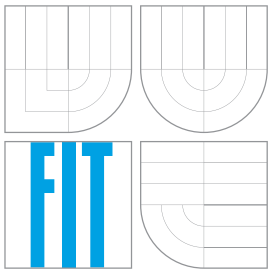
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PŘIPOJOVÁNÍ VESTAVĚNÝCH SYSTÉMŮ K INTRANETU

CONNECTING EMBEDDED SYSTEMS TO INTRANET

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN DAMBORSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. MIROSLAV ŠVÉDA, CSc.

BRNO 2009

Zadání práce

1. Prostudujte principy a architektury připojování vestavěných systémů k Internetu.
2. Navrhněte architekturu připojení kompatibilní se standardem IEEE 1451.1.
3. S využitím vývojového prostředku Ubicom resp. jeho obdoby navrhněte a implementujte prototyp programového vybavení komunikace vestavěného systému s rozhraním podle IEEE 1451.1 přímo připojeným na Intranet/Ethernet. Toto programové vybavení bude sestávat z prototypů klienta a WWW serveru s applety v jazyce Java a z programového vybavení uzlu Ubicom (resp. jeho obdoby) v jazyce C/C++.
4. Programové vybavení ověřte na modelové aplikaci.
5. Zhodnoťte dosažené výsledky a navrhněte možné pokračování projektu.

Licenční smlouva

Abstrakt

Práce zkoumá možnosti připojování vestavěných systémů k intranetu. Bere v úvahu možné technologie a prostředky použitelné k řízení a sběru dat vzdálených senzorů. Zabývá se možnostmi standardu IEEE 1451.1, který definuje abstraktní rozhraní převodníku pro připojování čidel k sítím, aniž by specifikoval konkrétní implementaci komunikace mezi těmito entitami. Dále práce pokrývá návrh a implementaci systému založeného na architektuře popsané standardem IEEE 1451.1.

Abstract

This thesis studies possibilities of connecting embedded systems to intranet. Available technologies and resources usable for control and capturing of data on remote sensors are considered. It goes into IEEE 1451.1 standard, which defines abstract interface of transducer for sensors connecting, while avoiding specification of concrete implementation of communication between these entities. Design and implementation of application with architecture based on IEEE 1451.1 standard takes also place in this thesis.

Klíčová slova

vestavěný systém, internet, intranet, převodník, chytrý sensor, ieee 1451.1, ubicom

Keywords

embedded system, internet, intranet, transducer, smart sensor, ieee 1451.1, ubicom

Citace

Roman Damborský: Připojování vestavěných systémů k intranetu, diplomová práce, Brno, FIT VUT v Brně, 2009

Připojování vestavěných systémů k intranetu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval zcela samostatně pod vedením pana Prof. Ing. Miroslava Švédy, CSc.

.....
Roman Damborský
25. května 2009

Poděkování

Děkuji Prof. Ing. Miroslavu Švédovi, CSc., vedoucímu mé diplomové práce, za zájem a věcné připomínky k tvorbě a organizaci práce. Také děkuji rodině a přátelům za podporu během studia.

Obsah

Kapitola 1

Úvod

Vestavěné systémy se staly součástí každodenního lidského života, lze je nalézt ve všech odvětvích lidské činnosti. Stejně tak se do lidského prostředí prosadil Internet, sloužící jako takřka neomezený komunikační a informační nástroj. Není těžké představit si možnosti, které nabízí spojení technologií Internetu a vestavěných systémů. Usnadnění práce, komunikace a informační hodnota takového sloučení je jistě přínosná.

Možnost ovládat zařízení ve výrobní hale, monitorování stavu ventilů v rozsáhlém komplexu nebo zapisování údajů z měřících čidel v laboratořích – to vše jistě vyžaduje dávku úsilí a vynaložení nezbytné energie. Proč tedy tyto činnosti nezjednodušit a pro člověka je nezpříjemnit. Právě k tomuto účelu lze využít vestavěné systémy a Internet nebo Intranet. Pracovník, dříve nucený absolvovat zdlouhavé obchůzky nebo zapisovat množství údajů tak může na vše pouze dohlížet z tepla kanceláře. Veškerou tuto činnost za něj může obstarat vhodná kombinace hardwaru a softwaru.

Protože si ale lze spojení "vhodná kombinace" vysvětlit různě, vznikají často vzájemně odlišné realizace těchto systémů a jejich částí. Jednotliví výrobci vyrábí nejrůznější čidla, senzory a akční členy a používají při tom vlastní specifikace rozhraní, pracovních podmínek apod. Stejně tak vznikají rozdíly v oblasti softwaru pro tyto systémy.

Aby bylo možné pohodlněji sestavovat řídící a kontrolní systémy s využitím libovolného softwaru a kombinací hardwaru, vznikl standard IEEE 1451. Jeho cílem je sjednotit aplikační rozhraní a zavést základní bloky pro zvýšení kompatibility jednotlivých částí systému, s ohledem na snadnou údržbu a rozšiřitelnost.

Předmětem této práce je navrhnout a implementovat architekturu postavenou na standardu IEEE 1451.1, která umožní připojení vestavěného systému na komunikační síť Ethernet a tím zpřístupnit ovládání a data vestavěného systému v rámci Internetu resp. Intranetu.

Do problematiky vestavěných systému uvede čtenáře kapitola 2, kde jsou vysvětleny základní pojmy a rysy vestavěných systémů, počítačových sítí a přenosových technologií. V kapitole 3 je představena architektura převodníku jako vestavěného systému a možné metody připojení převodníků a vestavěných systémů obecně do sítě Internet. Kapitola 4 se zabývá přiblížením a rozborem standardu IEEE 1451 a je charakterizován pojem chytrý senzor. Dále jsou v této kapitole uvedeny architektury komunikačních systémů standardu IEEE 1451.1 a přiblížení implementace tohoto standardu. Návrh připojení vestavěného systému k intranetu, tedy hlavní předmět této práce, je rozebrán v kapitole 5. Kapitola 6 popisuje detaily implementace navrženého systému na zadaných platformách. V závěru práce (kap. 7) jsou shrnuty dosažené výsledky a možné pokračování diplomové práce.

Kapitola 2

Přiblížení problematiky

V následujících podkapitolách jsou vysvětleny základní pojmy, týkající se tématiky práce.

2.1 Vestavěné systémy

Vestavěný systém je počítačový systém, navržený pro řízení jedné nebo několika málo specifických úloh. Často se jedná o součást jiného, robustnějšího systému, ať už z hlediska hardwaru nebo softwaru. Protože je vestavěný systém navrhován pro přesně definovanou sadu úloh, je možné provádět množství optimalizací, které umožňují snížit cenu systému a zmenšit jeho rozměry a naopak zvýšit spolehlivost a výkonnost. Je-li navíc vestavěný systém vyráběn ve velkých sériích, zvyšuje se i jeho ekonomický význam pro výrobce [13].

Vestavěné systémy našly uplatnění ve všech oblastech lidské činnosti. Lze je najít v dopravních prostředcích, spotřební elektronice, medicíně, potravinářském průmyslu atd. Jako vestavěný systém lze označit ABS systém v automobilech, MP3 přehrávač, systém pro řízení světelných křižovatek nebo kontrolní systém výrobní linky apod. Složitost těchto systémů je různá, od nejjednodušších jednočipových zařízení po velmi složité sestavy s několika mikrokontroléry a periferiemi v jednom celku.

Oproti osobním počítačům mají vestavěné systémy další typické vlastnosti:

- mohou pracovat v reálném čase – tzn. je definována nejhorší možná doba odezvy systému na výskyt určité události.
- vyšší nároky na bezpečnost (existence záložních systémů, automatické zotavení, patřičná indikace chybového stavu)
- zajištění odolnosti na okolní vlivy (práce v extrémních klimatických podmínkách, odolnost proti rušení a zároveň zabránění neúmyslného ovlivňování okolí)

Obecně nelze pojem *vestavěný systém* přesně definovat. Každé zařízení je možné zkoumat z několika hledisek, přičemž ne všechna musí zcela zapadat do uvedené formulace vestavěného systému. Příkladem mohou být současné mobilní telefony, jejichž základní funkce mohou být rozšiřovány instalací nových aplikací na softwarové úrovni.

2.1.1 Distribuované vestavěné systémy

V mnoha situacích je žádoucí, aby vestavěné systémy komunikovaly s dalšímy systémy ve svém okolí. Může se jednat např. o nejrůznější kontrolní terminály, kde je zobrazován souhrn měření několika čidel nebo analýzy jiných systémů.

Toto řešení přináší řadu výhod, zejména:

- jednodušší údržba a rozšiřitelnost
- nezávislost jednotlivých částí
- modularita
- skrytí detailů konstrukce a implementace a jejich omezení na rozhraní typu vstup/výstup

Nevýhodami takových systému jsou:

- složitější instalace (je nutné uvažovat rozvody kabeláže)
- synchronizace jednotlivých komponent

Vestavěné systémy spolu mohou komunikovat pomocí různých technologií, uvedených v následující kapitole.

2.1.2 Síťové komunikační technologie

Rozdělením zátěže a zodpovědnosti mezi více systémů vyžaduje zajištění spolehlivé komunikace. Navíc se může vyskytnout situace, kdy různé podsystémy pocházejí od různých výrobců a každý z nich vyvynul svůj systém v jiném vývojovém prostředí. Vzniká tak nutnost sjednotit formát rozhraní poskytovaný pro komunikaci mezi podsystémy vzájemně i mezi podsystémy a hlavní stanicí.

V současné době je dostupné množství standardů a technik, jak tuto komunikaci řešit na softwarové úrovni. Uvedeny jsou nejznámější technologie spolu s krátkou specifikací:

1. **CORBA** – mechanismus využívající jazyk IDL ke specifikaci rozhraní pro přístup k objektům. Pomocí nástrojů prostředí CORBA se z IDL definice vygeneruje mapování na specifický programovací jazyk, v němž je pak implementována potřebná funkcionalita. Server tak může být implementován v jiném jazyce než klient a naopak. Komunikace mezi dvěma systémy je zprostředkována ORB objektem. Současná implementace CORBA podporuje množství programovacích jazyků, např. C, Java, Smalltalk, Perl, Ruby... První verze CORBA 1.0 byla vydána v roce 1991 [8].

2. **RPC** – umožňuje volat vzdálené funkce, implementované v jiném adresovém prostoru. Programátor používá funkce jako by byly lokální – nemusí se zabývat detaily interakce. Rozhraní vzdálených metod je popsáno pomocí IDL jazyka.

3. **Java RMI** – programové rozhraní Javy ekvivalentní RPC. Původní implementace byla závislá na JVM a umožňovala volat vzdálené metody pouze mezi JVM. Novější implementace (Jini) využívá složitější architekturu, tvořenou klientem, serverem a vyhledávací službou. Server poskytuje služby, které jsou zaregistrované ve vyhledávači. Klient požádá vyhledávač o danou službu a dostane se mu informace, jak se ke službě připojit.

4. **.NET Remoting** – technologie firmy Microsoft, jejíž známějšími předchůdci byli OLE a COM. .NET Remoting umožňuje zpřístupnit objekt pro vzdálená volání z jiných aplikací, domén a počítačů. Požadavky na objekty jsou zprostředkovány pomocí kanálových objektů, které zapouzdřují aktuální přenosový režim, včetně TCP nebo HTTP. Správnou inicialiací kanálů lze tak přepnout přenosový režim na jiný protokol, bez nutnosti opětovné kompilace aplikace.

5. **Web Service** – nejčastěji dostupný prostřednictvím Web API na síti Internet. Komunikace probíhá pomocí HTTP protokolu. Zprávy jsou ve formátu XML dokumentu.

2.2 Internet

Internet je celosvětový globální systém vzájemně propojených počítačových sítí, umožňující výměnu dat mezi těmito sítěmi a počítači v nich [14]. Jednotlivé sítě slouží k různým účelům – sítě vzdělávacích institucí, vládní sítě – a jsou buď veřejně dostupné nebo privátní. Internet poskytuje velké množství informací a širokou škálu služeb, dostupných prostřednictvím specifických protokoků. Internet je založen na protokolu IP 2.4.1.

Internet sám o sobě je sítí veřejně dostupnou. Připojení k internetu nabízí poskytovatelé internetu jako placenou službu, nebo je dostupné prostřednictvím tzv. freepointů, poskytovaných městy, případně soukromými osobami.

2.3 Intranet

Intranet je privatní počítačová síť, využívající technologie internetu k bezpečnému sdílení informací organizace mezi zaměstnanci. Může se jednat o formu interních www stránek, častěji však jde o kompletní systém spojený s počítačovým systémem organizace. Intranet je postaven na stejných technologiích jako internet. Oproti internetu poskytuje ale jen omezený nebo pevně stanovený rozsah služeb, může poskytovat služby vlastní (proprietální) a vyžadovat různě stupně oprávnění k jejich použití (další srovnání v tab. 2.1).

Intranet lze také chápat jako privátní verzi internetu, nebo jako soukromé rozšíření internetu ve prospěch organizace [15].

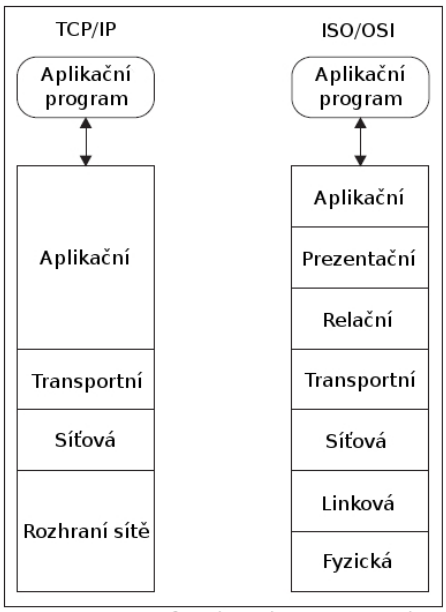
Vlastnost	**Internet**	**Intranet**
služby	neomezené	specifikovány organizací
potřebné oprávnění	žádné/minimální	údaje zaměstnance, veřejně nepřístupné
cílová skupina	neomezená	zaměstnanci organizace
vyžadovaná spolehlivost	nízká	vysoká (např. bankovní systémy)
kontrolovatelnost	minimální/žádná	vysoká (IT expertem organizaci)

Tabulka 2.1: Srovnání vlastností internetu a intranetu [3].

2.4 Komunikace v sítích

2.4.1 TCP/IP

TCP/IP model je čtyřvsrtvý model sloužící ke zpracování dalšími normami, zabývajícími se propojováním a komunikací mezi systémy. Je spravován organizací IETF a je na něm založen koncept internetu. Při přenosu informací spolu pak komunikují vždy odpovídající vrstvy, pomocí svých vlastních funkcí, definovaných standardem.



Obrázek 2.1: Vrstvový model architektury TCP/IP.

Jednotlivé vrstvy modelu TCP/IP (obr. 2.1):

1. **Aplikační vrstva** je nejblíže uživateli. Zajišťuje dostupnost informací pro přenos, přípravu dat pro přenos a základní kryptovací funkce. Typickými protokoly pracujícími v této vrstvě jsou aplikační protokoly Telnet, SNMP, FTP, SMTP a další.

2. **Transportní vrstva** je zodpovědná za doručení dat od zdroje k cíli, nezávisle na použitém médiu. Zajišťuje také kontrolu chyb, fragmentaci a řízení toku. Data přijatá z aplikační vrstvy zabalí do paketu a podle typu spojení s ním dále pracuje. Z pohledu TCP/IP se rozlišují dva způsoby komunikace mezi síťovými zařízeními:

   - **Spojovaný přenos** – využívá TCP pakety (viz kap. 2.4.2)
   - **Nespojovaný přenos** – pomocí UDP datagramů (viz kap. 2.4.3)

3. **Síťová vrstva** (také nazývaná Internetová) řeší problém jak přenést pakety v rámci jedné sítě (v rámci jediné domény). Příkladem protokolů této vrstvy jsou protokoly IP, ICMP, ARP, OSPF a další.

4. **Rozhraní sítě** je nejnižší vrstvou. TCP/IP model je navržen jako hardwarově nezávislý. Tato vrstva je tedy specifická pro každou síť a použitý hardware, v TCP/IP tedy není blíže specifikována. Zajišťuje přenos dat mezi dvěma body např. s využitím překladu MAC adres u technologie Ethernet.

2.4.2 TCP

TCP protokol je jedním ze základních internetových protokolů. Je podle něj pojmenován i model TCP/IP, což značí jistou důležitost. Zatímco IP protokol tvoří spojení mezi všemi počítači v rámci Internetu, přes které se přenáší určitá zpráva, TCP pracuje na vyšší úrovni a bere v potaz pouze zdrojový a cílový počítač (např. web server a internetový prohlížeč).

TCP poskytuje spolehlivé spojení a zajišťuje správné pořadí příjmu dat mezi aplikačními programy. TCP zprostředkovává požadavky aplikační vrstvy IP protokolu např. tím, že rozděluje velké objemy dat, které aplikace odesílá příjemci. Aplikace tedy může zavolat pouze jeden požadavek k přenosu, namísto složitějšího rozdělování dat a opakování požadavku pro každý segment.

Vlivem nepříznivých faktorů, působících v síti, může dojít k zahlcení nebo jinému nečekanému chování a přenášený paket se ztratí, nebo je doručen mimo pořadí. TCP je schopný tyto problémy detekovat a sám chyby napravovat. Dokáže znovu odesílat ztracené pakety nebo přeuspořádat pakety, které byly doručeny mimo pořadí. Po úspěšném sestavení přijatých dat jsou tato předána vyšší vrstvě ke zpracování. TCP tedy poskytuje abstrakci nad detaily komunikace. TCP je využíván mnoha Internetovými aplikačními protokoly, např. HTTP, SMTP, FTP, SSH a některými multimediálními streamovacími aplikacemi. Přenášená data jsou čtena jako *stream* neboli nepřetržitý tok dat (není stanoveno, kde jeden paket začíná a kde končí).

TCP se využívá pro spojovanou komunikaci (tzv. handshake), kdy vysílací strana zašle žádost o navázání spojení, příjemce potvrdí nebo odmítne a vysílací strana podle odpovědi buď začne data posílat nebo se o navázání spojení pokusí později. Po přenesení dat dochází k ukončení spojení. Tato komunikace před a po vlastním přenosu zajišťuje spolehlivější službu, na druhou stranu ale zatěžuje přenosové pásmo a dochází ke zpomalení komunikace. Další spoždění vznikne, musí–li se čekat na chybějící pakety, které se buď ztratily, nebo byly poslány jinou trasou a mají tak větší zpoždění. Proto je dostupný alternativní přenosový protokol UDP.

2.4.3 UDP

Pomocí UDP může aplikace poslat zprávy, označované jako datagramy, jiným aplikacím v síti bez nutnosti ustavování spojení, tak jako tomu je u TCP protokolu.

UDP používá jednoduchý přenosový model bez implicitní hand-shake procedury pro zajištění spolehlivosti, správného pořadí doručování datagramů a integrity. UDP tedy poskytuje nespojované a nespolehlivé spojení. Datagramy mohou přicházet mimo pořadí, mohou se objevovat duplikáty nebo mohou datagramy chybět a příjemce o tom nemusí vědet. UDP předpokládá, že kontrola chyb a jejich opravy nejsou nutné, nebo jsou prováděny aplikací. Tím se značně uvolní přenosovému pásmu a přenos je ve výsledku rychlejší. Aplikace citlivé na včasnost doručení často používají k přenosu dat právě UDP a dávají tak přednost zahození chybného datagramu před jeho opětovným ale pozdním doručením.

Bezstavovost UDP je také vhodná pro servery, které vysílají malé množství zpráv velkému počtu klientů. UDP je tedy použito pro zajištění broadcastu (rozeslání všem stanicím na síti) a multicastu (rozeslání registrovaným stanicím). Při přenosu nedochází k potvrzování přijatých datagramů nebo k ověření zda je příjemce schopný přijmout další datagram (kontrola zahlcení). Data jsou odesílána a čtena individuálně, mají tedy pevně určen počátek a konec bloku v celkové přenášené zprávě.

Mezi běžné služby využívající UDP patří DNS, VoIP, TFTP a další.

2.4.4 Adresace v TCP/IP

- **Aplikační vrstva** – adresace je aplikačně specifická – identifikace služeb např. e-mail adresou, www adresou, ftp serverem. Většinou formou URI.
- **Transportní vrstva** – identifikace služby na konkrétním počítači pomocí čísla portu. Číslování portů je rozděleno do několika skupin:
  1. specifické služby – 1 až 1023
  2. registrované – 1024 až 49151
  3. dynamické/privátní – 49152 až 65535
- **Síťová vrstva** – identifikace počítače/stroje v rámci sítě nebo Internetu pomocí IP adresy. Jedná se o 4 bajtové hodnoty, členěny podle tabulky 2.2. Mimo uvedené rozsahy se navíc používají speciální adresy, např. 127.0.0.1 pro rozhraní loopback a adresy ve formátu X.X.X.255 pro broadcast.
- **Rozhraní sítě** – adresace probíha na základě MAC adresy, což je 6 bajtová hodnota, zapisovaná v hexadecimální formě. Např. 00:1b:4c:22:7f:a3

Třída	**Rozsah**	**Maska podsítě**
A	1.0.0.1 – 126.255.255.254	255.0.0.0
B	128.0.0.1 – 191.255.255.254	255.255.0.0
C	192.0.0.1 – 223.255.255.254	255.255.255.0
D	224.0.0.1 – 239.255.255.254	–
E	240.0.0.1 – 254.255.255.254	–

Tabulka 2.2: Rozsahy IPv4 adres.

Kapitola 3

Způsoby připojení vestavěných systémů do Internetu

Cílem této práce je popsat a demonstrovat způsoby připojení vestavěných systémů do Internetu. V následujících podkapitolách bude vysvětleno několik přístupů, opírajících se o standard IEEE 1451 (viz kap. 4).

3.1 Převodníky

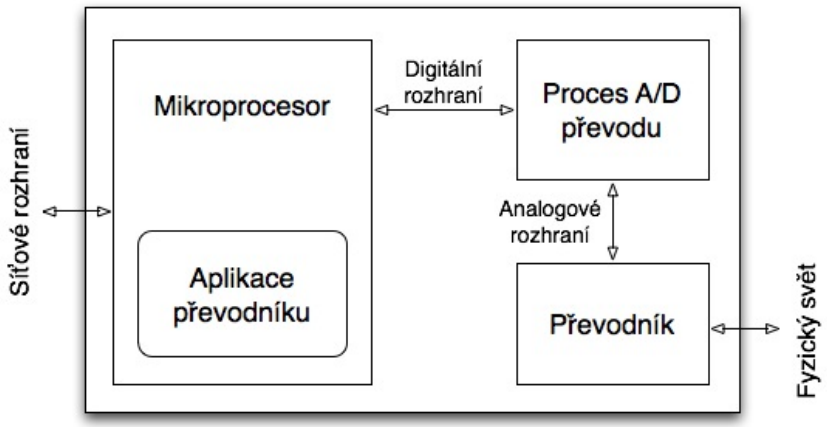
Převodník je zařízení, umožňující převod z jedné formy informace/energie do jiné. Mezi převodníky patří také senzory a akční členy. Senzor (nebo také čidlo) je schopno převést určitou fyzikální veličinu na elektrický signál, který je možné dále zpracovávat (např. měření teploty). Akční člen naopak pracuje v opačném směru, tj. převádí elektrickou informaci např. na práci (příkladem může být krokový motor).

Převodník komunikující s okolím pomocí komunikační sítě je tvořen mikroprocesorem, schopným převodu mezi elektrickými signály senzoru či akčního členu na signály použitelné pro komunikaci (obecné schéma převodníku je na obr. 3.1). Mezi nejznámější komunikační sítě patří ProfiBus, CAN, FlexRay a DeviceNet, z nichž každá má své specifické vlastnosti, výhody i nevýhody. Součástí převodníků je zpravidla i AD/DA převodník, zajišťující převod mezi analogovým a digitálním signálem a naopak.

Do oblasti vestavěných systémů se v posledních letech prosadil i Ethernet a bezdrátový Ethernet. Stále větší oblibě se těší bezdrátová komunikace mezi vestavěnými systémy, založená na optickém nebo rádiovém přenosu (např. IrDA, Bluetooth, ZigBee).

Převodník může být realizován čistě hardwarově (s pevným programem) nebo se může jednat o *inteligentní převodník*, umožňující např. konfiguraci převodu, identifikaci převodníku, diagnostiku chyb a stavu převodníku apod. Inteligentní převodník je vybaven programovatelnou pamětí a programem, který umožňuje výše uvedené funkce. Inteligentní převodník tak poskytuje možnost ovládání převodu ze sítě. Ve své podstatě tak může být jako převodník chápán i pouhý program v počítači [1].

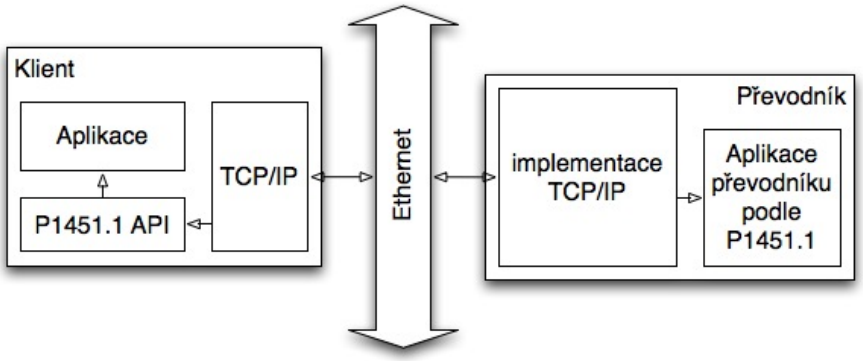
Protože neexistuje monopol na výrobu hardware a software, dochází k produkci převodníků, používajících různá rozhraní a komunikační protokoly. Pro případ, kdy je třeba připojit vzájemně nekompatibilní rozhraní převodníků, vznikl právě standard IEEE 1451. Jeho účelem je specifikovat množinu rysů a rozhraní – které v případě, že je převodník implementuje – zajistí možnost kompatibilního připojení převodníku s původně nesourodým rozhraním.



Obrázek 3.1: Obecné blokové schéma převodníku.

3.2 Přímé připojení

Aby bylo možné připojit převodník přímo k internetu nebo intranetu, musí zařízení implementovat rozhraní podle referenčního modelu ISO/OSI nebo TCP/IP, nebo alespoň vrstvy od fyzické po transportní, která poskytuje komunikaci pomocí TCP nebo UDP spojení. Je také nutné přidělit zařízení jednoznačnou IP adresu a port, na kterém budou dostupné služby převodníku. Komunikace poté probíhá přímo mezi klientem a převodníkem. Zapojení ilustruje obr. 3.2.



Obrázek 3.2: Přímé připojení převodníku k internetu.

Mezi výhody této architektury patří:

- rychlost komunikace (není zde žádný mezičlánek, který služby zprostředkovává)
- jednoduchost implementace (převodníky mají zpravidla implementovány potřebné vrstvy od výrobce, stačí tedy nastudovat jejich API rozhraní)
- robustnost systému (funkčnost implementovaných vrstev je zajištěna výrobcem)

Nevýhodou tohoto způsobu připojení je především ztráta možnosti přizpůsobit převodník použité konfiguraci komunikace. Dojde–li např. ke změně použitého šifrování přenosového média, není možné převodník jednoduše přeprogramovat.

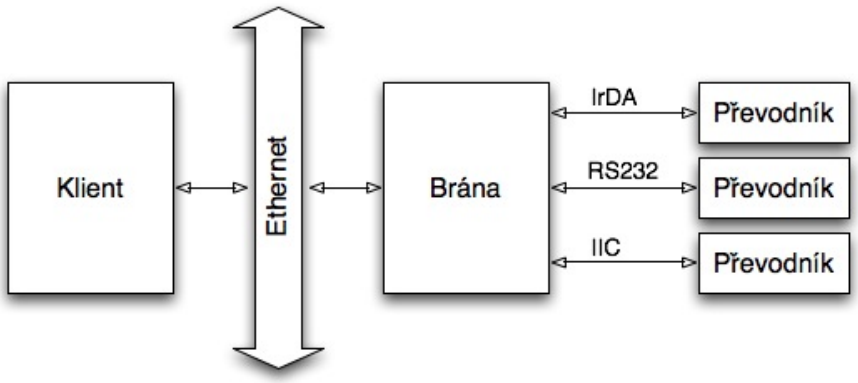
3.3 Připojení s využitím brány

Připojení pomocí brány znamená, že mezi převodník (v roli severu) a klienta (který zpracovává informace od převodníku) je vložen mezičlánek. Ten zprostředkovává komunikaci mezi oběma prvky.

Výhodou prostředníka je možnost jeho přeprogramování, změní–li se vlastnosti komunikační sítě. Můžeme tedy docílit zapouzdření jednotlivých senzorů a akčních členů, připojených k převodníku. Je pak možné použít jednotné rozhraní pro přístup k bráně, zprostředkovávající komunikaci s několika převodníky, z nichž každý může s bránou komunikovat pomocí vlastního rozhraní.

Nevýhodou této architektury je snížení rychlosti komunikace v důsledku zpracování a redistribuce dat bránou. Implementace brány je také složitější, jelikož je často nutné explicitně implementovat nižší vrstvy TCP/IP nebo ISO/OSI modelu. Obtížnější je také testování a je vyšší pravděpodobnost výskytu chyb.

Ilustrace zapojení s využitím brány je na obr. 3.3.



Obrázek 3.3: Připojení převodníku k internetu pomocí brány.

3.3.1 Brána zastupující převodník

Brána v tomto zapojení představuje zástupce daného převodníku. Klient, vyžadující informace nebo posílající příkazy převodníku, nemusí vůbec tušit že pracuje s prostředníkem. Brána se totiž chová jako plnohodnotný převodík a veškeré požadavky přizpůsobí a přesměruje na konkrétní převodník. Ten může být k bráně připojen pomocí libovolného protokolu nebo média (např. RS232, IIC, IrDA...). Odpověď od převodníku pak brána předá klientovi, jako by ji předával samotný převodník přímo. Brána pracuje zpravidla v aplikační vrstvě, výjimečně ve vrstvách nižších.

Tato architektura najde uplatnění v situacích, kdy je převodník již připojen k jiné síti, ale je vyžadováno jeho zpřístupnění z Internetu.

3.3.2 Brána zastupující klienta

V tomto případě brána nezastupuje jednotlivé převodníky, ale klienta. Brána pravidelně komunikuje s převodníky a získává od nich patřičné údaje. Ty pak podle požadavků klientů rozesílá žadatelům s tím, že může provést jistou formu předzpracování. Klienti ale nemají přímý přístup k datům převodníků nebo k jejich nastavení. Plyne z toho jisté omezení, které je však z pohledu připojení do Internetu přínosem – klient nemůže přímo ovlivňovat nastavení převodníku. Může získat pouze údaje, které mu náleží. Tím pádem je možné zajistit vyšší bezpečnost systému, ovšem za cenu snížení komfortu při konfiguraci převodníků.

Kapitola 4

Standard IEEE 1451.1

4.1 Standard IEEE 1451

Standard IEEE 1451 nazvaný Standard for a Smart Transducer Interface for Sensors and Actuators (Standard pro rozhraní chytrých převodníků pro senzory a akční členy) popisuje sadu otevřených, síťově nezávislých komunikačních rozhraní pro připojování snímačů (senzorů nebo akčních členů) k mikroprocesorům a řídícím nebo kontrolním systémům. Klíčovou vlastností této rodiny standardů je definice tzv. TEDS. TEDS je paměťové médium připojené ke snímači, které uchovává identifikaci, kalibrační údaje, rozsahy pro měření a výrobní informace pro konkrétní snímač. Cílem standardu 1451 je povolit přístup k datům snímače prostřednictvím obecné sady rozhraní, ať už je snímač připojen do systému nebo sítě pomocí kabelů nebo bezdrátově [7].

Pod pojmem chytrý senzor lze chápat senzor, který má následující vlastnosti [1]:

- je popsán pomocí strojově čitelného TEDS popisu
- řízení a data související se snímačem jsou digitální
- dotazování a řízení slouží k podpoře správné funkce snímače

4.1.1 Rodina standardů 1451.X

Standard je rozdělen do několika samostatných částí, z nichž každá popisuje určitou část komunikačního řetězce, od rozhraní sítě po rozhraní k fyzickému senzoru nebo akčnímu členu. Jednotlivé části standardu IEEE 1451 jsou popsány v následujících sekcích. Ilustrace vztahů mezi jednotlivými sekcemi standardu (komunikační řetězec) je uvedena v kapitole 4.1.2.

IEEE P1451.0

Definuje sadu obecných příkazů, operací a TEDS pro rodinu protokolů IEEE 1451. Pomocí těchto příkazů je možné přistupovat na jakýkoli senzor nebo akční člen v sítích podporujících standard 1451, ať už metalických nebo bezdrátových. Funkcionalita je nezávislá na fyzickém komunikačním médiu mezi snímačem a síťovým uzlem NCAP. To umožní mnohem jednodušší přidání standardů 1451.X popisujících fyzickou vrstvu podle vývoje technologií v této oblasti [7].

Tento standard je ve fázi vývoje.

IEEE 1451.1

Definuje obecný objektový model popisující chování chytrých snímačů. Definuje model měření který zjednodušuje a sjednocuje měřící procesy. Také specifikuje komunikační modely použité standardem, které zahrnují klient–server a publish–subscribe modely. Aplikační software běžící v NCAP založený na IEEE 1451 komunikuje se snímačem přes různé fyzické vrstvy určené standardy IEEE 1451.X tak, jak to vyžaduje konkrétní aplikace. Komunikace mezi NCAP entitami a komunikace do systémů vyšších úrovní je podporována způsobem, který je síťově neutrální [7].

Standard je schválený a dostupný u organizací IEEE a NIST. Tato část standardu je také hlavím předmětem této práce.

IEEE 1451.2

Definuje rozhraní mezi snímači a NCAP, a TEDS popisy pro konfiguraci spojení mezi dvěmi uzly. Originální standard popisuje komunikační vrstvu založenou na vylepšeném SPI s přidanými HW linkami pro řízení toku a časování [7].

Tento standard je v současné době rozšiřován o podporu sériového rozhraní UART.

IEEE 1451.3

Definuje rozhraní mezi snímačem a NCAP a dále TEDS pro zapojení s mnoha snímači použitím distribuované komunikační architektury. Umožňuje seskupit více snímačů do pole prezentovaného jako jeden uzel pomocí sdílení vodičů [7].

Standard je schválený a dostupný u organizací IEEE a NIST.

IEEE 1451.4

Definuje rozhraní pro vícerežimové analogové snímače s analogovým nebo digitálním operačním režimem. TEDS model byl upraven a umožňuje uložit minimální množství dat do malé fyzické paměti zařízení, tak jak je to vyžadováno menšími senzory. Pro popis TEDS jsou použity šablony, zahrnující např. akcelerometry, tenzometry, proudové smyčky, mikrofony a další [7].

Standard je schválený a dostupný u organizací IEEE a NIST.

IEEE P1451.5

Definuje rozhraní mezi snímačem a NCAP a dále TEDS pro bezdrátové snímače. Jako fyzické rozhraní tohoto standardu jsou uvažovány bezdrátové komunikační protokoly standardů 802.11 (WiFi), 802.15.1 (Bluetooth) nebo 802.15.4 (ZigBee) [7].

Tento standard je ve fázi vývoje.

IEEE P1451.6

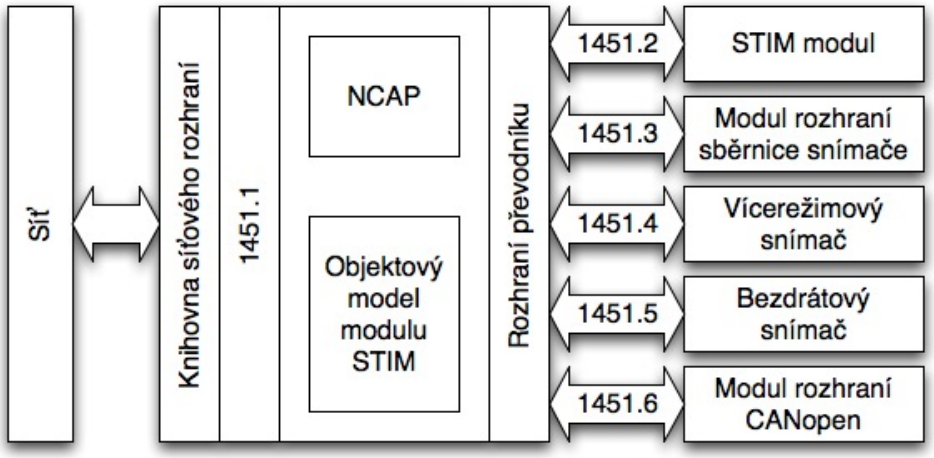
Definuje rozhraní mezi snímačem a NCAP a dále TEDS použitím vysokorychlostního CANopen síťového rozhraní. Specifikuje jak mapování 1451 TEDS na CANopen entity, tak i komunikační zprávy, zpracování dat, konfigurační parametry a diagnostické informace. Přizpůsobuje profil CANopen zařízení pro měřící zařízení a uzavřené kontroléry [7].

Tento standard je ve fázi vývoje.

4.1.2 Komunikační řetězec

Komunikační řetězec (ilustrace na obr. 4.1) se skládá ze dvou hlavních modulů. Prvním je modul STIM (samostatný snímač, schopný komunikovat přes TII rozhraní). Podle normy se může jednat obecně o jakýkoliv senzor, akční člen nebo jakékoliv digitální vstupy nebo výstupy. STIM modul může mít i větší počet nezávislých kanálů, z nichž každý může plnit jinou funkci.

Druhým článkem komunikačního řetězce je modul NCAP. Jedná se o síťový aplikační procesor tvořící prostředníka mezi snímačem a vnější komunikační sítí. Síť může být realizována jakoukoliv průmyslovou sítí (Ethernet, CAN, Profibus, DeviceNet, FlexRay a další). Úkolem NCAP procesoru je zajistit konverzi mezi TII a komunikační sítí [9].



Obrázek 4.1: Ilustrace vztahů rodiny standardů IEEE 1451.X a komunikačního řetězce.

4.2 Architektura systému podle IEEE 1451.1

Standard IEEE 1451.1 definuje softwarovou architekturu, aplikovatelnou na distribuované systémy skládající se z jednoho nebo více NCAP procesorů. Vestavěný systém pak komunikuje po síti a interakci s okolím zajišťuje NCAP procesor využívající připojený převodník. Standard poskytuje a definuje [2]:

- **síťovou abstraktní vrstvu** – je definováno rozhraní mezi aplikačními funkcemi NCAP procesoru a komunikační sítí bez ohledu na konkrétní detaily sítě.
- **převodníkovou abstraktní vrstvu** – standard definuje rozhraní mezi aplikačními funkcemi NCAP procesoru a převodníku, přepojeným k NCAP způsobem nezavislým na rozhraní ovladače převodníku.

Softwarová architektura je ve standardu IEEE 1451.1 definována třemi typy modelů:

- objektovým – pro softwarové komponenty
- datovým – pro popis informací přenášených přes specifikovaná rozhraní objektů
- dva síťové komunikační modely (klient/server a publish/subscribe)

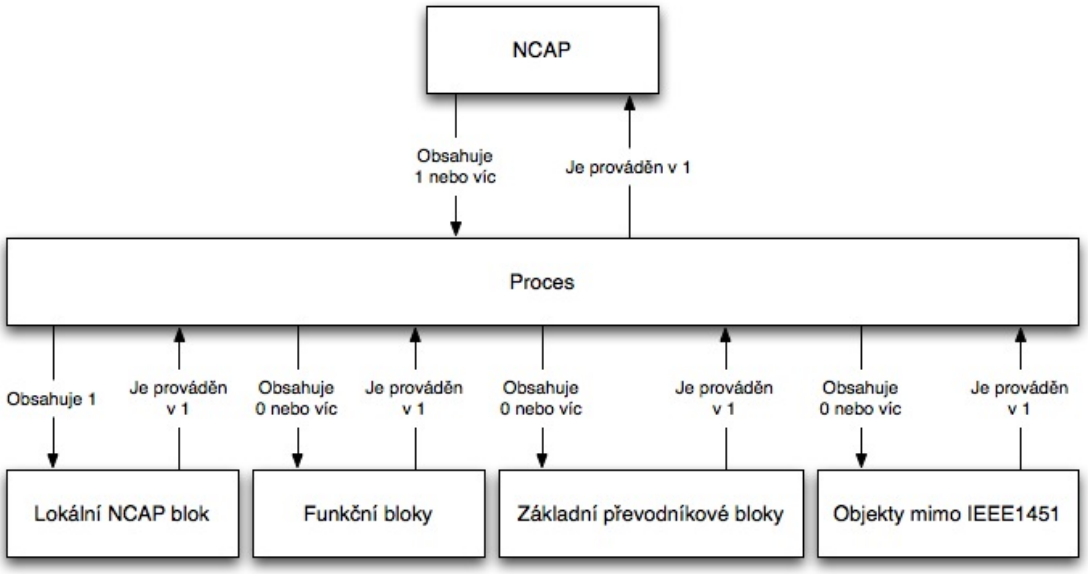
Jednotlivé modely jsou dále popsány podrobněji.

4.2.1 Objektový model

Objektový model specifikuje typy softwarových komponent použitých k návrhu a implementaci aplikačního systému (tj. skupina komponent vytvořená podle tohoto standardu). Model poskytuje základní stavební bloky pro aplikační systém. Objekt může být označen za objekt pouze tehdy, je-li jeho třída podtřídou `IEEE1451_Entity`. Aplikační systém je tvořen následujícími čtyřmi typy tříd:

1. **blokové třídy** – tvořeny třemi třídami
   - `IEEE1451_NCAPBlock` – poskytuje standardní softwarové rozhraní pro podporu síťové komunikace a systémové konfigurace
   - `IEEE1451_BaseTransducerBlock` – poskytuje standardní softwarové rozhraní mezi převodníkem a aplikačními funkcemi
   - `IEEE1451_FunctionBlock` – zapouzdřuje aplikačně specifickou funkcionalitu
2. **třídy komponent** – poskytují běžné aplikační konstrukce
   - strukturované informace (např. soubory)
   - kolekce souvisejících aplikačně specifických objektů
   - akce se stavem, jejichž provedení trvá relativně dlouhou dobu
3. **servisní třídy** – podporují
   - komunikaci mezi objekty na různých NCAP procesorech
   - synchronizaci celého systému
4. **objekty nepatřící do IEEE 1451.1** – veškeré objekty, které nejsou implementovány podtřídami hlavní třídy `IEEE1451_Entity`

Vztahy mezi NCAP procesorem, procesy a blokovými objekty jsou znázorněny na obrázku 4.2. Každý NCAP procesor v aplikačním systému obsahuje alespoň jeden softwarový proces (termín pouze popisuje adresový prostor, ve kterém jsou objekty v NCAP uloženy). V jednom NCAP procesoru může být několik aktivních softwarových procesů. Každý proces musí obsahovat alespoň jeden objekt třídy `IEEE1451_NCAPBlock` společně s případnými objekty tříd `IEEE1451_FunctionBlock`, `IEEE1451_BaseTransducerBlock` nebo objekty tříd nepatřících do IEEE 1451.1. Každý objekt v aplikačním systému je vždy přiřazen právě jednomu NCAP procesu.



Obrázek 4.2: Ilustrace vztahů mezi NCAP, procesy a blokovými objekty.

Identifikace objektu

K identifikaci objektů slouží následující vlastnosti [2]:

- **Class ID** – ID třídy objektu. Identifikuje pozici třídy v hierarchii tříd a pomáhá určit sémantiku objektu. Je stanoveno tvůrcem třídy a nemůže být změněno. Jedná se o unikátní hodnotu v rámci standardu.
- **Class Name** – Jméno třídy objektu. Poskytuje člověku přívětivější popis sémantiky třídy. Je stanoveno tvůrcem třídy a nemůže být změněno. Hodnota vlastnosti je unikátní v rámci standardu.
- **Object ID** – ID objektu. Je unikátní v rámci systému. Standard IEEE 1451 specifikuje několik algoritmů sloužících ke generování hodnot pro tuto vlastnost. Jednoznačně rozlišuje objekty. Není-li pevně přiděleno je generováno lokálním NCAP blokem objektu.
- **Object Tag** – Označení objektu. Definuje logický koncový bod pro serverovou stranu komunikace typu klient/server. Zpravidla je přiřazeno koncovým uživatelem jako součást procesu konfigurace systému. Mělo by být unikátní v rámci systému.
- **Object Name** – Jméno objektu. Poskytuje člověku bližší sémantický popis instance třídy. Je většinou stanoveno vývojářem IEEE 1451.1 komponenty nebo systémovým vývojářem. Je přiřazeno při tvorbě objektu.
- **Object Dispatch Address** – Vybavovací adresa objektu. Poskytuje jednoznačnou referenci na objekt v rámci síťové komunikace. Měla by být unikátní v běžícím systému a je závislá na typu komunikační sítě.

4.2.2 Datový model

Datový model specifikuje typ a formu informace přenášené přes rozhraní objektu při lokální i vzdálené komunikaci. Model je realizován implementací IEEE 1451.1 jako kolekce primitivních a strukturovaných datových typů.

Datové typy

Standard popisuje základní skupinu primitivních datových typů [2]:

- **Boolean** – logický datový typ nabývající hodnoty TRUE nebo FALSE
- **Octet** – 8 bitový datový typ bez číselného významu
- **Integer** – celočíselný typ - 8, 16, 32 nebo 64 bitové číslo, se znaménkem i bez znaménka
- **Floating point** – 32 nebo 64 bitové číslo s plovoucí desetinou čárkou
- **String** – řetězec znaků, nesoucí navíc informaci o použité znakové sadě

Dostupné jsou také strukturované datové typy.

- **Typy fyzikálních veličin** – navrženy pro použití v měřících a řídících systémech. Jsou to datové typy reprezentující čas, hodnotu fyzikální veličiny, doplňující informace o fyzikální veličině (metadata) jako např. jednotky.
- **Typy vlastností objektu** – obsahují informace o objektu. Jsou to typy `ObjectProperties`, `ClientPortProperties` a `PublisherInformation`.
- **Typy související se síťovou komunikací** – typy použité při řízení nebo filtrování skupinového vysílání (`PublicationTopic`, `SubscriptionQualifier` a `PubSubDomain`) a typy `Argument` a `ArgumentArray` sloužící pro zahrnutí jakéhokoliv výše uvedeného datového typu.

4.2.3 Síťové komunikační modely

Standard IEEE 1451.1 poskytuje 2 modely síťové komunikace mezi objekty v aplikačním systému:

- **těsně vázaný** – spojení bod-bod typu klient/server pro komunikaci pouze 2 objektů
- **volně vázaný** – model publish/subscribe pro spojení typu 1:N nebo M:N

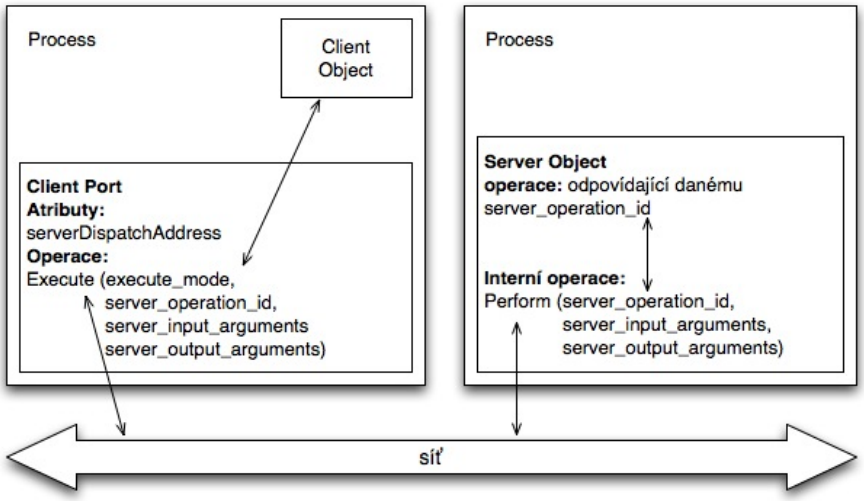
Komunikační modely definují syntaxi a sémantiku softwarového rozhraní mezi objekty aplikace a komunikační sítí. Standard ale nespecifikuje žádnou síťovou přenosovou syntaxi nebo protokol. Pro každou konkrétní síť se očekává, že bude dodán kód knihoven obsahující rutiny pro volání mezi IEEE 1451.1 operacemi a sítí.

Klient/server

Komunikační model Klient/Server je podporován dvěmi komplementárními operacemi na aplikační úrovni [2]:

- `Execute` – na straně klienta Client Port objekty
- `Perform` – na všech síťově viditelných objektech serverové strany

Operace `Execute` a `Perform` spolupracují a poskytují tak mechanismus vyvolání operace vzdáleného objektu (ilustrace na obr. 4.3). Postup, umožňující klientovi vyvolat vzdálenou operaci na straně serveru je popsán dále.



Obrázek 4.3: Komunikační model klient/server.

Při spouštění systému jsou svázány hodnoty `serverDispatchAddress` na straně klienta a vyřizovací adresa objektu na straně serveru. Během inicializace systému je pak klientovi předán odkaz na instanci objektu Client Port. Tím je zajištěno zpracování odesílaných a přijímaných dat.

Pokud chce klient provést vzdálenou operaci na straně serveru, zavolá operaci Execute, které předá id serverové operace a vstupní a výstupní parametry. S využitím síťové infrastruktury je zavolána operace Perform na straně serveru, která vyvolá požadovanou operaci s danými vstupními parametry.

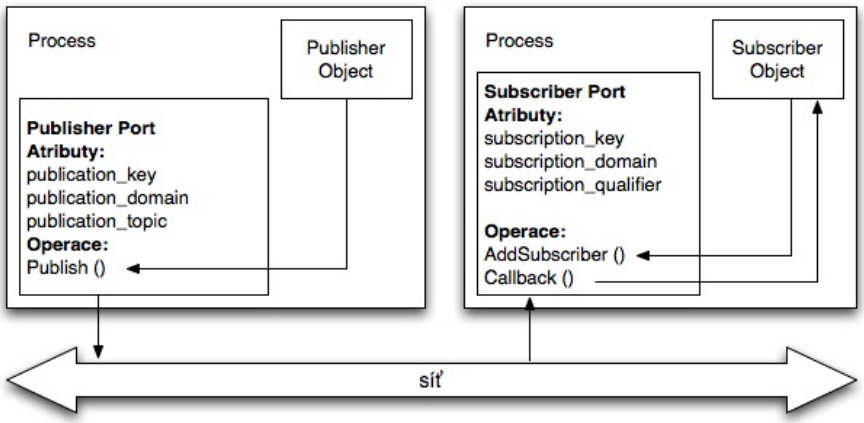
Jakmile operace na serveru skončí, vrací se výsledek zpět do funkce Perform, která prostřednictvím sítě předá výsledek klientovi. Na straně klienta se výsledek operace předá zpět do objektu klienta, který celou sekvenci komunikace inicioval.

Publish/subscribe

Komunikační model Publish/Subscribe je podporován dvěmi operacemi [2]:

- `Publish` – na straně vysílajícího objektu
- `AddSubscriber` – na straně přijímajících objektů spolu s přiřazenou callback funkcí

Každá publikace (vystavená žádost o data nebo provedení funkce) je v IEEE 1451.1 doplněna o další informace. Jde o publikační doménu (publication domain), definující rozsah distribuce pro publikaci. Dále je to publikační klíč (publication key) který identifikuje aplikačně nezávislou syntaxi a sémantiku publikace z pohledu vysílajícího objektu. A nakonec publikační téma (publication topic) které může být použito k identifikaci aplikačně specifické syntaxe a sémantiky obsahu publikace.



Obrázek 4.4: Komunikační model publish/subscribe.

Celý proces komunikace publish/subscribe modelu (obr. 4.4) lze rozdělit do tří částí.

1. Během inicializace systému je přiřazena hodnota publikačního klíče na straně vysílajícího objektu. Atribut přijímajícího klíče na straně příjemce je nastaven na hodnotu odpovídající hodnotě publikačního klíče pro žádanou publikaci. Jsou dokončeny inicializace vysílajících a přijímajících objektů.

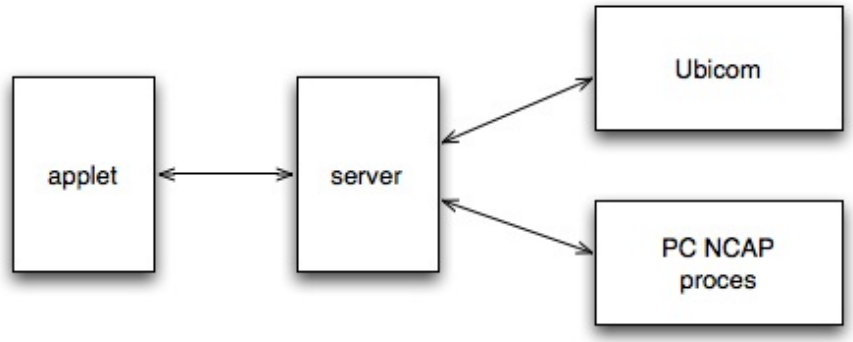
2. Při spouštění systému jsou přiřazeny hodnoty publikační domény a tématu u vysílajícího objektu. Jsou nastaveny hodnoty domény a kvalifikátoru přijímajícího objektu, čímž je určen typ publikací, které bude objekt přijímat a dále zpracovávat. Přijímající objekt zavolá funkci `AddSubscriber` pro zaregistrování do zpracování daných publikací. Této funkci je nutné předat jako argument referenci na funkci, která bude zavolána kdykoliv je přijata publikace určeného typu (tzv. callback funkce).

3. Kdykoliv po spuštění systému může vysílající objekt zavolat funkci `Publish`, s parametry obsahujícími data publikace. Pomocí síťové infrastruktury je publikace doručena na všechny odběratele v dané publikační doméně. Přijímající objekty pomocí svých atributů (klíč, doména a kvalifikátor) rozhodnou zda je doručená publikace zajímá, nebo je pro ně irelevantní. Pokud se jedná o žádanou publikaci, provede přijímací objekt všechny patřičné callback funkce, kterým předá jako vstupní parametr obsah publikace.

Kapitola 5

Návrh architektury připojení

Pro ověření možností poskytovaných standardem IEEE 1451.1 byl navržen systém s architekturou připojení podle obr. 5.1.



Obrázek 5.1: Architektura připojení uzlu Ubicom.

Klient je realizován appletem v programovacím jazyce Java, začleněným do www stránky. Ta je na požádání stažena z WWW serveru (viz 6.1.1) a zobrazena uživateli ve webovém prohlížeči. Klient prostřednictvím serveru (viz 5.1) komunikuje s uzlem Ubicom a dalšími NCAP procesy, běžícími na PC. Komunikace mezi klientem a NCAP procesy probíhá dle standardu IEEE 1451.1 – server je tedy z pohledu standardu neviditelný.

Navržené schéma je dostatečně obecné pro realizaci jakéhokoli systému, ať už konkrétní aplikace nebo aplikace ověřující standard samotný. Z praktického hlediska byla pro realizaci zvolena konkrétní aplikace, na níž je také možno lépe ověřit možnosti standardu IEEE 1451.1 v praxi.

Dále bude popsán návrh modelové aplikace, bude navržena cílová architektura aplikace a budou uvedeny části standardu, které tato aplikace využívá. Nakonec budou probrány postupy implementace standardu a aplikace a to jak pro část realizovanou v uzlu Ubicom, tak i pro aplikace v jazyce Java.

5.1 Modelová aplikace

Uvedená architektura připojení byla použita jako výchozí pro návrh a implementaci konkrétní modelové aplikace, realizující systém řízení vytápění rodinného domu.

Jednotlivé bloky tvořící aplikaci jsou následující:

- **Teplotní čidla** – Aplikace poskytuje uživateli přehled o venkovních a pokojových teplotách[1]. Venkovní teploty poskytuje NCAP proces Outside Sensors, pokojové teploty dodává NCAP proces Room Sensors. Oba tyto nezávislé procesy jsou umístěny na PC a jsou implementovány podle standardu IEEE 1451.1. Hodnoty jsou aplikaci dodávány periodicky formou publikací.
- **Řízení ventilů** – Pro regulaci teploty v místnostech jsou použity elektronicky ovladatelné ventily, které zajišťují průtok ohřáté vody do vytápěných místností. Ovládání ventilů zajišťuje další nezávislý NCAP proces označený jako Valves Control, implementovaný dle standardu IEEE 1451.1. Běží na PC a reaguje na žádosti o otevření nebo uzavření konkrétního ventilu. Při změně stavu některého z ventilů zasílá aplikaci souhrnnou informaci o aktuálním stavu všech ventilů a to formou publikace.
- **Databáze venkovních teplot** – Pro sledování venkovních teplot v průběhu roku slouží modul databáze, běžící jako NCAP proces na PC a přijmající publikace od procesu Outside Sensors. Jedná se o odběratele konkrétní publikace implementovaného dle standardu IEEE 1451.1. Přijaté hodnoty jsou pro jednoduchost zobrazeny pouze na výstup terminálu. Jejich další zpracování je nad rámec této práce.
- **Řízení bojleru** – Bojler slouží k ohřevu vody, obsahuje teplotní čidlo, zjišťující aktuální teplotu ohřívané vody a je schopný zahřát vodu na požadovanou teplotu. Zapínání a vypínání bojleru, stejně tak jako nastavování požadované teploty a zjišťování stavu bojleru je aplikaci přístupné prostřednictvím uzlu Ubicom. Řízení je implementováno dle standardu IEEE 1451.1 a je zpracováno v rámci NCAP procesu. Ten dále poskytuje pravidelné informace o aktuální teplotě vody formou publikace.
- **Klient** – Kompletní přehled a možnosti řízení vytápění jsou dostupné prostřednictvím appletu Control Station, umístěném na webové stránce. Tato stránka je stažena z WWW serveru a je tedy uživateli přístupná z libovolného místa v síti. Applet využívá knihovnu standardu IEEE 1451.1 a obsahuje NCAP proces, který zpracovává veškeré publikace v systému (pokojové a venkovní teploty, teplota vody v bojleru a stav ventilů) a poskytuje ovládací prvky pro řízení (ovládání ventilů, nastavování a zjišťování stavu bojleru).
- **Server** – Server je do systému vložen z důvodu omezení na straně appletu (viz 6.1.1). Z pohledu standardu se jedná o transparentní mezičlánek, z pohledu klienta a konkrétní implementace síťové komunikace jde však o zásadní prvek. Tento proces běží na PC a nevyužívá žádnou část standardu.

[1] Prezentované hodnoty nejsou reálné, jedná se o náhodně generované údaje. Z pohledu návrhu a implementace nejsou konkrétní hodnoty podstatné.

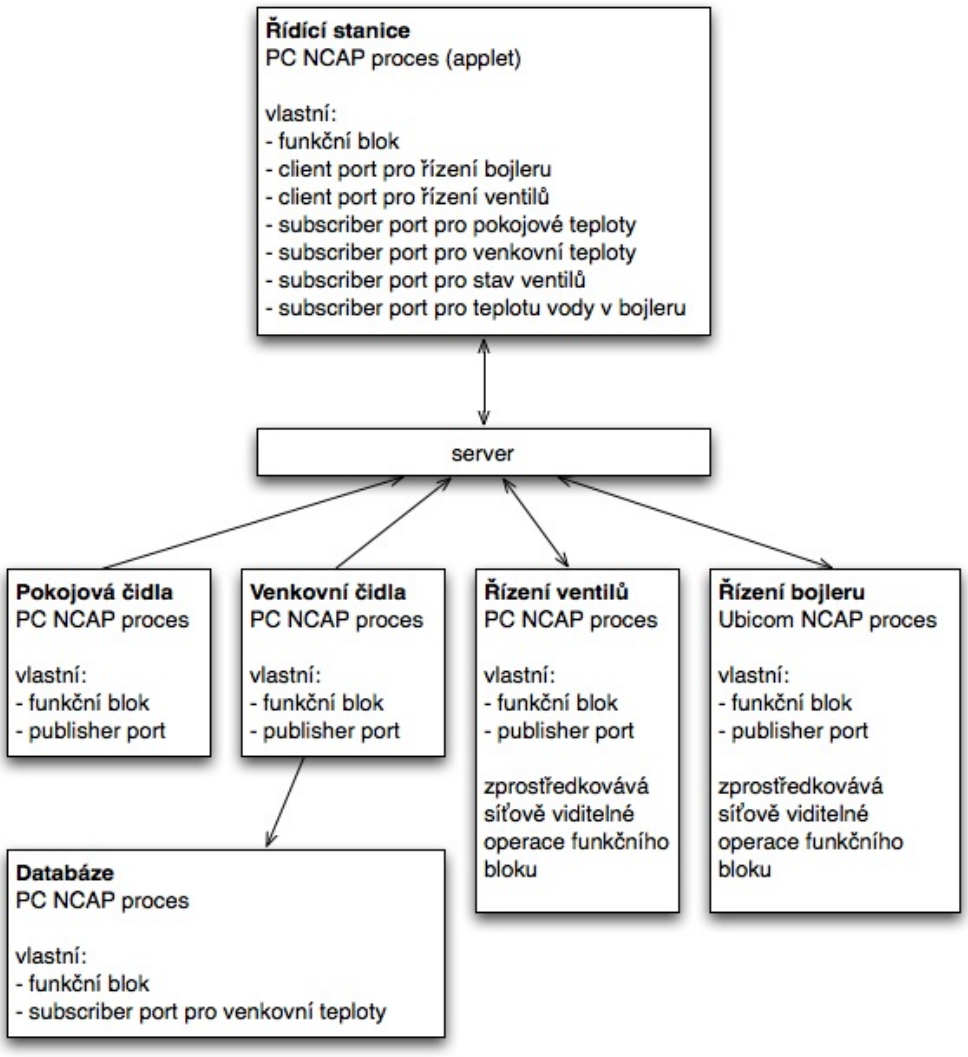
5.1.1 Architektura modelové aplikace

Podoba navržené modelové aplikace je uvedena na obr. 5.2. Jsou zde vyznačeny směry[2] komunikace mezi jednotlivými uzly a také použité bloky standardu IEEE 1451.1.

Z obrázku je patrné, že ne všechny bloky komunikují s řídící stanicí (server není uvažován, jedná se o blok začleněný do obrázku pro úplnost a ilustraci aplikace jako celku) oběma směry. To je dáno charakterem daných bloků. Pokojová a venkovní čidla budou aplikaci data pouze dodávat pomocí publisher portů.

Venkovní čidlo poskytuje data také bloku Databáze pro archivaci údajů. Databáze samotná je na řídící stanici zcela nezávislá.

Řídící bloky ventilů a bojleru komunikují s řídící stanicí oběma směry. Od řídící stanice příjmají pokyny k ovládání akčních členů. Řídící stanici pak poskytují informace o stavu ventilů a teploty vody v bojleru.



Obrázek 5.2: Architektura modelové aplikace.

[2] Směrem je chápán logický tok dat, nikoliv konkrétní síťová implementace a tok dat použitého protokolu.

5.1.2 Použité třídy standardu

Standard definuje rozsáhlou hierarchii tříd, jež má sloužit k přesné identifikaci funkčních celků systému. Jednotlivé třídy mají definováno rozhraní, signatury operací a specifické členské vlastnosti [2]. Standard je ale hodně abstraktní, co se týče chování daných metod a objektů. Chování některých metod je ponecháno zcela na programátorovi. Pro účely modelové aplikace byla vybrána podmnožina tříd, jejichž popis následuje. Kompletní hierarchie tříd definovaná standardem je uvedena v příloze A.

- `IEEE1451_Root` – Třída specifikuje operace, umožňující identifikovat každou třídu standardu. Poskytuje síťově viditelné operace pro získání jména a ID třídy.
- `IEEE1451_Entity` – Definuje základní vlastnosti objektů a identifikátory nezbytné pro komunikaci mezi NCAP procesy. Definuje metody pro operace s Object Tagy, jménem, ID a vybavovací adresou objektu. Mezi její metody patří i metoda Perform, určená k provádění síťově viditelných operací.
- `IEEE1451_Block` – Poskytuje základní operace pro práci s bloky systému. Specifikuje metody pro změny stavu bloku, získávání detailů definovaných výrobcem bloku.
- `IEEE1451_NCAPBlock` – Třída představuje instanci NCAP procesu. Umožňuje registrovat ostatní objekty uzlu, poskytuje metody pro vzdálenou konfiguraci uzlu, pro změnu stavu NCAP procesu a operace pro práci s cookie bloku. Dle standardu by na každém uzlu měla být právě jedna instance této třídy. V modelové aplikaci bude tato třída použita v každém uzlu, s vyjímkou serveru. V této třídě bude také obsažena síťově specifická funkcionalita.
- `IEEE1451_FunctionBlock` – Abstraktní třída sloužící jako bázová pro uživatelem definované funkční bloky systému. Poskytuje pouze metody pro práci se stavem funkčního bloku. Bloky v systému, které nabízí vzdáleně volatené metody by měly být potomky této třídy. V modelové aplikaci budou od této třídy dědit funkční bloky senzorů, řízení bojleru a ventilů, databáze i funkční blok aplikace klienta.
- `IEEE1451_Service` – Bázová třída pro třídy související s uvedenými komunikačními modely (viz kap. 4.2.3). Třídy od ní dědící reprezentují část na straně klienta.
- `IEEE1451_BasePort` – Bázová třída pro hierarchii tříd komunikačních portů, sloužících k odesílání informací přes síť. Poskytuje metody pro práci s vlastnostmi odesílaných zpráv.
- `IEEE1451_BaseClientPort` – Rodičovská třída pro třídy zajišťující komunikaci typu klient–server (viz kap. 4.2.3). Poskytuje standardem definované metody pro nastavení object tagu cílového objektu.
- `IEEE1451_ClientPort` – Klíčová třída pro komunikační model klient–server. Reprezentuje klientskou část komunikace a definuje pouze metodu Execute, která slouží k vykonání vzdálené operace na straně serveru. V modelové aplikaci bude tato třída využita pro zpřístupnění ovládání bojleru a ventilů klientovi.
- `IEEE1451_BasePublisherPort` – Rodičovská třída pro třídy poskytující komunikaci typu publish–subscribe (viz kap. 4.2.3). Standard u této třídy definuje síťově viditelné metody pro práci s tématem, doménou a klíčem publikace.

- `IEEE1451_PublisherPort` – Stěžejní třída pro komunikační model publish–subscribe. Poskytuje metodu pro zakódování a odeslání publikace přes síť. V modelové aplikaci budou tuto třídu využívat NCAP procesy poskytující venkovní a pokojové teploty, proces bojleru pro publikování teploty vody a proces řízení ventilů pro publikaci stavu ventilů.

- `IEEE1451_SubscriberPort` – Třída je využívána komunikačním modelem publish–subscribe. Poskytuje metody pro práci s tématem, klíčem a doménou přijímaných publikací. Dále poskytuje lokální metody pro správu operací, které budou zavolány v případě, že subscriber port obdrží vhodnou publikaci. V modelové aplikaci bude této třídy využito v uzlech klienta a databáze.

Ze specifikace uvedené v kap. 5.1.2 vyplývá, že aplikace bežící na PC v programovacím jazyce Java budou využívat všechny výše uvedené třídy. Uzel Ubicom oproti tomu využije pouze podmnožinu těchto tříd. Pro řízení bojleru není třeba využívat odběr publikací a s ním spojené třídy (`IEEE1451_SubscriberPort` a její rodičovská třída). Stejně tak není využita třída `IEEE1451_ClientPort` pro volání vzdálených operací a třída jí nadřazená.

Kapitola 6

Implementace

6.1 Použité technologie

Pro implementaci aplikace bylo použito několik technologií a dva programovací jazyky. Klient je implementován v programovacím jazyce Java, využívá technologii Java Applet a beží v prohlížeči uživatele. NCAP proces pro řízení bojleru je implementován v jazyce C na uzlu Ubicom. Ostatní NCAP procesy jsou implementovány v jazyce Java a běží na hostitelských stanicích, na kterých je nainstalována JVM.

6.1.1 Java

Programovací jazyk Java je plně objektově orientovaný. Jeho syntaxe vychází z jazyků C a C++. Některé konstrukce těchto jazyků vypouští za účelem přehlednější a pohodlnější implementace uživatelského softwaru. Naopak zavádí rysy, které jsou pro vývojáře přínosem. Mezi charakteristické vlastnosti tohoto jazyka patří:

- jednoduchá a přehledná syntaxe – V Javě není obsažen preprocesor, neexistují makra nebo přetěžování operátorů. Jednodušší je i navigace a přístup v hierarchii knihoven a tříd.
- podpora vyjímek – Pomocí mechanismu vyjímek lze zachytit chyby v programu, k nimž by docházet nemělo, ale jejichž výskyt nelze ovlivnit nebo vyloučit. V následné reakci na nastalou vyjímku je možné program uvést do konzistentního stavu a dále pokračovat, případně běh programu přizpůsobit vzniklé situaci.
- paralelizmus a synchronizace – Java podporuje paralelní běh částí programu pomocí vláken, jsou dostupné prostředky pro jejich synchronizaci a prostředky pro výlučný přístup k metodám.
- reflexe – Podpora reflexe, umožňující aplikaci zkoumat za běhu sebe samu a provádět nejrůznější analýzy.
- přenositelnost – Programy vytvořené v Javě je možné spustit na každém zařízení, které obsahuje patřičnou verzi JVM a potřebné zdroje. Zdrojový kód je přeložen do byte kódu, který dokáže JVM zpracovat a provést.
- archivy JAR – Formát pro distribuci knihoven a aplikací napsaných v Javě. Jedná se o platformově nezávislý formát. Ke spuštění aplikace v tomto archivu je zapotřebí virtuálního stroje JVM.

- serializace – Java umožňuje serializovat celé objekty a umožnit tak jejich jednoduché ukládání do souborů či databází, nebo jejich přenos po síti.
- automatické uvolňování použité paměti – Je dostupný Garbage collector, tedy nástroj pro automatické uvolňování použitých zdrojů.
- podpora databází – Java nabízí technologii JDBC, umožňující přístup k libovolné databázi. Ta může být realizována pomocí relační databáze, tabulkami nebo obyčejnými soubory.
- knihovna pro tvorbu GUI – Pro tvorbu uživatelských rozhraní je dostupná knihovna swing, případně AWT.
- obsáhlá dokumentace – Naprostá většina balíků Javy je výborně zdokumentována a existuje množství volně dostupných výukových kurzů a příkladů, vedoucích k osvojení si jazyka.
- univerzálnost – Pomocí jazyka Java lze vytvořit aplikace mnoha typů, od rozsáhlých serverových systémů, přes webové portály, až po aplikace pro mobilní telefony. Vše je závislé pouze na schopnostech cílového zařízení a použitého virtuálního stroje.
- velké množství knihoven – Java ve své současné 6. verzi SDK obsahuje stovky tříd a desítky knihoven [4].

Applet

Jedním z požadavků na vytvořenou aplikaci bylo implementovat klienta (řídící stanici) jako applet do www stránky. Applet je speciální formou programu v jazyce Java, který má jistá specifika a omezení [16]. Tento program je dostupný prostřednictvím sítě internet a internetového prohlížeče s podporou Javy.

Applet jako součást www stránky má několik výhod. Především je platformově zcela nezávislý (roli nehraje operační systém, ani použitý prohlížeč – podporuje-li grafický režim). Dalším důvodem proč se Java applet používá je možnost přístupu k datům na uživatelově PC, pokud s tímto uživatel souhlasí.

Z bezpečnostních důvodů ale applet nepovoluje některé funkce a nebo jejich provádění omezuje. Applet např. nemůže navazovat síťová spojení na jiný server, než ze kterého byl stažen, nemůže zapisovat do souborů na straně klienta nebo číst určité systémové informace. Pro spuštění je také vyžadován zásuvný modul Javy a doba samotného spuštění je prodloužena o dobu než se spustí JVM.

Web server

Pro potřeby appletu bylo nutné implementovat jednoduchý webový server. Ten uživateli poskytuje žádané www stránky a potřebné zdroje. Implementovaná verze není zcela kompletním web serverem, využita je pouze metoda GET pro získávání zdrojů z datového skladu web serveru. Pro navrženou aplikaci ale postačuje a umožní komunikaci appletu s aplikací na jiném než lokálním stroji.

Původní záměrem bylo spouštět aplikaci klienta pomocí HTML stránky, která si načte definovaný applet. Z důvodu bezpečnostních opatření [6], která zavádí JDK a JRE od verze 1.6.0_11 a vyšší, nemůže applet navazovat síťová spojení na lokální stanici v případě, že je tento načten z přímo spuštěného HTML souboru.

Je tedy nutné stránku s appletem umístit na webový server, na kterém bude přístupná přes odpovídající URL adresu.

Alternativním řešením by bylo upravit tzv. *applet.policy* soubor, který definuje bezpečnostní pravidla appletu. Protože má ale být aplikace přístupná odkudkoliv ze sítě internet, instalace webového serveru by stejně byla jedním z kroků, nutných pro splnění tohoto cíle.

6.1.2 Ubicom

Jako Ubicom je označena vývojová deska, obsahující procesor IP2022 z rodiny řady IP2000. Tato skupina procesorů představuje jednočipové procesory, požitelné pro síťovou komunikaci. Procesor IP2022, který je obsažen na vývojové desce, je optimalizován pro použití v sítích Internet a je použitelný také jako most nebo brána v infrastruktuře Internetu [12].

Hardwarová výbava Ubicomu sestává z následujících částí [10]:

- **IP2022 Internet Procesor** – 80 pinový procesor s instrukční sadou typu RISC, umožňující síťovou komunikaci pomocí IP protokolu. Obsahuje CPU, 64kB flash paměti pro program, 16kB programové RAM a 4kB datové RAM.
- **dvě RS-232 rozhraní** – umožňující komunikaci s terminálem podle specifikace RS-232, konfigurovatelné pomocí balíku ipUART.
- **konektory pro připojení rozhraní Ethernet nebo USB** – k Ubicomu je možné připojit rozšiřující karty, umožňující komunikaci pomocí rozhraní Ethernet nebo USB. Právě pomocí jedné z těchto karet bude Ubicom komunikovat s řídící stanicí na PC.
- **ISP/ISD konektor** – slouží pro připojení redukce z paralelního kabelu, pomocí kterého je Ubicom programován.
- **SPI flash paměť** – sériová flash paměť o velikosti 512kB, sloužící k ukládání souborů pomocí balíků ipFile a ipStorage.

Ubicom lze programovat pomocí balíčků ipModule [11], které již obsahují danou funkcionalitu (obdoba balíků v jazyce Java). Každý balíček lze konfigurovat z vývojového prostředí Unity a poté jeho funkce využívat při psaní softwaru.

Mezi dostupné balíčky patří např.:

- **ipOS** – obsahuje definici základních datových typů a funkcí pro práci s časovačem, pamětí, síťovými buffery, řetězci, vstupně/výstupními funkcemi a watchdogem. Také poskytuje funkce pro debugging a přístup do paměti procesoru.
- **ipStack** – knihovna funkcí pro práci se síťovými protokoly IP, ICMP, UDP, TCCP, SLIP a podpůrnými funkcemi pro DHCP.
- **ipEthernet** – dodává množinu funkcí podporujících komunikaci prostřednictvím Ethernetu (IEEE802.3)
- **ipI2C** – funkce pro komunikaci po sběrnici I^2C
- **ipWeb** – knohovna funkcí, podporující protokol HTTP a realizujících jednoduchý web server.
- **ipTFTP** – podpůrné funkce pro přenos souborů pomocí protokolu UDP

Po konfiguraci balíčku ve vývojovém prostředí jsou vývojáři dostupné jeho funkce. Např. u balíčku ipStack lze využívat:

- `tcp_init`, `tcp_connect`, `tcp_listen` a `tcp_close` – pro inicializaci TCP funkcí, inicializaci soketu pro klienta/server a uzavření spojení.
- `tcp_recv` – pro příjem dat pomocí TCP paketů.
- `tcp_connect_notify`, `tcp_establish_notify` a `tcp_close_notify` – pro signalizaci přijatého, navázaného a ukončeného spojení. Tyto události jsou zachyceny pomocí tzv. *callback* funkcí.

Uzel Ubicom lze programovat v assembleru nebo jazyce C. Pro implementaci softwaru modelové aplikace byl využit jazyk C, poskytující vyšší úroveň abstrakce a vzhledem k charakteru aplikace také přívětivější formu vývoje.

6.2 Komunikační protokol

Komunikace mezi jednotlivými bloky je postavena na IP protokolu a využívá spojovaného i nespojovaného přenosu. Princip komunikace mezi bloky modelové aplikace vychází z principů popsaných v kap. 4.2.3.

6.2.1 Komunikace klient–server

Spolehlivější spojovaný přenos, komunikující pomocí TCP paketů, je určen pro komunikaci podle schématu klient–server. Tento způsob se využívá především pro řízení cílového zařízení (bojler, ventily) a z principielního hlediska je vhodnější. Jistě by nebylo žádoucí, kdyby se paket nesoucí příkaz k vypnutí bojleru při přenosu ztratil.

U komunikace klient–server jsou přenášeny údaje uvedené v tab. 6.1 (směr od klienta k serveru) a 6.2 (směr od serveru ke klientovi, v případě že je očekávána odpověď). V paketech jsou data uložena v uvedeném pořadí.

Pořadí	Datový typ	Význam
1	UInteger8	Typ zprávy – vždy konstanta MSG_CS
2	ObjectTag	ObjectTag cílového objektu na serveru
3	UInteger16	Cookie uložená u klienta
4	UInteger16	ID cílové operace
5	ArgumentArray	Vstupní parametry pro cílovou operaci

Tabulka 6.1: Data přenášená od klienta k serveru.

Vzhledem k tomu, že byl do systému vložen blok serveru, zastupující klienta, bylo nutné upravit patřičně obsah odesílaných TCP paketů od klienta k jiným NCAP procesům. Klient na začátek paketu přidává informaci o adrese cílového objektu a tento paket posílá serveru. Server zjistí IP adresu a číslo portu, na který má být skutečně paket doručen a data (v podobě uvedené v tab. 6.1) na tuto adresu odešle. Podrobnosti funkce serveru jsou uvedeny v kap. 6.5.3.

Pořadí	Datový typ	Význam
1	UInteger8	Typ zprávy – vždy konstanta MSG_CS
2	ClientServerReturnCode	Návratová hodnota operace Perform
3	UInteger16	Nová cookie od serveru
4	ArgumentArray	Výstupní argumenty volané operace

Tabulka 6.2: Data přenášená od serveru ke klientovi.

6.2.2 Komunikace publish–subscribe

Nespojovaný přenos, realizovaný pomocí UDP paketů, slouží k rozesílání publikací, obsahujících teploty z čidel apod. Případná ztráta těchto údajů při přenosu není pro aplikaci kritická. Vzhledem k množství přenášených informací tohoto typu je nespojovaný přenos také přívětivější z hlediska vytížení sítě.

Data která se přenáší od publikujícího objektu k odebírajícímu jsou uvedena v tab. 6.3. Zde již komunikace probíhá vždy jedním směrem.

Pořadí	Datový typ	Význam
1	PubSubDomain	Publikační doména
2	UInteger8	Klíč publikace
3	PublicationTopic	Téma publikace
4	ArgumentArray	Vlastní obsah publikace

Tabulka 6.3: Data přenášená od klienta k serveru.

6.3 Datový model

Pro komunikaci mezi jednotlivými bloky systému je nutno dodržet jednotnou formu přenášených dat a zaručit, že data budou interpretována na všech zařízeních stejně.

Pro potřeby modelové aplikace byla implementována podmnožina datových typů popisovaných standardem:

- **Primitivní datové typy** – tedy `Integer8`, `Integer16`, `Integer32`, `UInteger8`, `UInteger16`, `UInteger32`, `Boolean`, `Octet`, `Float32` – pro 8, 16 a 32 bitová celá čísla se znaménkem i bez znaménka, hodnoty typu boolean, datový typ pro uložení znaku a datový typ pro čísla s plovoucí čárkou.
- **Strukturované datové typy** – jde o `ClientServerReturnCode`, `OpReturnCode`, `ObjectProperties`, `ObjectDispatchAddress` – pro informace o návratových hodnotách z operací a jejich vzdáleného volání, pro popis vlastností objektu a jeho adresy v rámci systému.
- **Datové typy polí** – `ArgumentArray`, `ObjectTagArray` – pro obecné pole argumentů a pole typů ObjectTag. Dále byly implementovány typy pro pole primitivních datových typů.

- **Speciální datové typy** – zejména `ClassID`, `ObjectID`, `ObjectTag`, `PubSubDomain`, `PublicationTopic`, `SubscriptionQualifier` – které vycházejí z dříve uvedených datových typů.

U zmíněných datových typů je zajištěno, že jejich interpretace na všech objektech modelové aplikace bude totožná. Vypuštěny byly datové typy, které nejsou pro aplikaci zajímavé (např. fyzická metadata ze senzorů, časové údaje apod.)

Standard definuje formát a reprezentaci jednotlivých datových typů pomocí jazyka IDL. Jejich implementace v jazycích Java a C je pro každý z těchto jazyků charakteristická.

6.3.1 Implementace v jazyce Java

Jazyk Java neobsahuje datové typy přímo odpovídající datovým typům standardu. Bylo proto nutné implementovat pro každý datový typ zapouzdřující objekt. Ten slouží k inicializaci instance datového typu na patřičnou hodnotu, její uchovaní ve vnitřní proměnné a k dalším operacím.

Aby bylo možné používat datový typ `ArgumentArray`, je nutné implementovat u každého datového typu rozhraní `IArgument`, díky kterému je možné pracovat s datovými typy dostatečně obecně a vytvářet tak pole různorodých objektů.

Rozhraní `IArgument` je definováno následovně:

```
public interface IArgument
{
   public void Encode(ByteArrayOutputStream stream);
   public OpReturnCode Decode(ByteArrayInputStream stream);
   public IEEE1451_String GetName();
   public Type.TypeCode GetType();
}
```

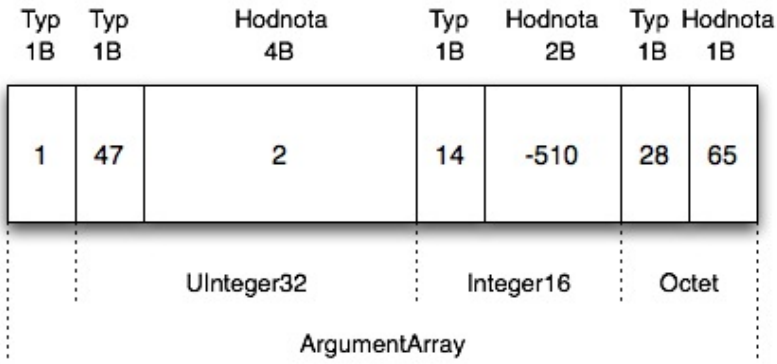
Metody `Encode` a `Decode` slouží k zakódování resp. dekódování datového typu do výstupního resp. ze vstupního datového proudu, použitého při odesílání resp. příjmání dat. Metoda `GetName` vrací jméno datového typu a metoda `GetType` vrací typový kód, použitý pro identifikaci datového typu při dekódování z přijímaných dat.

Dalším důležitým objektem je `MainArray`, což je abstraktní objekt, od kterého dědí všechny objekty implementující datové typy pole. Objekt má následující metody:

```
public void Encode(ByteArrayOutputStream stream);
public abstract OpReturnCode Decode(ByteArrayInputStream stream);
public IEEE1451_String GetName();
public abstract Type.TypeCode GetType();
public UInteger32 GetSize();
```

`MainArray` také implementuje rozhraní `IArgument`, navíc ale přidává metodu pro zjištění velikosti pole, vracející tento údaj ve formátu požadovaném standardem (ten definuje velikost pole jako číslo typu `UInteger32`). Objekt také přímo implementuje metodu `Encode`, jelikož je pro všechny objekty typu pole stejná.

Pro názornost je na obr. 6.1 uvedena podoba datového typu `ArgumentArray` tak, jak je zpracována operacemi `Encode` a `Decode` a jak je přenášena po síti. Jako ilustrativní bylo zvoleno pole, obsahující jedno číslo typu `Integer16` a znak typu `Octet`.



Obrázek 6.1: Formát datového typu ArgumentArray.

Při kódování datových typů se do výstupního proudu vždy ukládá nejdříve 1 bajt identifikující typ následujících dat. Další obsah závisí na konkrétním datovém typu – každý datový typ přidává do výstupního proudu odpovídající data. Při dekódování je pak vždy podle přečteného typu zavolána metoda `Decode` příslušného objektu reprezentujícího datový typ (zajištěno v objektu `Type`), ve které odpovídající objekt načte patřičná data v pořadí shodném s pořadím kódování.

6.3.2 Implementace v jazyce C

V jazyce C uzlu Ubicom je situace odlišná. Jazyk C disponuje datovými typy, které lze přímo použít pro datové typy definované standardem. U primitivních datových typů tedy stačí využít konstrukci `typedef` jazyka C, která zajistí přehlednější a jednotnější způsob pojmenování datových typů v rámci standardu IEEE 1451.1.

Pro implementaci strukturovaných datových typů je použita konstrukce `struct`, která je pro tento účel v jazyce obsažena.

Pole jsou implementována dvěma způsoby:

1. přímo jako pole datových typů (`typ[]`) – u těch datových typů, kde je možné zjistit pomocí jednoduchých funkcí jak je dané pole dlouhé. Tento způsob je použit např. u datového typu `String` nebo `OctetArray`.

2. využití konstrukce `struct` – u ostatních datových typů pole. Ve struktuře je obsažena informace o počtu položek pole a ukazatel na pole samotné.

Datový typ `ArgumentArray` je v jazyce C implementován jako struktura, obsahující informaci o počtu položek a dále ukazatele na dvě pole identifikující typy uložených prvků a uložené prvky samotné.

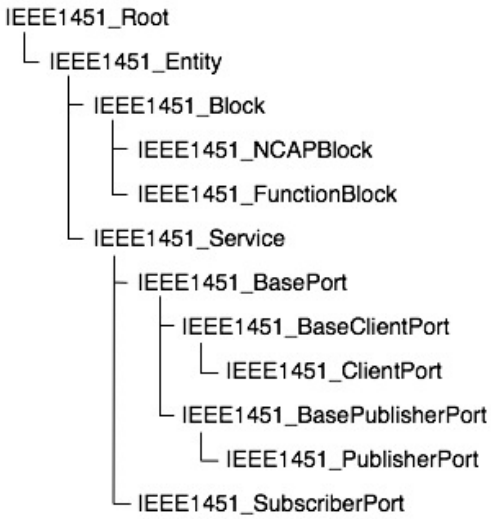
Kódování a dekódování dat je narozdíl od implementace v Javě umístěno do zvláštních metod objektu `IEEE1451_NCAPBlock`. Datové typy samotné se nemusí o kódování nebo dekódování starat a usnadňuje se tak jejich použití.

Dalším rozdílem oproti implementaci v Javě je absence datového typu pro obecný argument (rozhraní `IArgument`). Toto je umožněno díky využití datového typu `void`, u nějž se dle potřeby provádí přetypování na konkrétní datový typ.

6.4 Objektový model

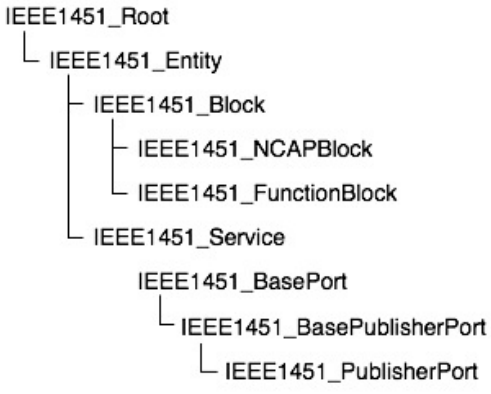
Třídy použité pro implementaci jsou uvedeny v kap. 5.1.2. Jejich konkrétní implementace je ale pro oba jazyky odlišná. Zatímco Java, jako jazyk čistě objektový, vychází svými prostředky vstříc požadavkům standardu, jazyk C pro uzel Ubicom již tak použitelný není.

Hierarchie v Javě implementovaných tříd je znázorněna na obr. 6.2. Z obrázku lze vyčíst, že uzly spuštěné na PC budou schopny vykonávat všechny základní operace nutné pro komunikaci pomocí popsaných komunikačních modelů (viz kap. 5.1).



Obrázek 6.2: Implementované třídy v jazyce Java.

Uzel Ubicom má mít z pohledu modelové aplikace omezenou funkcionalitu, a sice na odesílání publikací a komunikaci s klientem. Proto je jeho objektový model redukován na podmnožinu tříd, znázorněnou na obr. 6.3.



Obrázek 6.3: Implementované třídy v jazyce C.

V jazyce C je docíleno dědičnosti s využitím konstrukce `struct`. Třídy jsou definovány jako struktury, obsahující členské proměnné a metody. Dále je pro každou třídu vytvořena

inicializační funkce, která alokuje potřebnou paměť a provede mapování definovaných funkcí na metody třídy. Pokud má být třída potomkem jiné třídy, je tato uvedena na začátku definice struktury dědící třídy. Jazyk C pak toto interpretuje jako odkaz na rodičovskou třídu a po přetypování je schopen přistupovat k vlastnostem a metodám rodičovské třídy.

Struktura třídy `IEEE1451_NCAPBlock`, která dědí ze třídy `IEEE1451_Block` a tuto třídu rozšiřuje, vypadá schematicky následovně:

```
struct IEEE1451_NCAPBlock
{
   struct IEEE1451_Block Block;

   // ...

   void* registered_objects[MAX_OBJECTS_REGISTERED];

   // ...

   /*local*/ IEEE1451_NCAPBlock_RegisterObject RegisterObject;
   /*local*/ IEEE1451_NCAPBlock_GetBlockCookie GetBlockCookie;

   // ...
};
```

Vzhledem k paměťovým nárokům a dostupným prostředkům uzlu Ubicom byla implementace objektového modelu částečně optimalizována. Především byly sloučeny třídy `IEEE1451_Root` a `IEEE1451_Entity`, a dále byly sjednoceny třídy počínaje od `IEEE_Service` po `IEEE_PublisherPort`. Uvedené změny také výrazně zvýšily přehlednost aplikace a vývojář se v kódu lépe orientuje.

6.5 Bloky aplikace

Modelová aplikace má pevně definovanou architekturu (viz kap. 5) a stanoven význam a možnosti jednotlivých bloků. Každý blok je nakonfigurován tak, aby systému poskytoval potřebnou funkcionalitu a byl schopný reagovat na přijaté požadavky.

Jednotlivé uzly jsou identifikovány pomocí vlastností `ObjectDispatchAddress` a mají také potřebné informace o blocích, které potřebují ke své činnosti nebo se kterými musí komunikovat, aby byla zajištěna správná funkce aplikace.

Konfigurace jednotlivých bloků je pevně dána a jednotlivé bloky ji mají k dispozici v době kompilace ze zdrojových kódů. Pro změnu konfigurace je nutné upravit potřebná nastavení ve zdrojovém kódu a tento zkompilovat.

V následujících kapitolách budou podrobněji probrány jednotlivé bloky z hlediska jejich programové struktury, implementace a dalších aplikačních detailů. Příklady implementace některých částí systému jsou uvedeny v příloze B. Způsob jak jednotlivé aplikace přeložit a spustit je popsán v příloze C.

6.5.1 Web server

Pro umožnění komunikace klienta s okolními bloky je nutné stáhnout www stránku s appletem z webového serveru. Tím je umožněno navázat spojení pomocí soketů.

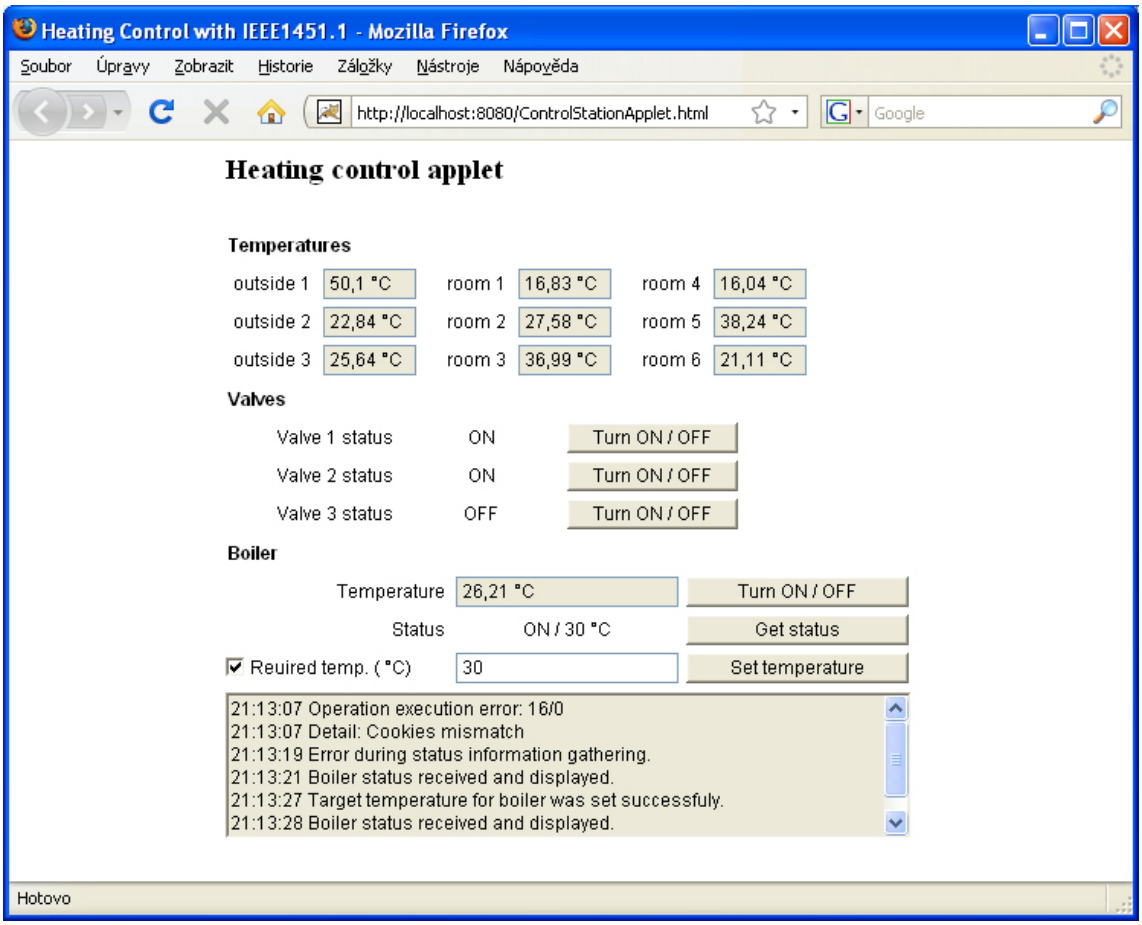
Web server je implementován v jazyce Java a běží jako samostatný proces na PC.

Jeho hlavní program běží v nekonečné smyčce a naslouchá požadavkům z internetového prohlížeče, na které odpovídá. Implementována je pouze obsluha metody GET a zpracování souborů typu HTML, jar a class, které jsou nutné pro provoz klienta.

Pokud na port web serveru přijde požadavek o stažení stránky, web server tuto žádost analyzuje a podle URL na ní odpoví odesláním patřičného dokumentu. Internetový prohlížeč pak zajistí zpracování přijaté odpovědi a pokud se jedná o HTML stránku, která obsahuje další zdroje umístěné na web serveru, požádá o ně automaticky webový server.

6.5.2 Klient – řídící stanice

Pro přístup k řízení systému musí uživatel navštívit www stránku s danou adresou, na které se nachází applet řídící stanice (viz obr. 6.4). Aby mohl klient navazovat síťová spojení, musí být applet stažen z PC, na které se chce připojovat. K tomu je využit web server (popsáno v kap. 6.1.1).



Obrázek 6.4: Applet běžící na www stránce.

Applet je do www stránky vložen pomocí HTML kódu. Jelikož je pro běh appletu vyžadována knihovna funkcí standardu IEEE 1451.1, překládaná zvlášť, musí applet vědět, odkud tuto knihovnu stáhnout. V HTML kódu stránky se toto nastavení provádí pomocí vlastnosti `archive` tagu `applet`.

```
<applet
   codebase="classes"
   archive="IEEE.jar"
   code="controlstationapplet/ControlStationApplet.class"
   width="450" "height=400"/>
```

Atribut `codebase` slouží ke specifikaci cesty, kde se nachází třídy appletu i knihovna s balíčky IEEE 1451.1. Dále je appletu pomocí tagu `code` předána informace o třídě, implementující samotný applet. Poslední atributy určují rozměr appletu na www stránce.

Applet poskytuje uživateli grafické rozhraní, implementované pomocí knihovny AWT. Uživatel má možnost sledovat venkovní a pokojové teploty, dodávané odpovídajícími čidly, je mu umožněno ovládat a sledovat stav ventilů a také má možnost ovlivňovat chod bojleru a sledovat jeho stav.

Chybové hlášky a výsledky některých operací se zobrazují v okně ve spodní části appletu.

Po spuštění je vyvtvořeno grafické uživatelské rozhraní appletu a je provedena konfigurace a inicializace aplikace podle standardu IEEE 1451.1.

V první fázi inicializace je vytvořena instance třídy `IEEE1451_NCAPBlock` se jménem `localNCAP`. Ta je využita pro registraci potřebných bloků a nastavení síťového spojení se serverem.

Pro zajištění požadované fukčnosti řídící stanice je nutné vytvořit instanci třídy funkčního bloku `ControlStationFunctionBlock` pojmenované `control_station_function_block`, jejíž význam bude popsán dále.

Inicializace pokračuje vytvářením instancí tříd `IEEE1451_ClientPort` pro řízení bojleru a ventilů a nastavením jejich cílových bloků pomocí IP adres, čísel portů a object tagů. Dále je nutné vytvořit instance tříd `IEEE1451_SubscriberPort` pro zajištění odběru publikací od bloků měřících venkovní a pokojové teploty, teplotu vody v bojleru a poskytující stav ventilů. Po vytvoření těchto instancí je u každého patřičného objektu zaregistrován odběratel dané publikace, kterým je metoda objektu `control_station_function_block`.

Jakmile jsou vytvořeny všechny potřebné instance, jsou tyto zaregistrovány u lokálního NCAP bloku. Tím se stávají síťově viditelnými a je možné s nimi dále nakládat podle standardu.

Posledním krokem při inicializaci aplikace řídící stanice je nastavení síťové komunikace.

Z důvodů zmiňovaných v kapitole 6.1.1 musí klient komunikovat s ostatními bloky prostřednictvím proxy serveru. Tuto skutečnost je nutno oznámit lokálnímu NCAP bloku pomocí metody `SetProxyServer`, která má jako parametry IP adresu a číslo portu proxy serveru, na který posílá data. Jakmile dojde k požadavku na komunikaci klient–server s cílovým blokem, přidá se na začátek dat odesílaných serveru IP adresa a číslo portu skutečného příjemce.

Pokud všechna nastavení proběhnou v pořádku, je lokální NCAP proces aktivován přechodem do aktivního stavu a je připraven k síťové komunikaci.

Aby bylo možné používat prvky GUI a zároveň obsluhovat požadavky na ovládání a zobrazování stavů bloků, bylo nutné applet implementovat s využitím vláken. Pokud by applet běžel pouze v jediném vlákně, veškerý výpočetní čas by byl věnován pouze obsluze GUI či zpracování příchozích zpráv od jiných bloků. Vlákna umožňují appletu zpracovávat oba tyto vstupy.

Nové vlákno je po spuštění appletu vytvořeno v metodě `start` a po jeho spuštění je řízení tohoto nového vlákna předáno metodě `run`. V této metodě probíhá detekce a zpracování příchozích zpráv od zbývajících bloků systému. Po každém zpracování je vlákno na krátký

okamžik uspáno, aby se čas procesoru mohl věnovat ostatním úlohám. V původním vláknu pak běží obsluha událostí grafického uživatelského rozhraní.

Funkcionalitu řídící stanice zajišťuje objekt `control_station_function_block`. Ten poskytuje metody zpracovávající podněty od uživatele. Jedná se o vlastní zavolání vzdálené operace na ostatních blocích aplikace. Např. při žádosti o spuštění/zastavení bojleru se zavolá metoda `ToggleBoiler`. Ta vytvoří vstupní parametry vyžadované k přechodu bojleru do žádaného stavu, zavolá operaci `Execute` klientského portu (který je spojen s cílovým funkčním blokem bojleru) a ošetří výsledek volání. Pokud se podařilo stav bojleru změnit nebo nastala chyba při komunikaci, je tato skutečnost oznámena uživateli textovou informací v appletu. Analogicky funguje řízení ventilů, zjišťování stavu bojleru nebo nastavování požadované teploty.

Funkční blok poskytuje také objekty, figurující jako odběratelé publikací. Tyto objekty implementují rozhraní `IEEE1451_ISubscriber`, které slouží jako mechanismus callback funkcí. Při inicializaci subscriber portů byl u každého takového portu nastaven odběratel, což byla právě instance třídy implementující toto rozhraní. Obdrží–li lokální NCAP blok publikaci, projde seznam všech u něj registrovaných objektů a předá publikaci metodě `FilterPublicationAndCallbackSubscribers` každého nalezeného subscriber portu. V této metodě jsou pak porovnána data publikace s údaji nastavenými subscriber portu a dojde–li ke shodě, jsou zavolány všechny registrované callback metody daného subscriber portu a je jim předán obsah publikace. V těchto metodách pak probíha vlastní zpracování obdržených údajů.

6.5.3 Server

Aplikace serveru slouží jako prostředník mezi klientem a ostatními bloky systému. Do architektury byl začleněn vzhledem k omezením Java appletu (kap. 6.1.1). Jeho hlavní činností je přeposílat obdržené pakety příslušným adresátům.

Síťová komunikace je v Javě řešena pomocí soketů [5]. Využívá se neblokujících soketů, vytvářených pomocí třídy `SocketChannel` balíku `java.nio`. Kanál je schopen vytvořit soket, ale jeho připojení již nechává na soketu samotném. Neblokující mód je možné nastavit metodou `configureBlocking` třídy `ServerSocketChannel` pro spojovanou komunikaci resp. `DatagramChannel` pro komunikaci nespojovanou.

Je–li třeba odeslat informaci na cílový uzel, vytvoří se nový soket, který se k cíli připojí metodou `bind`, data se odešlou a spojení se ukončí, případně je možné čekat na odpověď. Pokud se naopak na data čeká, testuje se jejich přijetí v nekonečné smyčce voláním metody `accept` třídy `ServerSocketChannel` resp. `receive` třídy `DatagramChannel`.

Server pracuje v nekonečné smyčce, ve které sleduje definované porty na stanici, na které je spuštěn. Pro modelovou aplikaci je nezbytné, aby sledoval pouze 2 typy přenosů:

1. **od řídící stanice k ostatním blokům** – zde je důležitá komunikace prostřednictvím spojovaného přenosu s TCP pakety. Server tedy naslouchá na přiřazeném portu a po obdržení paketu z něj extrahuje IP adresu a port skutečného adresáta, se kterým naváže spojení. Pokud se má čekat na odpověď (tato informace je taktéž přidána do paketu pro proxy server), čeká server na odpověď a tu pak přeposílá zpět klientovi.

2. **od bloků poskytující publikace k řídící stanici** – v tomto případě je situace jednodušší. Bloky poskytující publikace mají jako příjemce těchto publikací uveden proxy server. Ten naslouchá na patřičném portu a UDP pakety, které jsou na tento port doručeny přeposílá beze změny řídící stanici ke zpracování.

6.5.4 Pokojová a venkovní čidla

Bloky poskytující pokojové a venkovní teploty mají, co se funkce týče, totožné softwarové vybavení. Rozdíl je pouze v množství poskytovaných údajů a také významový z hlediska pozice v systému.

Tyto bloky poskytují v pravidelných intervalech hodnoty pomyslných teplotních čidel, které jsou doručovány řídící aplikaci. Je zde využit komunikační modelu publish–subscribe s tím, že pokojové teploty jsou odesílány pouze řídící stanici, kdežto venkovní teploty zpracovává navíc i blok Databáze (viz 6.5.7).

Software těchto bloků je implementován je v jazyce Java a oba běží na PC jako samostatné nezávislé procesy.

Každý z bloků provádí po spuštění inicializaci lokálního NCAP procesu a následně inicalizaci funkčního bloku (instance třídy `OutsideFunctionBlock` resp. `RoomFunctioBlock`). Poté je inicializován objekt třídy `IEEE1451_PublisherPort`, který zajišťuje rozesílání publikací.

Jakmile jsou instance úspěšně incializovány, jsou zaregistrovány u lokálního NCAP procesu a je možné s nimi dále pracovat podle standardu.

Po dokončení registrace instancí funkčních bloků a publisher portů je nastavena síťová komunikace. Zde je nutno zmínit metodu `EmulateMulticast`, která umožňuje rozesílat publikace na všechny registrované odběratele. Zájemci o publikace se přidávají pomocí metody `EmulateMulticastAddTarget` lokálního NCAPu. U bloku pro měření pokojových teplot je jako odběratel registrován pouze server (tedy řídící stanice), u bloku pro měření venkovních teplot je navíc zaregistrován ještě blok databáze. V případě databáze jsou data posílána přímo cílovému bloku, bez účasti serveru (jak je vidět na obr. 5.2).

Je–li úspěšně dokončena inicializace sítě, je lokální NCAP uveden do aktivního stavu. V nekonečné smyčce je pak testováno, zda jsou na blok kladeny žádosti o komunikaci typu klient–server nebo zda byly doručeny nějaké publikace, případně je provedeno obsloužení těchto žádostí. Obojí je v případě těchto dvou bloků pouze ilustrativní, hlavní funkčnost je obsažena v metodě `ProcessAndPublishMeasurement` funkčního objektu.

Funkční objekty ve svých konstruktorech nastaví náhodné výchozí teploty, které pak každou sekundu mění o nepatrný přírustek čí úbytek. Je tak simulován vývoj teplot měřených sadou čidel (konkrétní řešení není pro tuto práci podstatné). Funkční blok pak ve své metodě `ProcessAndPublishMeasurement` vždy zajistí aktualizaci naměřených teplot a aktuální hodnoty použije jako obsah publikace. Tu pomocí metody `Publish` příslušného publisher portu vyšle prostřednictvím sítě příjemcům.

Po odeslání každé publikace je proces na krátkou dobu uspán.

6.5.5 Řízení ventilů

Blok pro řízení ventilů poskytuje síťově viditelné operace pro ovládání jednotlivých ventilů a systému poskytuje informace o stavu těchto ventilů formou publikace.

Implementován je v Javě a běží na PC jako samostatný proces.

Po spuštění se provede incializace lokálního NCAPu, funkčního bloku (instance třídy `ValvesFunctionBlock`) a instance třídy `IEEE1451_PublisherPort` pro rozesílání publikací. Funkční blok i publisher port jsou poté zaregistrovány u lokálního NCAP procesu.

Registrace umožňuje používat dané bloky podle standardu. V případě bloku řízení ventilů poskytne systému síťově viditelné metody `GetValve`, `SetValve`, `CloseAll` a `OpenAll` pro obsluhu ventilů.

Po úspěšném zaregistrování instance funkčního bloku a publisher portu je provedeno nastavení síťové komunikace. Jedná se o konfiguraci rozesílání publikací a přidání příjemce těchto publikací, kterým je server (resp. řídící stanice). Jakmile je konfigurace síťové komunikace dokončena, je lokální NCAP proces uveden do aktivního stavu.

V nekonečné smyčce je pak testován výskyt žádostí od řídící stanice. Těmi se rozumí požadavky na vyvolání síťově viditelných metod. Je–li přijata zpráva obsahující pokyn k vykonání operace funkčního bloku, je zpracována lokálním NCAP procesem a předána metodě `Perform` instance třídy `ValvesFunctionBlock`. V této metodě je pak provedeno mapování vstupních argumentů, je zavolána příslušná metoda a případné výstupní argumenty jsou předány zpět lokálnímu NCAP procesu, který se postará o jejich zpětné zaslání žadateli. V případě výskytu chyby je žadateli zaslán také chybový kód.

Dojde–li ke změně stavu ventilů (ať už v důsledku zavolání patřičné vzdálené metody, nebo jejím přímým voláním), je ve funkčním bloku připravena publikace, obsahující informace o aktuálním stavu ventilů. Tato je pak pomocí publisher portu odeslána registrovaným příjemcům, v tomto případě řídící stanici.

Na konci iterace nekonečné smyčky je proces na krátkou dobu uspán.

6.5.6 Řízení bojleru

Blok řízení bojleru je realizován na hardwarovém uzlu Ubicom, připojeném k PC na kterém běží ostatní bloky aplikace. Jeho funkčnost je zajištěna programem v jazyce C, který je uložen ve flash paměti procesoru IP2022.

Proces komunikuje se serverem pomocí spojovaného i nespojovaného přenosu.

Na Ubicomu se pro síťovou komunikaci využívá funkcí balíku ipStack, který poskytuje operace s TCP i UDP pakety. Po inicializaci funkcí `tcp_init` se nastaví pomocí `tcp_listen` speciální callback funkce [11]. Ty budou zavolány v případech kdy Ubicom obdrží požadavek na spojení, v situaci kdy je spojení navázáno nebo v případě ukončení spojení. V příslušných funkcích se pak provádí zpracování přijatých dat a případně odesílá odpovědi.

Je–li potřeba odeslat z Ubicomu TCP resp. UDP paket, využívá se net bufferu, do kterého se ukládá obsah odesílaného paketu. Net buffer je speciální mechanismus, využívající techniku lazy-copy[1]. Jakmile jsou data připravena, využije se pro jejich rozeslání funkce `tcp_send_netbuf` pro komunikaci klient–server nebo `udp_send_netbuf` pro mechanismus publish–subscribe.

Na uzlu Ubicom je implementována skupina tříd tak, jak je popsána v kap. 6.4. Hlavní třídou, zajišťující potřebnou funkcionalitu je třída `IEEE1451_NCAPBlock` a k ní příslušející funkce `localNCAP_init`, která zajišťuje vytvoření právě jednoho NCAP procesu. V této funkci je provedeno namapování metod, nastavení výchozích vlastností NCAP procesu, registrace samotného NCAPu a inicializace síťové komunikace, především pak určení callback funkcí pro TCP spojení.

V hlavním programu je inicializován samotný Ubicom – je vymezena paměť pro uživatele a nastaven časovač. Dále je vytvořen lokální NCAP proces, pomocí výše zmíněné funkce.

Pro poskytnutí potřebné funkcionality je nutné vytvořit instance publisher portu a funkčního bloku. Publisher port bude systému pravidelně dodávat publikaci obsahující aktuální teplotu vody v bojleru. Funkční blok zase definuje síťově viditelné operace pro nastavování bojleru a vlastní funkcionalitu tohoto uzlu.

[1] Líné kopírování – data jsou ve výchozím stavu sdílena více prostředky a používá se jejich *shallow-copy* (mělká kopie). Kopírují se až v případě, kdy chce jeden z vlastníků data změnit [17].

Pomocí metody `RegisterObject` jsou oba uvedené bloky zaregistrovány u lokálního NCAPu a dále je pomocí metody `AddPublishTarget` přidán odběratel publikací, kterým je v modelové aplikaci server.

Po dokončení konfigurace přejde lokální NCAP do aktivního stavu, zavolá metody pro ohřev vody a publikaci teploty a začíná pracovat podle požadavků.

Vzhledem ke způsobu implementace síťové komunikace, tj. použití zpětných volání callback funkcí, je program Ubicomu stavěn na jiném principu než je tomu u implementace v Javě. Využívá se zde systému přerušení a časovače.

Hlavní program běží v nekonečné smyčce, v níž se kontroluje stav síťového rozhraní (ethernet device instance). V případě, že jsou na rozhraní ethernetu doručena data, je podle situace zavolána patřičná callback funkce a po jejím skončení se program vrací do hlavní smyčky. V jejím dalším kroku se vyhodnocuje hodnota časovače a v případě, že je nalezena registrovaná událost, přiřazena k aktuální hodnotě časového údaje, je tato událost vyvolána.

Zpracování požadavků klienta

Je–li navázáno spojení s Ubicomem, je tato skutečnost zachycena pomocí callback funkce `TCP_callback_connect`[2], která vytvoří pro spojení soket. Při příjmu dat je Ubicomem volána funkce `TCP_callback_recv`, která postupně přijatá data ukládá do bufferu. Jakmile je přijat celý paket, je zavolána metoda pro zpracování požadavku `TCP_ProcessRequest`. Po ukončení komunikace klientem volá Ubicom funkce `TCP_callback_close`, ve které uvolní prostředky dříve vytvořeného soketu.

Při zpracování požadavku funkcí `TCP_ProcessRequest` je nejdříve zkontrolován správný formát přijaté zprávy. To se děje ve funkci `TCP_CheckClientMessage`. Formát a pořadí dat musí odpovídat schématu, uvedenému v kap. 6.2.1. V této funkci se také vyhledá registrovaný blok, kterému je zpráva určena. Pokud není nalezen, zpracování končí neúspěšně. V opačném případě je otestována správná hodnota cookie a jsou uloženy informace o cílovém objektu a ID operace. Dále je provedeno dekódování vstupních argumentů pro tuto operaci.

Řízení se vrací zpět do funkce `TCP_ProcessRequest`, kde je zavolána metoda Perform cílového objektu. V případě modelové aplikace jde o instanci objektu `BoilerFunctionBlock`. Ta porovná ID cílové operace s těmi které poskytuje a pokud se ID shodují, je tato operace zavolána s případnými vstupními argumenty. Jakmile cílová operace skončí, jsou případné výstupní argumenty předány zpět do funkce `TCP_ProcessRequest`. Ta je předá funkci `TCP_SendReplyToClient`, která připraví a odešle odpověď klientovi pomocí dříve otevřeného soketu. Data jsou před odesláním zakódována do podoby odpovídající schématu v kap. 6.2.1.

Ohřev vody a publikace teploty

Požadovanou funkcionalitu bloku zajišťuje metoda `Boil` třídy `BoilerFunctionBlock`. Publikování aktuální hodnoty přísluší metodě `ProcessAndPublishMeasurement` stejného objektu.

Obě metody jsou specfické způsobem jejich volání. Jejich provádění je totiž zajištěno časovačem (oneshot timer) Ubicomu. První jejich zavolání probíhá těsně před nekonečnou smyčkou hlavního programu. V obou metodách je na konci jejich těl volána funkce časovače

[2] Veškeré funkce a metody zmíněné v této podkapitole jsou implementovány ve třídě `IEEE1451_NCAPBlock`.

`oneshot_attach`, která zajistí jejich opětovné vyvolání po uplynutí daného časového intervalu.

Ohřev vody je volán s menší frekvencí (časový interval je větší) a jeho cílem je upravovat hodnotu aktuální teploty, prezentované uživateli. Ta se mění podle toho, zda je bojler zapnutý či vypnutý nebo zda je nastavena určitá cílová teplota.

Metoda pro publikaci aktuální hodnoty má za úkol připravit obsah publikace a pomocí metody `Publish` příslušného publisher portu ji předat ke zpracování a odeslání. To se děje v metodě `MarshalAndPublishPublication` objektu `localNCAP`, kterou volá právě instance publisher portu. V této metodě dochází k vytvoření net bufferu, jeho naplnění hlavičkou a obsahem publikace podle kap. 6.2.2 a k následnému rozeslání publikace pomocí UDP paketu na všechny registrované odběratele.

Využití časovače má tu výhodu, že je možné stanovit interval volání obou funkcí bez toho, aby bylo v bloku použito aktivní čekání. Díky tomu může blok reagovat na žádosti klienta jen s minimálním zpožděním.

Kódování a dekódování datových typů

Kódování a dekódování použitých datových typů se provádí v metodách `Encode` a `Decode` objektu `localNCAP`.

Při kódování se příslušná data každého datového typu ukládají do net bufferu ve formátu, který je shodný s formátem implementovaným v Javě.

Během dekódování se pro každý přijatý datový typ alokuje paměť, do které jsou pak data typu uložena. Data se čtou pomocí funkce `MessageBuffer_GetByte` objektu `localNCAP`.

Protože má uzel Ubicom omezené paměťové možnosti a neposkytuje mechanismus podobný Garbage collectoru, je nutné veškerou použitou a nepotřebnou paměť ve vhodnou chvíli uvolnit. Vzhledem k množství datových typů a jejich specifické implementaci poskytuje objekt `localNCAP` metodu `FreeType`. Jejím účelem je uvolnit paměť podle zadaného datového typu tak, jak byla pro tento typ alokována.

6.5.7 Databáze

Blok databáze má za úkol zpracovávat data od bloku pro měření venkovní teploty. Je implementován v Javě a běží jako samostatný proces na PC.

Po spuštění je inicializován NCAP proces, je vytvořena instance funkčního bloku a subscriber portu a je provedena jejich registrace u lokálního NCAPu. Poté je nastavena síťová komunikace. V případě bloku databáze bude využita pouze komunikace podle modelu publish–subscribe. Jakmile je konfigurace dokončena, přechází blok do aktivního stavu.

Hlavní program bloku v nekonečné smyčce testuje příchozí publikace a tyto dále zpracovává. Po zpracování každé publikace je proces na krátkou dobu uspán.

Zpracování probíhá v metodě objektu `Callback_RoomTemperatures`, který byl registrován jako odběratel při inicializaci subscriber portu. V této metodě je ověřena správná struktura obsahu publikace a data publikace jsou následně zpracována (vzhledem k rozsahu se jedná pouze o jejich vypsání na standardní výstup).

Kapitola 7

Závěr

Diplomová práce navazuje na semestrální projekt, jehož náplní bylo prozkoumání způsobů připojování vestavěných systémů k internetu a prostudování možností standardu IEEE 1451.1. V rámci semestrálního projektu byla také navržena základní architektura systému podle uvedeného standardu.

Tento návrh byl v diplomové práci dále rozšířen a implementován na platformách Java a Ubicom. Při implementaci bylo nutno některé části návrhu přizpůsobit zvoleným architekturám bez ztráty obecnosti definované standardem.

Implementovaný systém podle navržené architektury byl úspěšně otestován a byla tak ověřena navržená funkcionalita a možnosti standardu IEEE 1451.1. Během implementace se ukázalo, že platforma Java je pro účely standardu zcela vyhovující a při realizaci příslušné části aplikace nebyly na této platformě zaznamenány žádné výrazné komplikace. Oproti tomu platforma systému Ubicom se projevila jako méně vyhovující. Hlavním důvodem byla absence podpory komunikace pomocí multicastu, což by zjednodušilo implementaci síťové komunikace mezi jednotlivými bloky systému. I přes tento nedostatek však Ubicom zastal očekávanou roli a z pohledu standardu pracoval bez potíží.

V dalším pokračování projektu by bylo možné využít realizace vybraných bloků pomocí speciálních jednočipových procesorů, podporujících platformu Java a obsahujících JVM. Takovým procesorem by mohl být nahrazen i uzel Ubicom a implementace celého standardu by tak byla provedena v jazyce Java, který je pro realizaci výhodnější pro své objektové vlastnosti.

Literatura

[1] Frank, R.: *Understanding Smart Sensors*. Artech House, Inc., 2000.

[2] IEEE: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model. IEEE 1451.1, 1999.

[3] May, G.: Differences and Commonalities. 1996, [Online; navštíveno 13.12.2008]. URL http://www.nim.com.au/inet_pub/in03006.htm

[4] Microsystems, S.: Java Platform, Standard Edition 6 Overview. 2006, [Online; navštíveno 20.5.2009]. URL http://java.sun.com/javase/6/docs/technotes/guides/index.html

[5] Microsystems, S.: All About Sockets. 2008, [Online; navštíveno 20.5.2009]. URL http://java.sun.com/docs/books/tutorial/networking/sockets

[6] Microsystems, S.: A Security Vulnerability in the Java Runtime Environment may Allow Code Loaded From the Local Filesystem to Access LocalHost. 2008, [Online; navštíveno 20.5.2009]. URL http://sunsolve.sun.com/search/document.do?assetkey=1-66-246387-1

[7] NIST: Brief Description of the Family of IEEE 1451 Standards. 2008, [Online; navštíveno 13.12.2008]. URL http://motion.aptd.nist.gov/

[8] OMG: Corba FAQ. 2007, [Online; navštíveno 13.12.2008]. URL http://www.omg.org/gettingstarted/corbafaq.htm

[9] Strach, M.: Diplomová práce. Technická zpráva, VUT Brno, 2002.

[10] Ubicom: Connectivity Kit User's Guide. Technická zpráva, Mountain View, California, 2001.

[11] Ubicom: IP2000 Family Software Development Kit Reference Manual. Technická zpráva, Mountain View, California, 2002.

[12] Ubicom: IP2012 / IP2022 Wireless Network Processors Data Sheet. Technická zpráva, Mountain View, California, 2003.

[13] Wikipedia: Embedded System. 2008, [Online; navštíveno 13.12.2008]. URL http://en.wikipedia.org/wiki/Embedded_system

[14] Wikipedia: Internet. 2008, [Online; navštíveno 13.12.2008].
URL http://en.wikipedia.org/wiki/Internet

[15] Wikipedia: Intranet. 2008, [Online; navštíveno 13.12.2008].
URL http://en.wikipedia.org/wiki/Intranet

[16] Wikipedia: Java applet. 2009, [Online; navštíveno 20.5.2009].
URL http://en.wikipedia.org/wiki/Java_applet

[17] Wikipedia: Object copy. 2009, [Online; navštíveno 20.5.2009].
URL http://en.wikipedia.org/wiki/Object_copy

Seznam použitých zkratek a symbolů

IEEE – Institute of Electrical and Electronics Engineers

CORBA – Common Object Requesting Broker Architecture

RPC – Remote Procedure Call

RMI – Remote Method Invocation

IP – Internet Protocol

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

IETF – Internet Engineering Task Force

HTTP – Hypertext Transfer Protocol

HTML – HyperText Markup Language

SNMP – Simple Network Management Protocol

FTP – File Transfer Protocol

SMTP – Simple Mail Transfer Protocol

ICMP – Internet Control Message Protocol

ARP – Address Resolution Protocol

OSPF – Open Shortest Path First

MAC – Media Access Control

SSH – Secure Shell

DNS – Domain Name System

VoIP – Voice over Internet Protocol

TFTP – Trivial File Transfer Protocol

URI – Uniform Resource Identifier

AD/DA – Analog to Digital / Digital to Analog

URL – Uniform Resource Locator

TEDS – Transducer Electronic Data Sheet

STIM – Smart Transducer Interface Module

TII – Transducer Independent Interface

NCAP – Network Capable Application Processor

NIST – National Institute of Standards and Technology

UART – Universal Asynchronous Receiver/Transmitter

SPI – Serial Peripheral Interface Bus

JVM – Java Virtual Machine

JAR – Java Archive

JDBC – Java Database Connectivity

GUI – Graphical User Interface

AWT – Abstract Window Toolkit

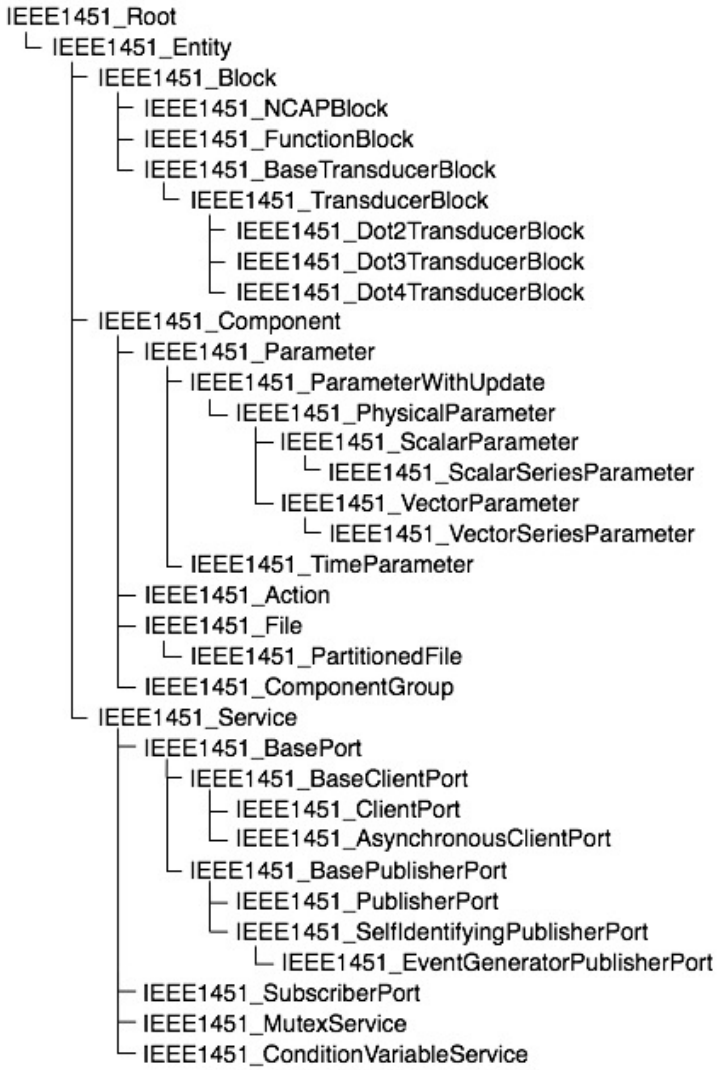
SDK – Software Development Kit

JDK – Java Development Kit

Seznam příloh

Příloha A

Hierarchie tříd standardu IEEE 1451.1



Příloha B

Ukázky zdrojových kódů

Uvedené zdrojové kódy jsou ilustrativní, nejedná se o kompletní výpisy.

Inicializace a registrace bloků v jazyce Java

```
IEEE1451_NCAPBlock local_ncap;
ValvesFunctionBlock valves_function_block;
IEEE1451_PublisherPort valves_publisher_port;

...

// result of operations
OpReturnCode result = new OpReturnCode();

// prepare this node dispatch address and initialize it
ObjectDispatchAddress local_ncap_dispatch_address = new ObjectDispatchAddress(
   _local_ncap_ip, _local_ncap_port, _local_ncap_object_tag);
local_ncap = new IEEE1451_NCAPBlock(
   local_ncap_dispatch_address, _local_ncap_object_name);

// init publisher port and set publication topic
valves_publisher_port = new IEEE1451_PublisherPort(local_ncap, _publisher_port_name);
result = valves_publisher_port.SetPublicationTopic(
   new PublicationTopic(new IEEE1451_String("valvesStates")));
if (!result.IsMajor(MajorReturnCode.MJ_COMPLETE)) { return result; }

// init function block
valves_function_block = new ValvesFunctionBlock(local_ncap, valves_publisher_port);

// register publisher port
ObjectDispatchAddress publisher_port_oda = new ObjectDispatchAddress();
result = local_ncap.RegisterObject(
   valves_publisher_port, valves_function_block, publisher_port_oda);
if (!result.IsMajor(MajorReturnCode.MJ_COMPLETE)) { return result; }

// register valves function block
ObjectDispatchAddress valves_function_block_oda = new ObjectDispatchAddress();
result = local_ncap.RegisterObject(
   valves_function_block, local_ncap, valves_function_block_oda);
if (!result.IsMajor(MajorReturnCode.MJ_COMPLETE)) { return result; }
```

Nastavení síťové komunikace a hlavní proces bloku v jazyce Java

```
// set up multicast emulation
local_ncap.EmulateMulticast(
   _local_ncap_ip.GetValueConcrete(), _local_ncap_port.GetValueConcrete());
local_ncap.EmulateMulticastAddTarget(_subscriber_ip, _subscriber_port);
local_ncap.DisableRealMulticast();

// all required block are instanciated - initialize network connections
if (!local_ncap.InitNetworkConnection()) {
   System.err.println("Client - Server communication couldn't be established.");
   System.exit(2);
}
if (!local_ncap.InitPubSubNetworking()) {
   System.err.println("Publish - Subscribe communication couldn't be established.");
   System.exit(3);
}

// set active state of ncap
local_ncap.Initialize();
local_ncap.GoActive();

while (valves_function_block.isOperating()) {
   // handle request from client, if there is any
   if (local_ncap.IsClientRequest()) { local_ncap.ProcessClientRequest(); }
   // handle ncap publications and subscriptions
   if (local_ncap.IsPubSubRequest()) {
      result = local_ncap.ProcessPubSubRequests();
      if (!result.IsMajor(MajorReturnCode.MJ_COMPLETE)) {
         System.err.println("ValvesControl PubSub processing failed: " + result.Major);
         System.exit(5);
      }
   }
   Thread.sleep(100);
}

// close network connections
local_ncap.FinishPubSubNetworking();
local_ncap.FinishNetworkConnection();
```

Implementace callback metody pro subscriber port

```
public class Callback_RoomTemperatures
   implements IEEE1451_SubscriberPort.IEEE1451_ISubscriber {

   public void callback_method(
      UInteger16 subscription_id,
      UInteger8 publishing_port_publication_key,
      PublicationTopic publishing_port_publication_topic,
      ArgumentArray publication_contents) {

      // check float array presence
      if (publication_contents.GetSize().GetValueConcrete() != 1) {
         System.err.println("Room temperatures set missing.");
         return;
      }
      // check floats count
      Float32Array values = (Float32Array) publication_contents.get(0);
      if (values.GetSize().GetValueConcrete() != Database.ROOM_SENSORS_COUNT) {
         System.out.println("Room temperatures set is not complete.");
         return;
      }

      // display temperatures
   }
}
```

Kostra aplikace Ubicomu

```
// initialize ubicom hardware
ubicomInit();

// initialize local NCAP
struct ObjectDispatchAddress local_ncap_oda;
local_ncap_oda.host = local_ncap_ip;
local_ncap_oda.port = local_ncap_port;
StringToObjectTag(local_ncap_object_tag, 2, &local_ncap_oda.object_tag);
localNCAP_init(local_ncap_oda, local_ncap_object_name);

// init publisher port
struct IEEE1451_PublisherPort *boiler_publisher_port = IEEE1451_PublisherPort_alloc(
   &localNCAP);
PublicationTopic publication_topic = "boilerMeasurement";
boiler_publisher_port->SetPublicationTopic(boiler_publisher_port, &publication_topic);

// init function block
boiler_function_block_init(boiler_publisher_port);
struct ObjectTag * boiler_function_block_ot =
   (struct ObjectTag *) heap_alloc(sizeof (struct ObjectTag));
StringToObjectTag(boiler_function_block_object_tag, 3, boiler_function_block_ot);
((struct IEEE1451_Entity *) &boiler_function_block)->SetObjectTag(
   (struct IEEE1451_Entity *) &boiler_function_block, *boiler_function_block_ot);

// register publisher port and function block
localNCAP.RegisterObject(boiler_publisher_port, &boiler_function_block, NULL);
localNCAP.RegisterObject(&boiler_function_block, &localNCAP, NULL);

// add publications target
localNCAP.AddPublishTarget(ip2dec(subscriber_ip), subscriber_port);

// initialize local NCAP
((struct IEEE1451_Block *) &localNCAP)->Initialize(
   (struct IEEE1451_Block *) &localNCAP);
((struct IEEE1451_Block *) &localNCAP)->GoActive(
   (struct IEEE1451_Block *) &localNCAP);

// start boiling process by invoking timer callback method
// timer is then reattached in that method
boiler_function_block.Boil(NULL);
boiler_function_block.ProcessAndPublishMeasurement(NULL);

// infinite loop for processing clients requests and managing node functionality
while (TRUE) {
   ip2k_eth_poll(edi);
   timer_poll();
}
```

Příloha C

Překlad a spuštění jednotlivých aplikací

1. Předpokládaná konfigurace systému

   Úspěšný překlad aplikací implementovaných v jazyce Java je podmíněn správným nastavením cest k JDK a nástroji Ant. Pro spuštění systému se předpokládá síťová konfigurace PC na IP adresu 10.1.1.1 třídy A. Dále se předpokládá že byl zkopírován obsahu adresáře `src` z přiloženého CD na disk C, který bude následně obsahovat tyto adresáře:

   ```
   C:\Ubicom
   C:\HeatingControl
   C:\WebServer
   C:\IEEE1451
   ```

2. Překlad zdrojových kódů pro Ubicom a jeho naprogramování

   Otevřít vývojové prostředí Unity.
   Otevřít projekt `BoilerControl.c_c` z adresáře `C:\Ubicom\BoilerControl`
   Přeložit projekt (v menu Build/Compile)
   Naprogramovat Ubicom (v menu Build/Start Programmer, pak kliknout na Program)

3. Knihovna IEEE1451

   V terminálu zadat posloupnost příkazů

   ```
   cd C:\IEEE1451
   ant
   ```

4. Aplikace pro jednotlivé bloky

   V terminálu zadat posloupnost příkazů

   ```
   cd C:\HeatingControl\BLOK
   ant
   java -jar dist\BLOK.jar
   ```

   Kde `BLOK` je vždy jeden z podadresářů `Database`, `OutsideSensors`, `RoomSensors`, `Server` a `ValvesControl`.

5. WebServer

   V terminálu zadat posloupnost příkazů

   ```
   cd C:\WebServer
   ant
   mkdir dist\classes
   mkdir dist\classes\controlstationapplet
   copy C:\IEEE1451\dist\IEEE.jar dist\classes
   copy C:\ControlStationApplet.html dist
   cd dist
   java -jar WebServer.jar
   ```

6. Applet

   V terminálu zadat posloupnost příkazů

   ```
   cd C:\HeatingControl\ControlStationApplet
   ant
   cd build\classes\controlstationapplet
   copy *.* C:\WebServer\dist\classes\controlstationapplet
   ```

7. Spuštění klienta

   Spustit internetový prohlížeč a zadat URL
   `http://localhost:8080/ControlStationApplet.html`