Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Implementace openEHR do elektronické zdravotní dokumentace

Plzeň 2015 Ondřej Havlíček

Prohlášení

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

V Plzni dne 13. května 2015

Ondřej Havlíček

Abstrakt

Práce je zaměřena na nový systém, elektronickou zdravotní dokumentaci (EZD), vyvíjený neuroinformatickou skupinou na Katedře informatiky a výpočetní techniky.

Systém je založen na konceptu *openEHR*, který odděluje popis doménových dat a implementaci pomocí vícevrstvého modelování (generický referenční model a doménově specifické archetypy).

Cílem této práce je vytvoření modulu pro EZD, který umožní automatické generování grafického uživatelského rozhraní na základě struktury archetypu. Modul je schopen zpracovat archetypy a vytvořit webové formuláře pro zadávání, editaci a zobrazení dat archetypu. Implementace využívá standardní EHR knihovny *openEHR Java libraries* a webový framework Vaadin.

Druhá část práce se zabývá zdokonalením elektrofyziologických doménových archetypů, kde navrhuje systém pro párování archetypů s odML terminologiemi a řeší automatizaci získávání odML identifikátorů pro dané terminologie.

Abstract

The work is focused on a new personal electronic health record (EHR) system; i.e. the system which is being developed bellow neuroinformatics research group at the Department of Computer Science and Engineering.

The system is based on the *openEHR* concept, which allows a separation of domain description and implementation details via two/three layer modelling (generic reference models and domain dependent archetypes).

The goal of this thesis is to develop a module for the system, which will handle an automatic GUI generation according to archetype structure. The archetypes are processed and the set of web-based forms for manual data input, edit and view is generated. Implementation itself uses web framework Vaadin and *openEHR Java libraries*.

Second part of the work deals with the electrophysiology domain archetypes improvement. It proposes the archetypes binding with the odML terminology and it solves an automatic odML terms identification.

Poděkování

Rád bych poděkoval vedoucímu diplomové práce za vedení, věcné připomínky a odborné rady během zpracování této práce.

Obsah

1 Úvod

Přehledné a intuitivní grafické uživatelské rozhraní (GUI) patří mezi základní předpoklady pro přežití jakéhokoli softwaru[1]. Tvorba takového rozhraní je zpravidla běh na dlouhou trať. V některých oblastech lidského života se však báze znalostí a nové potřeby vyvíjejí rychleji než software samotný a software zaostává prakticky okamžitě po jeho vydání. Následují iterace změn, které stojí nemalé finanční prostředky a vedou často ke znepřehlednění výsledné aplikace. V takových případech do hry vstupují automaticky generovaná GUI.

V oblasti zdravotnictví je vývoj taktéž nezastavitelný. Nejen, že jsou vyvíjeny nové a nové metody, postupy, způsoby léčby a vyšetření, ale do měření a uchovávání medicínských dat začínají velmi významně zasahovat i samotní pacienti.

Quantified self [2] tedy něco jako vyčíslené já – fenomén dnešní doby, se kterým souvisí sebesledování a evidence údajů, to všechno přináší potřebu vytvářet aplikace s „user-friendly“ rozhraním, jejichž vývoj nebude vyžadovat úpravu aplikace jako takové.

Obecně známé a používané způsoby data–driven development vývoje GUI jako MVC, Groovy, Spring nebo Ruby on Rails mají několik zásadních nevýhod, co se podkladových dat týče. Data, ale i datové modely zastarávají a pomocí klasických přístupů není jednoduché takové aplikace udržovat.

Na druhou stranu jsou tu formalismy, které se částečně s daným problémem vyrovnávají. Jedním z nich je *openEHR* – standard pro EHR aplikace založený na metodologii Multi-Level Modelling (MLM). S jejich pomocí dokážeme vytvořit aplikace, jejichž GUI bude řízené daty, které má zpracovávat, aniž by bylo potřeba při změnách zasahovat do kódu samotného programu.

V neposlední řadě se i v oblasti výzkumu (například medicínské informatiky) setkáváme s potřebou a nutností ukládat či sdílet medicínská

data. Pro výměnu dat mezi vědeckými pracovišti je důležité především to, aby byla data ukládána ve standardizovaných strukturách tak, aby byla data čitelná pro všechny. Tento problém také uspokojivě řeší standard *openEHR*.

V teoretické části bude popsána doména, v níž se tato práce pohybuje, využité technologie a dvě již existující řešení. V praktické části je popsán způsob, jakým funguje vytvořený modul, a je představeno grafické rozhraní, které generuje včetně jeho vlastností a výhod, které přináší.

2 Elektronická zdravotní dokumentace

Na KIV, v rámci výzkumu neuroinformatické skupiny, vzniká systém Elektronické zdravotní dokumentace (EZD). Jedná se o EHR systém, který bude schopen ukládat ve vhodném formátu medicínská data, včetně dat z EEG experimentů, a další osobní lékařsky relevantní informace.

Uživatelům by taková aplikace přinesla možnost přístupu ke krátkodobým i dlouhodobým zdravotním statistikám a vyhodnocování trendů sledovaných dat. Navíc by přispěla k možnosti včasné diagnózy případných zdravotních problémů či minimálně přispěla ke zlepšení preventivního zdravotního systému.

Podle [3] je cílovým uživatelem libovolný jedinec, ale hlavní oblast zájmu se soustředí na tři skupiny:

- Zdraví dospělí lidé – příležitostné monitorování zdravotního stavu, motivace ke zdravému životnímu stylu, prevence.
- Senioři – každodenní monitorování za účelem dlouhodobého sledování vývoje konkrétních zdravotních ukazatelů. Bude mít jak preventivní, tak monitorovací význam pro aktuální choroby.
- Aktivní lidé, sportovci – motivace, zvyšování a sledování výkonu, porovnání s ostatními, dietní plány.

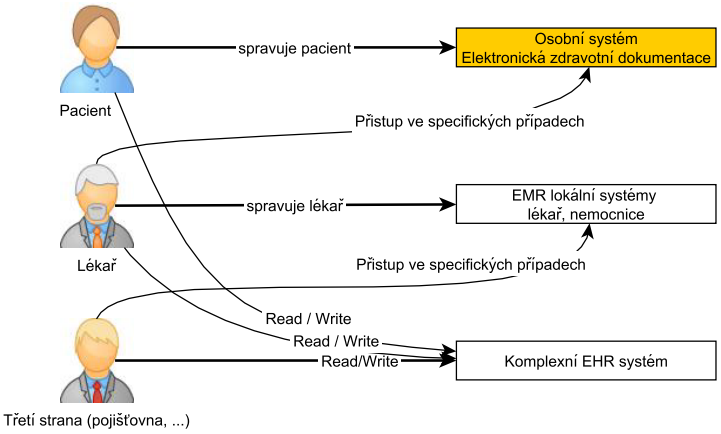
Ale bezesporu existují i další skupiny:

- Rodiče malých dětí – sledování životních funkcí (puls, kyslík v krvi) dítěte během spánku. Zde by šlo spíše než o prevenci o uklidnění samotných rodičů. Již dnes jsou vyvíjeny technologie pro taková měření[1].

[1] `https://www.owletcare.com`

2.1 Obecný EHR systém

Electronic Health Record (EHR) je z obecného pohledu systém pro ukládání medicínských dat. Podle [3] existují tři základní typy EHR systémů, viz obrázek 1.

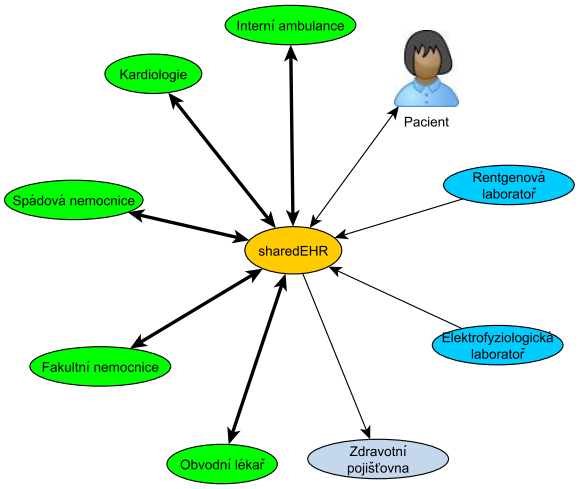


Obrázek 1: Základní typy EHR systémů, převzato z [3]

- Osobní EHR – pouze uživatelé mají přístup read/write do EHR. Většinou je nezávislý na standardech pro konkrétní zemi.
- Lékařské EHR – přístup mají pouze lékaři, nemocnice, specialisté. Závislé na standardech dané země.
- Kombinace obou – sdílený EHR systém viz obrázek 2 – přístup mají obvykle pacienti i lékaři. Často také třetí strana, jako pojišťovací společnosti/veřejné pojišťovny. Mohou se přes ně sdílet výsledky laboratorních vyšetření, nebo rentgenové snímky.

Existuje několik různých standardů, podle kterých lze vyvíjet obecný EHR systém. Podle [3] jsou to především následující:

- HL7 – set standardů pro přenos medicínských dat, terminologií a metodologií.
- CEN/ISO 13606 – evropský standard pro vývoj EHR. Očekává se, že bude mapováno na HL7 v3 [3].



Obrázek 2: Sdílený EHR systém

- *openEHR* – viz kapitola 5

2.2 Současný stav a výběr technologií

Jelikož vyvíjená EZD patří do kategorie osobních systémů, viz obrázek 1, byl pro implementaci vybrán standard *openEHR*, protože je více zaměřen na samotného uživatele, než například HL7 [3].

Vývoj aplikace byl rozdělen na čtyři hlavní části:

- Modelování archetypů a šablon (templates) – viz kapitola 9.1.
- Mapování modelů na perzistetní vrstvu – vývoj probíhá současně (není známé rozhraní k datové vrstvě).
- Poloautomatické generování GUI – je předmětem této práce.
- Analytické funkce a pluginy – zatím nerealizováno.

3 Doména: elektroencefalografie

Elektroencefalografie (EEG) patří mezi neinvazivní metody pro záznam aktivity lidského mozku. Principem metody je měření změn elektrických potenciálů na povrchu hlavy. Tyto změny potenciálů jsou způsobeny aktivitou neuronů v thalamu a mozkové kůře. Takové řešení má řadu výhod – finanční dostupnost, neinvazivnost, rutinní scénář vyšetření a měření spontánní aktivity [4]. Ale zároveň některé nevýhody. Významnou nevýhodou je podle [4] to, že výsledný EEG signál je velmi hrubý, protože se do něj promítá obrovské množství nerelevantních zdrojů neurální aktivity. Z toho plyne obtížnost odvození odpovídajících neurokognitivních procesů z EEG záznamu.

EEG se měří pomocí přístroje zvaného elektroencefalograf. Mohou se použít buď jednotlivé elektrody nebo elektrodová čepice (obrázek 3). Elektrody mohou být povrchové nebo podpovrchové (pak už se nejedná o neinvazivní metodu). Vodivou vrstvou jsou většinou tělní tekutiny nebo speciální vodivé gely. Dokonce existují jednoduché jednoelektrodové encefalografy, které se dají bezdrátově připojit k mobilnímu telefonu [3].

Podle [5] a [6] jsou mozkové vlny primárně rozdělené na:

1. Alfa vlny
    - frekvence 8-13 Hz
    - amplituda 30-80 μV
    - produkovány zdravým, bdělým a plně vyvinutým (od osmi let věku) mozkem
    - objevují se jen, pokud jsou zavřené oči
2. Beta vlny
    - frekvence 14-40 Hz
    - amplituda 10-20 μV někdy 20-30 μV

Obrázek 3: Elektrodová čepice, převzato z [5]

- produkovány častěji u žen
- neměly by být synchronní v obou hemisférách

3. Delta vlny
   - frekvence menší než 4 Hz
   - amplituda menší než 30 μV
   - obvykle symetrické na obou hemisférách
   - obvykle do jednoho roku věku
4. Theta vlny
   - frekvence menší než 4-7 Hz
   - amplituda 10-300 μV

- obvykle stav mezi bděním a sněním (někdy také při meditaci nebo modlení)

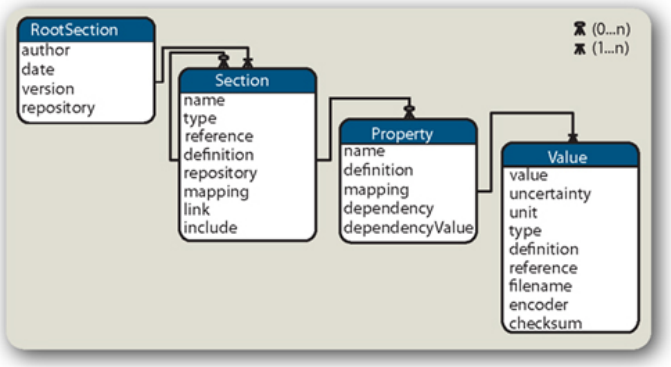
4 odML

Open metadata Markup Language – jazyk sloužící k přenosu metadat byl vyvinut k potřebám neurofyziologie. Snaží se zavést jednoduchý formát pro sběr, zpracování a výměnu dat v automatizované, strojově čitelné formě [7]. Data jsou zde ukládána stylem key–value dvojic v hierarchické struktuře. Zároveň definují vlastní terminologii pro část dat popsaných pomocí odML.

4.1 Datový model

OdML kombinuje obecný datový model s doménově specifickými terminologiemi a snaží se poskytovat maximální možnou flexibilitu. Celý model je navržen jak pro výměnu metadat, tak pro definici terminologií [7]. Díky tomu je velice flexibilní a adaptabilní vůči neustálým změnám, kterým čelí.

Na obrázku 4 je vidět datový model ve formě entity–relation diagramu.



Obrázek 4: odML datový model, převzato z [7]

Základní entitou je Property. `Property` představuje společně s `Value` zjednodušeně řečeno key–value pár. S tím, že `Property` má buď jednu nebo více `Values`. `Property` jsou dále uspořádány v sekcích, čímž se zajistí možnost použít stejné jméno na více místech a tím pádem se sníží i celkový objem přidaných metadat. Samotná struktura je čistě stromová a je reprezentovatelná pomocí XML se schématem[2].

[2] `http://www.g-node.org/projects/odml/odMLSchema.png`

4.2 odML terminologie

V rámci odML již byly vytvořené terminologie pro reprezentaci dat z neuroinformatické domény. Samotné termíny nemají v rámci terminologie, žádné vlastní ID a jsou definovány pouze použitím atributu <definition> uvnitř samotné terminologie. Tedy každý výskyt tohoto atributu definuje jeden konkrétní termín z terminologie.

Hotové odML terminologie jsou dostupné na

`https://github.com/G-Node/odml-terminologies`

ve formě úplných XML souborů, nebo pak na

`http://portal.g-node.org/odml/terminologies/v1.0/`

kde jsou v přehledných tabulkách vypsány všechny termíny z dané terminologie. Například pro terminologii pro experiment (viz příloha A) je vidět dané zobrazení na obrázku 5

Name	Value	Unit	Type	Definition
Description			text	A description of the experiment.
Type	electrophysiology		string	The type of experiment.
	behavior		string	
	simulation		string	
	imaging		string	
	psychophysics		string	
Subtype			string	
ProjectName			string	The name of the project this experiment belongs to.
ProjectID			string	The ID of the project this experiment belongs to.

Obrázek 5: odML Experiment – zobrazení terminologie

5 openEHR

OpenEHR je názvem nadace (openEHR Foundation), která vytvořila stejnojmenný formalismus, či lépe řečeno specifikaci pro vývoj plnohodnotných EHR systémů. Celá specifikace je postavena na metodologii známé jako Multi-Level modelling. V tomto případě konkrétně dvouvrstvé modelování archetype–template + referenční model[3].

OpenEHR architektura ztělesňuje 15 let výzkumu z mnoha projektů a standardů vzniklých po celém světě a byla vytvořena na základě požadavků sbíraných několik let. [8]

Hlavní výhodou *openEHR* je její vysoká genericita, díky níž je možné ji využít prakticky kdekoli mimo původní doménu, klinické EHR. Podle [8] ji lze stejně snadno použít například pro veterinární záznamy nebo dokonce správu budov pro města a podobně.

Toho se dá dosáhnout především díky tomu, že referenční model zachycuje pouze velmi obecné vztahy a používá generické objekty. Teprve archetypy a šablony definují popisované entity a administrativní události s nimi spojené. Jedná se vlastně o odstínění vazby na konkrétní doménu od samotného referenčního modelu.

Klasický software přináší nové možnosti jen v okamžiku vydání nové verze. Výhodou softwaru vytvořeného podle *openEHR* architektury je to, že ho lze efektivně upravovat, aniž by bylo nutné vydávat nové verze či dokonce vůbec kontaktovat výrobce. Je možné z jednoho solidně navrženého systému vytvořit několik specifických subsystémů podle domény a jejich výstupy pak efektivně shromažďovat v centrálním úložišti.

Na obrázku 2 je vidět typický EHR systém. Zelené bubliny představují elektronické záznamy pacienta (EPR). Jsou to obecně záznamy od specializovaných lékařů, obvodních lékařů či zprávy z nemocnic. Jednotlivé

[3] Archetypy a šablony společně tvoří jednu vrstvu, ačkoli se někdy mluví o třívrstvém modelování archetyp + template + referenční model.

systémy mohou být díky správně vybrané množině archetypů a především šablon připraveny „na míru“ pro konkrétní pracoviště.

Především pro lékaře specialisty bude výběr správné množiny šablon klíčový. Čím více se množina omezí, tím přehlednější bude výsledná aplikace. Na druhou stranu nesmí chybět žádná důležitá součást.

Díky velice jednoduchým referenčním modelům mohou také výrobci různé medicínské nebo nositelné elektroniky, tzv. wearables připravit rozhraní, které dovolí pacientovi ukládat svá vlastní data do svých záznamů. Tato data mohou být přísně medicínská jako například domácí Holterovská monitorace krevního tlaku (dále TK), hladina krevního cukru nebo hladiny ketolátek v moči.

Stejně tak se však může jednat o nemedicínská data. Například záznamy ze sport–testerů, subjektivní záznamy o sportovních výkonech či přijímaných kaloriích nebo v poslední době velmi častá spánková data z chytrých mobilních telefonů. Také tato data však mohou být relevantní pro celkový obraz, který lékař o pacientovi dostane.

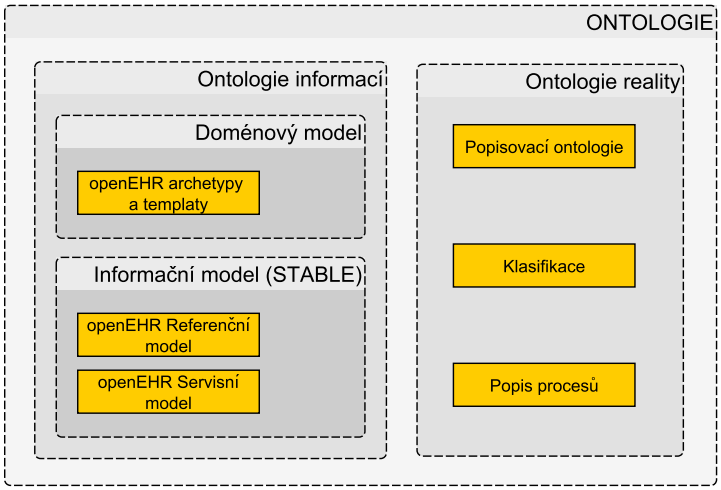
5.1 Oddělení ontologií

Jedním ze základních principů *openEHR* je oddělení ontologií [8]. Na obrázku 6 jsou vidět dvě základní oddělení.

Primární je separace informačních ontologií od ontologií reality. Toto oddělení je zcela logické a pramení z reálného oddělení autorů těchto ontologií.

Ontologie reality musí popisovat odborník na danou doménu. Jedná se například o pozice lidského těla při měření krevního tlaku nebo typy bakteriálních infekcí. Oddělení ontologií reality zároveň výrazně zjednodušuje možnosti vícejazyčných aplikací.

Ontologie informací pak vytváří zpravidla softwarový architekt, ačkoli i on musí mít jistou znalost dané domény. Zde se jedná zejména o způsoby uložení dat (informací) neboli referenční model – neměnné generické datové typy pro popis informací.



Obrázek 6: Oddělení ontologií

Sekundární oddělení je mezi doménovým modelem a informačním modelem. Informační model je naprosto nezávislý na doméně, kterou popisuje. Jedná se především o základní datové objekty, které by měly být znovupoužitelné a dostatečně obecné. Takový model je pak neměnný a jednotlivé datové typy jsou perzistovatelné.

Doménový model je těsně svázán, jak název napovídá, s konkrétní popisovanou doménou. V tomto modelu jsou popisy konkrétních datových struktur, které patří do dané domény jako například měření krevního tlaku, které se skládá z několika datových typů informačního modelu.

5.2 Dvouvrstvé modelování

Jedno z klíčových paradigmat konceptu *openEHR* je podle [8] Multi Level Modeling [9]. Dvě úrovně modelu představují:

- neměnný referenční model
- formální definice dat / doménového modelu

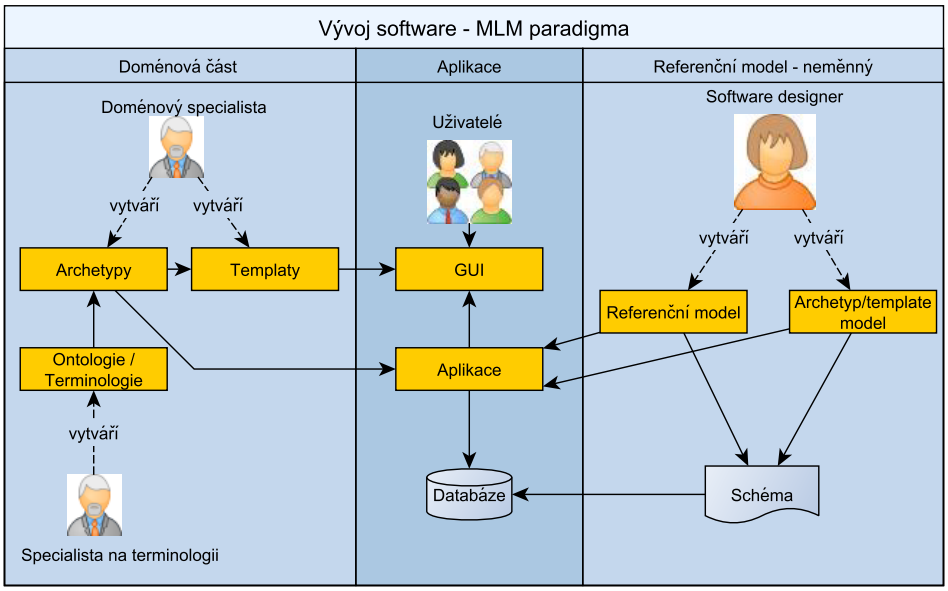
Důležité je, že pouze první část je implementována programově. Druhá část je implementována ve formě archetypů a šablon a stojí obvykle mimo program. Jako důsledek takového oddělení je především vysoká znovupoužiti-

telnost a možnost vytvořit software pomocí málo provázaných komponent. Takový systém je díky tomu samozřejmě mnohem robustnější a podle [8] zároveň velice adaptabilní právě tím, že dokáže konzumovat archetypy tak, jak jsou vyvíjené nezávisle na samotném softwaru.

Celý systém popisuje [1] jako stavebnici LEGO®, kdy referenční model představuje neměnné atomické prvky (jednotlivý díly stavebnice), ze kterých lze jejich libovolnou kombinací stavět složitější a komplexnější objekty, v našem případě archetypy a šablony. Ty tvoří základ celého systému.

Společně s grafickými objekty a logikou, která mapuje jednotlivé prvky referenčního modelu na konkrétní grafické objekty dostáváme základní model výsledného softwaru.

Vývoj systémů postavených na paradigmatu dvouvrstvého modelování je z velké části vývojem generické bussiness logiky, robustního systému pro ukládání a dotazování, které jsou poté velmi stabilní a obvykle neměnné. Samotná sémantika je doménově závislá a tvoří ji specialista na danou doménu. Obrázek 8 ukazuje základní schéma pro vývoje aplikace s použitím MLM paradigmatu.



Obrázek 7: Vývoj software – MLM paradigma

Podle [8] je klíčovým výsledkem tohoto přístupu to, že archetypy představují technologicky nezávislé vyjádření doménové sémantiky, které je použito k řízení databázových schémat, bussiness logiky i definici GUI a systémových zpráv.

5.3 Archetypy

Archetypy tvoří spolu s šablonami dynamickou část MLM paradigmatu uvnitř *openEHR*. Jedná se v podstatě o množinu omezení, která jsou aplikována na referenční model. Tím vytváří množinu instancí, které odpovídají danému předmětu archetypu - tím může být například velmi obecný archetyp „laboratorní výsledek“[8]. Tento velmi obecný archetyp se dále upravuje, omezuje a derivuje se z něj obecnější (například doménově specifický) objekt typu „laboratorní výsledek“.

Koncepty definované na této úrovni jsou již doménově závislé a tvoří jakési jasně uchopitelné jednotky z pohledu dané domény. Pro představu může se jednat o krevní tlak, hladinu cukru v krvi a další.

Tyto jednotky jsou sice dále dělitelné, ale jejich rozdělením by z pohledu domény mohlo dojít ke ztrátě informací. Například systolický tlak bez hodnoty diastolického je jen částečná informace.

Všechny informace, které lze popsat pomocí *openEHR* referenčního modelu lze popsat pomocí archetypu. To znamená pomocí libovolného jazyka, který může stát jako serializace archetypu (ADL, XML, JSON, ...). Samotné archetypy jsou obvykle uloženy ve vlastním úložišti mimo samotné programové řešení a běžně se používají především archetypy ze známých online úložišť archetypů [8].

Existují však specializovaná odvětví pro něž veřejně dostupné archetypy neexistují nebo nesplňují kvalitativní požadavky, které na ně uživatel klade. Příkladem budiž doména elektroencefalografie, pro kterou neexistují ve veřejných repozitářích archetypy, které by vyhovovaly potřebám neuroinformatické skupiny na KIV.

5.3.1 Archetypy uvnitř EHR

Do výsledného programu se archetypy zpravidla nenačítají samostatně, ale obalené šablonou, která definuje určitou množinu archetypů. Ty pak představují obecný typ vyšetření. Například vyšetření krevního oběhu může být složeno z krevního tlaku, tepu a plicního arteriálního tlaku. Nebo se může jednat o výsledky krevního obrazu, který obsahuje mnoho archetypů jako krevní plyny, destičky, enzymy atp.

5.3.2 Reprezentace archetypů

Archetypy v *openEHR* jsou formalizovány v Archetype Object Modelu (AOM). Jedná se o model sémantiky archetypů - pokud je reprezentace archetypu načtena v paměti, je uložena jako instance třídy právě z tohoto modelu.

Serializace archetypů je možná prakticky v jakémkoli jazyce, ale standardem je Archetype Definition Language (ADL)[8] viz kapitola 6. Pro ADL je garantována 100% bezztrátovost dat, ale existují i jiné serializace. Například XML, HTML, JSON. Podle [8] navíc v budoucnu přibudou další definice pro jiné formáty serializace.

5.4 Templates

Templates (šablony) viz výše, obalují množinu archetypů. Template většinou těsně koresponduje s formulářem na obrazovce, tiskovým reportem, nebo jinou reprezentací dat. Obecně jsou vyvíjeny a používány lokálně, zatímco archetypy jsou široce rozšířené[8].

Jsou to také šablony, které dále definují to, co archetyp nechává na uživateli. Tím jsou například sloty pro archetypy. Archetyp definuje, že na určitém místě se má použít obsah jiného archetypu a může definovat požadavky, jaké na takový archetyp klade – omezení pomocí názvu archetypu ve formě regulárního výrazu. Slot pro archetyp je vidět v ukázce 1. Zde je vidět, že lze použít obsah celého archetypu, jehož jméno odpovídá

jednomu ze tří definovaných regulárních výrazů. Důležitou poznámkou je, že se nekladou omezení na zanoření do sebe, archetyp tedy může obsahovat sám sebe.

Ukázka kódu 1: Slot pro archetypy

```
allow_archetype CLUSTER occurrences matches {0..*} matches{
    include
        archetype_id/value matches {/health_event\.v1/}
        archetype_id/value matches {/symptom-pain\.v1/}
        archetype_id/value matches {/symptom\.v1/}
}
```

5.5 Cesty – paths

Další a především všudypřítomnou částí *openEHR* jsou paths neboli cesty. Jedná se o způsob, jakým se adresují elementy uvnitř archetypu. Tyto cesty patří mezi základní principy, které *openEHR* zavádí a díky nim má každý element svou unikátní adresu nejenom v rámci archetypu, ale také v rámci celé šablony nebo dokonce v rámci celého EHR systému.

Cesty se skládají z posloupnosti jmen atributů a identifikátorů uzlů a používají podobnou syntaxi jako XPath u klasického XML.

Jsou to cesty, které zajišťují možnosti perzistence dat. Jelikož neexistují (a ani nemohou existovat) databázové modely konkrétního archetypu, není možné data ukládat do konkrétních tabulek nebo podobným způsobem pro nerelační databáze. Navzdory tomu, existují schema–less databáze, které modely striktně nevyžadují (alespoň ne v tradičním pojetí). Příkladem mohou být key–value databáze, kde klíčem je obvykle unikátní textový řetězec [10] (v našem případě právě cesta k datovému elementu) a hodnotou je pak libovolná serializace ukládaných dat (String, binární data, serializovaný Java object, atd.).

Z druhého pohledu nám cesty nabízejí v podstatě dotazovací jazyk nad archetypy. Lze s jeho pomocí dotazovat (zacílit) konkrétní elementy pro

potřeby šablony, či v datech vyhledávat hodnoty pro konkrétní elementy bez nutnosti procházet všechny hodnoty.

6 Archetype Definition Language

Archetype Definition Language (ADL)[4] je formální jazyk navržený pro popis archetypů. ADL je jedním více možných typů serializace archetypu, ale prakticky je nejpoužívanějším formátem, a to především proto, že je bezeztrátový a umožňuje uložit všechna data, která umožňuje ukládat referenční model.

ADL se skládá ze tří dalších syntaxí:

- cADL – constraint ADL
- dADL – data definition ADL
- FOPL – first-order predicate logic

na obázku 8 je ukázáno použití jednotlivých syntaxí v konkrétních sekcích.

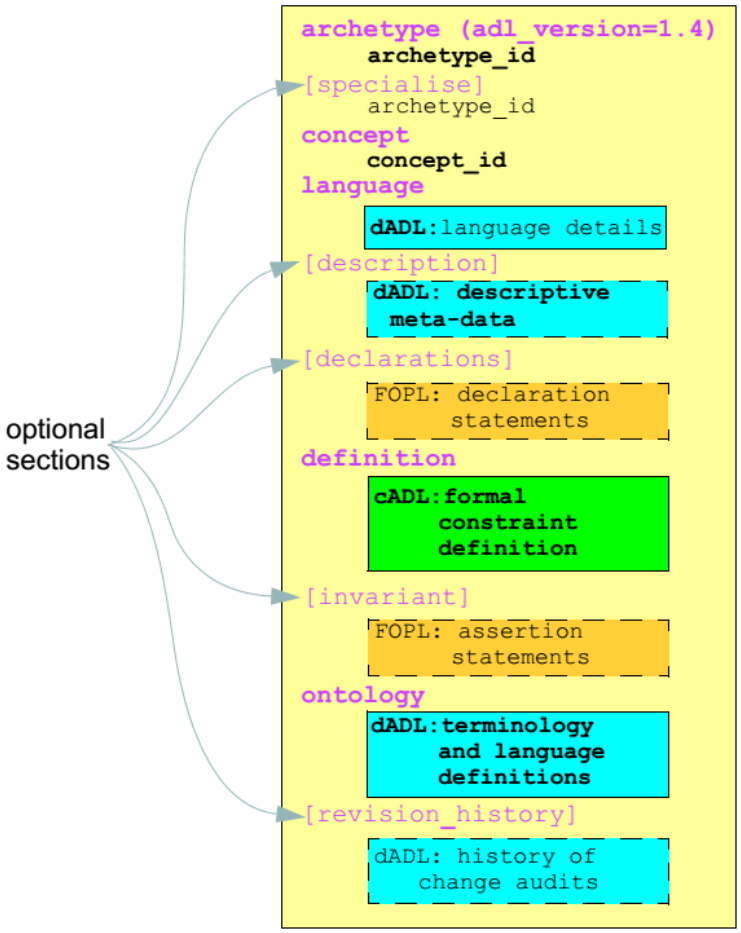
ADL včetně všech jeho syntaxí je detailně popsaný v [11], jedná se o kompletní definici jazyka a nepovažuji proto za nutné popisovat celou syntaxi jazyka. Uvedu zde pouze několik částí, které budou potřeba k pochopení dalších částí textu.

6.1 Termíny a internacionalizace

V rámci archetypů se běžně nepoužívají konkrétní překlady lékařských termínů. Místo toho se používají kódy pro definici termínů. Používá se formát `atxxxx` a jedná se v podstatě o názvy proměnných.

Důvodem je především možnost internacionalizace. Pokud jsou termíny definovány jako proměnné, lze pak snadno definovat překlady pro konkrétní jazyky. Svázání překladu s termínem je vidět na příkladu 2.

[4] V celé práci je za verzi ADL považována verze 1.4.



Obrázek 8: Struktura archetypu z pohledu ADL, převzato z [11]

6.2 Svázání definic a terminologií

Terminologie je soubor definic pro konkrétní doménu, který se používá především pro jasnou kodifikaci daných termínů.

Například pojem krevní tlak je sám o sobě relativně nejasný. Aby si všichni zúčastnění (doménoví specialisté, lékaři, vývojáři archetypů, vývojáři šablon, uživatelé, ...) byli jisti, že mluví o stejném krevním tlaku, přiřadí se k danému elementu v archetypu konkrétní termín z dané terminologie.

Terminologie pak například popisuje, že se jedná o tlak krve na plicní artérii. Pokud by toto svázání nebylo definováno, mohla by být data dezinterpretována, jelikož plicní arteriální tlak má od toho běžného výrazně

posunuté normotenzní hranice. To by mohlo způsobit nesprávnou diagnózu a v důsledku toho i medikaci.

V jazyce ADL se svázání provádí v sekci pro ontologii viz příklad 2. Na příkladu je vidět, že konkrétní termín je svázaný s konkrétním ID v konkrétní terminologii. Tato ID si každá terminologie definuje sama.

Ukázka kódu 2: Termíny a jejich svázání s terminologií

```
ontology
    terminologies_available = <"ODML", ...>
    term_definitions = <
        ["en"] = <
            items = <
                ["at0000"] = <
                    text = <"Environment">
                    description = <"The environment conditions
                        for the experiment.">
                >
                ["at0001"] = <
                    text = <"Environmental conditions">
                    description = <"*">
                >
                ["at0002"] = <
                    text = <"Weather">
                    description = <"*">
                >
                ["at0003"] = <
                    text = <"Description">
                    description = <"*">
                >
            >
        >
    >
    term_bindings = <
        ["ODML"] = <
            items = <
                ["at0000"] = <[ODML::ODMLID001001]>
                ["at0002"] = <[ODML::ODMLID001002]>
```

```
                ["at0003"] = <[ODML::ODMLID001005]>
            >
        >
    >
```

7 Vaadin

Vaadin je webový framework, jehož obliba v posledních letech stoupá, což ukazují i data z vyhledávače Google na obrázku 9, kde je vidět denní počet vyhledávání slova Vaadin mezi roky 2004 a 2015.

Vaadin slouží především k rychlému a efektivnímu programování rozsáhlých Rich Internet Applications (RIA). Jeho obrovskou výhodou je, že se programátor nemusí učit nový jazyk, ale vystačí si se znalostí Javy a povědomím o tom, jak se píší desktopové aplikace ve Swingu[5] nebo AWT. Vaadin totiž funguje pro programátora na stejném principu – tedy vytváření komponent, které jsou následně umísťované do komponentových kontejnerů.

Vaadin znamená ve finštině „požadavek“ a pravě požadavky programátorů a uživatelů reflektuje ve své jednoduchosti a uživatelské přívětivosti výsledných aplikací. Programátor nepotřebuje pro samotné naprogramování výsledné aplikace znalost ani HTML, CSS nebo JavaScriptu a uživatel vidí profesionálně vypadající aplikaci.

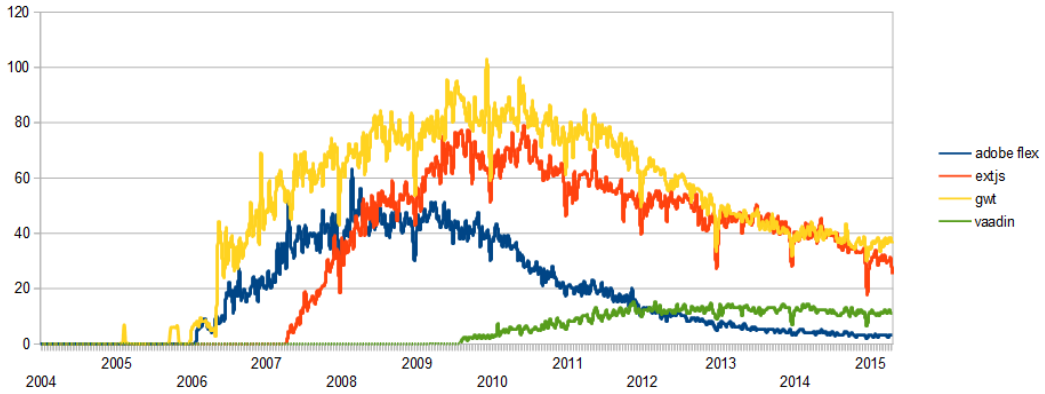
Právě to, že není nutné používat HTML výrazně zvyšuje znovupoužitelnost komponent vzniklých pod Vaadin frameworkem. Nejsou potřeba žádné HTML (Wicket), JSP nebo GSP(Groovy) šablony. Na druhou stranu umožňuje znalost HTML a především CSS výrazně upravovat chování a vzhled výsledných aplikací včetně vytvoření nových vzhledů pro celé aplikace.

Framework dnes ve verzi 7.6 nabízí více než 400 standardních komponent, které je možné využívat rozšiřovat či upravovat jejich funkčnost. Existuje dokonce veřejné úložiště[6], kam mohou vývojáři ukládat své vlastní komponenty a nabídnout je ostatním členům komunity, a to ať už zdarma, či za poplatek spojený s licencí. Vaadin je nabízen pod licencí Apache Licence 2.0. Existují však také placené nadstavby tzv. Vaadin Pro Tools, ty

[5] `http://docs.oracle.com/javase/7/docs/technotes/guides/swing/`

[6] `https://vaadin.com/directory#!browse`

obsahují například podporu pro mobilní aplikace, podporu grafů, komponentu tabulkového editoru a mnoho další užitečných nástrojů. Pro nekomerční projekty je možné požádat o licenci zdarma[7].



Obrázek 9: Vývoj vyhledávání slova Vaadin mezi roky 2004 a 2015 [8]

7.1 Historie

Původní jméno projektu bylo Millstone 3 a vznikl v roce 2005 [12]. Nejprve používal pro komunikaci AJAX, poté přešel na vlastní implementaci pomocí JavaScriptu a nakonec využil Google Web Toolkit (GWT), který Vaadin používá dodnes.

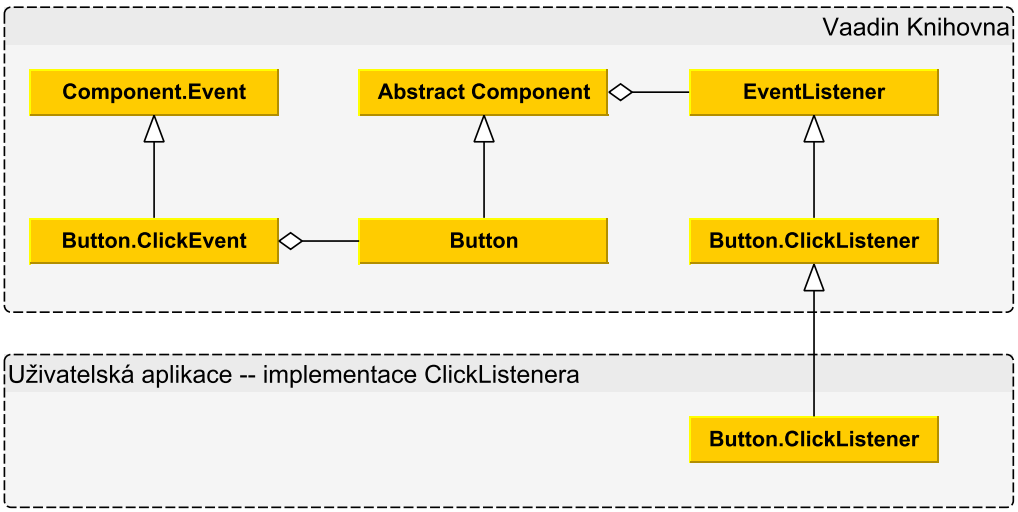
V roce 2009 pak vznikl projekt Vaadin a byl vydán Vaadin framework 6. V této verzi však byly pozorovány problémy s výkonností aplikací a s rychlostí vykreslování stránek. Tyto problémy pak odstraňuje Vaddin 7, který byl vydán v roce 2013 [13]. Vaadin 7 narozdíl od verze 6 výrazně využívá možnosti HTML 5, především co se týče výpočtů pro layout na výsledné stránce. Tím se výrazně zrychlilo původně problematické načítání aplikací.

[7] `https://vaadin.com/blog/-/blogs/non-commercial-licenses-for-pro-tools`
[8] `http://www.google.com/trends/explore#q=adobe++flex,extjs,gwt,vaadin`

7.2 Data, události, modely

Základními stavebními prvky celého frameworku jsou data, datové elementy / komponenty, obsluha událostí a datové modely. V této části bude popsán základní princip obsluhy událostí a dále modely dat pro komponenty, neboť především v těchto částech se liší od konvenčních frameworků a připomíná spíše Swing, jak jej známe z Javy SE. Existují jistě další zajímavé části Vaadin frameworku, ale tyto dvě vybrané se mi jeví jako nejvíce a nejčastěji použité prvky.

7.2.1 Události a jejich obsluha



Obrázek 10: Vaadin Observer - obsluha událostí tlačítka

Co se týče obsluhy událostí, využívá Vaadin dobře známý návrhový vzor Observer [14]. Pomocí tohoto modelu jsou řízeny obsluhy všech běžných událostí v uživatelském rozhraní, ale i servisní záležitosti, jako je třeba řízení sessions. Na obrázku 10 je vidět základní diagram pro obsluhu události „click“ nad tlačítkem. Programátor se zajímá pouze o implementaci jedné třídy a její přidání do seznamu listenerů konkrétního tlačítka, jak je vidět v ukázce kódu 3.

Ukázka kódu 3: Vaadin – přidání posluchače k tlačítku

```
button.addClickListener(new Button.ClickListener() {
 public void buttonClick(ClickEvent event) {
  layout.addComponent(new Label("Thank you for clicking"));
 }
});
```

7.2.2 Datové modely a data

Základní práce s jednotlivými komponentami a jejich datovými modely může být velmi snadná. Do konkrétního modelu (předpokládejme pro názornost ComboBox) se přidají objekty, tak jak jsme je definovali, v comboboxu se objeví jejich `toString()` reprezentace, která se zároveň použije jako unikátní identifikátor daného objektu uvnitř modelu. To však přináší problémy, pokud by dva objekty měly stejný výstup z metody `toString()`.

Proto se ve Vaadinu definovaly kontejnery (Containers). To jsou vlastně kolekce dat v datovém modelu – objektů typu Item. Důležité je, že každý item má svůj unikátní identifikátor, ale není problém, aby se více položek zobrazovalo se stejným jménem.

Z toho důvodu byly zavedeny property. Každý kontejner má definovaný seznam „vlastností“, které jeho itemy mají. Property mohou být libovolného typu (String, Integer, vlastní datový typ) a jsou dále využívány při práci s komponentou. V ukázce kódu 4 je vidět, jak se k datovému modelu přidávají dvě property

- caption typu `String`
- quantityItem vlastního typu `CDvQuantityItem`

a ty jsou pak plněny při každém vložení itemu do kontejneru. Jedním z důležitých prvků je `ItemCaptionMode`, který definuje jaká property bude použitá jako textová reprezentace daného objektu v modelu.

Ukázka kódu 4: Vaadin – přidání posluchače k tlačítku

```
ComboBox units = new ComboBox();
```

```
units.addContainerProperty("caption", String.class, "");
units.addContainerProperty("quantity", CDvQuantityItem.class, "
    ");

units.setItemCaptionPropertyId("caption");

Item item = units.addItem(cDvQuantityItem.getUnits());
item.getItemProperty("caption").setValue(cDvQuantityItem.
    getUnits());
item.getItemProperty("quantityItem").setValue(cDvQuantityItem);
```

Tento poměrně jednoduchý model, kdy kontejner samotný stojí jako tabulka, item jako její řádek a property jako sloupec je často opomíjený a nesprávně se využívá `toString()` metoda. Takový přístup však dříve nebo později přinese nějaké „ALE“.

8 Současná řešení

Pro automatické nebo alespoň poloautomatické generování GUI z archetypů s použitím *openEHR* v současné době existují dvě řešení GastrOS[9] a EHRgen[10]. V této kapitole budou popsána obě řešení s jejich klady a zápory. Předně je důležité říci, že ani jedno z řešení není možné integrovat do EZD.

8.1 GastrOS

GastrOS je podle mého názoru velmi zdařilá implementace EHR systému, ačkoli vznikla pouze jako studie použitelnosti v doméně gastroenterologie.

Řešení je napsáno na platformě .Net v jazyce C#. Jedná se tedy o klasickou desktopovou aplikaci. Aplikace důsledně dodržuje paradigma model–view–controller. Program pracuje s tzv. operational template, což je kombinace archetypu a šablony. Vlastně jsou v něm zahrnuta všechna data, která jsou pro generování GUI potřebná.

Z mého pohledu aplikace velmi zajímavě řeší celý systém a přináší velké množství GUI directives [1]. Tyto direktivy řídí zobrazení GUI a mohou stejný datový element zobrazit různým způsobem. GUI s nástinem možností gastrOS je vidět na obrázku 11. Zeleně označená data jsou direktivy.

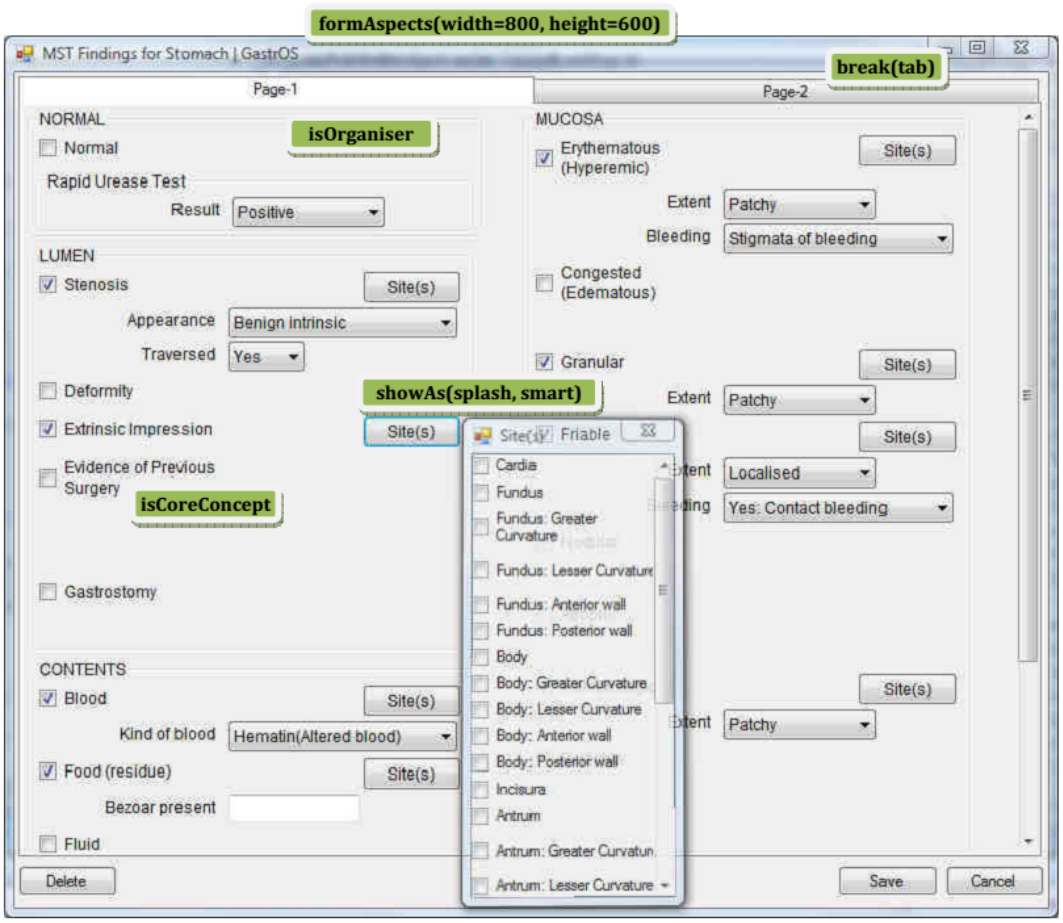
Nevýhodou celého systému je podle mého názoru to, že se nejedná o webovou aplikaci.

8.2 EHRgen

Aplikace napsaná v jazyce Groovy. Zároveň používá graficky mírně nepřehledné řešení klasického webového formuláře rozděleného do dvou sloupců. Při používání mě překvapilo poměrně velké množství výjimek, kdy občas bylo nutné celou aplikaci restartovat.

[9] `https://gastros.codeplex.com`

[10] `https://openehr.atlassian.net/wiki/display/projects/Open+EHR-Gen+Framework`



Obrázek 11: GastrOS GUI s direktivami, převzato z [1]

Program v tomto případě pracuje s vlastní implementací šablon, která nemá tak bohaté direktivy, jako gastrOS, ale umožňuje definovat v jakém sloupci se daný element nebo archetyp zobrazí.

Nevýhodou je bohužel opět jazyk. Groovy je spíše prototypovací jazyk a nejeví se jako příliš vhodný pro rozsáhlé aplikace. Programové řešení mi přišlo poměrně komplikované a především je popsáno pouze ve španělštině. Španělština je také jazyk všech oficiálních dokumentací nebo dokumentů, které jsem k aplikaci našel. Další nevýhodou je nutnost používat jsp templaty pro elementy.

9 Implementace

9.1 Archetypy pro elektrofyziologii

Vzhledem k tomu, že ve standardních úložištích neexistují archetypy pro elektroencefalografii, vznikly potřebné archetypy na KIV v rámci výzkumu neuroinformatické skupiny[3]. Částečně byly využity i existující archetypy z *openEHR*. Podle [3] jsou to Device, Device details a Environmental Conditions.

Vývoj archetypů byl postaven na existujícím popisu terminologie odML[11]. Byly vybrány terminologie, které jsou v rámci elektroencefalografie využitelné a byly převedeny do ADL. Jedná se o:

- Experiment (OBSERVATION),
- Stimulus (CLUSTER),
- Software (CLUSTER),
- Scenario (CLUSTER),
- Electrode (CLUSTER),
- Environment (CLUSTER).

Všechny archetypy jsou ve stádiu konceptu a čekají na formální schválení celým týmem neuroinformatické skupiny[3].

To, že jsou archetypy svázány s odML terminologií (kapitola 4.2), je nutné zavést do samotných archetypů. odML není terminologie v klasickém smyslu a nepoužívá vlastní ID pro konkrétní termíny. Proto bylo nutné vytvořit pro každý jednotlivý element unikátní identifikátor, který se použije jako ID do terminologie a sváže s konkrétními termíny v archetypu.

Původně navržený systém je následující

- ID má tvar ODMLIDXXXXX, kde XXXXX je celé číslo (rozsah 1 – 99999 pro jednu terminologii),

[11] `http://www.g-node.org/projects/odml/`

- hodnota je složením jména terminologie a XPath k `<definition>` elementu uvnitř XML reprezentace terminologie.

Takové řešení se zdá být vhodné, nicméně není snadno automatizovatelné a především zavádí nutnost vést další soubor s mapováním ID na hodnoty. Další nevýhodou by bylo to, že při změně v jedné terminologii by bylo nutné znovu vygenerovat všechny identifikátory pro celý soubor terminologií, a ty pak znovu zavést do mapování v archetypech.

Proto byl navržen následující jednoduchý systém:

- ID má tvar REPO#TYPE#NAME#XPATH

V rámci této práce jsem vytvořil automatický skript pro generování seznamů těchto identifikátorů jednotlivých termínů.

Jedná se o XSL transformaci, která vypíše všechny definované termíny v předané terminologii ve formátu popsaném výše.

Ukázka kódu 5: XSL transformace terminologie na seznam XPaths

```
<xsl:template match="/">
    <xsl:for-each select="//definition">
        <xsl:value-of select="//repository"/>
        <xsl:text>&#35;</xsl:text>
        <xsl:value-of select="//type"/>
        <xsl:text>&#35;</xsl:text>
        <xsl:value-of select="//name"/>
        <xsl:for-each select="ancestor::*">
            <xsl:text>/</xsl:text>
            <xsl:value-of select="name()"/>
            <xsl:if test="(preceding-sibling::*|following-
                sibling::*)[name()=name(current())]">
                <xsl:text>[</xsl:text>
                <xsl:value-of select="count(preceding-
                    sibling::*[name()=name(current())]) + 1
                    "/>
                <xsl:text>]</xsl:text>
            </xsl:if>
        </xsl:for-each>
```

```
            <xsl:text>/definition</xsl:text>
            <xsl:if test="position()!=last()">
                <xsl:text>&#10;</xsl:text>
            </xsl:if>
        </xsl:for-each>
    </xsl:template>
```

Výhodou je, že ID jednoznačně identifikuje termín v odML terminologii bez nutnosti existence mapovaní. Zároveň, pokud by to bylo nutné, není problém skript jednoduše upravit, aby generoval zároveň i ID ve formátu:

- ODMLIDYYY/XXXXX, kde XXXXX je celé číslo (rozsah 1 – 99999 pro jeden archetyp) a YYY je mapování čísla na konkrétní terminologii,
- hodnota pak může zůstat generovaná stejnou XSL transformací.

Pak by bylo možné zavést mapování, které by vycházelo přímo z této transformace.

9.2 Templates

Součástí dvouvrstvého modelování jsou rovněž šablony. Pro jejich návrh jsem využil klasické XML – (Schéma v příloze B).

Základní funkcí template je sdružovat archetypy nebo jejich částí do větších celků, které tvoří výsledný formulář. V této kapitole budou popsány mechanismy, které současná implementace šablony dovoluje a zároveň nastíněné další směry, jakými by se mohl vývoj ubírat.

9.2.1 Základní elementy

Jazyk pro popis šablon je velice jednoduchý. Kromě elementů pro archetypy se v něm vyskytují pouze elementy<name> a <uniqueID>.

Jméno slouží především pro základní zařazení šablony a tvoří popisek okna, které obsahuje formulář pro daný template.

Unikátní ID pak slouží pro další zpracování uvnitř aplikace. Na prezentační vrstvě by mohl sloužit k identifikaci okna pro potřeby cache již vytvořených formulářů pro danou šablonu. Na vrstvě databázové ho pak bude možné použít jako identifikátor souboru dat.

Nejdůležitějším elementem je však <archetype>. Každý <archetype> element obsahuje cestu k jednomu konkrétnímu archetypu. Cesta je udávána relativně vůči kořenovému adresáři archetypů, definovanému v konfiguračním souboru (viz kapitola 9.4) pomocí atributu `file`. Druhým důležitým atributem tohoto elementu je `useAll`. Tento boolean atribute definuje, zda se z konkrétního archetypu použijí všechna data nebo bude dále specifikován seznam uzlů daného archetypu, které budou ve formuláři figurovat.

Ukázka kódu 6: Příklad <archetype> elementu.

```
<archetype file="/ehr/cluster/openEHR-EHR-CLUSTER.a2.v1.adl"
    useAll="true"></archetype>
```

9.2.1.1 Využití části archetypu

Pokud je potřeba využít nějakou specifickou část (či více částí) archetypu, lze v šabloně definovat seznam použitých uzlů.

Jako příklad může posloužit archetyp pro krevní tlak. V jeho obecnosti poskytuje jak obecné měření krevního tlaku, tak jeho průměrnou hodnotu. Součástí obvyklého vyšetření však bude pravděpodobně pouze jednoduché neopakované měření krevního tlaku. Průměry mohou být použity například při komplexnějších vyšetřeních jako je Holterovská monitorace.

V těchto případech se pomocí atributu `useAll` nastaveného na false a definováním části archetypu do templatu přidá pouze požadovaná hodnota pro měření. Použití je znázorněno v ukázce 7.

Ukázka kódu 7: Definice použitých částí archetypu – příklad využití pouze průměrné hodnoty krevního tlaku

```
<archetype file="openEHR-EHR-OBSERVATION.blood_pressure.v1.adl"
     useAll="false">
    <includes xPath="/data[at0001]/events[at1042]"/>
```

```
</archetype>
```

9.2.1.2 Další možný vývoj šablon

V budoucnu by bylo možné do šablon přidat další řídící prvky. Inspirace ve směru řízení GUI by se dala brát v projektu GastrOS (kapitola 8.1). Bylo by možné například zavést záložkové zobrazení (TabView) a v v šabloně řídit, jakým způsobem by se jednotlivé archetypy mapovaly na dané záložky.

Dalším možným rozšířením by mohlo být exclude některých elementů z archetypu. V současném stavu by se takový požadavek řešil vypsáním všech elementů, které budou použité, ale bylo by možné definovat seznam elementů, které budou z archetypu odebrané a tím vývoj šablon zjednodušit.

9.2.1.3 Obsazení slotu

V kapitole 5.4 jsou popsány sloty a jejich definice v archetypu. Dosazení do konkrétního archetypu je pak záležitostí šablony a příklad je vidět v ukázce 8.

Ukázka kódu 8: Dosazení archetypu do slotu

```
<archetype file="openEHR-EHR.health_event.v1.adl" useAll="true">
    <slot path="/items[at0015]/items[at0008]/items" >
        <archetype file="openEHR-EHR.a2.v1.adl" useAll="true">
    </slot>
</archetype>
```

Element <slot> má povinný atribut path. Definuje konkrétní místo, do kterého se má daný archetyp umístit. Sloty mohou vyžadovat specifický archetyp a jeho jméno definovat přes regulární výraz. V tom případě je před umístěním prováděna kontrola, zda je zde uvedený archetyp skutečně povolen na tomto místě. Tuto kontrolu lze vypnout pomocí aplikační konfigurace viz kapitola 9.4.

9.3 Výběr technologie pro implementaci aplikace

Při rozhodování, jaký použít framework, přicházeli v úvahu hlavně dva kandidáti. Wicket, který je použitý na několika dalších projektech v rámci katedry a Vaadin (viz kapitola 7).

Oba frameworky jsou hojně používány napříč programátorským světem a poskytují prakticky stejný prostor pro vytvoření libovolné webové aplikace.

Co se týče dokumentace, je Vaadin jedním z nejlépe dokumentovaných frameworků, které znám. Základní materiál Book of Vaadin [13] je velice obsáhlý, psaný srozumitelným jazykem a plný jasných a výstižných příkladů. Kromě online edice existuje také papírová forma. Obrovskou výhodou jsou online dostupné demo aplikace, které poskytují náhled do toho, jak mohou aplikace psané ve Vaadinu vypadat a ukazují zároveň kompletní zdrojové kódy[12].

Wicket naproti tomu obsahuje z mého pohledu méně kvalitní dokumentační materiály, které nejsou zcela přehledné a příklady nejsou dostatečně výstižné.

Hlavním hlediskem však byla použitelnost na konkrétním projektu. Pro Vaadin zde mluví především to, že se celý obsah výsledné aplikace skládá v samotném kódu Javy. Nejsou potřeba žádné šablony ani zásahy do HTML. Celá aplikace je dynamicky tvořena za běhu aplikace.

Wicket pak nutí programátora používat (X)HTML šablony a tím pádem minimálně znepříjemňuje tvorbu automaticky generovaných formulářů. Do modelu by vstoupily další šablony, které by bylo potřeba udržovat a to pro každou šablonu zvlášť. Z automatického generování by se tak stalo poloautomatické.

Největší devizou výsledné aplikace by mělo být to, že bude kompletně řízená archetypy a šablonami. Především z tohoto důvodu byl zvolen Vaadin jako hlavní technologie.

[12] `https://vaadin.com/demo`

9.4 Konfigurace

Aby byla aplikace co nejvíce znovupoužitelná a obecná, byl vytvořen systém externí konfigurace. Konfigurační soubor se defaultně nachází v `c:/Resources/config.properties`[13]. Aplikace, která modul využívá si může před jeho použitím nastavit cestu ke svému vlastnímu konfiguračnímu souboru. Aplikace je však zodpovědná za to, že soubor existuje buď v defaultním umístění, nebo že modulu předala validní cestu k jinému property souboru.

Konfigurační soubor obsahuje základní nastavení modulu a některá další volitelná nastavení viz příloha D. Povinné jsou pouze property:

- `archetypeFolder` – cesta ke složce archetypů
- `templateFolder` – cesta ke složce šablon
- `uploadFolder` – složka, kam budou nahrávány binární soubory z formulářů.

Další property se týká validace archetypu vloženého do slotu. Tu lze pomocí `validateSlotsArchetypes` property vypnout a spolehnout se tak na to, že byl do slotu vložen správný archetyp.

Ostatní hodnoty slouží k internacionalizaci aplikace. Jak již bylo řečeno v kapitole 6.1, archetypy jsou vytvářeny tak, aby byla zajištěna jazyková nezávislost. Aplikace však musí vědět, jaký jazyk má pro konkrétní archetyp zvolit.

V aplikaci se proto definuje pomocí property `defaultLanguage` defaultní jazyk. V případě, že je archetyp přeložen do takto definovaného jazyka, využijí se tyto překlady. Pokud ne, zvolí se překlad, který je v archetypu definován jako primární.

Aplikace však používá i jiné texty, které je možné internacionalizovat. Jedná se například o popisky tlačítek nebo hlášky validátorů. I pro tyto

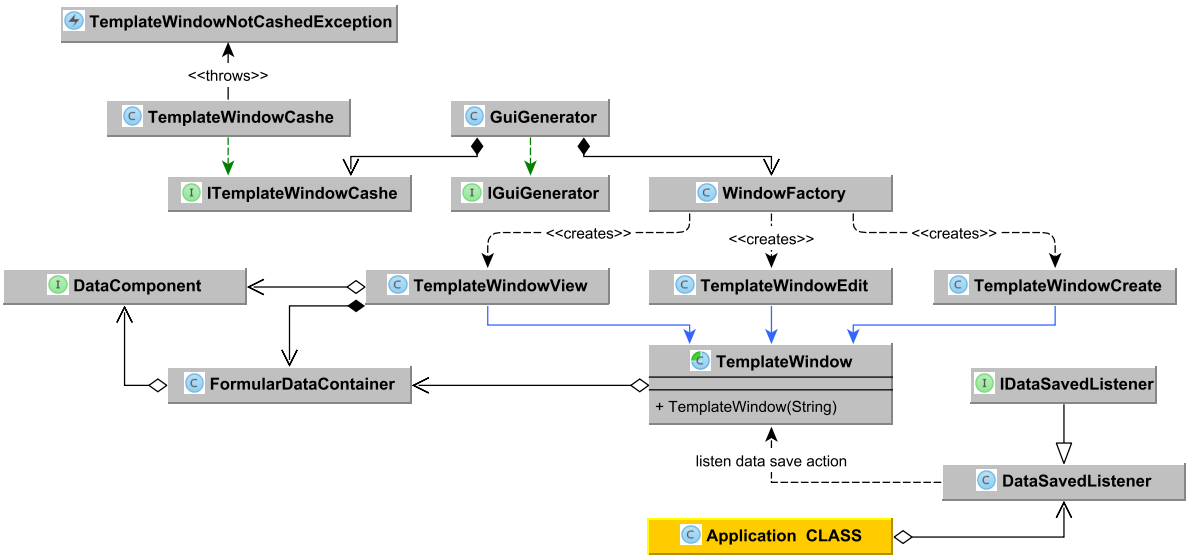
[13]Konfigurační soubor musí být uložen v ANSI kódování, pokud překlady obsahují české znaky.

texty lze definovat vlastní překlady. Jednoduše se přidá překlad do konfiguračního souboru spolu s postfixem definujícím konkrétní jazyk a tento překlad aplikace využije.[14]

Poslední důležitou property je `theme`. Definuje Vaadin styl pro celou aplikaci – viz 9.7.5. Použitelné hodnoty jsou například `valo`, `runo`, `chameleon` nebo `reindeer`.

9.5 Základní struktura a použití modulu

Aplikace je vyvíjena podle konvencí objektově orientovaného programování. Na obrázku 12 je vidět základní struktura jádra programu.



Obrázek 12: Jádro aplikace

Jedním z důležitých cílů této práce bylo vytvořit aplikaci ve formě modulu. To vyžaduje nemalé úsilí již při návrhu samotné kostry aplikace. Jak je vidět na obrázku 12, zvolil jsem pro vytváření samotných jednoduchou Factory třídu, která na základě požadavku z generátoru vytvoří instanci požadovaného typu. Všechny 4 typy formulářů (na obrázku není vidět typ `TemplateWindowSimpleView` viz kapitola 9.7.1) dědí od abstraktní

[14] U této funkce se jedná o proof–of–concept. Ne všechny texty, které v aplikaci jsou mají definované překlady.

třídy `TemplateWindow`, která zprostředkovává všechny společné činnosti, ale zároveň ponechává předkům volnost v definovaný samotného obsahu formuláře.

Způsob získávání dat z formulářů je také nastaven tak, aby byla aplikace co nejvíce znovupoužitelná a ruku v ruce s tím co nejméně závislá na svém okolí. Formuláře, které manipulují s daty (create, edit) po stisku příslušného tlačítka vyvolávají `dataSaved` event, který obsahuje všechna data, která daný formulář obsahoval. Tyto eventy jsou pak zpracovány v samotné aplikaci (obvykle mimo samotný modul).

9.6 Integrace a testovací Vaadin aplikace

Jako proof–of–concept možnosti integrace EHR modulu byla vytvořena aplikace, jenž se i se svými zdrojovými kódy nachází na přiloženém CD. Oba projekty (modul a testovací aplikace) jsou *Maven*[15] projekty. Modul se tak do aplikace přidává jako její závislost v projektovém modelu (`pom.xml`)[16] .

Ukázka kódu 9: Maven dependency – EHR modul

```
<dependency>
    <groupId>cz.zcu.fav.kiv</groupId>
    <artifactId>EHR-GuiGen</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

Aplikace po spuštění vezme všechny šablony ze složky definované v konfiguraci a vytvoří pro ně tlačítka, jejichž stisknutí vyvolá vygenerování formuláře typu CREATE[17].

Žádost o vygenerování nového okna se zasílá `singleton` objektu `GuiGenerator`, viz ukázka kódu 10.

[15] `https://maven.apache.org`

[16] Modul musí být samozřejmě nainstalovaný v lokálním repozitáři. K tomu se využije *Maven* volání `mvn clean install`.

[17] Šablony lze přidávat i za běhu aplikace. Po přidání stačí obnovit stránku.

Ukázka kódu 10: Žádost o vygenerování nového okna

```
GuiGenerator
    .getInstance()
        .createWindow(file.getAbsolutePath(), GuiType.VIEW,
            data);
```

Zároveň je potřeba aby se aplikace přidala jako posluchač `dataSaved()` události, viz ukázka 11.

Ukázka kódu 11: Zaregistrování dataSaved listeneru.

```
window.addDataSavedListener(new DataSavedListener() {
      @Override
      public void dataSaved(FormDataContainer data) {...}
```

Testovací aplikace po stisku tlačítka save data uloží a vygeneruje editační a view formulář. Tím představí všechny základní možnosti modulu.

9.7 GUI

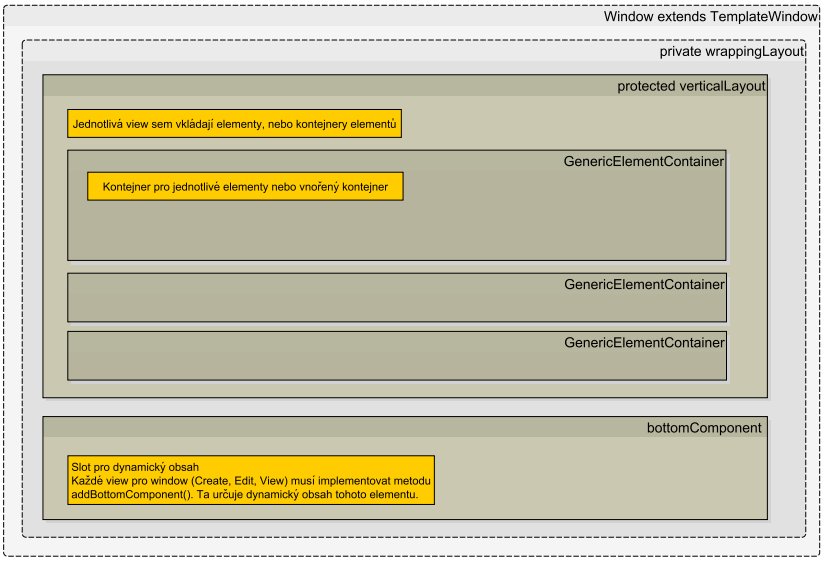
Generované formuláře musejí být i ve své komplexnosti co nejvíce přehledné, snadno použitelné a především jednoduše uživatelsky přívětivé. Navržená podoba obecného formuláře je zobrazena na obrázku 13 a konkrétní podoba jednoho z formulářů na obrázku 14.

V následujících kapitolách budou popsány typy generovaných formulářů, komponenty, které slouží ke zobrazování některých zajímavých elementů s jednoduchým popisem jejich funkcionality a implementace spolu s popisem procesu, kterým jsou formuláře generované.

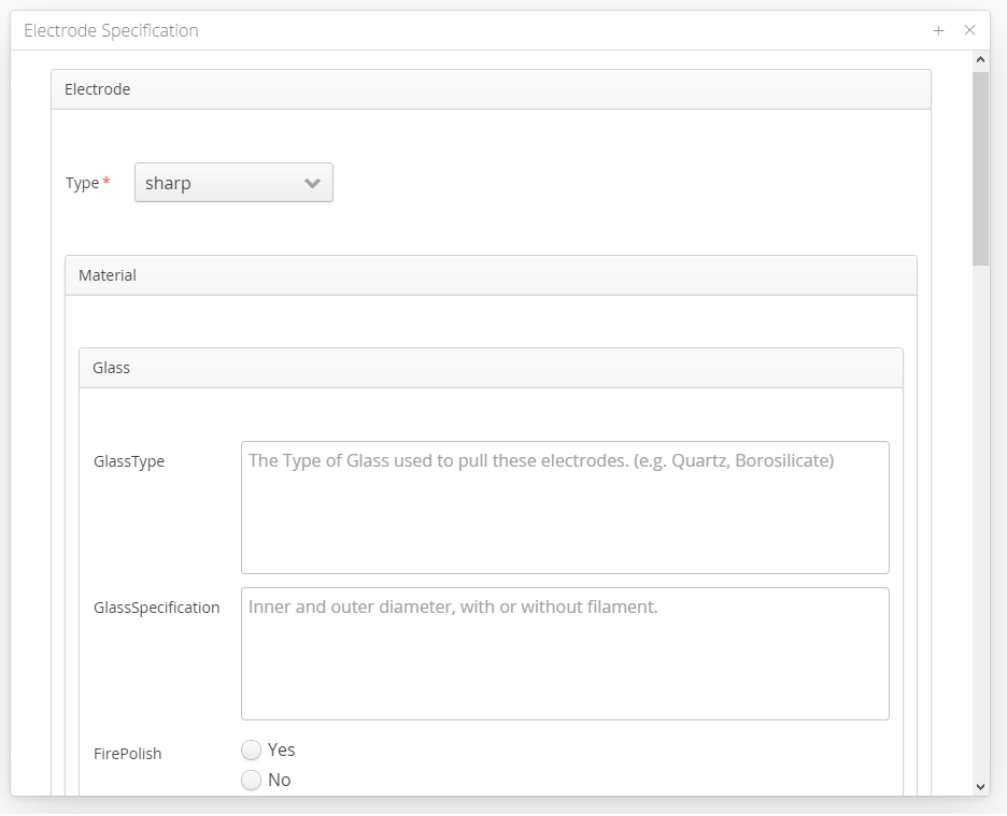
9.7.1 Typy generovaných formulářů

Pro uspokojení všech základních požadavků na obecný GUI modul jsem navrhl 4 typy formulářů, které je schopen generátor vytvořit.
Jedná se o:

- `TemplateWindowCreate` – základní formulář pro zadávání nových dat,



Obrázek 13: Obecná struktura formulářového okna



Obrázek 14: Konkrétní příklad formulářového okna

- `TemplateWindowEdit` – editace již vytvořených a uložených dat,

- `TemplateWindowView` – zobrazení uloženého formuláře bez možnosti úprav,
- `TemplateWindowSimpleView` – zobrazení dat v jejich nejprostší formě, kdy datová část je textovou reprezentací objektu z referenčního modelu. Tento formulář slouží pouze jako proof–of–concept. Ukazuje, jaká data a v jaké formě je možné získat ze standardních datových modelů pro dané komponenty.

Tyto 4 formuláře pokrývají všechny požadavky, které jsou kladeny na modul tak, aby mohl plnit svou funkci. Zde již je vidět, že daný modul není využitelný jen v jedné aplikaci. Jedná se o znovupoužitelný modul, který by měl být bez potíží implementovatelný do libovolné Vaadin aplikace, ale rovněž do aplikací jiných frameworků.

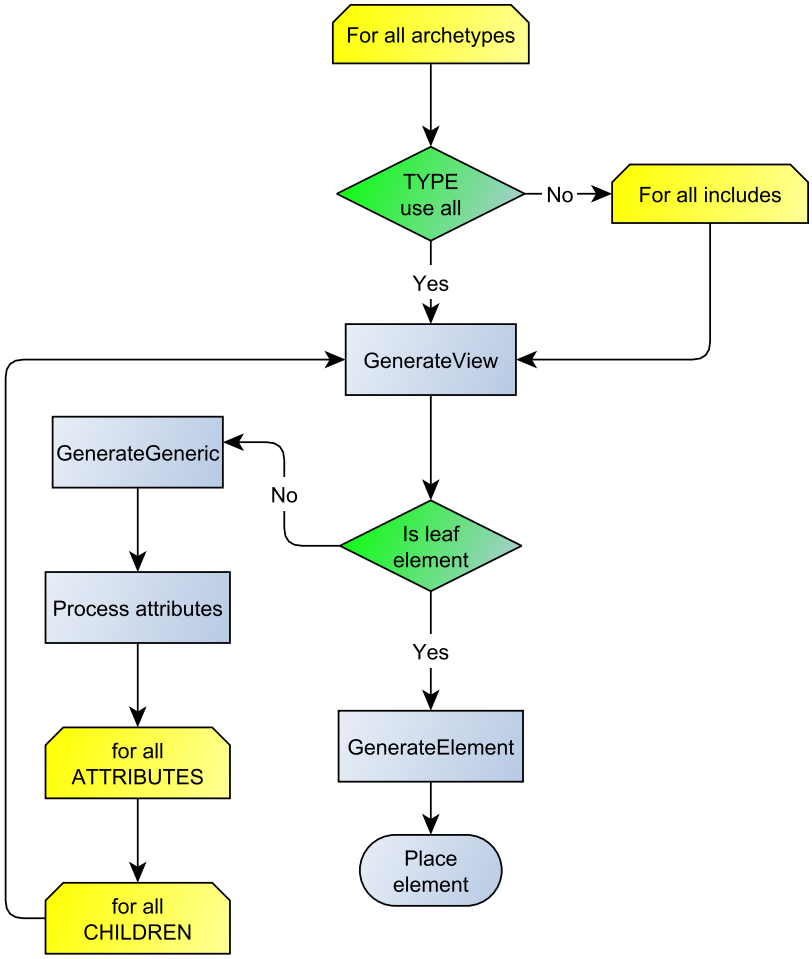
9.7.2 Proces generování GUI

Jelikož je ADL rekurzivní jazyk[11], je potřeba rekurzivně zpracovat archetypy při procesu generování. Flow diagram na obrázku 15 popisuje systém procházení archetypu a generování jednotlivých prvků rozhraní.

Odpovědnost za samotné generování je účelně rozdělena mezi základní třídu `TemplateWindow` a její potomky. Rodičovská třída je zodpovědná za celkové řízení procesu generování, rekurzivní procházení archetypu nebo vytváření kontejnerů pro elementy a jejich umísťování do layoutu.

Potomci jsou zodpovědní za vytváření samotných elementů, párování s daty, vytváření stromu komponent jejich reprezentaci a udržování kontejneru elementů. Na obrázku 15 jsou aktivity potomků zobrazeny v oranžové barvě.

Toto oddělení umožňuje pro každý formulář definovat jiný layout, používat pro stejné elementy rozdílné komponenty nebo definovat vlastní panely, které budou automaticky umístěné na konci formuláře.



Obrázek 15: Flow diagram – generování formulářů

9.7.2.1 Implementace vlastních formulářů

Každý formulář je potomkem abstraktní třídy `TemplateWindow` a musí implementovat tři abstraktní metody:

- `Component addBottomComponent()` – vrací komponentu, která je umístěna na konci každého formuláře, typicky tlačítko na uložení v create formuláři,
- `void generateELEMENT()` – zpracování elementů a vytváření příslušných komponent, případné navázání předaných dat (view a edit formuláře).

- `void addElement()` – způsob přidávání elementů do layoutu formuláře, možnost vlastní definice typu kontejneru (panel, accordion, tab view, atd.).

Každý formulář si zároveň udržuje objekt typu `FormDataContainer` a je zodpovědný za jeho obsah. V tomto objektu uchovává kolekci všech definovaných komponent (reprezentací elementů). Jedná se o mapu, kde klíčem je cesta k elementu a hodnotou komponenta, která daný element představuje. Z tohoto objektu lze získat kolekci samotných dat opět reprezentovaných key–value mapou, kde klíčem je znovu cesta k původnímu elementu, ale hodnotou je již objekt z referenčního modelu.

Data z `FormDataContainer` objektu jsou dále ve formulářích, která data zobrazují (view, simpleViev a edit) navázána na dané elementy.

9.7.3 Typy elementů a jejich GUI reprezentace

Pro každý typ elementu z *openEHR* existuje komponenta, která umožňuje snadné zadání požadované hodnoty, základní validaci a jednoduché načítání a ukládání vložených hodnot.

Všechny komponenty implementují rozhraní `DataComponent` a všechny umožňují zakázat vstup hodnot, čehož se využívá u formuláře typu view. Společnou vlastností je také schopnost zobrazit negativní výsledek validace nebo skutečnost, že daný element je v rámci formuláře povinný.[18]

Pro snadnější orientaci ve formuláři se při najetí na jednotlivé části okna ukazují informační bubliny. Pokud kurzor ukazuje na komponentu reprezentující element, je zobrazen popis z ontologie pro daný element. Pokud kurzor ukazuje do komponentového kontejneru, zobrazí se název daného kontejneru. Pro větší formuláře je toto pro orientaci v aplikaci nezbytné.

[18] V kapitole jsou používány názvy komponent z Vaadin framevorku. Bližší informace o jednotlivých komponentách v [13].

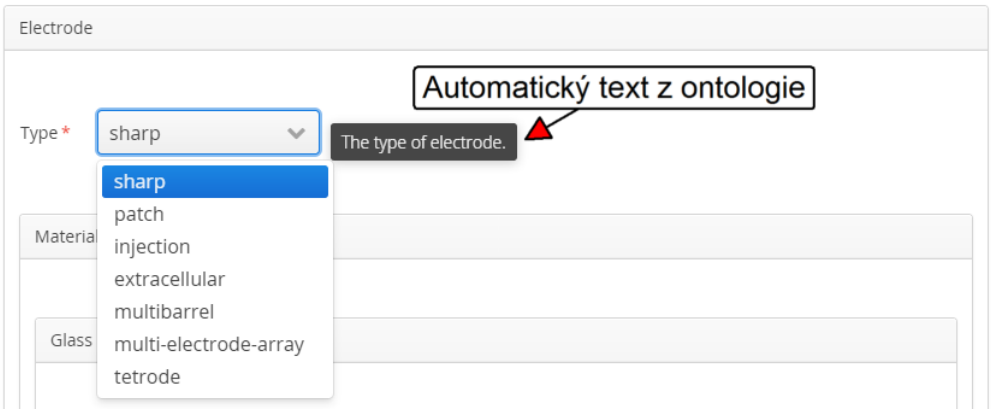
9.7.3.1 Boolean

Pro typ DV_BOOLEAN představuje komponenta jednoduchý dvouhodnotový OptionGroup viz obrázek 16.

Obrázek 16: Komponenta pro typ DV_BOOLEAN

9.7.3.2 Coded text

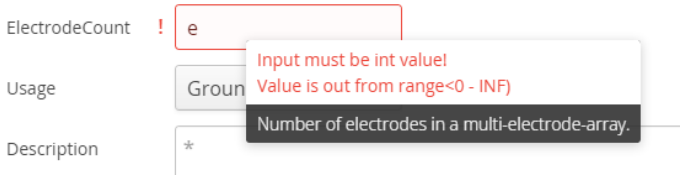
DV_CODED_TEXT je zobrazený jako ComboBox. Na obrázku 17 je vidět informační bublina při najetí myší na element Type.



Obrázek 17: Komponenta pro typ DV_CODED_TEXT

9.7.3.3 Count

DV_COUNT slouží k zadávání celých čísel. Archetyp můžu definovat interval pro validní hodnotu. Komponenta kontroluje zadání správného typu a zároveň interval, do kterého se musí hodnota vejít. Pokud hodnota neodpovídá požadavkům, zobrazí se u ní červený vykřičník a červený rámeček. Po najetí na komponentu se zobrazí informační bublina s informací, které validace byly neúspěšné - viz obrázek 18, kde zadaná hodnota evidentně nesplní ani jedno z daných kritérií.

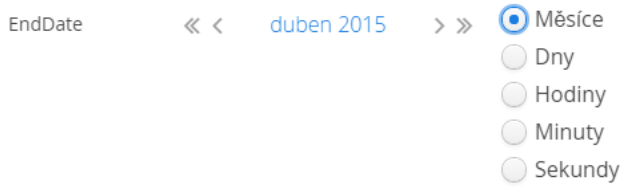


Obrázek 18: Komponenta pro typ DV_COUNT

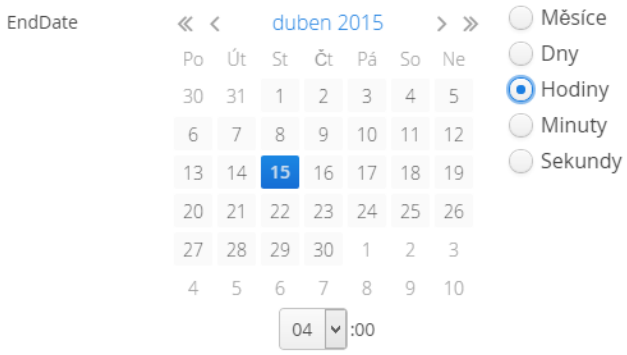
9.7.3.4 Date and time

Komponenty pro zadávání času jsou si velmi podobné, ať se jedná o DV_DATE_TIME, DV_DATE nebo DV_TIME, proto bude popsána jenom komponenta pro DV_DATE_TIME element.

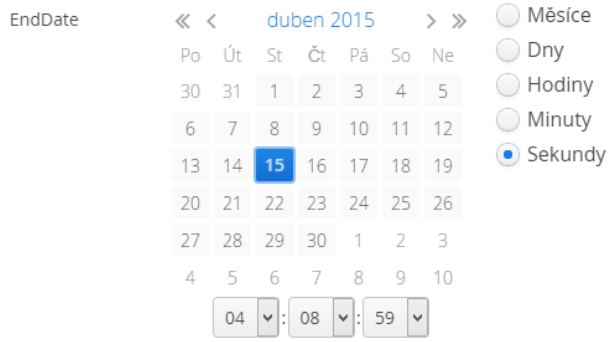
Komponentu tvoří obvyklý kalendář spolu s OptionGroup komponentou. OptionGroup komponenta slouží pro nastavení rozlišení zadávaného času v případě, že archetyp ponechává na uživateli zvolení tohoto rozlišení. Komponenta se pak při zvolení daného rozlišení překreslí tak, aby bylo možné zadat datum ve zvolené přesnosti. Na obrázcích 19, 20 a 21 je vidět stejná komponenta s různě zvolenými rozlišeními.



Obrázek 19: Komponenta pro typ DV_DATE_TIME – rozlišení měsíců



Obrázek 20: Komponenta pro typ DV_DATE_TIME – rozlišení hodin



Obrázek 21: Komponenta pro typ DV_DATE_TIME – rozlišení sekund

9.7.3.5 Quantity

DV_QUANTITIY element je navržen pro zadávání obecných číselných hodnot. Celých i desetinných čísel. Stejně jako DV_COUNT je možné omezit množinu vstupních hodnot pomocí intervalu, ale u tohoto datového typu se přidává možnost definovat jednotky, ve kterých bude hodnota zadána. To samotnou komponentu komplikuje a zobrazení komponenty musí být rozděleno na dva základní případy:

- je definována jedna nebo žádná jednotka
- nebo je definováno jednotek více[19].

V prvním případě tvoří komponentu InputField. V případě, že element má definovánu alespoň jednu jednotku, zobrazí se tato jednotka jako nápověda pro field. Pokud byl definován interval, složí se nápověda z obou hodnot viz obrázek 22.

Obrázek 22: Komponenta pro typ DV_QUANTITY – jedna jednotka

V druhém případě, kdy je možnost volit mezi jednotkami, ve kterých bude hodnota zadána, je k InputFieldu přidán ComboBox, s výčtem jednotek. Nápovědou do InputFieldu je pouze interval, pokud byl pro konkrétní jednotku definován.

[19]Například krevní tlak může být měřen v mm[HG] nebo N/m^2.

Intervaly zadávaných hodnot samozřejmě mohou záviset na zvolené jednotce[20]. Na obrázcích 23 a 24 je zobrazen stejný element, který dovoluje zadávat hodnoty ve více jednotkách. Obrázky ukazují, jak se změní nápověda v InputFieldu pokud se změní jednotka.

Obrázek 23: Komponenta pro typ DV_QUANTITY – více jednotek

Obrázek 24: Komponenta pro typ DV_QUANTITY – změna intervalu

9.7.3.6 Multimedia

Vkládání souborů se provádí přes element DV_MULTIMEDIA, jehož reprezentací je komponenta FilePicker. Tu jsem upravil tak, aby byly vizuálně znázorněné výsledky uploadu. Na obrázku 25 je vidět komponenta po úspěšném uploadu souboru.

Ke standardní komponentě pro výběr souboru je přidán Label, který po úspěšném uploadu zobrazí v zeleném rámečku jméno souboru a jeho cestu tak, jak bude reprezentován na serveru. Tento label zároveň slouží pro download uploadovaného dokumentu, čehož se využívá hlavně u view a edit oken, pro kontrolu uploadovaného souboru.

Obrázek 25: Komponenta pro typ DV_MULTIMEDIA – úspěšný upload

Obrázky jsou na serveru ukládány ve formátu `fileName` + časová známka. To proto, aby se mohly do příslušné složky nahrávat soubory

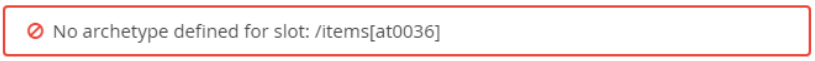
[20]Například například teplota ve stupních Celsia a Fahrenheita budou mít jiné intervaly, kdy je hodnota validní.

se stejným jménem. Při stažení souboru se časová značka ořízne a soubor je uložen s původním jménem.

Jedná se o jednu z nejkomplexnějších komponent modulu, protože zajišťuje upload (implementace `Upload.Receiver`, `Upload.Receiver`, `Upload.SucceededListener`, `Upload.StartedListener` a `Upload.FailedListener`) a zároveň i download (FileDownloader) souborů.

9.7.3.7 Sloty archetypů

Pokud není obsazen slot v archetypu nebo je do něj vkládán nevalidní archetyp, zobrazuje se chybový Label s příslušnou hláškou. Na obrázku 26 je vidět příklad pro neobsazený slot.



Obrázek 26: Label pro neobsazený slot v archetypu

9.7.3.8 Ostatní komponenty

Ostatní komponenty jsou buď variacemi předchozích nebo jednoduché textové elementy bez zajímavé implementace nebo použití a nebudou zde uváděny.

9.7.4 Validace vstupních dat

Komponenty jsou schopné (viz předchozí kapitola) používat validátory a zobrazovat výsledky validace. Validátor je potomkem třídy `AbstractValidator<T>`, kde T představuje datový typ, který se ve validátoru zpracovává.

Každý validátor přepisuje abstraktní metodu `isValidValue`, která je zodpovědná za samotnou validaci předané hodnoty.

Validátory jsou konstruovány tak, aby co nejvíce využívaly poskytované možnosti *openEHR*. Příkladem je validace rozsahu, která využívá objekt `Interval` z referenčního modelu. Tento objekt má metodu `has(T value)`,

která vrací `true`, pokud předaná hodnota patří do intervalu a `false` v případě opačném.

9.7.5 Možnosti personalizace vzhledu

Vzhled aplikace se dá díky použití Vaadinu snadno upravovat. Nejjednodušší možností je přepnutí vzhledu pomocí Vaadin tématu. Jedná se o soubor CSS definic, které dohromady tvoří jednotné styly napříč celou aplikací. Vzhled se mění pomocí konfiguračního souboru viz 9.4.

Druhou možností pro personalizaci look&feel aplikace je přímá editace CSS souborů. Každá komponenta, každý nadpis nebo field mají ve Vaadinu své vlastní CSS styly. V [13] je lze dohledat a předefinovat v příslušném CSS souboru.

Tento systém umožňuje přizpůsobit výsledný software bez zásahu do samotného kódu aplikace libovolnému Look&Feel stylu podle požadavků aplikace, do které se bude modul integrovat. Umožňuje jak minoritní úpravy jako změnu barvy pozadí okna, až po komplexní definice vzhledu celých komponent.

9.8 Testování

Jelikož se jedná především o grafický modul, který nemá mnoho business logiky, nebylo zde mnoho prostoru pro smysluplné automatické jednotkové testy.

Testování grafických komponent je obecně složité a tím spíše, pokud se jedná o automaticky generované rozhraní. Existují sice nástroje typu Selenium[21], ale ty jsou vhodné spíše pro testování případů užití než pro testování samotného zobrazení GUI.

Celkově mě nenapadla možnost, jak samotné vygenerované GUI automaticky testovat a otestoval jsem ho proto ručně. Použil jsem sadu archetypů, které jsem vložil do šablony a pozoroval, jakým způsobem se generuje

[21] `http://www.seleniumhq.org`

rozhraní. Testoval jsem validaci vstupních hodnot, hraniční hodnoty této validace, ale i data, která formuláře vracejí.

9.8.1 Testování validace

Validátory se jako jediná entita v modulu jevili dobrými kandidáty na automatické jednotkové testy. Pro všechny validátory byly vytvořeny testovací případy pomocí testovacího frameworku JUnit[22].
Příklad pro `IntegerQuantityRangeValidator` je vidět v příloze C

[22] `http://www.junit.org`

10 Závěr

Hlavním cílem práce bylo vytvořit modul, který bude z navržených archetypů a šablon automaticky generovat uživatelské rozhraní pro nově vznikající systém elektronické zdravotní dokumentace, který je založený na standardu *openEHR*.

Pro doménové archetypy, které vznikly převedením z terminologie odML jsem navrhl a implementoval systém pro automatizaci vytváření identifikátorů odML terminologie, které budou sloužit ke svázání nově vzniklých archetypů s původní odML terminologií.

Povedlo se vytvořit funkční systém pro generování GUI z navržených elektrofyziologických archetypů, jehož použití sahá za hranice tohoto systému, jelikož je schopen generovat GUI pro libovolné domény a data popsatelná pomocí ADL.

Navíc je modul konfigurovatelný převážně externě. Tedy konfigurační soubor, archetypy a šablony jsou mimo samotný program a lze je do systému nahrávat za běhu aplikace. Současně lze jednoduše upravit také vizuální styl generovaných formulářů podle požadavků výsledného systému. To bude v budoucnu velkou výhodou při integraci modulu do elektronické zdravotní dokumentace.

Dále byla vytvořena jednoduchá proof–of–concept aplikace, na které se dají předvést možnosti vytvořeného modulu.

Modul je připraven na další vylepšení, kterými mohou být například ukládání vytvořených formulářů do paměti nebo vytvoření úložiště pro archetypy a šablony, které by dovolovalo používat různé verze archetypů a šablon v jednom systému.

Všechny body zadání jsou splněné a modul je navržen s ohledem na další možná vylepšení. Věřím, že se v budoucnu povede systém integrovat do projektu elektronické zdravotní dokumentace, která by se mohla zařadit mezi úspěšné aplikace na poli personálních EHR systémů.

Seznam zkratek

ADL Archetype definition language – Jazyk pro popis dat pomocí archetypů.

AOM Archetype Object Modelu

EEG Elektroencefalografie

EHR Electronic Health Record

EPR Electronic Patient Record

EZD Elektronická Zdravotní Dokumentace

GUI Graphical user interface

GWT Google Web Toolkit

KIV Katedra informatiky a výpočetní techniky

MLM Multi Level Modelling

odML Open metadata Markup Language

RIA Rich Internet Applications

TK Krevní tlak

Reference

[1] K. Atalag and Y. Yang, Hong, *From openEHR domain models to advanced user interfaces: a case study in Endoscopy*, University of Auckland Department of Computer Science, 2010.

[2] W. Gary, *QS The Macroscope*, 2004. [Online]. Available: http://antephase.com/themacroscope

[3] R. Mouček and V. Papež, *Archetypes Development in Electrophysiology Domain – Electroencephalography as a Personal EHR System Module*, Department of Computer Science and Engineering, University of West Bohemia, Pilsen, 2015.

[4] R. Mouček and P. Mautner, *Pozornost řidiče při dvojí zátěži – EEG/ERP*, 2009. [Online]. Available: http://dai.fmph.uniba.sk/events/kuz2009/prispevky-pdf/moucek.pdf

[5] J. Diviš, *Driver's attention during monotonous driving and visual stimulation (ERP experiment)*, Západočeská Univerzita v Plzni, 2012. [Online]. Available: https://portal.zcu.cz/stag?urlid=prohlizeni-prace-detail&praceIdno=49898

[6] R. Mouček and P. Mautner, *Neuroinformatika – metoda evokovaných potenciálů*, ZČU Plzeň: Interní materiály, 2008.

[7] J. Grewe, T. Wachtler, and J. Benda, "A bottom-up approach to data annotation in neurophysiology," *Frontiers in Neuroinformatics*, vol. 5, no. 16, 2011. [Online]. Available: http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2011.00016/abstract

[8] T. Beale and S. Heard, *Architecture Overview*, The openEHR Foundation, 2008. [Online]. Available: http://www.openehr.org/releases/1.0/architecture/overview.pdf

[9] T. Beale, *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*, The openEHR Foundation, 2002. [Online]. Available: http://www.deepthought.com.au

[10] M. Seeger, *Key Value Stores: A Practical Overview*, Computer Science and Media Ultra-Large-Sites SS09 Stuttgart, Germany, 2009. [Online]. Available: http://blog.marc-seeger.de/2009/09/21/key-value-stores-a-practical-overview/

[11] T. Beale and S. Heard, *The openEHR Archetype Model – Archetype Definition Language*, The openEHR Foundation, 2008.

[12] N. Frankel, *Learning Vaadin*. Packt Publishing, 2011.

[13] M. Grönroos, *Book of Vaadin: Vaadin 7 Edition - 5th Revision*, Vaadin Ltd, 2014. [Online]. Available: https://vaadin.com/download/book-of-vaadin/vaadin-7/pdf/book-of-vaadin.pdf

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

Seznam obrázků

A odML terminologie – Experiment

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml stylesheet type="text/xsl" href="odmlTerms.xsl"
    <repository>http://portal.gnode.org/odml/terminologies/v1
        .0/terminologies.xml</repository>
        <version>1.0</version>
        <date>20110121</date>
    <section>
        <type>experiment</type>
        <name>Experiment</name>
        <definition>Specification of an experiment. The
            Experiment is part of a Project and the recorded
            data for a specific Experiment are found in
            Datasets. Possible subsections are, for example,
            Dataset and Stimulus.</definition>
     <property>
         <name>Description</name>
          <value>
            <type>text</type>
          </value>
         <definition>A description of the
         experiment.</definition>
      </property>
                                  ⋮
      <property>
         <name>ProjectID</name>
          <value>
            <type>string</type>
          </value>
         <definition>The ID of the project this experiment
             belongs
         to.</definition>
      </property>
  </section>
</odML>
```

B Schéma pro jazyk templatů

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/
    XMLSchema">

  <xs:element name="template" type="template"/>

  <xs:complexType name="template">
    <xs:sequence>
      <xs:element name="base" type="templateArchetype"
          minOccurs="0"/>
      <xs:element name="name" type="xs:string" minOccurs="0"/>
      <xs:element name="archetype" type="templateArchetype"
          maxOccurs="unbounded"/>
      <xs:element name="uniqueID" type="xs:string" minOccurs="0
          "/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="templateArchetype">
    <xs:sequence>
      <xs:element name="includes" type="include" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:element name="slot" type="slot" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="file" type="xs:string"/>
    <xs:attribute name="type" type="archetypeType"/>
    <xs:attribute name="useAll" type="xs:boolean" use="required
        "/>
  </xs:complexType>

  <xs:complexType name="include">
    <xs:sequence/>
    <xs:attribute name="xPath" type="xs:string"/>
  </xs:complexType>
```

```
  <xs:complexType name="slot">
    <xs:sequence>
      <xs:element name="archetype" type="templateArchetype"/>
    </xs:sequence>
    <xs:attribute name="path" type="xs:string"/>
  </xs:complexType>

  <xs:simpleType name="archetypeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="OBSERVATION"/>
      <xs:enumeration value="EVALUATION"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

C JUnit testcase – validátor

```
@Test
public void testIsValidValue() {
  validator = new IntegerQuantityRangeValidator("", new
      Interval(1, 10, true, true));
  assertTrue(validator.isValidValue("1"));
  assertTrue(validator.isValidValue("10"));
  assertTrue(validator.isValidValue("3"));
  assertFalse(validator.isValidValue("3e"));
  assertFalse(validator.isValidValue(""));
  assertFalse(validator.isValidValue(null));
}

@Test
public void testIsValidValueExclusiveBounds() {
  validatorBoundsExclusive = new
      IntegerQuantityRangeValidator("", new Interval(1, 10,
      false, false));
  assertFalse(validatorBoundsEx.isValidValue("1"));
  assertFalse(validatorBoundsEx.isValidValue("10"));
  assertTrue(validatorBoundsEx.isValidValue("3"));
}

@Test(expected = ClassCastException.class)
public void testIsValidValueException() {
  try {
    validatorDouble = new IntegerQuantityRangeValidator("",
        new Interval(1.8, 10, false, false));
    validatorDouble.isValidValue("1.9");
  } catch (Exception e) {
    assertEquals("java.lang.Double cannot be cast to java.
        lang.Integer", e.getMessage());
  }
}
```

D Externí konfigurační soubor

```
archetypeFolder=c:\\Resources\\Archetypes
templateFolder=c:\\Resources\\Templates
uploadFolder=c:\\uploads

#Validation / is archetype allowed in slot
validateSlotsArchetypes=false

#Languages and locale settings
defaultLanguage=en

#Language specific translations (English translation are
    default - wired into code)

booleanTrue.es=Si
booleanFalse.es=No
dayResolution.es=Dia
weekResolution.es=Semana
monthResolution.es=Mes
yearResolution.es=Ano
hourResolution.es=Hora
minuteResolution.es=Minuto
secondResolution.es=Segundo
#integerValidationError.es=
#button.save.es=
#button.upload.es=

booleanTrue.de=Ja
booleanFalse.de=Nein
dayResolution.de=Tag
weekResolution.de=Woche
monthResolution.de=Monat
yearResolution.de=Jahr
hourResolution.de=Stunde
minuteResolution.de=Minute
secondResolution.de=Sekunde
```

```
#integerValidationError.de=
#button.save.de=
#button.upload.de=

booleanTrue.cs=Ano
booleanFalse.cs=Ne
dayResolution.cs=Dny
weekResolution.cs=Týdny
monthResolution.cs=Měsíce
yearResolution.cs=Roky
hourResolution.cs=Hodiny
minuteResolution.cs=Minuty
secondResolution.cs=Sekundy
#integerValidationError.cs=
#button.save.cs=
#button.upload.cs=
```

E Instrukce pro uživatele

Na přiloženém DVD se nachází binární verze EHR modulu i testovací aplikace. Pro spuštění testovací aplikace stačí nahrát webový archiv `Vaadin-ProofOfConcept.war` soubor na webový server[23] a konfigurační soubory uložit do defaultního umístění – složka `Resources` z přiloženého DVD se zkopíruje do kořenového adresáře disku C.

Samozřejmě je možné znovu binární soubory vygenerovat pomocí nástroje Maven. V tom případě se nejprve musí nainstalovat do lokálního Maven repozitory *openEHR* knihovny[24], na kterých je EHR modul závislý.

EHR modul lze poté nainstalovat pomocí příkazu `mvn clean install` a až poté instalovat testovací aplikaci pomocí příkazu `mvn clean package`.

Umístění složek pro archetypy a šablony jsou definovány v konfiguračním souboru – defaultně umístěném v `c:/Resources/config.properties`.

Lze změnit i defaultní umístění samotného konfiguračního souboru a to pomocí kontextového parametru

```
<context-param>
  <param-name>propertyFile</param-name>
  <param-value>c:/Resources/config.properties</param-value>
</context-param>
```

v deployment deskriptoru `web.xml`.

[23] Vyzkoušeno s Tomcat 7.0.56

[24] urlhttps://github.com/openEHR/java-libs