Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Rozšíření frameworku pro ověřování kompatibility softwarových komponent

Plzeň, 2012 Zbyněk Růžička

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně s konzultační pomocí vedoucího bakalářské práce Ing. Kamila Ježka. Použité literární prameny a zdroje informací jsou uvedeny v seznamu literatury.

V Plzni 7. května 2012 .....

Poděkování

Rád bych zde poděkoval především panu Ing. Kamilu Ježkovi za jeho ochotu, čas, trpělivost a velmi podnětné rady, které věnoval mně a vedení této práce.

Současně chci také poděkovat panu Ing. Jiřímu Kučerovi, který mi poskytl velmi hodnotné informace při mém počátečním seznamování s komponentovým úložištěm CRCE.

Taktéž děkuji panu Ing. MSc. Přemyslu Bradovi Ph.D. za možnost osobních konzultací v průběhu zahraniční stáže pana Ježka a za podnětné komentáře k závěrečné formě tohoto dokumentu.

Abstract

Extension of a Component Repository supporting Compatibility Evaluation

Component-based development is considered to be one of today's standards. One of the tools that can help in this programming process is a Component Repository supporting Compatibility Evaluation (CRCE).

Software components may contain Extra-Functional Properties (EFP) metadata. Currently the focus is on extending the CRCE's possibility of EFP metadata indexing. Indexed EFP metadata assists with the compatibility evaluation process.

This work is focused on the design and implementation of an extension module for CRCE repository. The designed module ensures the indexing of EFP entries into the CRCE's additional metadata bound to corresponding software components. The indexing process starts after the component upload into the CRCE repository.

The first part of this thesis provides a brief introduction of the problem domain and describes a proposed format for the transcription of EFPs to the OSGi Bundle Repository (OBR) format and reasons for this format. The second part describes the implemented solution and partial implementation tasks. Class diagrams and parameters of the solution are also included.

Obsah

1 Úvod

Komponentové systémy umožňují stavět aplikace z jednotlivých programových komponent. Pro možnost vzájemné spolupráce musí být související komponenty vzájemně kompatibilní. Při vyhodnocování kompatibility se nabízí dvě hlavní kritéria – funkční a mimofunkční charakteristiky.

Mezi funkční charakteristiky patří to, že si komponenty vzájemně poskytují data prostřednictvím metod definovaných v rozhraní API (Aplication Programming Interface). Seznam dostupných metod a předávaných datových typů je jednoznačně určen rozhraním a při nedodržení těchto předpisů dojde při kompilaci nebo běhu programu v chybné části kódu k nevyhnutelné výjimce.

Mimofunkční charakteristiky nejsou přímo patrné z API. Popisují vlastnosti dané komponenty a jejich vzájemným porovnáním umožňují stanovit kompatibilitu porovnávaných komponent. Pro správný běh a odezvu programu je tedy třeba tyto vlastnosti zohledňovat při vývoji komponentového systému. Mezi mimofunkční charakteristiky patří např. doba odezvy, paměťové nároky, verze komponenty atd.

Tato práce je součástí projektu komponentového úložiště CRCE (Component Repository supporting Compatibility Evaluation), které je vyvíjeno na Katedře informatiky a výpočetní techniky na Západočeské univerzitě v Plzni. CRCE úložiště je orientováno především na ukládání OSGi (Open Services Gateway initiative) komponent.

Každá OSGi komponenta v sobě nese popis exportovaných a importovaných balíků, ale součástí tohoto popisu závislostí mohou být také mimofunkční vlastnosti zapsané definovaným způsobem.

Cílem této práce je doplnit funkčnost CRCE o indexování mimofunkčních charakteristik OSGi komponent v úložišti. Indexovaná metadata jsou možností, jak úložiště seznámit s mimofunkčními vlastnostmi komponenty a zpřístupněné informace je poté možno využít k ověřování vzájemné kompatibility komponent, což je také hlavním cílem CRCE úložiště.

Obsahem této práce je nejdříve seznámení se stávajícími aplikacemi a způsoby uchovávání metadat v těchto systémech. Poté je navrhnuta možnost vzájemného přepisu informací mezi těmito systémy a zdůvodněna volba jednoho z navržených formátů. Další část práce se zabývá popisem implementovaného řešení a jeho charakteristickými parametry. Závěrem jsou shrnuty dosažené výsledky. V příloze lze nalézt seznam odkazovaných datových zdrojů a uživatelskou dokumentaci.

2 Seznámení s doménou problému

V následujících kapitolách budou popsány jednotlivé prvky, kterých se práce dotýká a jsou zapojeny do procesu indexování mimofunkčních charakteristik (EFP) uvnitř CRCE úložiště.

Vztah a závislosti mezi jednotlivými prvky jsou zachyceny ve schématu obr. 1, který je na konci této druhé kapitoly spolu s vysvětlujícím popisem schématu.

2.1 Programová komponenta

Jedná se o samostatný celek, který poskytuje určitou funkcionalitu orientovanou na konkrétní problém. Komponenty bývají často distribuovány ve formě přeloženého binárního kódu. Využitelnost a funkce komponent bývá dána skrze jejich definované rozhraní [1].

Pro řešení složitějších a komplexnějších problémů vyvstává často potřeba propojit funkcionality více komponent i od různých skupin vývojářů. Tento přístup bývá nazýván jako komponentově orientovaný softwarový vývoj.

Mezi hlavní výhody takto orientovaného vývoje patří především urychlení vývoje cílového produktu a znuvupoužitelnost samostatných komponent při vývoji dalších aplikací.

Szyperski [2] přirovnává užitečnost programových komponent k významu integrovaných obvodů. Též uvádí analogii komponentově orientovaného vývoje s používáním lego kostek. Dále Szyperski zdůrazňuje ekonomický význam používání komponent a upozorňuje, že s existencí ekonomického trhu může vhodná míra používání komponent zvýšit konkurenceschopnost firem zabývajících se softwarovým vývojem.

S rostoucí mírou propojování komponent a „outsource“ řešení také roste robustnost (objemnost), množství závislostí a zároveň klesá flexibilita řešení. S menší mírou používání cizích komponent zase rostou náklady na vývoj vlastní implementace [2].

Nezbytnou součástí softwarového vývoje je proces testování. Při vývoji systému sestaveného z komponent vyvstává krom potřeby testování funkcionality a rozhraní samostatných komponent také potřeba ověřovat, zda komponenty společně pracují správně, jsou schopny si předávat korektní data a zda jsou v souladu jejich parametry [1].

Při ověřování vzájemné spolupráce komponent mají významnou roli mimofunkční charakteristiky komponent.

2.2 Extra-functional property (EFP)

V oblasti našeho zájmu zkratka EFP znamená mimofunkční vlastnosti programové komponenty. Jedná se o atributy, které přímo nespecifikují funkci komponenty, ale vypovídají o jejích vlastnostech, jako jsou například rychlost odezvy komponenty, spotřeba paměti, verze atd. [3].

Tyto mimofunkční vlastnosti by měly být přiřazeny programové komponentě při dokončení jejího vývoje, aby také mohly být tyto vlastnosti zvažovány a vyhodnocovány při sestavování systému z programových komponent. Mimofunkční charakteristiky zohledňuje vývojář komponentového systému.

2.3 Metadata

Metadaty rozumíme doplňující popisné údaje vztahující se k objektu, který popisují. Zjednodušeně lze říci, že metadata jsou „data o datech“.

Metadata dávají význam ve vztahu k nadřazenému objektu, který popisují a umožňují tak například snazší třízení nebo vyhledávání tím, že poskytují o svém předmětu určité sumarizované informace, které jsou stěžejní pro danou doménu.

Fotografie může například obsahovat metadata o velikosti souboru, o rozlišení obrazu, počet barev a též datum a čas expozice. Textové dokumenty pak mohou obsahovat informace o svém autoru, datum publikace, počet stránek, žánr nebo kategorii dokumentu a stručný abstrakt [11].

V našem kontextu může být nadřazeným objektem metadat programová komponenta a jejími popisnými metadaty mohou být již zmíněné mimofunkční charakteristiky. Význam těchto metadat pak spočívá v možnosti vycházet z nich při určování vzájemné kompatibility komponent.

2.4 Projekt EFFCC, programy pro práci s EFP

Programy pro správu EFP a nástroje umožňující ověřování kompatibility mimofunkčních charakteristik programových komponent jsou vyvíjeny v rámci projektu Extra Functional Property Featured Compatibility Checks (EFFCC).

V prostředí tohoto projektu jsou pro práci s EFP vlastnostmi používány tři hlavní produkty. Prvním přínosem jsou nástroje pro provozování a ovládání repositáře, který umožňuje centralizovanou správu a úschovu mimofunkčních charakteristik. Dále sem patří nástroj pro přiřazení mimofunkčních charakteristik programovým komponentám a rovněž nástroj pro vyhodnocování vzájemné kompatibility softwarových komponent obohacených o mimofunkční charakteristiky. Každá z vyjmenovaných aplikací používá pro svoji činnost řadu dalších low-level EFP programových modulů.

a) Úložiště EFP vlastností

Pro možnost centralizované správy vlastností se používá úložiště mimofunkčních charakteristik – tzv. EFP repository. Ke správě tohoto úložiště slouží aplikace *EfpRegistryGUI*. Popisem jejího ovládání a problematiky správy EFP úložiště se zabývá bakalářská práce [5].

V rámci úložiště se realizují činnosti týkající se správy mimofunkčních vlastností. Příkladem mohou být operace definování nového globálního registru, vytváření lokálních registrů uvnitř globálního. Dále plnění registrů vytvořenými EFP vlastnostmi.

b) Spojení EFP vlastností s programovou komponentou

Přiřazení EFP vlastností z EFP repository k vybrané (JAR) programové komponentě se v rámci EFFCC projektu realizuje skrze aplikaci *EfpAssignmentGUI*. V současnosti umožňuje *EfpAssignmentGUI* spojení vlastností s komponentami modelů CoSi a OSGi. Předmětem této práce je indexování EFP metadat z OSGi komponent. Popisem aplikace *EfpAssignment* se zabývá zdroj [6].

c) Vyhodnocování kompatibility porovnáním EFP vlastností

Problematiku porovnání EFP vlastností, výpočet jejich matematických formulí a vyhodnocení jejich vzájemné kompatibility řeší programový modul *EfpComparator*. Popisu této aplikace se věnuje bakalářská práce [7].

2.5 Open Services Gateway initiative (OSGi)

Informace k této kapitole jsou čerpány z informačního zdroje [8].

OSGI je modulární systém používaný v jazyce Java. Jeho předností je možnost správy modulů aplikace za běhu komponentového kontejneru.

Komponentový model OSGi se vyskytuje v několika implementacích. Používané OSGi frameworky jsou např. Apache Felix, Concierge, Equinox nebo Knopflerfish. CRCE úložiště běží v OSGi frameworku Apache Felix [4].

Jednotlivé moduly aplikace běžící v komponentovém kontejneru tvoří samostatné funkční celky. V prostředí OSGi se modul nazývá *bundle*. Toto pojmenování bude opakovaně používáno napříč dokumentací.

Vlastnosti OSGi *bundlů* jsou specifikovány v jejich manifest souboru MANIFEST.MF. Tento soubor moduly nutně potřebují pro možnost jejich provozování v OSGi kontejneru. Manifest soubor obsahuje řadu atributů. Budou zde vyzdvihnuty především tři následující atributy:

- **Bundle-Activator** - Atribut aktivátoru uvádí třídu, která bude volána po spuštění modulu. Zpravidla se nazývá rovněž *Activator* a obsahuje metody *start*() a *stop*() nebo *init*().
- **Export-Package** - Atribut obsahuje výčet balíků, které modul poskytuje k veřejnému použití pro další moduly. Každý balík může být doplněn informací o jeho verzi.

- **Import-Package** - Specifikuje balíky (a případně i jejich verzi), které modul vyžaduje pro svoji činnost.

Tyto údaje v manifest souboru jsou také určitým druhem metadat.

2.6 Component Repository supporting Compatibility Evaluation (CRCE)

Dalším částí je CRCE úložiště komponent. Jedná se o úložiště programových komponent orientované na kontrolu kompatibility uložených modulů. Informace do této kapitoly byly čerpány ze zdroje [4].

Ke každému uloženému artefaktu bývají při ukládání přiřazena popisná metadata, která vystihují charakter artefaktu. Metadata jsou ukládána ve formátu OBR (OSGi Bundle Repository), který je popisován v kapitole 3.3.

Předpokládá se především používání úložiště v souvislosti s komponentami modelu OSGi. Zároveň je ale úložiště navrženo tak, aby bylo schopné uchovávat artefakty libovolného typu, tedy například také protokoly výsledků testování nebo konfigurační soubory a u takových artefaktů je indexována jen základní množina metadat vypovídající o parametrech těchto souborů.

Ukládaný artefakt bývá nazýván též jako „resource“. Často jsou tak označeny na úrovni CRCE zdrojových kódů instance symbolizující uložené artefakty, a proto bude tento termín používán zejména v implementační části dokumentace.

Úložiště umožňuje práci s komponentami i jejich metadaty. Uložené komponenty je možné opět libovolně stáhnout z úložiště a není překážkou do úložiště nahrát vícekrát identickou komponentu, neboť při jejím nahrání se shodnému modulu přiřadí index umožňující jejich rozlišení a identifikaci novější verze komponenty.

Důležitou vlastností úložiště je možnost spouštět nad vybranými komponentami různé druhy testů dle definovaných testovacích scénářů. Tato funkcionalita je předmětem současného vývoje a je usilováno o budoucí možnost ověřovat vzájemnou kompatibilitu komponent na základě zaindexovaných EFP metadat.

Existují tři hlavní typy uživatelů CRCE úložiště. První skupinou jsou vývojáři komponent, kteří vytvořené komponenty nahrají do úložiště. Další uživatelem je vývojář systému sestaveného z komponent. Ten jednotlivé komponenty využívá s ohledem na jejich kompatibilitu. Komunikovat s CRCE úložištěm by měl být schopen také vytvořený komponentový systém, který bude ověřovat existenci aktualizací u používaných komponent.

Úložiště je ovládáno prostřednictvím webové aplikace, která používá technologie JSP a Java Servlety.

2.6.1 CRCE moduly

CRCE úložiště je navrženo jako modulární systém, jehož funkcionality zajišťují dílčí implementované moduly. CRCE moduly spolu s podpůrnými moduly třetích stran běží

v OSGi kontejneru a je možno s nimi pracovat (instalovat je, startovat nebo zastavovat jejich běh) za běhu systému. Některé moduly jsou pro běh CRCE systému nezbytné, jiné jsou volitelné a jejich použití přináší především rozšíření funkcionalit úložiště.

Například webové rozhraní systému zajišťuje modul *crce-webui*. Hlavními produkty této bakalářské práce jsou moduly *crce-efp-indexer* a *crce-efpAssignment*.

2.6.2 Modul crce-efp-indexer

Modul načítá EFP data ze zpracovávaného OSGi *bundlu* a po jejich načtení jsou tyto metadata zaindexovány a uloženy v CRCE úložišti v OBR formátu, jehož návrhem se zabývá čtvrtá kapitola. Podrobnější informace o činnosti indexeru lze nalézt v pozdější části dokumentu zabývající se implementací modulu.

2.6.3 Modul crce-efpAssignment

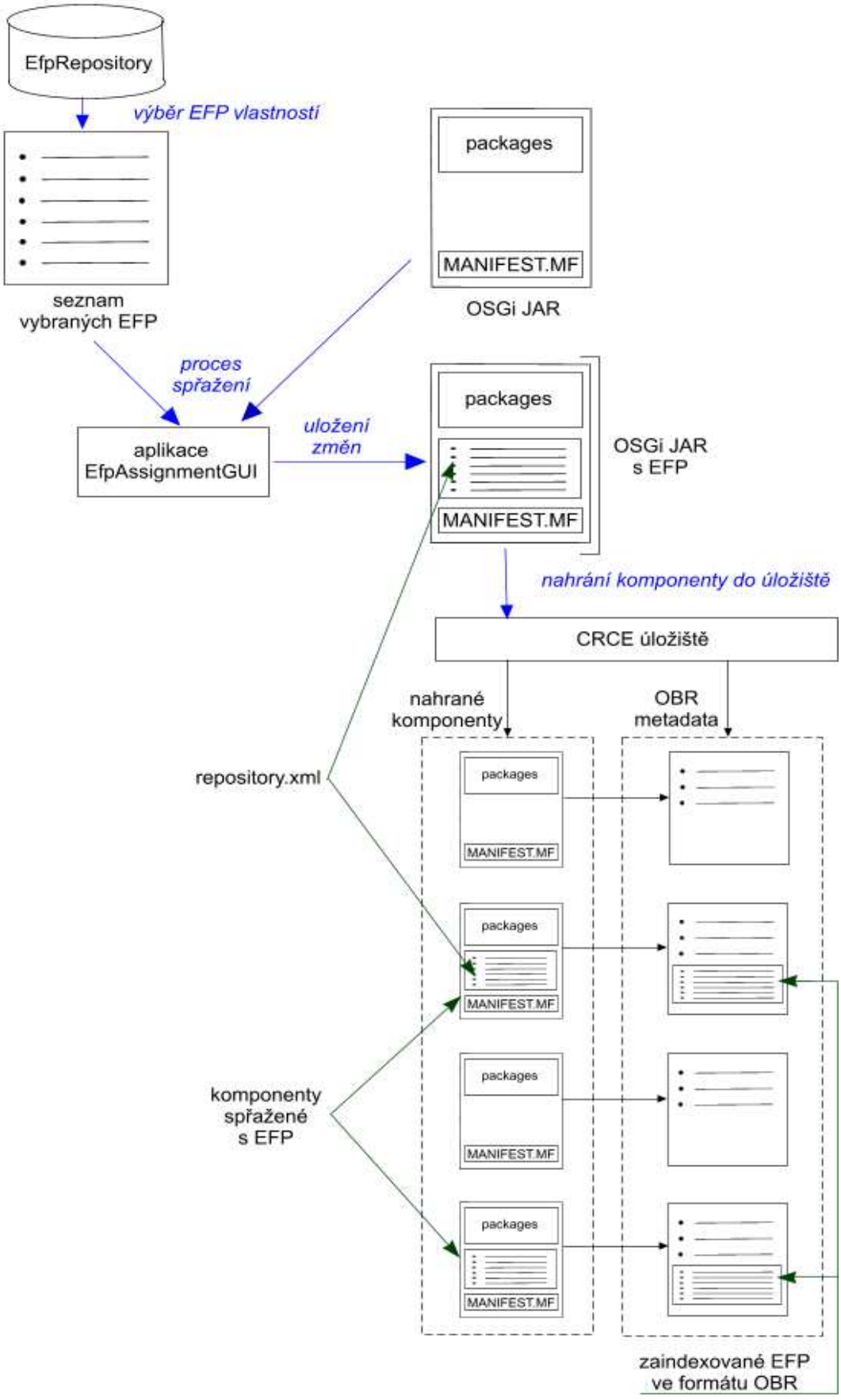
Tento modul zapouzdřuje a v prostředí OSGi kontejneru zpřístupňuje některé programy z EFFCC projektu. V současné době modul poskytuje knihovny pro načtení EFP informací z OSGi *bundlů*. V budoucnu pak může modul obsahovat i další EFFCC programy pro vyhodnocování EFP kompatibility modulů. Vytvoření tohoto samostatného modulu je podrobněji odůvodněno v implementační fázi dokumentace.

2.7 Schéma řešení

K zachycení a vysvětlení vztahů mezi právě popsanými prvky a systémy je použit obrázek obr. 1 (následující strana).

EFP repositář (ve schématu vlevo nahoře) sdružuje všechny definované a dostupné EFP vlastnosti. Aplikace *EfpAssignmentGUI* zajišťuje spřažení načtené JAR komponenty (typu OSGi nebo CoSi) spolu se zvolenými EFP vlastnostmi vybranými z EFP repositáře. Po procesu spřažení a uložení změn jsou již vybrané EFP vlastnosti zaznamenány uvnitř upravované JAR komponenty. Informace o přiřazených EFP jsou zaznamenány v souborech `MANIFEST.MF` a `repository.xml` viz další kapitola *Uložení metadat v zúčastněných systémech*.

Dalším postupem je načtení takto obohacené komponenty do CRCE úložiště. Po načtení komponenty spustí svoji činnost nově implementovaný modul *crce-efp-indexer*, který provede indexování EFP vlastností do OBR formátu. V úložišti jsou pak nahrané komponenty dostupné k opakovatelnému stáhnutí, nebo ke spouštění testů kompatibility. Každá komponenta má přiřazena popisná metadata, která vystihují charakter nahraného artefaktu a která mohou obsahovat také zaindexované EFP vlastnosti. Veškerá popisná OBR metadata jsou uložena mimo originální artefakt v přidruženém META souboru.



Obr. 1: Schéma vztahů a závislostí v problematice indexování EFP uvnitř CRCE úložiště.

3 Uložení metadat v zúčastněných systémech

Nyní jsou stručně popsány jednotlivé základní složky schématu a lze se zaměřit na to, jak jsou mimofunkční popisná metadata uložena v jednotlivých systémech.

3.1 Uložení EFP metadat v OSGi komponentě

Vstupem procesu indexování je OSGi komponenta spojená spolu s jí příslušejícími EFP vlastnostmi. Po spřažení jsou v *bundlu* EFP data uložena v manifest souboru `MANIFEST.MF` a také ve XML souboru, který bývá nazýván `repository.xml` a je umístěn ve stejné složce `META-INF` spolu se souborem manifestu.

Program, který umí tyto data načíst a zpřístupnit již existuje v rámci modulu *efpAssignment* v EFFCC projektu. Součástí úkolů této práce je využít existující programové vybavení pro načtení EFP metadat z OSGi *bundlu* a navrhnout formát a implementaci ukládání těchto dat uvnitř CRCE systému tak, aby CRCE dokázalo tato data na požádání zobrazovat a uživatel je mohl procházet v prostředí CRCE úložiště.

Výhodou toho bude, že CRCE úložiště komponent bude zahrnovat řadu funkcionalit pro správu komponent a ověřování jejich kompatibility a je vhodné, aby uživatel mohl zobrazovat také EFP data uvnitř systému a nemusel uložené komponenty stahovat mimo úložiště a jejich EFP vlastnosti číst další aplikací. Další výhodou seskupení funkcionalit do CRCE bude, že uživatel systému se nebude pro každou funkcionalitu správy komponent muset učit ovládat jinou aplikaci.

3.2 Metadata v CRCE úložišti komponent

V CRCE jsou metadata nahraných programových modulů seskupeny též v XML souboru `repository.xml`. Krom tohoto hromadného souboru má každý nahraný *bundle* ještě svůj vlastní soubor s popisnými metadaty. Je-li v CRCE realizována nějaká změna popisných metadat *bundlu*, pak je nejdříve změna uložena do tohoto individuálního META souboru a do globálního `repository.xml` je změna uložena až později, když není aplikace zatěžována uživatelskými požadavky.

Metadata jsou v CRCE ukládána v OBR formátu, který se skládá z položek Requirements a Capabilities. Tyto položky jsou popsány v kapitolách 3.3.1 a 3.3.2.

Programový modul pro zápis metadat do individuálního META souboru v CRCE již existuje a bude při realizaci řešení použit.

3.3 OSGi Bundle Repository (OBR)

Jedná se o specifikaci popsanou dokumentem OSGi RFC 112 [9]. Specifikace umožňuje popisovat každý soubor v komponentovém úložišti definovaným formátem. Formát zahrnuje položky *Capabilities* a *Requirements*.

OSGi modul, artefakt zpracovávaný v komponentovém úložišti, je v terminologii OBR označován jako *resource*. Jemu příslušející popisná OBR metadata jsou ukládána v XML souboru.

OBR formátem lze k jednotlivým zdrojům v úložišti uchovávat informace například o tom, jaké jiné programové komponenty tento *bundle* potřebuje pro svoji činnost a zároveň, jaké své balíky modul nabízí ostatním komponentám k jejich použití. Obdobně lze v OBR formátu uchovávat také informace mimofunkčních charakteristik, které již měla komponenta v sobě zabudované při nahrání do úložiště. Formátem transformace EFP do OBR metadat se věnuje čtvrtá kapitola.

3.3.1 Capabilities

Tato položka obsahuje atributy a data, kterými daný *resource* disponuje a které nabízí k použití dalším zdrojům. Jinak řečeno, uspokojuje požadavky.

Každá *capability* položka má nejdříve své popisné jméno a teprve v ní jsou uvedeny jednotlivé popisné atributy souvisejícího charakteru vzhledem k názvu dané *capability*. Každý obsažený atribut má také svoji přiřazenou hodnotu.

Běžně *capabilities* popisují balíky, které daný *bundle* exportuje, což je také případ následující ukázky ze zdroje [10]:

```
<capability name='package'>
   <p n='package' v='org.foo.bar'/>
   <p n='version' t='version' v='1.0'/>
</capability>
```

Jednotlivé `<p>` elementy (property) mohou zahrnovat následující vlastnosti:

`n` – Název vlastnosti artefaktu.
`v` – Hodnota vlastnosti.
`t` – Typ hodnoty vlastnosti. Typy hodnot mohou být `string`, `version`, `uri`, `long`, `double` nebo `set`.

Uvedený příklad popisuje export balíku *org.foo.bar* ve verzi 1.0

3.3.2 Requirements

Položka popisuje požadavky a nároky příslušného *bundlu*. Konkrétní požadované vlastnosti a jejich hodnoty jsou vyjmenovány ve filtru.

Následuje příklad použití ze zdroje [10]:

```
<require name='package' extend='false'
  multiple='false' optional='false'
  filter='(&(package=org.foo.bar)(version>=1.0.0))'>
  Import package org.foo.bar
</require>
```

Položka popisuje požadavek na import balíku *org.foo.bar* ve verzi alespoň 1.0.0.

4 Formát přepisu EFP do OBR

Porovnáme-li popis EFP vlastností, který je používán v rámci aplikace *EfpAssignment*, spolu s charakterem OBR metadat, pak velkou výhodu je fakt, že oba popisné formáty umožňují třídit metadata ve smyslu „nabízím/propaguji k použití“ a v obráceném smyslu „vyžaduji/vyhledávám ke svému použití“.

Na úrovni OBR jsou to položky *Capabilities* a *Requirements*, které jednoznačně třídí metadata artefaktu dle výše uvedeného smyslu. A současně na úrovni popisu EFP lze každou dílčí EFP vlastnost klasifikovat, zda je přiřazena k *feature* s typem *PROVIDED* nebo *REQUIRED*.

EFP vlastnosti typu *PROVIDED* lze přepisovat do OBR metadat jako *capabilities* a obráceně EFP typu *REQUIRED* budou popsány skrze *requirements*.

K popisu OBR *capability* se nabízí prvky:

1) Název položky *capability*.
2) Element s párem dat: název vlastnosti – hodnota vlastnosti
3) Element s trojicí dat: název vlastnosti – typ vlastnosti – hodnota vlastnosti

V případě OBR *requirement* se nabízí následující popisné prvky:

1) Název položky *requirement*.
2) Filtr, který specifikuje vlastnosti a hodnoty požadavku.
3) Doplňující atributy jako *extend*, *multiple* a *optional*.

4.1 Současný praktický požadavek na formát přepisu

Závazná specifikace formátu přepisu se odvíjí od praktických požadavků na využití těchto zaindexovaných EFP metadat. Tedy například od toho, jaká bude činnost a cíl programového modulu, který bude zaindexovaná EFP metadata využívat a jaké všechny popisné znaky EFP dat využije (zda např. uvádět hodnoty *efp-id* a *gr-id*, nebo nikoliv).

V současnosti neexistují striktně závazné požadavky na formát zápisu EFP vlastností do OBR metadat, neboť pro zpracování zaindexovaných EFP vlastností zatím neexistuje jiná CRCE programová funkcionalita, která by tyto data nyní zpracovala. Využitelnost a zpracování zaindexovaných EFP metadat bude předmětem dalšího vývoje. Současným požadavkem na formát zaindexovaných EFP vlastností je především vyhovující čitelnost a přehlednost metadat pro člověka, uživatele CRCE.

Z toho vyplývá, že důležitější, než striktní formát přepisu EFP vlastností do OBR metadat, je návrh a vlastní realizace programového modulu, který bude součástí CRCE a který umožní zaindexovat EFP vlastnosti z OSGi *bundlů*. Výsledkem této bakalářské práce bude použitelný programový modul v CRCE a lidsky čitelný formát přepisu EFP metadat. Striktní formát přepisu zajistí další studenti, až budou existovat konkrétní

požadavky na formát přepisu vyplývající z využitelnosti zaindexovaných vlastností jiným CRCE programovým modulem.

4.2 Formát přepisu EFP PROVIDED do OBR Capability

4.2.1 Formát přepisu bez rozlišení datových typů

Nejdříve se nabízí velmi přímočarý styl přepisu:

```
<capability name='EFP'>
   <p n='parent-feature' v='org.foo.bar'/>
   <p n='efp-name' v='memory_consumption'/>
   <p n='efp-id' v='76'/>
   <p n='value' v='[3000.0; 6000.0]'/>
   <p n='value-name' v='average'/>
   <p n='gr-name' v='Education'/>
   <p n='gr-id' v='41'/>
   <p n='type' v='simple'/>
   …
</capability>
```

Další EFP vlastnosti budou zaznamenány v dalším bloku `<capability name='EFP'> … </capability>`, který se ale bude lišit obsaženými hodnotami atributů.

Pro tento první typ popisu je charakteristické, že se skládá vždy z dvojice atributů (String) *attr_name* a (String) *attr_value*. Datový typ jednotlivých *attr_value* se nerozlišuje a předpokládá se pro všechny hodnoty jednotný datový typ textového řetězce. Obecně tedy jeho `<p>` element vypadá takto:

```
<p n=(String)'attr_name' v=(String)'attr_value'/>
```

Tento navrhnutý formát přepisu umožňuje snadnou identifikaci zaindexovaných metadat. Programový modul, který bude metadata využívat, ze seznamu všech metadat identifikuje mimofunkční vlastnosti dle názvu *capability* `'EFP'`. Identifikace příslušnosti k určitému balíku je zajištěna přes atribut `'parent-feature'`. Obecně pak hodnoty všech potřebných vlastností lze dohledat skrze název atributu *attr_name*.

I přes svoji jednoduchost poskytuje tento formát dobrou vyjadřovací schopnost, ale na rozdíl od následujícího formátu, tento v sobě standardně nenese datový typ hodnoty.

4.2.2 Formát přepisu s rozlišením datových typů

Naproti tomu je možné se pokusit o stupeň složitější formát vyjadřování, kde by jednotlivé popisné elementy mohly využívat k popisu též další atribut „*type*“ a nevázat se tak pouze na standardní typ atributů String. Obecně se tedy nabízí následující formát `<p>` elementu:

```
<p n=(String)'attr_name' t=(String)'attr_value_type'
v=(attr_value_type)'attr_value'/>
```

U některých elementů by tak mohl jejich charakter vypadat následovně:

```
<p n='meta-list' t='set' v='veryHigh, verLow, low, average, high'/>
```

případně u některých specifických EFP vlastností (avšak určitě ne obecně u všech) by mohl být u atributu s názvem `value` použit tvar

```
<p n='value' t='long' v='123456789'/> nebo
<p n='value' t='double' v='3,141592'/>
```

Této druhé variantě popisu EFP s využitím atributu *type* v praxi ale velmi zásadně vadí naprosto malý počet datových typů, který sice vyhovuje původnímu záměru použití OBR, ale pro využívání atributu *type* v souvislosti s EFP chybí například často využívané typy hodnot jako *NumberInterval*, *Boolean* atd. Číselný interval by sice bylo možné nahradit dvouprvkovým setem, ale problém nedostatku datových typů by to zcela nevyřešilo a navíc by bylo těžké identifikovat, zda se v takovém případě jedná z pohledu významu dat o interval, nebo skutečně o dvouprvkový set.

Výhodou tohoto formátu je vyšší informační hustota. Jediným elementem lze vyjádřit název atributu, hodnotu atributu a oproti prvnímu formátu navíc také omezenou množinu datových typů atributu.

4.2.3 Formát získaný kombinací kladů předešlých formátů

Přestože má první formát oproti druhému nižší vyjadřovací schopnost, informace o použitých datových typech lze přenášet i prvním zmíněným formátem. Stačí v rámci používaných n='(String)attr_name' definovat určitou množinu klíčových slov, které by vyjadřovaly datový typ zvolených atributů.

Lze tedy definovat, že attr_name odpovídající klíčové hodnotě „value-type“ bude specifikovat použitý datový typ atributu s názvem „value“. Attr_value elementu „value-type“ pak může specifikovat konkrétní datový typ např. „EfpNumberInterval“, „EfpBoolean“, „EfpFormule“ a tato další klíčová slova umožní určit, jak bude interpretována attr_value u elementu s názvem „value“.

Tato konečná množina použitých datových typů je již součástí informací přenášených v EFP datech. Lze ji proto rovnou využívat bez nutnosti definování vlastního výčtového typu v implementovaném modulu pro popis datových typů hodnot atributů.

4.2.4 Zvolený formát přepisu pro implementaci

V implementaci bude u Capability bloků využíván uvedený kombinovaný formát přepisu s využitím atributu „**value-type**“, který specifikuje datový typ hodnoty atributu „value“ a který je převzat z druhého navrženého formátu.

Výsledný formát přepisu tedy bude vypadat například takto:

```
<capability name='EFP'>
   <p n='parent-feature' v='org.foo.bar'/>
   <p n='efp-name' v='memory_consumption'/>
   <p n='efp-id' v='76'/>
   <p n='value' v='[3000.0; 6000.0]'/>
```

```
    <p n='value-type' v='NumberInterval'/>
    <p n='value-name' v='average'/>
    <p n='gr-name' v='Education'/>
    <p n='gr-id' v='41'/>
    <p n='type' v='simple'/>
    …
</capability>
```

Důvody k rozhodnutí pro zvolený formát:

- Použitím atributu „value-type“ bude vyvážena celková vyjadřovací schopnost obou formátů.
- Druhý formát poskytuje velmi omezený počet datových typů (string, version, uri, long, double, set), který není schopen plně pokrýt všechny datové typy využívané při popisu EFP.
- Hodnota „value-type“ bude v každé EFP vlastnosti využita vždy jen jednou. Pokud by ale byl datový typ klíčový pro každý zaindexovaný atribut, bylo by nutné přehodnotit možnosti použití druhého formátu.

  Konkrétní příklad: Pokud je pro nás důležitých N atributů, ale jen u jediného chceme poznamenat datový typ, pak v současnosti spolu s „value-type“ zaindexujeme (N+1) atributů. Pokud by nás ale zajímal datový typ všech atributů, museli bychom v případě použití prvního formátu indexovat 2N atributů, což už není žádoucí.
- Implementovaný funkční kód bude trochu jednodušší a rychlejší, než kdybychom zaznamenávali a později rozlišovali datový typ úplně každé hodnoty, jako u druhého formátu.
- Toto rozhodnutí je v souladu se současnými požadavky viz podkapitola 4.1.

4.3 Formát přepisu EFP typu REQUIRED do OBR Requirement

V případě přepisu EFP do OBR *Requirement* jsou možnosti formátu přepisu poměrně jednoznačně dány a není zde mnoho prostoru pro invenci. Klíčové je zde především to, že veškeré indexované vlastnosti s jejich hodnotami je třeba zaznamenat v položkách záznamu filtru. Jednotlivé položky filtru jsou mezi sebou ve vztahu logického součinu. Tím je umožněno zaznamenat potřebný počet EFP atributů spolu s jejich hodnotami.

Nevýhodou je zde přehlednost záznamu filtru pro člověka, neboť všechny položky spolu s jejich daty jsou seskupovány do řádky, a tak při zaindexování záznamů mnoha položek bude řetězec filtru snadno působit nepřehledně. Při běžném použití *Requirement* formátu totiž filtr obsahuje především název požadovaného balíku a požadovanou verzi. Více položek u standardního použití není třeba. Teoreticky ale v případě našeho použití můžeme zaindexovat libovolný počet položek, stejně jako u typu *Capability*, neboť při programovém zpracování obsahu filtru je možné záznam filtru jednoznačně rozložit (rozparsovat) a každou položku vyhodnocovat zvlášť.

4.3.1 Zvolený formát přepisu pro implementaci

Seznam indexovaných atributů a jejich význam je uveden v následující podkapitole 4.4.

Pro položky *extend*, *multiple* a *optional* se ve smyslu použití s EFP daty v současnosti nenachází uplatnění, proto jsou tyto vlastnosti zatím ponechány nevyužité.

Indexované EFP typu *REQUIRED* budou mít pro ilustraci například následující podobu:

```
<require name='EFP' filter='(&amp;(parent-
name=cz.zcu.kiv.osgi.example.inventory.inventorydata.storequeryif)
(parent-type=package)(efp-name=response_time)(assignment-type=NAMED)
(value-name=high)(value=[7500.0; 10000.0]))' extend='false'
multiple='false' optional='false'>
</require>
```

4.4 Základní atributy pro indexaci v OBR typech Capability i Requirement

Všechny atributy a hodnoty uvedené v OBR *Requirement* musí být uspokojeny odpovídajícím typem *Capability*. Příslušná *Capability* tedy obsahuje všechny požadované atributy a nabízí všechny požadované hodnoty. S tímto základním požadavkem je nutno počítat při návrhu základních atributů.

Základní atributy musí umožňovat jednoznačnou identifikaci konkrétní EFP vlastnosti. Současná implementace indexuje tyto základní atributy:

parent-type – Typ rodičovského prvku (*bundle* nebo častěji *package*).

parent-name – Jméno rodičovského prvku. Často je to jméno balíku.

efp-name – Jméno dané EFP vlastnosti.

assignment-type – Typ přiřazení EFP hodnoty (*named*, *direct*, a *formula)*.

Dalším základním atributem je jeden z těchto: *value*, *formula* nebo *deriving-formula*. Volba těchto atributů však závisí na hodnotě atributu *assignment-type* dané EFP.

U typu OBR *Requirement* současná implementace nepřidává žádné další atributy, neboť tyto základní jsou pro jednoznačnou identifikaci konkrétní EFP vlastnosti dostačující a přidávání dalších vlastností by bylo na úkor snadné přehlednosti pro člověka.

Na druhou stranu u typu OBR *Capability* je v CRCE zobrazován každý atribut na samostatný řádek, což je pro člověka přehlednější. U typu OBR *Capability* si tedy současná implementace dovoluje indexovat několik dalších atributů (informace o globálních a lokálních registrech, typ *SIMPLE* nebo *DERIVED*, *value-type* i *efp-id*). Ustálení počtu a typu atributů se bude v budoucnu odvíjet od konkrétních praktických požadavků, jak je uvedeno v kapitole 4.1.

5 Realizace EFP indexeru pro CRCE

5.1 Výchozí stav před implementací CRCE-EFP indexeru

Nyní lze zrekapitulovat stav aplikací před jejich úpravou.

a) Existuje programové vybavení na čtení EFP vlastností z JAR archivu obohaceného o EFP vlastnosti. Tento program *EfpAssignment* je standardně využíván v EFFCC projektu a je spouštěn přímo v Java virtuálním stroji.

b) V rámci zdrojových kódů CRCE úložiště existují metody pro zápis metadat ve formátu OBR. Tyto OBR metadata jsou uchovány v individuálním META souboru příslušejícímu k artefaktu, který je nahrán do CRCE úložiště. Velmi důležité je zde zmínit, že celý CRCE systém běží v OSGi modulárním systému.

Z různorodosti prostředí provozu propojovaných aplikací vyplynuly při práci ve fázi implementace nepříjemné starosti a zdržení vývoje funkčního kódu.

5.2 Dílčí úkoly implementace

V rámci implementace bylo nutné vyřešit následující dílčí kroky.

a) Vytvoření programového crce-efp-indexer modulu, který se bude spouštět při definované události.

b) Přidání programu *EfpAssignment* a dalších EFFCC závislostí do knihovního modulu, aby v prostředí OSGi kontejneru implementovaný crce-efp-indexer mohl využívat třídy a metody programu *EfpAssignment*.

c) Pomocí tříd a metod programu *EfpAssignment* sestavit algoritmus, který z OSGi artefaktů načte jejich EFP vlastnosti, pokud artefakt tyto vlastnosti obsahuje.

d) Vhodným způsobem zpracovat všechny načtené EFP a postupně je metodami CRCE úložiště přiřadit do objektů metadat zpracovávaného artefaktu a v závěru práce realizované změny uložit

5.2.1 Vytvoření crce-efp-indexer modulu

Nově vytvořený CRCE modul obsahuje aktivační třídu nazvanou *Activator*. Na tuto třídu ukazuje „Bundle-Activator,“ což je vlastnost *bundlu* uvedená v manifest souboru `MANIFEST.MF`. Zmíněná třída Activator obsahuje metodu *init*(), která zajišťuje přidání nové komponenty do instance *DependencyManageru*. Bývá zde také určena obslužná třída, která implementuje požadované chování na vzniklé události. V našem případě je obslužnou třídou *ResourceActionHandler*. Dále je v metodě *init*() řešen

import využívaných (případně export poskytovaných) služeb technologií zvanou dependency injection (česky vkládání závislostí).

Vytvořený modul je nezbytné uvést také v hlavním kořenovém konfiguračním souboru `crce-root/pom.xml` v bloku `<modules> … </modules>`, čímž je kompilačnímu a sestavovacímu (= „buildovacímu“) nástroji Maven udělen pokyn, aby uvedený modul překládal a sestavoval spolu s ostatními.

Spuštění modulu v OSGi kontejneru současně s dalšími moduly na příkaz `mvn pax:provision` lze docílit zanesením odkazu závislosti na modul do souboru `crce-root/provision/pom.xml`.

Události v CRCE předcházející předání obsluhy modulu crce-efp-indexer

Po výběru artefaktu k nahrání do repositáře a spuštění operace ukládání se vybraný artefakt nejdříve uloží do bufferu. Tam jsou artefaktu přiřazena popisná metadata a poté je artefakt spolu s popisným META souborem uložen do složky `crce-root/runner/store`. V tomto adresáři jsou uchovávány všechny nahrané artefakty spolu s jejich popisnými metadaty.

CRCE systém umožňuje na úrovni zdrojového kódu reagovat na vzniklé události v průběhu činnosti úložiště. Událost, při které jsou volány obslužné metody crce-efp-indexeru, se nazývá *afterUploadToBuffer*. Je to stav, kdy uživatel odeslal požadavek k nahrání artefaktu do CRCE úložiště a systém uložil zpracovávaný artefakt do svých datových struktur.

5.2.2 Propagace EFFCC závislostí do OSGi kontejneru

Předmětem tohoto problému je fakt, že OSGi kontejner, ve kterém CRCE běží, standardně neposkytuje modul EfpAssignment a ostatní EFFCC závislosti modulu crce-efp-indexer. Pro možnost jejich použití v prostředí kontejneru je třeba potřebné EFFCC moduly zapouzdřit do OSGi *bundlu*, čehož lze docílit sestavovacím nástrojem Maven.

Do budoucna lze očekávat, že v rámci CRCE budou provozovány ještě další „CRCE-EFP funkcionality“ jako například EfpComparator. Tyto postupně vyvíjené dílčí funkcionality/moduly budou používat řadu shodných modulů z EFFCC projektu.

Z toho důvodu byl vytvořen jeden samostatný knihovní modul crce-efpAssignment, který zapouzdřuje a skrze atribut Export-Package zpřístupňuje ostatním modulům s crce-efp funkcionalitami třídy z programu efpAssignment a jeho dalších efp-doplňků. Tento modul nepotřebuje vlastní *ActionHandler* pro obsluhu událostí vzniklých při běhu CRCE, pouze slouží jako jeden centrální knihovní modul s EFFCC programy.

Výhodou tohoto přístupu je, že jednotlivé CRCE-EFP moduly ve svém OSGi *bundlu* nemusí obsahovat vždy identické balíky z EFFCC projektu. V první řadě se tím zabrání nežádoucí redundanci dat. Dále při práci na vývoji jednotlivých crce-efp funkcionalit nejsou při každém dílčím sestavení upravovaného crce-efp modulu vždy znovu a znovu přibalovány identické programy z EFFCC projektu a nové sestavení upravovaného

modulu je rychlejší. Současně je jednotnou *crce-efpAssignment* knihovnou zajištěno, že všechny EFP funkcionality pracují se stejnou verzí podpůrných EFFCC balíků.

Implementovaný *crce-efp-indexer* modul je spouštěn jako samostatný OSGi *bundle* plugin. Tvoří tak dílčí volitelnou funkcionalitu rozšiřující možnosti CRCE.

Klíčová opatření při vytváření crce-efpAssignment knihovny

a) Aby crce-efpAssignment knihovna obsahovala požadované EFFCC moduly, je třeba v konfiguračním `pom.xml` souboru v bloku `<dependencies>` nastavit Maven závislost na příslušné požadované moduly, která vypadá např. takto:

```
<dependency>
    <groupId>cz.zcu.kiv.efps</groupId>
    <artifactId>efpAssignmentOSGi</artifactId>
    <version>1.3-SNAPSHOT</version>
</dependency>
```

b) Při buildování *crce-efpAssignment* modulu pluginem *maven-bundle-plugin* je třeba do *bundlu* vložit používané závislosti a nastavit na ně cesty uvnitř manifest souboru. Tuto operaci lze automatizovat zápisem následujících instrukcí v `pom.xml` pod umístněním `project-build-plugins-plugin-configuration-instructions`:

```
<Include-Resource>{maven-resources},{maven-dependencies}
</Include-Resource>
<Bundle-ClassPath>., {maven-dependencies}</Bundle-ClassPath>
```

Modul *crce-efpAssignment* tak využívá své vlastní nastavení *maven-bundle-pluginu* a obráceně modul *crce-efp-indexer* se sestavuje dle přednastavené konfigurace *maven-bundle-pluginu* společné pro naprostou většinu CRCE modulů.

c) Pro potřeby sestavování maven nástrojem a možnost dotažení EFFCC závislostí je třeba uvést v rodičovském konfiguračním pom.xml souboru také odkazy na repositář EFFCC projektu viz kapitola B.1 přílohy.

d) Balíky třetích stran, které EfpAssignment používá a zároveň nejsou standardně poskytovány OSGi kontejnerem jsou do kontejneru prosazeny uvedením potřebných balíků v bloku:
`<org.osgi.framework.system.packages.extra>`, který je součástí nastavovaných `<properties>` v rodičovském konfiguračním souboru pom.xml.

e) Součástí problému jsou balíky, které obsahují závislosti na další knihovny třetích stran, ale tyto knihovny třetích stran při běhu crce-efp-indexeru nejsou využívány. Nepoužívané balíky třetích stran jsou z nastavení závislostí v Import-Package vykázány jejich kompletním výčtem v konfiguračním BND souboru zápisem:

```
Import-Package: \
      !javax.swing.*; \
      !com.thoughtworks.*; \
```

{výčet balíků byl pro ilustrativní potřeby zkrácen}

f) Závěrem pro parsování XML a možnost validace souborem XSD schématu bylo nutné do OSGi kontejneru doplnit propagaci validačního balíku

`com.sun.org.apache.xerces.internal.jaxp.validation`. Modul obsahující tento balík bylo možné stáhnout z MVN Repository [JAXP].

Vyřešení této kapitoly propagace *EfpAssignment* do OSGi kontejneru představovalo největší komplikace. Dosud balíky *EfpAssignment* nebyly používány v rámci OSGi kontejneru a neexistoval zažitý způsob a postup jejich propagace do OSGi kontejneru.

5.2.3 Postup načtení EFP vlastností

Před načítáním EFP metadat z artefaktu se nejdříve ověřuje, zda je artefakt OSGi *bundlem*. Dále je zvolen typ *ComponentLoaderu*. Každý dílčí typ *ComponentLoaderu* rozumí vnitřní struktuře určitého druhu komponent a dokáže v nich lokalizovat mimofunkční charakteristiky. Našemu typu OSGi komponent odpovídá *OSGiAssignmentImpl*. Dále je *ComponentLoaderu* předána cesta k obsluhovanému artefaktu a jsou načteny soubory `MANIFEST.MF` a `repository.xml` s EFP metadaty. V dalším kroku je načítán seznam balíků (features) a obsažených mimofunkčních vlastností. Jednotlivé balíky s jejich EFP vlastnostmi lze postupně procházet a získávat dílčí EFP atributy a jejich hodnoty.

5.2.4 Přepis EFP do OBR formátu

Je třeba zaindexovat všechny načtené EFP vlastnosti. Formát přepisu odpovídá návrhu popsanému a odůvodněnému v kapitole 4 (*Formát přepisu EFP do OBR*). Poskytované vlastnosti typu *PROVIDED* se indexují jako *Capability* a naopak požadované *REQUIRED* EFP vlastnosti se přepisují jako *Requirement*. Každý z *Capability*/*Requirement* bloků obsahuje zvolené EFP atributy a jejich hodnoty. V závěru práce s artefaktem je nutné zaindexovaná metadata uložit do META souboru.

5.3 Přehled implementovaných tříd modulu crce-efp-indexer

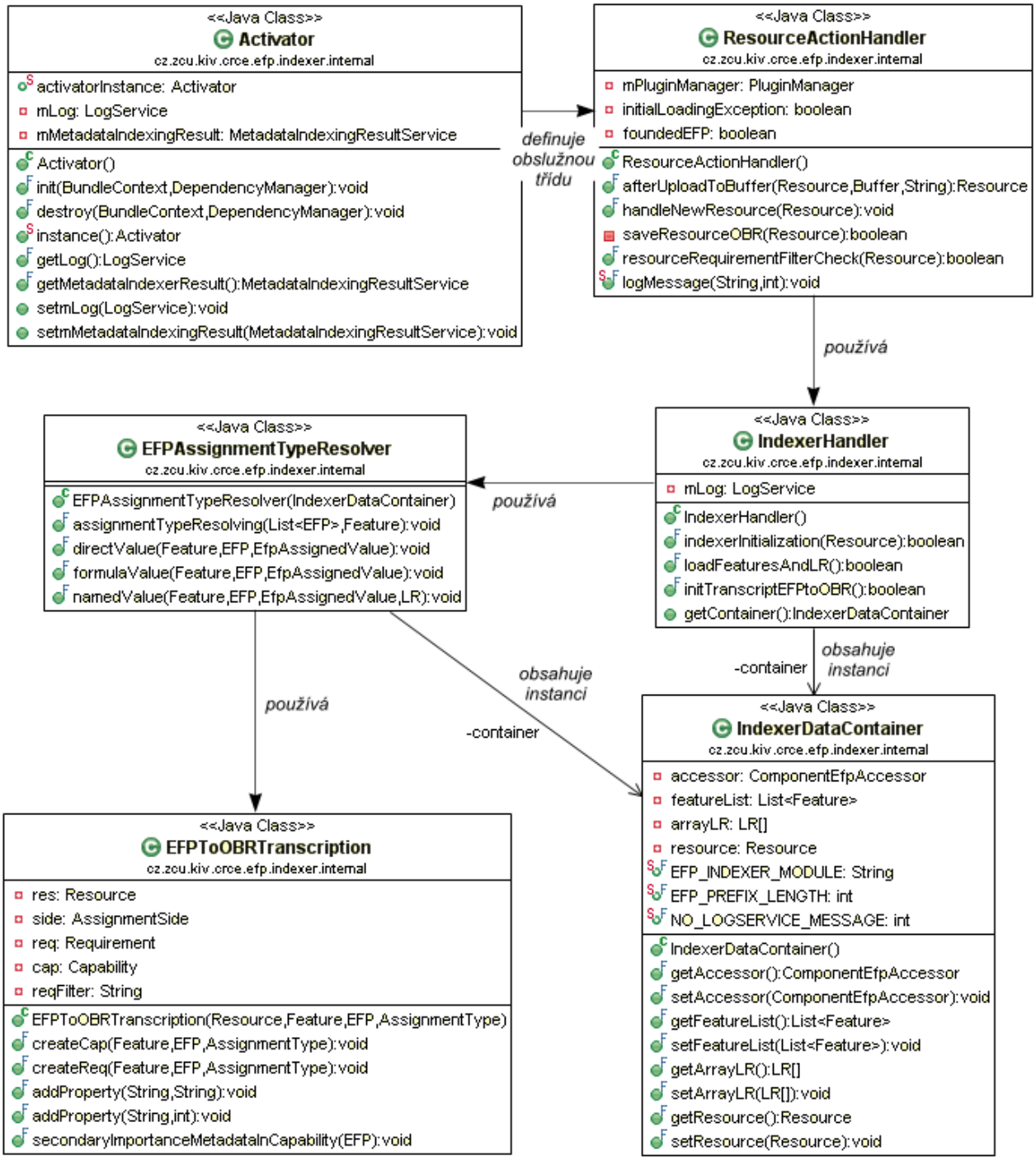
Implementované třídy se nacházejí v balíku

`cz.zcu.kiv.crce.efp.indexer.internal`.

Tato kapitola se zabývá popisem hlavních rysů jednotlivých tříd. Následuje diagram tříd obr. 2, který pro každou třídu zachycuje seznam metod včetně jejich argumentů a návratových typů. Současně jsou pro každou třídu uvedeny její privátní atributy.

Activator

V aktivační třídě je uvedena obslužná třída událostí *ResourceActionHandler* a pomocí technologie dependency injection je zpřístupněna „logovací“ služba a služba *PluginManager*, která umožňuje uložení změn v metadatech artefaktů. Současně je zde definována závislost na též nově implementované službě *MetadataIndexingResultService*, která předává zprávy o výsledku indexování EFP metadat do modulu webového rozhraní pro možnost informování uživatele úložiště.



Obr. 2: Diagram tříd modulu crce-efp-indexer.

ResourceActionHandler

Při požadované události *afterUploadToBuffer* se nejdříve ověřuje, zda je zpracovávaný *resource* OSGi komponentou a poté jsou skrze třídu *IndexerHandler* z *bundlu* načítány mimofunkční charakteristiky. V případě pozitivního nálezu EFP vlastností je spouštěn sled operací k jejich postupnému zaindexování do metadat artefaktu.

Po úspěšném zaindexování EFP a před uložením metadat do META souboru se ověřuje, zda žádný z *Requirement* objektů nepostrádá povinnou položku *filter*. V případě absence filtru by hrozila při ukládání výjimka *NullPointerException*. Požadovaný filtr

se nastavuje nad instancí třídy *Requirement* metodou *setFilter(String filter)*. Příklad řetězce filtru je možné nalézt v kapitole 3.3.2, případně též v 4.2.

Pokud zpracovávaný zdroj není OSGi komponentou, nejsou nalezena EFP metadata, nebo dojde při zpracování artefaktu k výjimce, je CRCE uživatel o těchto událostech informován výpisem do webového rozhraní.

IndexerHandler

Třída zpřístupňuje metody, které jsou volány třídou ResourceActionHandler. Jedná se o metodu načtení exportovaných a importovaných balíků (tzv. *features*) OSGi komponenty a jejích EFP vlastností. Tuto činnost umožňují třídy *EfpAwareComponentLoader* a *ComponentEfpAccessor*, které jsou součástí API poskytovaného modulem EfpAssignment.

Pokud artefakt obsahuje EFP metadata, ale s nepodporovanou verzí XSD schématu, skončí operace zachycením výjimky, indexer výjimku loguje a propaguje informativní výpis do webového rozhraní a dále již indexer s artefaktem nepracuje.

IndexerHandler zpřístupňuje také metodu *initTranscriptEFPtoOBR*() pro zahájení procesu indexování. Indexování spočívá v prohledání jednotlivých *features* a načtení jejich EFP. Pokud je seznam EFP příslušející k danému balíku nenulový, jsou tyto parametry (daný balík a seznam jeho EFP) předány metodě *assignmentTypeResolving()* třídy *EFPAssignmentTypeResolver*.

IndexerDataContainer

Zapouzdřuje datové instance používané třídami *IndexerHandler* a *EFPAssignmentTypeResolver* při procesu načítání EFP informací a následně při přepisu EFP do OBR.

Tento přístup propagování datových instancí skrze zapouzdřující objekt byl upřednostněn namísto předávání nezanedbatelného počtu několika instancí mezi odlišnými třídami skrze argumenty jejich metod.

Metody této třídy jsou výhradně *gettry* a *settry* pro manipulaci s datovými instancemi.

EFPAssignmentTypeResolver

Třída poskytuje metodu pro rozpoznání *AssignmentType* atributu u každé EFP a dále metody pro zacházení s danou mimofunkční vlastností s ohledem na rozpoznaný *AssignmentType*. Tento výčtový typ nabývá vždy jednu z hodnot *direct*, *formula* nebo *named*. Podle typu atributu se pak odlišují vlastnosti, které jsou třídou *EFPToOBRTranscription* zaznamenány do OBR formátu. Například u *AssignmentType.direct* je zaznamenávána vlastnost „value“, která nabývá *true* nebo *false* hodnoty. U *AssignmentType.formula* se zaznamenává vlastnost „formula“ s hodnotou vzorce pro výpočet. A pro *AssignmentType.named* může být zaznamenána určitá hodnota vlastnosti „value“, nebo vlastnost „deriving-formula“ u EFP s typem *DERIVED*.

EFPToOBRTranscription

Třída je zodpovědná za formát přepisu EFP vlastností do OBR formátu a odděluje tento proces od zbylých operací v průběhu indexování EFP. Toto oddělení umožňuje,

aby osoba v budoucnu zodpovědná za úpravy ve formátu přepisu se mohla při plnění úkolu soustředit výhradně na způsob indexování dat a aby nebyla rozptylována nebo matena dalším funkčním kódem.

V souvislosti s budoucím upřesněním požadavků na formát přepisu EFP dat do OBR se očekává, že metody této třídy budou v budoucnu upravovány v první řadě. Každá třída zahrnuje své dokumentační komentáře, ale pro možnost rychlejšího a snazšího pochopení kódu této třídy zde bude následovat několik doplňujících informací.

Zaindexování základních a nezbytných hodnot mimofunkční vlastnosti je možné skrze argumenty předané při volání konstruktoru třídy. Z nich je možné rozlišit vlastnost *AssignmentSide.PROVIDED* nebo *AssignmentSide.REQUIRED*, což je rozhodující vlastností pro definování OBR Capability nebo Requirement a konkrétní zaindexované vlastnosti jsou popsány v kapitole 5.3 - *Základní atributy pro indexaci v OBR typech Capability i Requirement*.

Indexování dalších vlastností závisí na hodnotě atributu *AssignmentType*. Třída *EFPAssignmentTypeResolver* podle této hodnoty EFP volí doplňující vlastnosti k zaindexování a jejich název a hodnota jsou vždy předávány do této třídy *EFPToOBRTranscription* skrze metodu *addProperty*. Tato metoda je deklarována jako přetížená s odlišnými datovými typy argumentu hodnoty. Rozlišení datových typů *String* a *Integer* má význam u OBR Capability, do které se přidávají vlastnosti metodou *setProperty* a ta je též přetížená. Webové rozhraní CRCE pak v seznamu zaindexovaných OBR metadat u Capability výpisu zobrazuje také datový typ indexovaných hodnot a bylo by matoucí, aby celočíselné hodnoty byly označovány typem String. Tímto je přetížení metod odůvodněné.

U typu OBR Capability jsou pak indexovány ještě některé doplňující vlastnosti týkající se globálních, případně i lokálních registrů a také některé další atributy. Tyto doplňující vlastnosti zatím nejsou vyžadovány OBR typem Requirement, pouze jsou další informací pro uživatele dle zásad uvedených v kapitole 5.1 - *Současný praktický požadavek na formát přepisu*.

Pro připomenutí lze zmínit, že OBR Requirement vyžaduje definování řetězce filtru. Podrobněji na tento požadavek upozorňuje odstavec u třídy *ResourceActionHandler*.

6 Ověření funkčnosti

6.1 Funkční a jednotkové testy

Funkčnost implementovaného kódu byla ověřována přes JUnit testy. Nebylo třeba ověřovat již dříve fungující funkcionality CRCE systému (například ukládání OBR metadat) nebo EFFCC projektu (načítání EFP metadat z OSGi *bundlu* do datových instancí). Testovány byly pouze ty funkcionality, kde byl pro řešení problému implementován určitý nově navržený algoritmus nebo rozhodovací logika.

IndexerHandlerTest

Záměrem testu je ověřit, zda algoritmus pro zpřístupnění EFP metadat se chová podle očekávání pro různé vstupní artefakty.

Nejdříve je testován OSGi *bundle* s přiřazenými EFP metadaty dle aktuální verze XSD schématu. Zvolený artefakt prochází tímto testem s pozitivním výsledkem.

Dále je testován OSGi modul, který též obsahuje přiřazené EFP vlastnosti, avšak tyto metadata jsou přiřazena dle již neaktuální verze XSD schématu. U tohoto artefaktu se očekává selhání procesu načtení EFP metadat.

V závěru tohoto testu vstupuje do algoritmu artefakt, který není OSGi artefaktem. Očekává se opět neúspěšné načtení metadat. Tento chybový stav by správně při běhu CRCE systému neměl nastat, neboť zpracování artefaktů, které nejsou OSGi komponentou, končí již při úvodní podmínce.

Algoritmus indexeru v obou nepříznivých případech korektním způsobem zachycuje vzniklé výjimky a o jejich vzniku při reálném běhu CRCE systému informuje uživatele skrze webové rozhraní.

ResourceActionHandlerTest

Test si klade za cíl ověřit metodu *handleNewResource*() a nepřímo ověřuje také funkcionality, které jsou v rámci této metody volány a které představují funkcionality nižší úrovně. V rámci testu jsou vytvořeny dva Resource objekty. Jeden resource je vytvořen skrze testovanou metodu *handleNewResource*(), která vrací resource objekt se zaindexovanými EFP metadaty. Druhý resource objekt je vytvořen skrze metody crce-metadata-metafile modulu tak, že je jim na vstupu předán dříve vytvořený META soubor se zaindexovanými EFP vlastnostmi. Tento META soubor obsahuje EFP metadata zaindexované stávající implementací crce-efp-indexeru a představuje vzor, s nímž se mají shodovat zaindexovaná EFP metadata. Metody modulu crce-metadata-metafile přetransformují daný META soubor v *resource* objekt s OBR metadaty a dále v průběhu testu jsou porovnávány OBR metadata obou vytvořených objektů. Tímto způsobem se ověřuje, že stávající implementace zaindexuje ze vstupní

OSGi komponenty metadata ve stejném formátu a se stejným obsahem, který odpovídá dříve vzniklému vzoru vytvořeného ze stejné OSGi komponenty.

MetadataIndexingResultServiceImplTest

Tento test se zaměřuje na ověření implementace služby *MetadataIndexingResultService*, která je popsána v kapitole 7.1 - *Služba poskytující uživateli výsledek o běhu crce-efp-indexeru*. Test ověřuje, že vstupní řetězec odpovídá výstupnímu řetězci a že je službou možné předat více zpráv. Dále se ověřuje odstraňování položek seznamu zpráv a že služba správně odpovídá na dotaz, zda je seznam zpráv prázdný nebo nikoliv.

6.2 Parametry a omezení implementovaného řešení

6.2.1 Množství ukládaných OBR metadat

U prověřovaných modulů [OSGiB] z pohledu všech indexovaných metadat zabíraly mimofunkční EFP vlastnosti indexované crce-efp-indexer modulem v horní mezi přibližně 40% - 45% objemu metadat, přičemž objem všech zaindexovaných metadat (nejen EFP) k *bundlu* se pohybuje v počtu 3000 nebo 4000 znaků, což odpovídá velikosti 4 kB. Některé moduly mohou být pochopitelně výrazně menší a jiné výrazně větší. Při vyhodnocování této zjednodušené statistiky byly uvažovány moduly s relativně více EFP vlastnostmi, konkrétně komponenty *InventoryApplication OSGi Bundle*, *CardReaderController OSGi Bundle* a *InventoryData OSGi Bundle*.

Za zmínku také stojí, že indexovaná OBR metadata jsou v CRCE systému uložena dvakrát. Zdvojení se netýká OBR s názvem „package“ nebo „bundle“ - prakticky jsou tedy zdvojeny indexované EFP vlastnosti a *Capability* s názvem *„file*“ uchovávající původní název souboru. Jednou jsou tyto data uloženy v individuálním metadata souboru (*res_hascode.meta*), který náleží pouze příslušnému *bundlu* a podruhé v globálním metadata souboru repository.xml, který na jednom místě shromažďuje veškerá indexovaná metadata. Zmiňované soubory se nachází pod cestou crce-`root\runner\store\`. Zdvojení OBR metadat má ten význam, že lze realizovat změny metadat jednoho souboru bez nutnosti uzamykat přístup ke všem ostatním souborům v úložišti. Realizované změny se zapíší do globálního souboru repository.xml později, když systém neobsluhuje uživatelské požadavky.

Při růstu počtu mimofunkčních vlastností přiřazených ke komponentě roste lineárně i množství indexovaných OBR metadat.

6.2.2 Programové požadavky

K provozování CRCE systému včetně jeho rozšíření je doporučováno prostředí Java JDK 1.6 a sestavovací nástroj Maven verze 3. V rámci této práce byla ověřena funkčnost sestavování a spouštění i s dřívější verzí Maven 2.

Pro vývojové účely se osvědčilo vývojové prostředí *Eclipse* s *m2eclipse* pluginem. Při práci s SVN repository byly používány SVN klienti *RabbitVCS* pod systémem GNU/Linux a *TortoiseSVN* pod systémem Windows.

6.2.3 Nároky na paměť

Při běhu celého CRCE systému (tedy OSGi kontejneru a CRCE webového rozhraní) vykazuje každý ze dvou Java procesů využití přibližně 60 MB paměti pod systémy Windows i GNU/Linux.

Pro možnost alespoň přibližného určení nárůstu paměťové náročnosti CRCE systému při spuštění také EFP indexeru a jeho přidružených komponent bylo provedeno základní měření.

V první fázi měření byly zaznamenávány hodnoty využití paměti u obou procesů při opakovaném spuštění CRCE systému a uploadu několika *bundlů*. Průměrné hodnoty využití paměti Java procesů při tomto postupu dosahovaly 61 724 kB a 63 162 kB.

V druhé fázi byly z konfiguračních pom.xml souborů vyřazeny moduly *crce-efp-indexer* i *crce-efpAssignment* a také související balíky ze systémové proměnné *org.osgi.framework.system.packages.extra*. Bylo provedeno nové sestavení projektu a opět byl opakovaně startován CRCE systém a vykonány uploady identických *bundlů* z předešlého případu. Nyní průměrné hodnoty využití paměti Java procesů dosahovaly hodnot 61 258 kB a 54 912 kB.

Z výsledků tohoto improvizovaného měření je patrné, že nárůst využití paměti CRCE systému při provozování také *crce-efp-indexeru* a jeho závislostí představuje necelých 9 MB, což z pohledu celého systému představuje necelých 8 % paměti.

Paměťové nároky implementovaného řešení jsou tedy již i jednoduchým způsobem měřitelné, ale jejich přirovnáním k využívané paměti celého systému a také z pohledu zdrojů, které poskytuje běžný počítač, nepředstavují problém a jsou přijatelnou cenou za poskytovanou funkcionalitu napomáhající k vyhodnocování kompatibility komponent.

6.2.4 Časové nároky

Ovládání CRCE systému probíhá v reálném čase. Časově náročnější je pouze překlad a sestavení projektu a také spouštění projektu knihovnou Pax Runner. Neurčitý pojem „časově náročnější“ se odvíjí od rychlosti stroje, ale řádově se jedná až o jednotky minut.

Programově byly zaznamenány doby činnosti implementovaného modulu *crce-efp-indexer*. Pro vybrané programové komponenty [OSGiB] byly naměřeny průměrně následující časové hodnoty:

CardReaderController-1.0-SNAPSHOT.jar:	735 [ms]
CashDeskApplication-1.0-SNAPSHOT.jar:	744 [ms]
InventoryApplication-1.0-SNAPSHOT.jar:	784 [ms]
InventoryData-1.0-SNAPSHOT.jar:	533 [ms]

InventoryDatabase-1.0-SNAPSHOT.jar: 425 [ms]
InventoryGui-1.0-SNAPSHOT.jar: 384 [ms]
PrinterController-1.0-SNAPSHOT.jar: 94 [ms]
ScannerController-1.0-SNAPSHOT.jar: 663 [ms]

Z měření vyplývá, že indexer plní svoji funkci v čase menším než jedna sekunda. Tyto časové nároky lze z pohledu uživatelského ovládání CRCE úložiště považovat za akceptovatelné.

6.2.5 Jiná omezení

CRCE komponentové úložiště je možno provozovat na systémech, které splňují především programové požadavky.

Při počátečním sestavení a spuštění systému je velmi důležitá internetová konektivita. K provozování CRCE v offline režimu nestačí mít na lokálním disku pouze checkout CRCE z repository. V online režimu je také nutné spustit překlad a sestavení projektu, kdy Maven stahuje do lokální repository chybějící závislosti včetně modulů z EFFCC projektu. Až teprve po úspěšném spuštění CRCE knihovnou Pax Runner je možné přejít do offline režimu.

S tímto omezením je nutné počítat, plánujeme-li provádět vývojové práce na CRCE projektu v prostředí bez možnosti internetové konektivity.

6.3 Možnosti rozšíření

6.3.1 Reorganizace modulů v Maven repositáři

Oba CRCE i EFFCC projekty používají odlišné Maven repositáře, neboť tyto projekty vznikaly odděleně. V současné době, kdy došlo k propojení projektů, by bylo vhodné seskupit programové moduly využívané CRCE systémem do jednoho repositáře. Jedná se o všechny CRCE moduly, CRCE závislosti a závislosti implementovaných crce-efp modulů z EFFCC projektu .

CRCE používá Assembla Maven repositář. EFFCC projekt se odkazuje na Safe and Comprehensible software Components (SaCCo) Maven Repository [SaCCo].

Po rozšíření CRCE Maven odkazů o SaCCo Maven Repository se celková doba překladu a spuštění prodloužila, protože se Maven snaží hledat aktualizace na uvedených úložištích. Určitou dočasnou alternativou je po prvním úspěšném překladu a spuštění provádět další kroky překladu nebo spuštění s Maven přepínačem pro offline režim, tedy například

mvn -o install nebo *mvn -o pax:runner*

6.3.2 Hustší pokrytí implementovaných metod JUnit testy

Současné implementované JUnit testy pokrývají a ověřují klíčové funkcionality crce-efp-indexer modulu spíše ve stylu základních „smoke“ testů. Pro hustší pokrytí implementovaných metod jednotkovými JUnit testy by ale bylo možné usilovat o vytvoření samostatných testů i pro třídy *EFPAssignmentTypeResolver* a *EFPToOBRTranscription*. Tyto třídy jsou sice také nepřímo testovány nadřazeným *ResourceActionHandlerTestem*, ale testy cílené přímo a pouze na tyto třídy v současnosti chybí.

6.3.3 Ověření doby odezvy pro moduly s řádově odlišným množstvím metadat v manifest souborech

Zdroj [OSGiB] nabízí OSGi komponenty se vzájemně podobnou velikostí (v kB) a objemem přiřazených EFP dat. Test časové odezvy v kapitole 6.2.4 by vypovídal více, kdyby vycházel z porovnání několika modulů, jejichž velikost by se vzájemně lišila v řádech. Tím je na mysli řádově rozdílné množství metadat v manifest souborech než ve velikosti obsažených balíků. Z naměřených údajů by pak bylo možné sestavit graf vypovídající o nárůstu časové odezvy s ohledem na vyšší množství indexovaných metadat.

Pro účely těchto testů by bylo možné vytvářet OSGi komponenty s podvrženým manifest souborem a souborem repository.xml, který také uchovává přiřazená EFP metadata komponenty. Vytvoření podvržených testovacích modulů by spočívalo v takové úpravě manifest souboru, aby manifest vykazoval exportování a importování řádově velmi vysokého počtu balíků, které by k sobě vždy měli přiřazeny určité EFP vlastnosti. Generování seznamu balíků programově by nebylo náročné, neboť jednotlivé balíky se mohou lišit vždy i jen o jeden znak. Navíc podvržený OSGi modul nemusí tyto balíky skutečně obsahovat, neboť *bundle* by sloužil pouze k testování časové odezvy CRCE indexerů metadat.

Navíc lze během jednoho celého procesu indexace posbírat více časových značek mezi jednotlivými fázemi procesu a jejich porovnání by umožňovalo identifikovat úzká hrdla stávající implementace a vybízely by k výkonnostní optimalizaci kódu.

7 Další realizovaná rozšíření

V průběhu realizace *crce-efp-indexeru* vyvstala také potřeba informovat uživatele úložiště o výsledku indexace EFP metadat, což ve výsledku znamenalo implementovat rozšíření v modulech *crce-plugin-api* a *crce-webui*.

Dále bylo realizováno rozšíření *efpAssignment* modulu projektu EFFCC a byly v něm implementovány dvě metody pro poskytnutí doplňujících dat.

Řešení těchto potřeb jsou popsána v následujících dvou kapitolách.

7.1 Služba poskytující uživateli výsledek o běhu crce-efp-indexeru

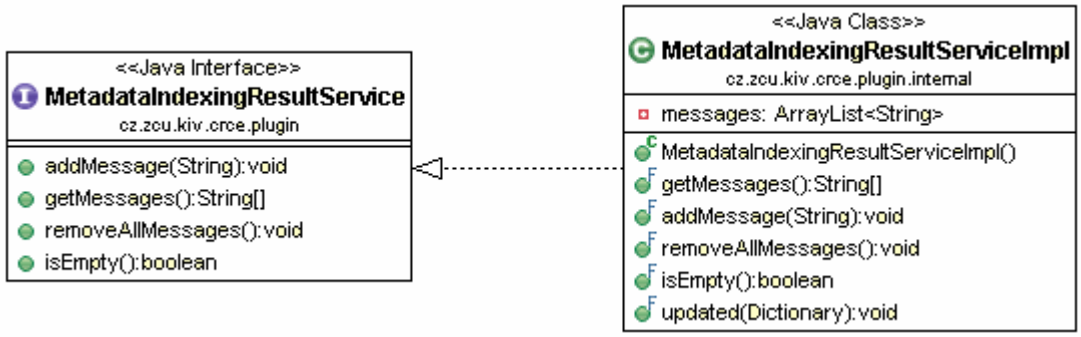
Snaha o zaindexování EFP metadat může skončit s více různými výsledky v závislosti na nahrávaném artefaktu a jeho obsahu. V ideálním případě jsou všechny nalezené EFP vlastnosti zaindexovány a uloženy. V některém *bundlu* se ale nemusejí nacházet žádné EFP vlastnosti, se kterými umí pracovat *efpAssignment* modul z EFFCC projektu. Též je možný výsledek, že bude do úložiště nahráván OSGi *bundle* s EFP vlastnostmi v neaktuální verzi (konkrétně, že EFP vlastnosti budou uloženy v XML souboru se zastaralou verzí XSD schématu). Vyvstává tedy potřeba informovat CRCE uživatele o výsledku indexování EFP metadat.

Dosud CRCE úložiště informovalo uživatele pouze o výsledku nahrání celého artefaktu do úložiště a tyto zprávy byly generovány v rámci modulu *crce-webui*, který představuje webové uživatelské rozhraní. Jiné moduly dosud neměli možnost propagovat své informativní zprávy pro uživatele do webového rozhraní. Tato potřeba vyvstává s existencí *crce-efp-indexeru*. Reakcí na tuto potřebu je implementace služby MetadataIndexingResultService.

Původně bylo rozhraní i jeho implementace umístněno přímo v *crce-efp-indexer* modulu, ale aby *crce-webui* mohlo volat metodu této služby, musela být v konfiguraci *crce-webui* uvedena závislost na modul *crce-efp-indexer*. Při sestavování a spouštění všech modulů dohromady vše fungovalo v pořádku, ale byl-li z procesu spouštění v kontejneru vyřazen *crce-efp-indexer*, nedokázal se nastartovat ani *crce-webui* modul, neboť v *Import-Package* obsahoval odkaz na balík s touto službou v modulu EFP indexeru. Tato nová závislost na provozování *crce-webui* modulu se jeví jako nežádoucí, a proto byla tato služba informující o výsledku *crce-efp-indexeru* přesunuta do *crce-plugin-api* modulu a implementována obecněji, aby ji případně mohly využít později i další moduly pro indexování metadat.

7.1.1 Implementace služby

Obecné rozhraní i implementace služby se nacházejí v *crce-plugin-api* modulu. Obr. 3 s diagramem tříd zachycuje jednotlivé metody služby včetně atributů a návratových hodnot.



Obr. 3: Diagram tříd služby MetadataIndexingResultService.

MetadataIndexingResultService

Rozhraní definuje obslužné metody. Obecný návrh počítá s možností vložit řetězec zprávy pro uživatele. Protože může být služba využívána i jiným indexerem metadat, je navržena možnost opakovaného přidání více zpráv. Pro cílový modul je navržena možnost dotazování na pole řetězců zpráv a možnost jejich vymazání poté, kdy jsou zprávy vyzvednuty. Rozhraní zahrnuje také dotaz, zda je seznam zpráv prázdný.

Pro doplnění zde lze zmínit, že crce-efp-indexer modul využívá metody *addMessage*() k zapsání předávané informace a cílový modul crce-webui pro vyzvednutí zprávy používá metody *isEmpty*(), *getMessages*() a poté seznam zpráv maže a připravuje pro další použití metodou *resetMessages*().

MetadataIndexingResultServiceImpl

Třída implementuje rozhraní *MetadataIndexingResultService* a splňuje výše uvedené principy jednotlivých metod definovaných v rozhraní. Řetězce zpráv jsou uchovávány ve struktuře ArrayList.

Implementace většiny metod je velmi jednoduchá díky použití standardních metod struktury ArrayList. Pouze u metody *getMessages*():String[] bylo třeba napsat vlastní převod seznamu na pole řetězců, neboť pole získané metodou *toArray*():Object[] nelze přetypovat na pole řetězců.

7.2 Implementované metody v EfpAssignment

Součástí provedených úprav bylo přidání dvou doplňujících metod do rozhraní `cz.zcu.kiv.efps.assignment.values.EfpAssignedValue` a implementace těchto metod ve třídách, které implementují zmíněné rozhraní.

7.2.1 Metoda getValueName

Některé hodnoty mimofunkčních vlastností jako například `["EAN-13","EAN-8"]` postrádají pro nezasvěcené vypovídající hodnotu a při větším množství těchto technických údajů se stávají snadno nepřehledné. Naštěstí bývají tyto hodnoty pro snazší přehlednost doplněny atributem `valueName`, který obsahuje pro člověka čitelnější a přehlednější zobecňující název. Součástí úkolu je indexovat EFP vlastnosti a jejich atributy v lidsky čitelné podobě, a proto je tento atribut zpřístupněn a též indexován.

Implementovaná metoda *getValueName* vrací v textovém řetězci hodnotu zmíněného atributu `valueName`. Pro ilustraci jsou uvedeny data pochází ze souboru `repository.xml` z OSGi modulu *InventoryDatabase* [OSGiB].

```
<lr id="463" name="CashDesk">
            <simpleAssignment id="499" efp_id="472" valueName="europe">
                  <efpValue>{"efpEnum":["EAN-13","EAN-8"]}</efpValue>
            </simpleAssignment>
</lr>
```

Při volání metody *getValueName*() nad instancí třídy *EfpNamedValue* implementující rozhraní *EfpAssignedValue* získáme z výše uvedených dat řetězec „europe“.

V případě EFP vlastností, kde atribut valueName nemá smysl, vrací tato metoda `null` hodnotu. Tento případ se týká instancí tříd *EfpDirectValue* a *EfpFormulaValue*.

7.2.2 Metoda getAssignmentType

Tato metoda reaguje na potřebu rozlišení, zda je aktuálně zpracovávaná EFP vlastnost instancí třídy *EfpNamedValue, EfpDirectValue*, nebo *EfpFormulaValue*. Od příslušnosti EFP hodnoty k jedné z vyjmenovaných tříd se odvíjí, jaké datové atributy budou k dané mimofunkční vlastnosti indexovány.

Pro možnost snadného rozlišení instancí byl do rozhraní připsán veřejně viditelný výčtový typ *AssignmentType*, který může nabývat hodnot: {*named*, *direct*, *formula* }.

Metoda *getAssignmentType* vrací jeden z prvků výčtového typu *AssignmentType*. Na základě této znalosti lze rozlišit, jaké datové atributy jsou pro danou EFP vlastnost relevantní.

8 Závěr

Cílem práce bylo prozkoumat možnosti přepisu mimofunkčních charakteristik do OBR formátu a realizovat programový modul, který by umožnil zaindexování mimofunkčních charakteristik OSGi komponent v požadovaném OBR formátu. Indexovaná metadata bude možné využít při vyhodnocování vzájemné kompatibility komponent.

Počátečním úkolem bylo seznámit se s aplikacemi EFP Assignment a CRCE úložištěm do té míry, aby bylo možné na úrovni zdrojového kódu využívat metody obou systémů pro požadované účely.

Dalším úkolem bylo zaměřit se na propojení obou systémů tak, aby v CRCE úložišti bylo možné provozovat funkční kód EFFCC projektu.

Byly navrhnuty formáty přepisu EFP vlastností do OBR metadat. Jsou zhodnoceny jejich klady a nevýhody s ohledem na současné požadavky zadavatele a jeden vyhovující formát byl zvolen pro implementaci.

Dále byly navrženy a implementovány dva OSGi moduly, které jsou nyní součástí CRCE systému. Modul *crce-efp-indexer* obsahuje implementaci zvoleného formátu indexování metadat i obslužné třídy a metody zajišťující proces indexování. Druhý modul *crce-efpAssignment* představuje knihovní modul zapouzdřující v sobě balíky z EFFCC projektu, které umožňují načítat EFP metadata z OSGi modulů.

Zdrojový kód zahrnuje dokumentační komentáře a bylo usilováno o dodržování formátu checkstyle. V používaném vývojovém repositáři je zajištěn pro ostatní CRCE vývojáře a uživatele návod související s provozem celého CRCE úložiště spolu s implementovaným *crce-efp-indexer* modulem.

Byla ověřena stabilita kódu a je zajištěno, aby byly zachyceny možné výjimky související se zpracováním vstupního artefaktu. Ke vzniklým výjimkám je propagováno odpovídající chybové/informativní hlášení do logu CRCE systému a o zásadních výjimkách, ale také o výsledku indexování EFP metadat je uživatel informován skrze webové rozhraní systému.

V kapitole 6.2 jsou zhodnoceny parametry a omezení implementovaného řešení. Časové i paměťové nároky jsou vyhovující a z pohledu zdrojů, které poskytuje běžný počítač, představují přijatelnou cenu za poskytovanou funkcionalitu napomáhající při vyhodnocování kompatibility komponent.

Přehled zkratek

API (Aplication Programming Interface)

Rozhraní, které umožňuje programátorovi pracovat s aplikací druhé strany skrze metody poskytnuté tímto rozhraním.

CoSi (Components Simplified)

CoSi je komponentový model inspirovaný modelem OSGi. Je psán též v jazyce Java. Model je používán při výzkumu nahraditelnosti komponent na Katedře informatiky a výpočetní techniky na Západočeské univerzitě v Plzni [12].

CRCE (Component Repository supporting Compatibility Evaluation)

Komponentové úložiště orientované na kontrolu kompatibility uložených programových komponent.

EFP (Extra-Functional Property)

Prostředek k popisu mimofunkčních vlastností programové komponenty a zároveň lze pomocí těchto metadat vyhodnocovat vzájemnou kompatibilitu programových komponent.

EFFCC (Extra Functional Property Featured Compatibility Checks)

Projekt zahrnuje soustavu modulů zabývajících se mimofunkčními charakteristikami programových komponent a ověřováním jejich mimofunkční kompatibility. Moduly pracují s komponentami frameworků OSGi a CoSi.

OBR (OSGi Bundle Repository)

Specifikace pro popis souborů v komponentovém úložišti. Tato specifikace je určena dokumentem OSGi RFC 112.

OSGi (Open Services Gateway initiative)

Jedná se o modulární systém pro jazyk Java. Moduly aplikace lze instalovat, spouštět i stopovat za běhu kontejneru a při tom procházejí tzv. životními cykly. Vlastnosti modulu jsou specifikovány manifest souborem.

SaCCo (Safe and Comprehensible software Components)

V rámci tohoto projektu je používán odkazovaný SaCCo Maven Repository k ukládání release verzí, snapshotů a knihoven třetích stran. Na tento repositář je odkazován nástroj Maven při sestavování modulů EFFCC projektu.

Citovaná Literatura a informační zdroje

[1] KOHOUT, Josef. *Programování a užití komponent*. Pomocný učební text pro studenty předmětu KIV/PUK. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2010.

[2] SZYPERSKI, Clemens. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley a ACM Press, 1998. ISBN 0-201-17888-5.

[3] JEŽEK, Kamil., BRADA, Přemysl. *Correct matching of components with extra-functional properties*. ENASE 2011. Lisabon: SciTePress, 2011. s. 155-166. ISBN 978-989-8425-65-2.

[4] KUČERA, Jiří. *Úložiště komponent podporující kontroly kompatibility*. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2011.

[5] ŠTULC, Martin. *Uživatelské rozhraní pro úložiště mimofunkčních charakteristik*. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2011.

[6] ŠVÁB, Jan. *Obecná reprezentace mimofunkčních charakteristik na komponentách*. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2011.

[7] VLČEK, Lukáš. *Vyhodnocování mimofunkčních charakteristik na komponentách*. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2011.

[8] ŠNAJBERK, Jaroslav. *Úvod do komponent, OSGi*.
[online]. URL: <http://wiki.kiv.zcu.cz/UvodDoKomponent/OSGi>.
[aktualizováno 2011-03-28], [cit. 2012-03-04].

[9] HALL, Richard S. *RFC-0112 Bundle Repository*. OSGi Alliance.
[online]. URL: <http://www.osgi.org/download/rfc-0112_BundleRepository.pdf>.
[2006-02-23], [cit. 2012-01-14].

[10] *Apache Felix OSGi Bundle Repository (OBR)*. The Apache Software Foundation.
[online]. URL: <http://felix.apache.org/site/apache-felix-osgi-bundle-repository.html>.
[aktualizováno 2009-12-04], [cit. 2012-01-14].

[11] *Metadata*. The Tech Terms Computer Dictionary.
[online]. URL: <http://www.techterms.com/definition/metadata>. [cit. 2012-03-26].

[12] LIŠKA, Vojtěch. *Pokročilé aspekty komponentových modelů*. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, 2009.

Příloha A – Odkazované datové zdroje

[CRCE] Component Repository supporting Compatibility Evaluation.
[online]. URL: <http://www.assembla.com/spaces/crce/>.

[EFFCC] Extra Functional Property Featured Compatibility Checks.
[online]. URL: <http://www.assembla.com/spaces/efps/ >.

[JAXP] Apache ServiceMix Bundles. MVN Repository.
[online]. URL: <http://svn.apache.org/viewvc/servicemix/smx4/bundles/tags/org.apache.servicemix.bundles.jaxp-ri-1.4.2_1>. [cit. 2012-01-15].

[OSGiB] Vybrané OSGi *bundly* s přiřazenými EFP a používané v EFFCC.
[online]. URL: <http://www.assembla.com/code/efps/subversion/nodes/bundles/osgi/cashdesk/Bundles>.

[SaCCo] SaCCo Maven Repository.
[online]. URL: <http://gforge.kiv.zcu.cz/gf/project/sacco-mvn-repo/>.

Příloha B – Instalační a uživatelský manuál

B.1 Nastavení přístupu na SaCCo repository pro Maven

Používáme-li CRCE úložiště spolu s implementovaným *crce-efp-indexer* modulem, je nutné nastavit sestavovacímu nástroji Maven přístup k Safe and Comprehensible software Components (SaCCo) Maven Repository [SaCCo], kde se nachází přeložené balíky projektu EFFCC. Maven přibaluje do *bundlu* indexeru konkrétně moduly *efpAssignment*, *efpAssignmentOSGi* a *efpTypes*.

Podrobný návod na zpřístupnění SaCCo repository pro Maven byl též v rámci této práce uveden na webovém prostoru [CRCE] na stránce „*Running CRCE with a module crce-efp-indexer*“. Návod je inspirován obsahem podobné stránky „*Set up environment*“, která je určena vývojářům projektu EFFCC a která se nachází na webovém prostoru [EFFCC].

Nastavení přístupu spočívá v nalezení Maven konfiguračního souboru `settings.xml`, který bývá v Linux systému pod cestou

`/etc/maven2/settings.xml`

a v OS Windows pod cestou

… `\maven-x.x.x\conf\settings.xml`

V tomto souboru je třeba nalézt blok `<servers>` … `</servers>` a vložit do něho přihlašovací údaje k úložišti uvedené na výše uvedené webové stránce.

Aktuální podoba těchto údajů je:

```
<server>
    <id>sacco.maven.repository</id>
    <username>anonymous</username>
    <password>anonymous</password>
</server>
<server>
    <id>sacco.maven.snapshots.repository</id>
    <username>anonymous</username>
    <password>anonymous</password>
</server>
<server>
    <id>sacco.maven.third-parties.repository</id>
    <username>anonymous</username>
    <password>anonymous</password>
</server>
```

Dále je možné pokračovat překladem CRCE projektu.

B.2 Překlad modulů CRCE

Po provedení checkoutu z CRCE repository a nastavení přístupu na EFFCC úložiště lze spustit překlad a sestavení celého CRCE projektu. Tyto kroky je možné vykonat buildovacím nástrojem Maven.

Překlad projektu se spustí zadáním příkazu

```
mvn install
```

do příkazové řádky, ve které máme otevřené umístnění překládaného projektu a nacházíme se na stejné úrovni, jako je kořenový konfigurační pom.xml soubor.

Na zadání příkazu `mvn install` se spouští s překladem současně také automatické testy, které bývají součástí sestavovaných modulů. Provádíme-li sestavení modulů opakovaně jen s malými změnami, nebo máme-li jiný důvod pro přeskočení testů, lze vykonávání testů vynechat modifikací příkazu do tvaru

```
mvn install –Dmaven.test.skip
```

Po vykonání překladu a sestavení modulů je úspěšné dokončení prováděných operací oznámeno informační zprávou „BUILD SUCCESSFUL“ s údaji o uplynutém čase a paměťovými nároky.

B.3 Spuštění CRCE úložiště

Sestavený projekt lze následně spustit zadáním příkazu

```
mvn pax:provision
```

Spuštění projektu zajišťuje knihovna Pax Runner. Po zavedení jednotlivých modulů je do konzole vypsána zpráva *„Runner has successfully finished his job!“* a dále je možné přistoupit k ovládání aplikace nebo vlastního OSGi kontejneru.

Pokud se nepodařilo některé moduly spustit, je k příslušnému modulu vypsána informativní chybová hláška, která zpravidla upozorňuje na chybějící závislosti. Přehled o zavedených modulech lze získat příkazem „`lb`“ viz další kapitola.

B.4 Ovládání OSGi kontejneru

Bude zde zmíněno několik základních příkazů, které byly při práci v konzoli opakovaně používány. Příkazy vyhodnocuje interpretr OSGi kontejneru.

`lb` – Příkaz vypíše seznam zavedených modulů a stav jejich životnosti. Vhodné pro ověření úspěšného zavedení všech modulů. V současné době je po spuštění projektu zavedeno necelých šedesát modulů.

`install file:``/cesta-k-osgi-bundlu/bundle.jar` – nainstalování modulu do komponentového OSGi kontejneru.

`uninstall` *`id_modulu`* – odinstalování a vymazání modulu z běžícího kontejneru.

`start` *`id_modulu`* – spuštění modulu, který je buď ve stavu *Resolved* nebo ve stavu *Installed*. Pro úspěšné spuštění musí být dostupné všechny jeho požadované závislosti.

`stop` *`id_modulu`* – zastavení činnosti aktivního modulu.

Informace o životním cyklu modulů v kontejneru a stručnou charakteristiku jednotlivých stavů modulu lze nalézt ve zdroji [8].

B.6 Zpřístupnění CRCE úložiště skrze webové rozhraní

Po dokončení spuštění projektu je možné zadat do adresní řádky webového prohlížeče adresu pro uživatelské webové ovládání CRCE aplikace:

`<http://localhost:8080/crce>`

Zobrazení podrobných informací o zavedených modulech v kontejneru lze získat přes záložku Bundles webového rozhraní Apache Felix, které je přístupné na webové adrese `<http://localhost:8080/system/console/>`. Přihlašovací údaje do konzole jsou uživatelské jméno: `admin`, heslo: `admin`.

B.6 Uložení OSGi modulu s EFP vlastnostmi do CRCE

Pro zobrazení výsledků indexování mimofunkčních vlastností je třeba mít k dispozici OSGi komponentu s přiřazenými EFP vlastnostmi. Tyto moduly je možné stáhnout z EFFCC úložiště v sekci [OSGiB]. Uložení *bundlu* do CRCE úložiště lze provést skrze CRCE webové rozhraní na stránce pro Upload. Vybereme artefakt a potvrdíme jeho upload. Měli bychom obdržet zprávu ve smyslu „*Upload was succesful. crce-efp-indexer: EFP metadata were succesfully saved.*“ Rozkliknutím uloženého artefaktu zobrazíme jeho podrobné informace a v případě, že měl přiřazeny EFP vlastnosti podporované verze, budou tyto EFP metadata též zaindexovány v seznamu metadat.

B.7 Překlad modulů projektu EFFCC

Podobně, jako byly během této práce do modulu efpAssignment přidány metody getValueName a getAssignmentType, mohou v budoucnu vyvstat požadavky na implementaci dalších podpůrných metod.

V této souvislosti je možné zmínit praktickou zkušenost, že moduly projektu EFFCC lze v pořádku překládat Maven verzí 2, ale naopak překlad verzí 3 byl opakovaně neúspěšný. Tyto zkušenosti potvrzuje také aktualizovaná informace na stránce „*Set up environment*“ projektu [EFFCC].