
Elektronický zdravotní záznam a telemedicína

doktorand:

MGR. JOSEF ŠPIDLEN

Ústav Informatiky AV ČR, EuroMISE Centrum,
Pod Vodárenskou věží 2, 182 07 Praha 8
spidlen@euromise.cz

školitel:

RNDR. ANTONÍN ŘÍHA, CSc.

Ústav Informatiky AV ČR, EuroMISE Centrum,
Pod Vodárenskou věží 2, 182 07 Praha 8
riha@euromise.cz

obor studia:
Biomedicínská informatika

Abstract

Dizertační práce zkoumá možnosti modelování a reprezentace medicínské informace a tzv. elektronického zdravotního záznamu. Vzhledem ke klíčovému požadavku univerzálnosti a dynamické modifikovatelnosti množiny sbíraných typů je pro uchování informací navržena a matematicky popsána grafová struktura zahrnující tzv. znalostní bázi a datové složky. Kromě způsobu ukládání informací text popisuje třívrstvou architekturu systému a způsob získávání uložené informace pomocí definovaného aplikačního rozhraní. Na závěr jsou zmíněny možnosti mobilního přístupu k záznamu, využití v telemedicínských aplikacích a závěry plynoucí z prvních testování.

1. Úvod

1.1. Cíle dizertačního projektu

Elektronický zdravotní záznam má potenciál uzavřít cyklus mezi klinickou praxí, výzkumem a výukou. Vzhledem k náročnosti jeho obsahu i použití musí být založen na nejlepších výsledcích informatiky i počítačových technologií. Na druhé straně, protože nelze snadno opustit ani přepracovat dosavadní systémy natož čekat s tvorbou nových na dobu, kdy technologie i informatika pokročí natolik, aby bylo možno splnit všechny požadavky na takový záznam kladené, otevřenost a modulárnost použitých systémů umožňující integraci nehomogenních komponent a jednoduchou modifikovatelnost množiny uchovávaných dat zde nabývá nezastupitelného významu.

V projektu “Elektronický zdravotní záznam a telemedicína” se snažíme o zvládnutí současného stavu problematiky elektronického zdravotního záznamu i souvisejících oblastí a prostředků, které poskytují současné softwarové – zejména databázové – technologie a výsledky medicínské informatiky, a o jejich rozpracování ve směru dokonalejšího naplnění kladených požadavků.

Vzhledem k aplikované povaze tohoto výzkumu je cíl projektu směřován k vytvoření a otestování funkční distribuované prototypové aplikace. Těžisko práce je v prozkoumání vhodnosti využití různých technik uložení dat, otestování soudobých XML komunikačních trendů, zvolení vhodných

metod pro začlenění multimediálních atributů do elektronického zdravotního záznamu a zvážení možností různého způsobu integrace formalizovaných lékařských doporučení. Celá práce je vytvářena s důrazem kladeným na podporu v telemedicínské oblasti, převážně je počítáno s možnostmi vzdáleného přístupu ke zdravotnímu záznamu pomocí Internetu a různých mobilních zařízení. Prototypovou platformou pro vývoj je Pocket PC 2002 a testovacím zařízením T-Mobile Mobile Digital Assistant.

1.2. Inspirace

Jedním z výzkumných směrů Evropského centra pro medicínskou informatiku, statistiku a epidemiologii – Kardio (EuroMISE Centra – Kardio) je aplikovaný interdisciplinární výzkum v různých oblastech medicínské informatiky. Součástí tohoto výzkumu jsou otázky reprezentace medicínských znalostí. Prototypová aplikace, na které pracujeme, je inspirována některými evropskými projekty z oblasti elektronického zdravotního záznamu (EHR), zejména projektem I4C/TripleC, na kterém EuroMISE centrum spolupracovalo a ve kterém vznikla dvouvrstvá pilotní aplikace EHR ORCA (Open Record for Care) [1]. Vyvíjená aplikace rozvíjí elektronický zdravotní záznam, který jsem implementoval v rámci své diplomové práce a je inspirována konzultacemi s lékaři – převážně kardiology – a českými, evropskými a mezinárodními standardy jako HL7, DASTA a komunikačními standardy ISO TC 215 a CEN TC 251 [2].

2. Architektura vyvíjeného EHR

2.1. Architektura MUDR

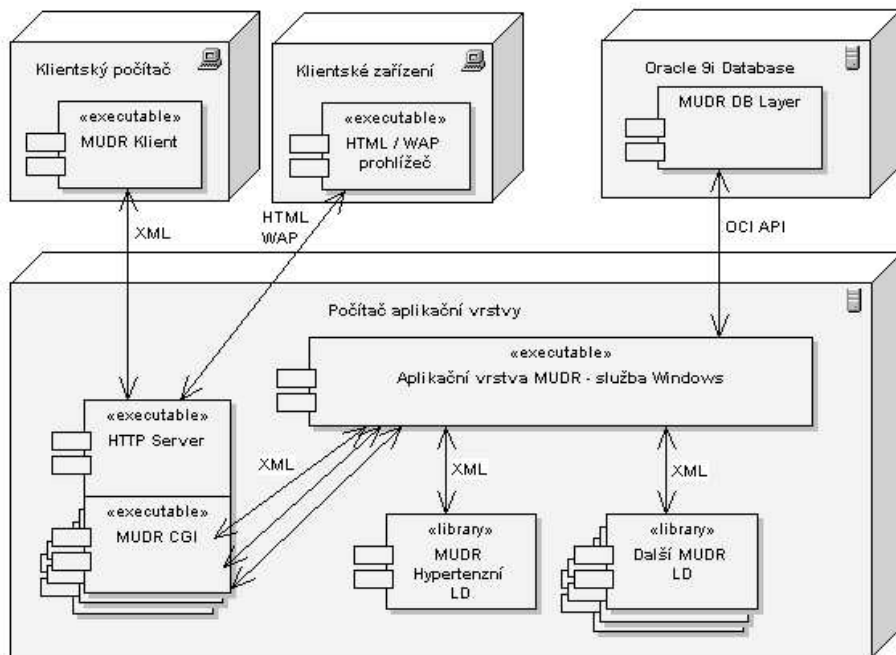
V rámci diplomové práce jsem pracoval na vývoji EHR MUDR (MUltimedia Distributed Record) [3], který je nyní modifikován a dále rozvíjen. Elektronický zdravotní záznam MUDR je založen na třívrstvé architektuře s datovou, aplikační a klientskou vrstvou. Tato dekompozice umožňuje oddělit jednotlivé funkční části systému a umožňuje uživatelským rozhraním pracovat na různých platformách. Funkce datové vrstvy spočívá v ukládání informací a kontrole základní referenční integrity dat. Aplikační (funkční) vrstva zajišťuje zjednodušený pohled na databázi a zpřístupňuje data uživatelským rozhraním a různým klientským aplikacím. Tito klienti komunikují s aplikační vrstvou pomocí MUDR API definovaného ve formě platných XML dokumentů odpovídajících příslušnému XML schématu. K přenosu tohoto XML je využito HTTP serveru na aplikační vrstvě. XML je na straně klienta zabaleno do HTTP POST žádosti a tím je přeneseno na vstup Gateway Interface (CGI) skriptu, který jej dále předává aplikační vrstvě.

Ve formě dynamických knihoven jsou ke službě aplikační vrstvy připojována lékařská doporučení. Jejich komunikace probíhá na úrovni win32 prostřednictvím exportovaných funkcí a sdílené paměti. Těmto funkcím je předáváno XML odpovídající MUDR API. V tomto ohledu získávají knihovny informace ze zdravotního záznamu pacienta stejným způsobem jako MUDR klienti. Architektura je zobrazena na obrázku číslo 1.

2.2. Architektura MUDR²

Architektura EHR MUDR² vychází ze základní koncepce třívrstvé architektury MUDR, kterou dále dekomponuje způsobem zobrazeným na obrázku číslo 2. Datová vrstva MUDR² již nemusí být nutně implementována na databázovém stroji Oracle. Komunikace obecného MUDR DB Serveru s aplikační vrstvou je zajištěna pomocí adaptačních modulů – “MUDR DB Connection Module”. Služba aplikační vrstvy – “MUDR Application Layer Service” – zvolí příslušný modul ve formě dynamicky připojitelné knihovny dle konkrétního databázového stroje. S využitím tohoto modulu již služba aplikační vrstvy komunikuje s datovou vrstvou plně transparentně.

Dále služba aplikační vrstvy MUDR² plní logicky obdobné funkce jako služba aplikační vrstvy MUDR; zajišťuje funkční logiku aplikace, připojuje knihovny lékařských doporučení (LD) apod. Rozdíl patrný na první pohled je ve zpřístupnění svých služeb. Pro komunikaci směrem ke klientům integruje služba aplikační vrstvy tzv. komunikační moduly. V první verzi probíhá implementace komunikačního modulu MUDR WS. Tento modul obsahuje a zpřístupňuje objekty implementující rozhraní “MUDR .NET Remoting API” (MUDRNRAPI). Pomocí RPC technologie založené na



Obr. 1: Schéma distribuované architektury EHR MUDR.

.NET Remotingu [4] je možno vzdáleně volat metody těchto objektů. Toto využívá další komponenta aplikační vrstvy – MUDR Web Service. Tato webová služba zpřístupňuje klientům aplikační rozhraní “MUDR Web Service Application Interface” (MUDRWSAPI). Typický lékař tedy pracuje s k EHR MUDR ze své pracovní stanice, využívá aplikačního HTTP serveru pro komunikaci a SOAP protokolu pro kódování příkazů a parametrů a přistupuje takto ke službě MUDR Web Service [5].

Pro případné využití EHR MUDR tenkými klienty ve formě HTML či WAP klientů na jednoduchých zařízeních je dále počítáno s vytvořením tzv. MUDR WS Proxy Service. Tato komponenta ve formě CGI by měla vystupovat na jedné straně jako klient MUDR Web Service a na straně druhé by měla zpřístupnit EHR MUDR ve formě HTML či WAP stránek.

3. Reprezentace dat

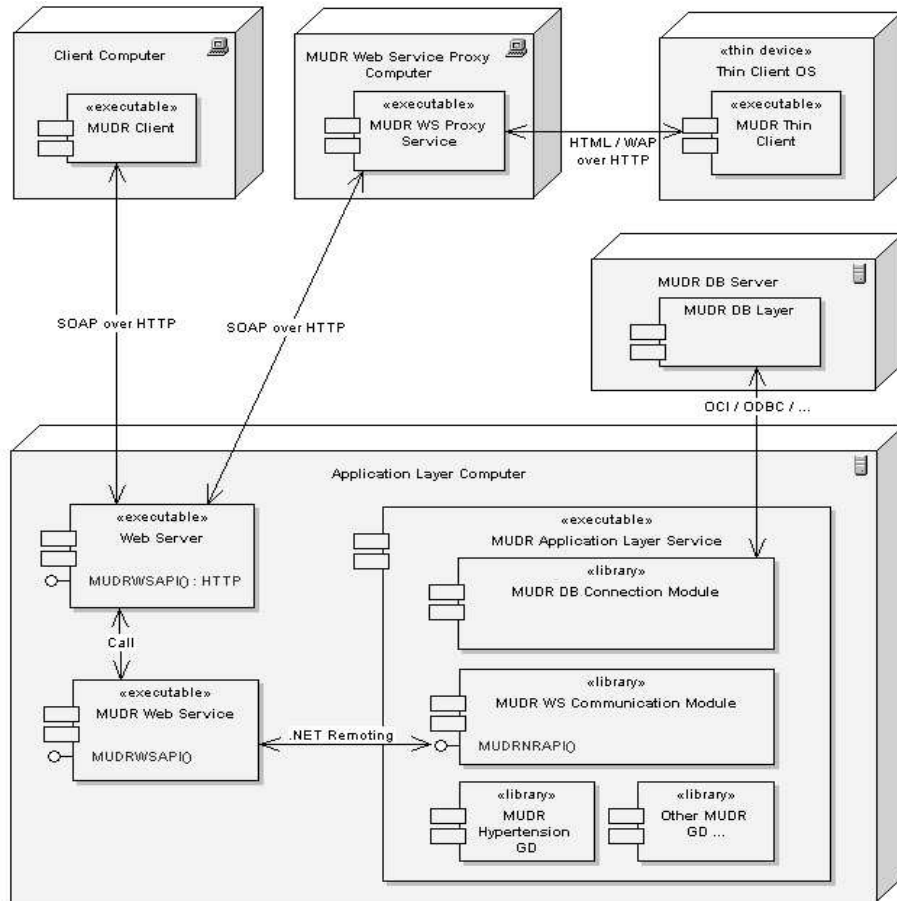
Z důvodů nezávislosti záznamu na použitém databázovém stroji nepopisuje práce fyzické schéma databáze. Místo toho je poskytnuto matematizované zobecnění, které je možno fyzicky implementovat pomocí různých databázových strojů různým způsobem. Některé databázové systémy umožňují využít hnížděných tabulek či objektově relačních technologií, v jiných systémech je třeba si vystačit s relačními tabulkami. Příslušnou transparentnost přístupu zajišťuje MUDR DB Connection Module.

Vzhledem k požadavku na dynamicky se měnící množinu sbíraných druhů údajů není možné využít jako základ pro zaznamenání hodnot klasickou relační tabulku se sloupěčky odpovídajícími jednotlivým ukládaným veličinám. Místo toho jsou k ukládání dat použity dále popsané struktury teorie grafů, tzv. znalostní báze v kombinaci s datovými složkami.

3.1. Znalostní báze

Hlavním úkolem znalostní báze je zachycovat druhy sbíraných údajů a vztahy mezi nimi. Formálně znalostní bázi *MUDR KB* definujeme jako orientovaný graf:

$$MUDR KB = (\mathbf{V}_{kb}, \mathbf{E}_{kb}). \quad (1)$$



Obr. 2: Schéma architektury MUDR².

Vrcholy grafu, prvky $n \in \mathbf{V}_{kb}$, označujeme jako uzly znalostní báze. Každý uzel znalostní báze je čtveřice:

$$n = (\gamma, \varphi, \omega, \epsilon_{nd}), \quad (2)$$

kde γ je jednoznačný identifikátor v rámci všech uzlů, φ je mnemotechnický, maximálně desetiznakový, řetězcový identifikátor, ω je datový typ a ϵ_{nd} obsahuje základní administrativní údaje například o tom, kdo a kdy vložil uzel do znalostní báze.¹ Hrany grafu $e \in \mathbf{E}_{kb}$ jsou také čtveřice:

$$e = (\alpha, \beta, \tau, \epsilon_{ed}), \quad (3)$$

kde $\alpha, \beta \in \mathbf{V}_{kb}$ označují odkud kam hrana vede, τ je typem hrany určujícím druh vztahu mezi koncovými uzly a ϵ_{ed} uchovává administrativní údaje, tentokrát týkající se příslušné hrany.

Mezi typy hran existuje jeden dominantní typ – tzv. *inferior* – vedoucí od rodiče k potomkovi a určující tímto hierarchický vztah mezi uzly znalostní báze. Vyjmeme-li ze znalostní báze hrany všech ostatních typů, požadujeme, aby vznikl orientovaný les s několika málo stromy. Tyto stromy označujeme jako *domény znalostní báze*. Každá doména sdružuje uzly sloužící ke stejnému účelu. Typicky právě jedna doména slouží pro uchování množiny všech sbíraných údajů o pacientovi. Uzly této domény nazýváme *sémantické typy*. Množinu sémantických typů značíme \mathbf{V}_s ; přirozeně

¹ Dále si z oblasti programování vypůjčíme indexové značení, například zápisem $n[\omega]$ budeme myslet třetí složku, tj. datový typ uzlu n .

platí $\mathbf{V}_s \subseteq \mathbf{V}_{kb}$. Jiné domény uchovávají například hierarchicky strukturovanou mezinárodní klasifikaci nemocí MKN10, anatomicko-terapeuticko-chemickou klasifikaci chemických názvů ATC či klasifikaci měrných jednotek SI nebo seznam léků dostupných na českém trhu. Hrany dalších typů umožňují vložit medicínskou znalost do této struktury. Představit si můžeme typy hran, které označují ekvivalenci dvou sémantických typů nebo například indikace a kontraindikace některého léku.

Uzly se stejným otcem nazýváme *bratry* a značíme $n_1 \diamond n_2$. Formálně je-li $n_1, n_2 \in \mathbf{V}_{kb}$, pak:

$$n_1 \diamond n_2 \stackrel{def}{\iff} \exists e_1, e_2 \in \mathbf{E}_{kb}, e_1[\alpha] = e_2[\alpha], e_1[\beta] = n_1, e_2[\beta] = n_2, e_1[\tau] = e_2[\tau] = \text{"inferior"}. \quad (4)$$

Triviálně vidíme, že bratrství je relací ekvivalence. Logicky, bratři slouží k podobným účelům, např. zaznamenání ulice a města v kontaktní adrese pacienta.

Uzly nemající otce nazýváme *kořeny domén* a pro $n \in \mathbf{V}_{kb}$ píšeme:

$$n \in \mathbf{R}_{kb} \stackrel{def}{\iff} (\forall e \in \mathbf{E}_{kb}, e[\tau] = \text{"inferior"} \implies e[\beta] \neq n). \quad (5)$$

Požadujeme, aby jméno φ bylo mezi bratry vždy jednoznačné. Stejnou jednoznačnost vyžadujeme mezi kořeny domén, tedy:

$$(n_1 \diamond n_2) \vee (n_1, n_2 \in \mathbf{R}_{kb}) \implies (n_1 = n_2) \vee (n_1[\varphi] \neq n_2[\varphi]). \quad (6)$$

Datový typ uzlu je významný především pro sémantické typy. Rozlišujeme základní datové typy jako číslo, boolean či text, multimediální datové typy: obraz, audio, video a binární soubor a speciální, tzv. referenční datové typy. Dále existuje pomocný datový typ adresář. Datové složky (viz def. 8 na str. 5) mající datový typ adresář mají prázdnou hodnotu a slouží například k seskupování svých podsložek. V nejnovější verzi MUDR² je na multimediální data uplatňován jednotný pohled a rozlišování je prováděno podle "Multipurpose Internet Mail Extensions Type" (mime-type, viz RFC 2048) atributu. V této verzi byl také přidán výčetový datový typ "enum".

3.2. Datové složky

Konkrétní zadaná data tvoří orientovaný les $MUDR DF$, formálně:

$$MUDR DF = (\mathbf{D}_{df}, \mathbf{E}_{df}). \quad (7)$$

Záznamy jednoho pacienta odpovídají vždy jednomu stromu v tomto lese. Vrcholy $d \in \mathbf{D}_{df}$ nazýváme datové složky a definujeme jako čtveřice:

$$d = (\delta, \sigma, \lambda, \epsilon_{df}). \quad (8)$$

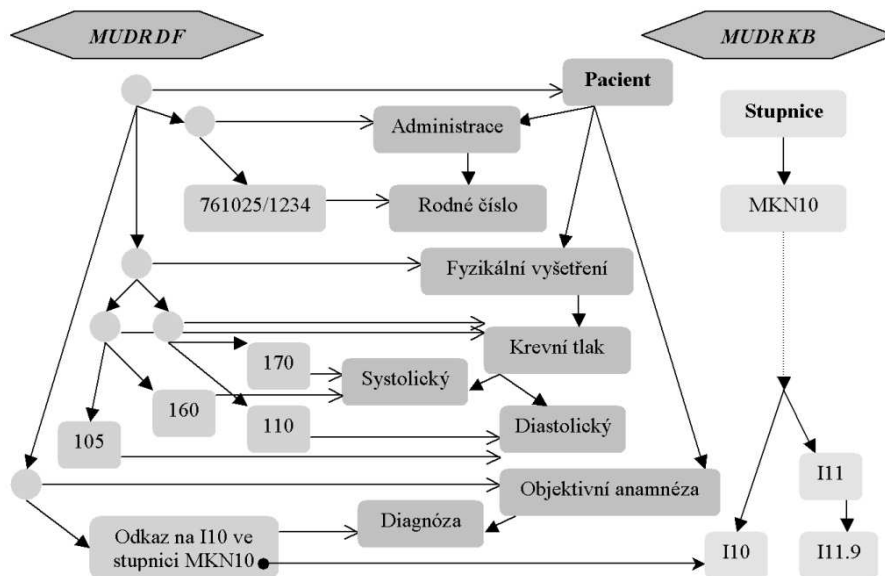
Zde δ je jednoznačný identifikátor v rámci všech datových složek, σ je sémantický typ ve znalostní bázi ($\sigma \in \mathbf{V}_s$)² a λ je tzv. hodnotou datové složky. Doména této hodnoty je určena datovým typem datové složky, který implicitně vyplývá z sémantického typu σ . Pod položkou ϵ_{df} se skrývá množství administrativních informací určujících kdo a kdy, zadal, smazal či potvrdil příslušnou datovou složku, jaké je období platnosti hodnoty, spolehlivost zadaného údaje apod.

Hrany $e \in \mathbf{E}_{df}$ jsou tentokrát netypané, vytvářejí pouze hierarchický vztah "otec - syn", tedy $\mathbf{E}_{df} \subseteq \mathbf{D}_{df} \times \mathbf{D}_{df}$. Navíc pro $MUDR DF$ musí platit:

² Implementace samotná je typicky prováděna formou odkazu na sémantický typ ve znalostní bázi.

1. $\forall d \in \mathbf{D}_{df} (\exists d' \in \mathbf{D}_{df}, (d', d) \in \mathbf{E}_{df} \vee d[\sigma] \in \mathbf{R}_{kb})$
2. $d, d' \in \mathbf{D}_{df}, (d', d) \in \mathbf{E}_{df} \implies \exists e \in \mathbf{E}_{kb} e[\alpha] = d[\sigma], e[\beta] = d'[\sigma], e[\tau] = \text{"inferior"}$

Slovně popsáno první podmínka vyjadřuje, že každá datová složka má otce nebo její sémantický typ je kořenem domény. Druhá podmínka tvrdí, má-li datová složka d otce d' , pak sémantický typ d je syn sémantického typu d' . Zjednodušeně řečeno, podmínky zaručují, že datová část odpovídá znalostní bázi a že v ní “neplavou” volné uzly. Malý příklad několika uložených dat a části znalostní báze je znázorněn na obrázku číslo 3.



Obr. 3: Příklad reprezentace dat v EHR MUDR².

4. Získávání informací z MUDR

4.1. Rozhraní pro přístup k datům

Pro účely komunikace s aplikační vrstvou EHR MUDR bylo definováno aplikační rozhraní MUDR API. Toto rozhraní má formu platných XML dokumentů odpovídajících XML Schématu MUDR API.xsd [6]. Existují 2 druhy platných XML dokumentů: směrem k aplikační vrstvě je používán dokument sestávající z identifikace uživatele a posloupnosti příkazů, směrem od aplikační vrstvy je používán dokument složený z posloupnosti odpovědí. Každému příkazu je přiřazen jednoznačný identifikátor v rámci XML dokumentu. Tento identifikátor kopíruje aplikační vrstva do odpovědi, aby klient správně a jednoduše rozpoznal, k jakému příkazu se příslušná odpověď váže. Ke každému příkazu jsou definovány možné odpovědi.

Aplikační rozhraní MUDR API bylo navrženo tak, aby obsahovalo relativně malou množinu příkazů a přitom umožňovalo plnohodnotně pracovat s elektronickým záznamem včetně úprav znalostní báze, správy uživatelů a jejich pravomocí, práce s výrazy v různých světových jazycích apod.

4.2. Typické použití aplikačního rozhraní

Jak je patrné z obrázku číslo 2 aplikační rozhraní pro EHR MUDR² je řešeno odlišně po technické stránce. Komponenta MUDR Web Service zpřístupňuje tzv. MUDRWSAPI. Toto aplikační rozhraní sestává z množiny funkcí, které je možno volat vzdáleně pomocí technologie webových služeb. Pro příkazy původního MUDR API nalezneme v MUDRWSAPI funkce provádějící obdobnou službu v kontextu nového EHR. Opačně toto pravda není. V původním EHR bylo analyzováno chování klientů a na základě získaných poznatků bylo API rozšířeno. Klienti nyní mohou požádat například o celý strom $T \subseteq \text{MUDR DF}$ odpovídající určitému pacientovi nebo získat určitý fragment

znalostní báze $K \subseteq \text{MUDRKB}$. V původní implementaci získávali klienti postupně (maximálně “po patrech”) jednotlivé datové složky $d \in \mathbf{D}_{df}$ či uzly znalostní báze $n \in \mathbf{V}_{kb}$ s přidanou informací o některých hranách v příslušné struktuře. Klienti implementují různé modifikace a kombinace klasických “Depth First Search” (DFS) a “Breadth First Search” algoritmů k prohledávání získaných grafových struktur. Celkem existuje 21 základních příkazů, z nichž pro potřeby prohledání informací ze záznamu pacienta jsou relevantní 4 příkazy uvedené v tabulce číslo 1.

Příkaz	Popis příkazu
<i>get.knowledge.domains</i>	Vypíše kořenové uzly znalostních domén.
<i>get.knowledge.node</i>	Vypíše detailní informace o uzlu znalostní báze.
<i>get.node.neighbours</i>	Vypíše seznam okolních uzlů k uzlu znalostní báze.
<i>get.data.files</i>	Vrátí určitou(-é) datovou(-é) složku(-y).

Tab. 1: Vybrané příkazy aplikačního rozhraní MUDR API.

Na příkladě v jazyce C++ si ukážeme algoritmus načtení podstromu znalostní báze. Každé volání metody `AppendChildren` načte další patro stromu znalostní báze. Vnější cyklus přes `pN2` prochází uzly z patra, které právě zkoumáme. Vnitřní cyklus přes `pN1` přidává postupně do `xmlNextLayer` požadavky na prozkoumání všech synů. Je-li alespoň jeden takovýto požadavek přidán, je před koncem metody voláno zpracování `xmlNextLayer.ProcessCmds()` a metoda `AppendChildren` je rekurzivně volána na výsledek. Objekty třídy `CMUDRXMLDoc` sdružují dva XML dokumenty, jeden shromažďuje příkazy a do druhého (`m_pXMLResp`) je uložena odpověď po `ProcessCmds()`.

```
bool CKnowledgeTree::LoadTree(int nRootNodeID)
{
    CMUDRXMLDoc xmlDoc;
    xmlDoc.AppendCmd.GetKnowledgeNode(nRootNodeID);
    if(xmlDoc.ProcessCmds()) return AppendChildren(xmlDoc);
    else return false;
}

bool CKnowledgeTree::AppendChildren(CMUDRXMLDoc& xmlDocNodes)
{
    IXMLDOMElementPtr pELRoot = xmlDocNodes.m_pXMLResp->GetdocumentElement();
    IXMLDOMNodePtr pN1, pN2 = pELRoot->GetfirstChild();
    CMUDRXMLDoc xmlNextLayer;

    while(pN2)
    {
        pN1 = pN2->GetfirstChild(); pN2 = pN2->GetnextSibling();
        if(!pN1) continue;

        do {
            CKnowledgeNode *pNode = new CKnowledgeNode(pN1);
            StoreKnowledgeNode(pNode);

            for(int i = 0; i <= pNode->m_dwaInfs.GetUpperBound(); i++) {
                xmlNextLayer.AppendCmd.GetKnowledgeNode(pNode->m_dwaInfs.GetAt(i));
            }

        } while ((pN1 = pN1->GetnextSibling()));
    }

    if(!xmlNextLayer.IsEmpty()) {
        if(xmlNextLayer.ProcessCmds()) return AppendChildren(xmlNextLayer);
        else return false;
    }

    return true;
}
```

Příkaz *get_data_files* lze použít třemi základními způsoby:

1. Dotaz na datovou složku $d \in \mathbf{D}_{df}$ pomocí jejího identifikátoru $d[\delta]$. Vracena je kompletní informace o datové složce. V případě multimediálních dat lze dokonce specifikovat některé vlastnosti hodnoty datové složky $d[\lambda]$ a systém provede příslušné transformace jako například převod do určitého formátu či nastavení úrovně komprese.
2. Dotaz na datové složky pomocí jejich společného otce. Vraceny jsou všechny datové složky $d \in \mathbf{D}_{df}$, které mají otce $d' \in \mathbf{D}_{df}$ specifikovaného dotazem, tj. pro vrácené datové složky d platí $(d', d) \in \mathbf{E}_{df}$.
3. Vyhledávání datových složek $d \in \mathbf{D}_{df}$ pomocí zadané klíčové hodnoty λ' , sémantického typu $d[\sigma]$ a upřesňujících parametrů vyjadřujících vztah mezi $d[\lambda]$ a λ' . Tento způsob je hodně flexibilní a lze jím vyhledat například datové složky systolického krevního tlaku většího než 180 mmHg nebo příjmení pacientů začínající řetězcem "Nov".

Při prohledávání datových složek i znalostní báze vyvstává potřeba odkazování se na určitý sémantický typ či libovolný jiný uzel znalostní báze. Jednou možností, jak se odkázat na uzel $n \in \mathbf{V}_{kb}$ je využít identifikátor $n[\gamma]$. Pro případ, že klient tímto údajem nedisponuje, lze využít tzv. *úplné jméno uzlu*. Úplné jméno uzlu n definujeme jako posloupnost $\varphi_0, \varphi_1, \dots, \varphi_k$, jestliže existují uzly $n_0, n_1, \dots, n_k \in \mathbf{V}_{kb}$, kde $\forall i \in 0..k \ n_i[\varphi] = \varphi_i$, $n_0 \in \mathbf{R}_{kb}$, $n_k = n$ a $\forall i \in 1..k \ \exists e_i \in \mathbf{E}_{kb}, e_i[\alpha] = n_{i-1}, e_i[\beta] = n_i, e_i[\tau] = \text{"inferior"}$. Úplné jméno uzlu zapisujeme tečkovou notací " $\varphi_0.\varphi_1.\varphi_2. \dots .\varphi_k$ ", tedy například "PATIENT.PHYS_EXAM.BP.SYSTOLIC".

Úplné jméno uzlu jednoznačně identifikuje uzel znalostní báze. Důkaz provedeme indukcí. Pro n_0 předpokládáme $n_0 \in \mathbf{R}_{kb}$. Dle (6) tedy $\forall n' \in \mathbf{R}_{kb}, n' \neq n \Rightarrow n'[\varphi] \neq n_0[\varphi] = \varphi_0$, tedy " φ_0 " jednoznačně identifikuje nějaký kořen znalostní domény. Předpokládejme nyní, že řetězec " $\varphi_0.\varphi_1.\varphi_2. \dots .\varphi_i$ " jednoznačně identifikuje uzel n_i . Postupujeme sporem, necht' řetězec " $\varphi_0.\varphi_1.\varphi_2. \dots .\varphi_i.\varphi_{i+1}$ " odpovídá dvěma uzlům $n_j, n_k \in \mathbf{V}_{kb}, n_j \neq n_k$. Uzel n_i je ale jednoznačně určen posloupností " $\varphi_0.\varphi_1.\varphi_2. \dots .\varphi_i$ ", tedy $\exists e \in \mathbf{E}_{kb}, e[\alpha] = n_i, e[\beta] = n_j, e[\tau] = \text{"inferior"}$. Stejně tak pro n_k máme $\exists e' \in \mathbf{E}_{kb}, e'[\alpha] = n_i, e'[\beta] = n_k, e'[\tau] = \text{"inferior"}$. Tedy z (4) plyne, že $n_j \diamond n_k$, pak ale dle (6) musí být $\varphi_{i+1} = n_j[\varphi] \neq n_k[\varphi] = \varphi_{i+1}$, což je spor.

5. Mobilní přístup k záznamu

5.1. Využití tenkých klientů

Ze schématu architektury na obrázku 1 je patrné, že pro typickou komunikaci mezi klienty a aplikační vrstvou je využíváno XML definované v [6]. Kvůli příkazově orientovanému charakteru tohoto XML je nutné, aby standardní klientské aplikace disponovaly nezanedbatelnou výpočetní silou.

Pro přístup k záznamu z tenkých klientů je využívána množina služeb na straně aplikační vrstvy, která transformuje příkazově orientované XML do HTML či WML jazyka. Tyto utility jsou implementovány jako speciální preprocesory ve formě CGI programů či HTTP Server modulů. Tímto způsobem může být veškerá složitost a aplikační logika přesunuta na prostřední vrstvu MUDR záznamu, což zjednoduší výpočetní nároky na uživatelské rozhraní a umožní využít klienty ve formě WWW prohlížečů, Pocket, Handheld či Tablet PC, PDA nebo mobilních telefonů. Jedním z prvních testovacích modulů byl modul pro Nokia 9110i Communicator. U tohoto zařízení bylo nutné pamatovat na to, že WWW prohlížeč nepodporuje ani tabulky ani rámce. Obecně je velmi důležité přizpůsobit výstup podmínkám malého přenosného zařízení. V porovnání z osobními počítači je třeba počítat s malým a často monochromatickým displejem, omezenými možnostmi ovládání, menší pamětí a výpočetní silou, pomalejším datovým přenosem atd. S přihlédnutím k těmto omezením zvažujeme řešení, které by umožňovalo předtrdit data dle specifikace lékaře a znalosti informací o pacientovi tak, aby pouze relevantní data byla přenášena k lékaři a zobrazována na mobilním zařízení. Ostatní data by byla přístupná až na speciální vyžádání. Tímto bylo došlo ke zmenšení objemu přenášených dat a snížení potřebné přenosové kapacity. Bohužel,

tento způsob stále naráží na obavy z možných následků chybného rozhodnutí na základě neúplné informace.

Jak je vidět ze schématu na obrázku 2, v nové verzi je použití tenkých klientů přesunuto až za tzv. “MUDR WS Proxy” službu. Je tomu tak proto, že využití tenkých klientů je čím dál více potlačováno. Důvody k tomuto jsou dva. Příliš jednoduché zařízení nedokáže dostatečně přehledně zobrazit potřebné informace tak, aby lékaři byli ochotni s těmito informacemi pracovat. Kromě toho s vývojem v oblasti mobilních komunikací přichází na trh mnoho mobilních zařízení, která umožňují implementace tzv. tlustých mobilních klientů, kterým se ve své práci snažím věnovat více.

5.2. Tlustí, ale mobilní klienti

Nejnovější trend výzkumu mobilního přístupu k EHR MUDR spočívá v tvorbě tzv. “.NET Compact Klienta”. K vývoji je používán “.NET Compact Framework” [7], který je přirozeně podporován ve vývojovém nástroji “MS Visual Studio .NET 2003” [8, 9]. Tento nástroj v kombinaci s programovacím jazykem C# přispívá ke vzniku MUDR mobilního klienta na platformách “Pocket PC” a “Smart Phone 2002”. Pomocí .NET Studia rozšířeného o tzv. “Smart Device Extensions”, případně v kombinaci s “Microsoft Mobile Internet Toolkit”, lze v současné době jednoduše vyvíjet pro více než 200 různých zařízení, přičemž tento počet velmi rychle vzrůstá. Díky tomu, že stále více zařízení používá Windows XP Embedded and Windows CE .NET [10] operační systémy, je vývoj do jisté míry transparentní záležitostí.

V současné době jako testovací a vyvíjecí platformu pro mobilního MUDR Klienta využíváme zařízení T-Mobile MDA. Tento mobilní digitální asistent kombinuje výhody mobilního telefonu podporujícího GPRS s osobním digitálním asistentem (PDA) na platformě Pocket PC 2002 Phone Edition (WinCE 3.0). Kombinací výkonu procesoru Intel Arm SA-1110 206MHz, barevného TFT 240 x 320 pix. dotykového displeje a 32 MB RAM rozšiřitelné pomocí MMC karet je poskytována dostatečná funkčnost pro použití v medicínském prostředí.

Další vývoj směřujeme k použití tzv. ultra osobních počítačů, které představila firma Microsoft na “Windows Hardware Conference 2002” (WinHEC 2002). Tyto počítače slibují posunout éru osobních počítačů kupředu srovnatelným způsobem jako posunul příchod mobilních telefonů svět telekomunikací. První produkt této řady by měl být k dispozici koncem roku 2003. Jde o plně funkční všestranný wireless handheld počítač, který jednoduše poslouží i jako notebook. Počítač měřící cca. 10,5 cm x 7,4 cm x 2,3 cm a vážící ne více než 250 gramů využívá operační systém Microsoft Windows XP Professional, zahrnuje 1GHz Crusoe TM5800 processor od firmy Transmeta Corporation, 4 palcový barevný VGA LCD displej s dotykovou obrazovkou, USB, zvukový výstup a podporu pro bezdrátové sítě 802.11b and Bluetooth.

5.3. Komunikace mezi mobilními klienty a MUDR záznamem

Jak již bylo řečeno, tlustý mobilní klient využívá MUDRWSAPI ke komunikaci s elektronickým zdravotním záznamem MUDR. V praxi to znamená, že na straně klienta je třeba vytvořit tzv. “proxy objekt” webové služby MUDR Web Service. Tento proxy objekt nazveme např. MUDRWSProxy a podědíme od třídy System.Web.Services.Protocols.SoapHttpClientProtocol. Pro každou metodu rozhraní MUDRWSAPI jsou vytvořeny 3 metody proxy třídy, jedna stejného názvu jaký má veřejná funkce rozhraní, jedna s prefixem Begin a jedna s prefixem End. První metoda slouží pro normální synchronní vyvolání určité funkce MUDRWSAPI, další dvě jsou určeny pro asynchronní komunikaci. Ve vývojovém prostředí MS .NET Studia je tvorba proxy třídy automatizována na základě WSDL dokumentu generovaného webovou službou [5]. Máme-li k dispozici proxy třídu, lze jednoduchým kódem vzdáleně volat rozhraní MUDRWSAPI:

```
try
{
    cz.euromise.unix.MUDRWSProxy mws = new cz.euromise.unix.MUDRWSProxy();
    mws.Url = "http://unix.euromise.cz:6004/MUDRWebServices/MUDRWS1.asmx";
    KnowledgeNode nd = mws.get_knowledge_node("PATIENT.PHYS_EXAM.BP.SYSTOLIC");
}
}
```

```
catch (Exception ex)
{   Msg.Text = "Chyba při volání služby: " + ex.Message; }
```

6. Závěr

Využití a možnosti mobilního přístupu k elektronickému zdravotnímu záznamu závisí podstatnou mírou na vývoji v oblasti mobilních komunikací a na výzkumu a rychlém vývoji podpůrných nástrojů pro lékaře. Praktické testování ukázalo, že na příliš malých a jednoduchých zařízeních není možné strukturovat medicínské informace tak, aby lékaři byli ochotni s elektronickým zdravotním záznamem tímto způsobem pracovat. Nicméně částečné využití se nachází i zde například v rychlém zobrazení přehledu stavu pacienta s možností online vkládání jednoduchých poznámek do zdravotního záznamu.

Na druhou stranu se v dnešní době objevuje na trhu spousta malých zařízení s relativně velkým displejem a dostatečnou silou na to, aby bylo možné implementovat plnohodnotné mobilní klienty elektronického zdravotního záznamu. Ve své práci se snažíme toto respektovat při vývoji MUDR mobilních modulů i při dalším výzkumu v oblasti telemedicíny. V oblasti mobilního přístupu k datům využíváme nové technologie a nejvíce podporované a rozšířené nástroje.

Bohužel tomu, aby mobilní přístup k elektronickému zdravotnímu záznamu přešel z testovacích laboratorních podmínek do běžné praxe každého lékaře, brání zatím dva zásadní problémy. Jedním je stále poměrně vysoká cena za přenesená data a druhým poměrně malá úspěšnost těchto zařízení při rozpoznávání rukopisu, zvláště pak jde-li o český jazyk. První problém by bylo možné řešit "cacheing technikami" v kombinaci s částečnou synchronizací dat občasným přímým propojováním s pevnou pracovní stanicí, na řešení druhého problému pracují vývojáři komerčních firem.

Poděkování

Práce je částečně podporována projektem LN00B107 Ministerstva školství, mládeže a tělovýchovy České Republiky.

Literatura

- [1] Zvárová J., Hanzlíček P., Přibík V.: "Application of ORCA multimedia EPR in Czech hospitals", *Proceedings of 3rd European Conference on Electronic Healthcare Records*, Sevilla 1999, ss. 160–165.
- [2] CEN/TC251, ENV 13606, části 1–4.
- [3] Špidlen J.: "Databázová reprezentace medicínských informací a lékařských doporučení", *Diplomová práce*, UK MFF, Katedra softwarového inženýrství, 2002.
- [4] McLean S., Naftel J., Williams K.: "Microsoft .NET Remoting", Microsoft Press, 2002, ISBN 0-7356-1778-3.
- [5] Shortm S.: "Building XML Web Services for the Microsoft .NET Platform", Microsoft Press, 2002, ISBN 0-7356-1406-7.
- [6] Špidlen J., EuroMISE Centrum - Kárdio: "Definice aplikačního rozhraní MUDR API", <http://www.euromise.cz/MUDRAPI.xsd>, 2002.
- [7] Wigley A., Wheelwright S., Burbidge R., MacLeod R., Sutton M.: "Microsoft .NET Compact Framework (Core Reference)", Microsoft Press, 2003, ISBN 0-7356-1725-2.
- [8] Kačmář D.: "Programujeme .NET aplikace ve Visual Studiu .NET", Computer Press, 2001, ISBN 8072265695.
- [9] Prosis J.: "Programming Microsoft .NET", Microsoft Press, 2002, ISBN 0-7356-1376-1.
- [10] Boling D.: "Programming Microsoft Windows CE .NET, Third Edition", Microsoft Press, 2003, ISBN 0-7356-1884-4.