MASARYKOVA UNIVERZITA V BRNĚ
FAKULTA INFORMATIKY

API pro tvorbu otázek a testů

DIPLOMOVÁ PRÁCE

Bc. Martin Tomola

Brno, jaro 2005

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Tomáš Pitner, Ph.D.

Poděkování

Rád bych poděkoval RNDr. Tomáši Pitnerovi, Ph.D., vedoucímu mé diplomové práce, za vstřícnost a ochotu vést tuto práci. Dále děkuji svým rodičům za podporu a trpělivost, kterou se mnou měli během tvorby této práce a celého studia. Mé díky patří také kolegovi Bc. Tomáši Udržalovi za pomoc a podnětné rady.

Zvláštní poděkování patří mým oblíbeným hudebním labelům, hlavně pak labelům Warp, Skam a Rephlex, za hudbu, kterou vydávají.

Shrnutí

V dnešní době informační společnosti se počítače stále více využívají pro testování vědomostí. Značnou pozornost je nutné věnovat formátu uložení testových materiálů. Je jistě možné zvolit pro jejich reprezentaci proprietární formát. Ale raději než ten je lepší využít nějaký již existující, široce rozšířený a používaný a standardizovaný. Odměnou za tuto volbu bude snadná interoperabilita uložených testových materiálů napříč různými systémy, které zvolený formát podporují. Tato práce si klade za cíl vytvořit aplikační programové rozhraní pro práci s jedním z existujících formátů testových materiálů – formátem podle specifikace Question & Test Interoperability organizace IMS. Pro realizaci rozhraní byl použit programovací jazyk Java.

Klíčová slova

otázky, testy, elektronická podpora výuky, interoperabilita, IMS, QTI, API, XML, transformace, Bronchus

Obsah

Kapitola 1

Úvod

V současné době zažíváme rozmach elektronické podpory výuky. Její významnou součást představuje možnost testování znalostí pomocí počítače. S tím vyvstává otázka vhodného nástroje pro tvorbu testů a otázek a také vhodného formátu pro jejich uložení.

Tato práce se zabývá formátem testových dat podle specifikace IMS Question & Test Interoperability a vytvořením aplikačního programového rozhraní (API), které by umožňovalo s tímto formátem pracovat. Práce také popisuje i konkrétní použití vytvořeného API v případě, kdy bylo využito k realizaci vstupních a výstupních operací pro aplikaci pro tvorbu testových materiálů.

Následující text je rozdělen do několika částí. První část (kapitoly 2 a 3) slouží k uvedení do problematiky. Obsahuje obecné popisy existujících standardů v oblasti elektronické podpory výuky a technologií použitých při tvorbě API. V další části (kapitola 4) je detailněji popsána specifikace IMS Question & Test Interoperability. Pátá kapitola se zabývá tvorbou API pro tuto specifikaci a šestá pak použitím API pro implementaci vstupních a výstupních operací.

V práci se několikrát objevují termíny standard a specifikace. Termín standard označuje všeobecně uznávaný postup, který je zaštítěn některou z uznávaných standardizačních organizací. Protože je proces schválení standardu poměrně zdlouhavý, objevují se postupy zvané specifikace. Ty vytvářejí organizace, které nemají potřebnou váhu pro prohlášení postupu standardem, nicméně umožňují svými specifikacemi pružně reagovat na aktuální potřeby. Pokud je specifikace široce podporována, může být uznána standardizační organizací jako standard. Pro jednoduchost nebude mezi těmito termíny dělán rozdíl.

Kapitola 2

Existující standardy

V minulosti se ukázalo, jak důležitou roli hraje přijetí standardů pro široké rozšíření nějaké technologie. Příkladem může být obrovský rozvoj Internetu po zavedení standardů TCP/IP, HTTP či HTML. Nejinak je tomu i při tvorbě testových materiálů, kde má formát jejich uložení zásadní význam. Je zřejmé, že použití proprietárního formátu omezuje možnost jednoduché výměny zaznamenaných dat. Tím přispívá k uzavřenosti celého systému, který s testovými materiály pracuje. Naopak dodržení standardizovaného formátu nám zajistí, že data budou snadno přístupná, nezkreslená a budeme mít zaručenu interoperabilitu se systémy dodržujícími stejný standard jako my.

Co se vlastně požaduje od takového standardu? Především by měl umět odpovědět na následující otázky:

- Jak prolínat testová data získaná z různých zdrojů?
- Jak vytvořit snadno vyměnitelný obsah, který bude moci být rychle a jednoduše uložen, načten a opětovně použit?
- Jak zajistit, že nebudeme omezováni proprietárním formátem?

V současné době existuje pro oblast elektronické podpory výuky standardů několik. Přiblížíme si ty, které mají v této oblasti největší váhu.

2.1 LTSC LOM

Standard LOM (Learning Objects Metadata)[1] vytvořila pracovní skupina LTSC (Learning Technology Standards Committee)[2] spadající pod IEEE (Institute of Electrical and Electronics Engineers, Inc.)[3]. Jedná se o první standard v oblasti elektronické podpory výuky. Specifikuje způsob, jak systematicky a konzistentně popsat výukové objekty, aby se daly jednoduše sdílet, vyměňovat atd. Popisuje je pomocí kódování, kterým může být například značkování pomocí jazyka XML. Je jedním z velmi často používaných standardů – převzaly jej například společnosti IMS a ADL (viz. dále). LTSC koordinuje svoji práci na vývoji

1. <http://ltsc.ieee.org/wg12/20020612-Final-LOM-Draft.html>
2. <http://ltsc.ieee.org/>
3. <http://www.ieee.org/>

standardů i s dalšími organizacemi, mimo jiné s ISO/IEC (International Organization for Standardization/International Electrotechnical Commission)[4].

2.2 AICC

Profesní organizace AICC (Aviation Industry CBT Committee)[5] vytvořila soubor pravidel vztahujících se ke školení pomocí počítače. Pravidla přesně popisují například jak vyvinout systém, aby umožňoval interoperabilitu, jaký použít hardware pro daný systém nebo jak má konkrétně vypadat uživatelské rozhraní. AICC si stanovila tyto cíle:

- Pomoci leteckým společnostem ve vývoji pravidel, která pomohou efektivní implementaci školení pomocí počítače.
- Vytvořit pravidla umožňující lepší interoperabilitu.
- Poskytnout otevřené fórum pro diskuzi o školení pomocí počítače a dalších školících technologiích.

Ačkoli původním zaměřením AICC byl letecký průmysl, pravidla jsou tvořena dostatečně obecně a dají se aplikovat i mimo letectví. Díky tomu byla postupem času tato pravidla přijata i dalšími společnostmi a AICC je dnes široce respektovaným standardem. Systém, vytvořený ve shodě s pravidly AICC, může po posouzení nezávislou testovací laboratoří (AICC Independent Test Lab) získat označení AICC-Certified.

2.3 IMS

Společnost IMS Global Learning Consortium[6] je nevýdělečná organizace vyvíjející otevřené a platformově nezávislé specifikace pro oblast elektronické podpory výuky. V současnosti má přes 50 přispívajících členů, kteří zahrnují akademické, komerční a vládní organizace. Klíčové záměry IMS jsou:

- definovat technické standardy pro interoperabilitu aplikací a služeb pro distanční vzdělávání a
- propagovat celosvětové začlenění IMS specifikací do produktů a služeb a vytvořit tak prostředí různých kooperujících výukových systémů

IMS specifikace jsou založeny na jazyku XML. V současné době existují například již tyto specifikace:

4. <http://www.iso.org> a <http://www.iec.ch/>
5. <http://www.aicc.org/>
6. <http://www.imsglobal.org/>

Content Packaging – tvorba standardizovaných „balíčků“, které obsahují výukové objekty a jimi odkazované soubory, spolu s instrukcemi pro výukové systémy, jak objekty organizovat.

Learning Resource Metadata – převzetí standardu LOM pro popis výukových objektů.

Question & Test Interoperability – specifikace XML formátu pro kódování otázek a testů.

Learner Information Packaging – definice XML struktur pro výměnu informací o studentovi mezi kooperujícími systémy.

Accessibility – rady, doporučení a pravidla pro zpřístupnění výukových materiálů i zdravotně postiženým.

Digital Repository – specifikace pro interoperabilitu digitálních informačních zdrojů.

Learning Design – struktura pro popis návrhu výukového procesu formálním způsobem.

Několik IMS specifikací je dnes celosvětově bráno jako de facto standard. Specifikací IMS Question & Test Interoperability se detailněji zabývá kapitola Specifikace IMS QTI.

2.4 ADL SCORM

Za zřízením iniciativy ADL (Advanced Distributed Learning)[7] americkým ministerstvem obrany stojí snaha o propojení akademických a průmyslových standardů a vytvoření nového prostředí, ve kterém bude fungovat interoperabilita na globální úrovni. Mezi její cíle patří:

- Založení komunity, která bude chápat nutnost vytvoření společných standardů.
- Vytvářet a implementovat návody pro efektivní distribuované vzdělávání.
- Sdílet výukové materiály a urychlit vytvoření robustního, vysoce strukturovaného, objektově orientovaného prostředí pro ADL.
- Označit a podporovat obchodní modely a ekonomické pobídky na podporu uživatelů a poskytovatelů distribuovaného vzdělávání.

7. <http://www.adlnet.org/>

- Stimulovat rozvoj spolupráce mezi obdobnými vzdělávacími institucemi.

Jedním z výsledků této snahy je vytvoření souboru pravidel SCORM (Shareable Courseware Object Reference Model)[8]. Je to referenční model definující vzájemný vztah mezi součástmi kurzů, datovými modely a protokoly. Učební materiály pak mohou být sdíleny mezi systémy, které jsou s tímto modelem v souladu. Obsahuje například všechny dosud zmíněné specifikace. Systém vyhovující pravidlům SCORM, bude splňovat:

Přístupnost – bude schopen zpřístupnit výukové objekty z různých vzdálených úložišť použitím jejich metadat.

Přizpůsobivost – bude schopen přizpůsobit se potřebám jedince nebo celé organizace.

Trvalost – bude schopen odolávat technologickým změnám.

Interoperabilitu – bude schopen používat výukové objekty vytvořené různými autorskými nástroji na různých platformách.

Znovupoužitelnost – bude schopen využít výukové objekty ve více aplikacích v různém kontextu.

8. <http://www.adlnet.org/scorm/index.cfm>

Kapitola 3

Použité technologie

3.1 XML

Je mnoho způsobů jak v informačním prostředí použít značkovací jazyk XML (eXtensible Markup Language)[1]. Jedním z nich je použít jej jako formát pro ukládání dat. Tato volba nám přinese některé výhody. Předně, data budeme moci nejen uložit, ale XML umožní i popis logického obsahu ukládaných dat. Tedy dovolí nám přesně popsat strukturu a význam těchto dat pomocí elementů a atributů. Díky tomu je snazší takto uchovaná data prohledávat a upravovat. Jelikož se jedná o univerzální formát, nezávislý na architektuře či operačním systému, uložená data budou přístupná i v rámci různých platforem. Mezi další výhody tohoto formátu patří:

Standard – odpadává omezenost proprietárních formátů, v dnešní době je široce rozšířen a používán.

Otevřenost – jedná se o volně dostupný formát, jehož specifikace je každému zdarma k dispozici.

Flexibilita – umožňuje popsat data jakéhokoliv typu.

Striktnost – pevně daná pravidla struktury, jejichž dodržení se dá prověřit parserem.

Založenost na textu – ačkoli je určen pro strojové zpracování, je snadno čitelný a upravitelný i v obyčejném textovém editoru.

Mezinárodní podpora – podporuje kódování znaků v UNICODE[2], čímž odpadají problémy s přechodem mezi různými jazykovými prostředími.

1. <http://www.w3.org/XML/>
2. <http://www.unicode.org/>

Uložení v XML má samozřejmě i své nevýhody. Tou může být větší výsledná velikost uloženého souboru, což je důsledek „upovídanosti“ XML. Je ji však možné redukovat vhodnou kompresí. Dále zpracování takového souboru může být procesorově a paměťově náročné, zvláště u velkých objemů dat. Obecně se ale dá říci, že výhody XML formátu jednoznačně převažují.

3.2 Java

Programovací jazyk Java vyvinula společnost Sun Microsystems, Inc[3]. Jedná se o objektově orientovaný programovací jazyk. Syntakticky je shodný s jazyky C a C++, nicméně není ani na jednom z nich založený. Vznikl jako náhrada jazyka C++, který se ukázal nevhodným pro některé problémy.

Jedním z nejdůležitějších rysů Javy je platformová nezávislost a z toho vyplývající snadná přenositelnost mezi různými systémy. Snadno přenositelný je nejen zdrojový kód, ale i binární tvar kódu. To umožňuje virtuální stroj JVM (Java Virtual Machine), což je vlastně jakýsi abstraktní počítač. Java je totiž, na rozdíl od zmiňovaného C++, interpretovaný jazyk a při překladu zdrojového kódu se vytvoří tzv. *bytecode*. Ten pak interpretuje JVM, který vytvoří rovnocenné prostředí pro programy, nezávislé na konkrétně použitém hardware a operačním systému. Jediným požadavkem pro běh programů v Javě je tedy existence JVM pro příslušnou platformu. Další vlastnosti Javy jsou:

- Podpora mechanismu vláken (*multithreading*).
- Automatické přidělování a uvolňování paměti.
- Implementace mechanismu výjimek, takže veškeré chyby vzniklé za běhu programu je možné odchytit a zpracovat.
- Implementace bezpečnostních mechanismů, jako například podepisování kódu nebo přidělování práv pro různé akce.
- Možnost *serializace* vytvořených objektů, tj. lze je ukládat do souboru, zasílat po síti atd.
- Přímá podpora sítě a tím splnění základní podmínky pro tvorbu distribuovaných aplikací.

S Javou jsou také dodávány standardní knihovny, s jejichž obsáhlostí se asi nemůže srovnávat žádný běžně používaný jazyk. K dispozici jsou knihovny pro tvorbu grafického uživatelského rozhraní, vstup/výstup, práci s textem, komunikaci s SQL databázemi, práci s komprimovanými soubory a mnoho dalších.

3. <http://www.sun.com/>

3.3 dom4j

Pro XML existuje abstraktní (tj. platformově neutrální a na programovacím jazyku nezávislé) API, umožňující programům přistupovat, číst a aktualizovat jeho obsah a strukturu. Je jím DOM (Document Object Model)[4] a specifikovalo ho konsorcium W3C[5]. Strukturu XML lze přirozeně modelovat pomocí stromu a DOM právě s touto reprezentací XML pracuje. DOM definuje objektový model XML, rozděluje XML na jednotlivé objekty, které odpovídají uzlům v XML stromu – elementy, atributy, textový obsah atd. Každý takový objekt nabízí metody pro práci s ním – například na zjištění jeho typu, hodnoty nebo potomků.

Pro prostředí Javy existuje několik implementací tohoto rozhraní. Jedním z nich je dom4j[6], řešení od Jamese Strachana. Kromě DOM podporuje také práci se SAX (Simple API for XML), rozhraním sloužícím výlučně pro čtení XML souboru. Pro práci s XML souborem používá dom4j standardní javovský soubor knihoven Java Collections, ale také dovoluje programátorovi použít i jiné řešení, které si sám navrhne. Jeho výhodou je poměrně dobrá rychlost načítání XML souboru.

3.4 ANTLR

ANTLR (ANother Tool for Language Recognition)[7] je parser a jazykový nástroj. Jeho autorem je profesor Terence Parr z univerzity San Francisco. ANTLR umožňuje vytváření rozpoznávačů textu, překladačů a převaděčů z jednoho formátu do druhého pomocí gramatických předpisů. Vytváří je ve zdrojových kódech programovacích jazyků Java, C#, C++ nebo Python. Do svých gramatik umožňuje vkládat i prvky těchto jazyků.

Gramatiky pro ANTLR se skládají ze tří částí, přičemž každou část je možné vynechat. Tyto části jsou *Lexer*, *Parser* a *TreeParser* a gramatická pravidla určená pro jejich tvorbu jsou prakticky totožná. Lexer slouží pro logické rozdělení znaků vstupního textu do různých skupin, odpovídajícím definici v gramatice. Příkladem takové skupiny může být skupina *CISLICE* sdružující číslice ze vstupního textu nebo *OPERATOR* sdružující například matematické operátory. Vytváří tím jakýsi slovník pro Parser. Zároveň provádí lexikální analýzu vstupního textu, čili zda se v něm vyskytují pouze znaky povolené gramatikou. Parser kontroluje syntaxi vstupního textu, tedy jestli odpovídá pravidlům zadané gramatiky. Gramatika pro Parser může také obsahovat instrukce pro tvorbu parsovacího stromu, který lze posléze využít TreeParserem pro převod vstupního textu z jednoho formátu do jiného.

4. <http://www.w3.org/DOM/>
5. <http://www.w3.org/>
6. <http://www.dom4j.org/>
7. <http://www.antlr.org/>

Kapitola 4

Specifikace IMS QTI

Jak již bylo řečeno, při tvorbě testů a otázek je dobré vycházet z existujících standardů. Pak budeme mít zaručeno, že pro náš systém nebude problém převzít testové materiály vytvořené někým jiným, pomocí jiného nástroje, ale používajícího stejnou specifikaci. A naopak, že on nebude mít problém s těmi našimi. Takovou, celosvětově stále častěji používanou specifikací je IMS QTI (Question & Test Interoperability). Tato specifikace popisuje základní strukturu pro reprezentaci otázek a testů ve formátu XML, kterým je zaručena plná přenositelnost těchto testových materiálů.

V této kapitole si specifikaci QTI přiblížíme podrobněji. Ukázky některých typů otázek budou uvedeny v dalších kapitolách, příklad XML kódu je uveden v Příloze B.

4.1 Historie QTI

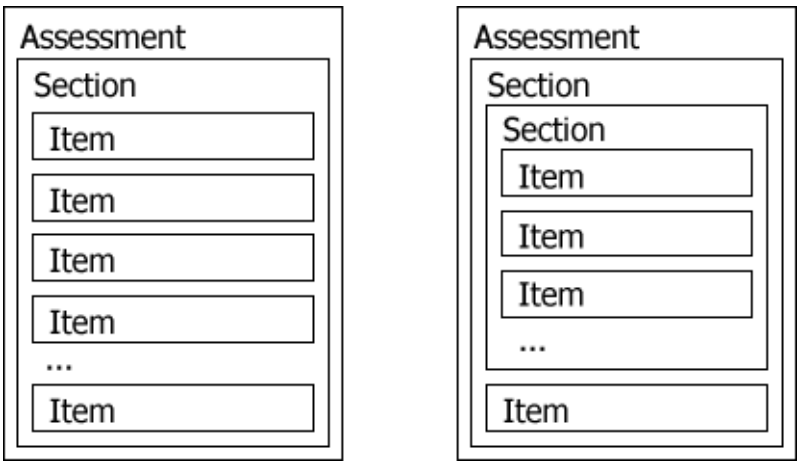
První verze QTI se objevila v roce 1999. Od této doby prošla specifikace vývojem a zdokonaleními. Verze 1.0 specifikovala informační model pro zachycení struktury testových materiálů. Implementace této specifikace se ukázala poměrně náročnou, proto k ní ve verzi 1.1 přibyla specifikace QTI Lite, která přinesla zjednodušenou verzi QTI (bude o ní řeč dále). Ve verzi 1.2 došlo k některým změnám ve struktuře XML formátu (například změna přístupu při ukládání informací – *metadat* – o testu nebo zavedení repositáře otázek). Původní model byl rozšířen o kontrolu nad prezentací testů koncovému uživateli (náhodné pořadí otázek atd.). Objevila se také potřeba standardizovat zprávy o výsledcích testů. Tím se v této verzi objevila samostatná nezávislá specifikace Result Reporting. Verze 1.2.1 přinesla pouze opravu některých chyb předchozí verze.

Další text popisuje právě verzi 1.2.1. Nicméně v průběhu tvorby této práce došlo k uvolnění QTI verze 2.0, která kompletně nahrazuje některé části specifikace. Proto budou na konci kapitoly v krátkosti uvedeny některé změny, které verze 2.0 přinesla. Ke každé verzi jsou na stránkách IMS k dispozici popisy struktury XML formátu v podobě DTD (Document Type Definition) a XML Schema souborů.

4.2 ASI informační model

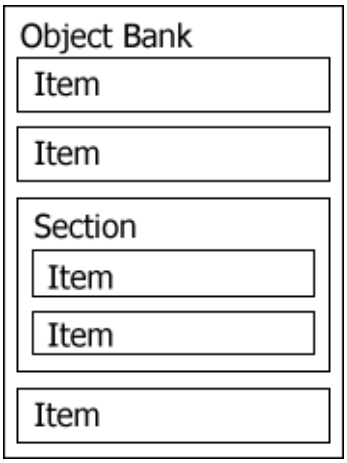
Aby se předešlo problémům s nejednoznačností pojmů, je v QTI definována terminologie. Kompletní test je zde nazýván *Assessment*. Test se samozřejmě skládá z jedné nebo více otázek.

Samotná otázka může ještě nést doplňující informaci, jako například počet bodů za správnou odpověď, jak se má otázka zobrazit nebo nápovědu. Pro otázku s jejími přidruženými údaji se používá označení *Item*. V dalším textu bude slovo položka označovat právě tento termín. V reálném světě je někdy potřeba sdružit otázky do skupin, které odpovídají například tématu otázek. Taková skupina je nazývána *Section* a v tomto textu pro ni bude používán termín sekce. Sekcí může být v rámci testu libovolné množství a každá sekce může mít svoje podsekce. Výsledný ASI (Assessment, Section and Item) informační model testových dat ilustruje obrázek 4.1.



Obrázek 4.1: Možné uspořádání částí ASI modelu

Shrnuto – test se skládá z alespoň jedné sekce. Každá sekce se skládá z podsekcí a/nebo položek.



Obrázek 4.2: Příklad repositáře otázek

Další strukturou v ASI modelu je repositář otázek, který je v QTI specifikaci pojmenován *Object Bank*. Slouží k vytvoření jakési databáze otázek, včetně jejich rozdělení do sekcí, pro pozdější použití a jednoduché výměně.

Díky ASI modelu máme jasně zachycenu hierarchickou skladbu testových dat. Každou část této struktury lze importovat či exportovat, čímž je zajištěna snadná vyměnitelnost testových dat. Nejmenším vyměnitelným prvkem je položka. Aby bylo snadno zjistitelné, jaká data se přesně vyměňují, dovoluje QTI každou část ASI modelu popsat metadaty.

Již bylo zmíněno, že položka ve smyslu QTI nenese pouze vlastní otázku, ale obsahuje také přidružené informace. Těmito informacemi jsou: informace o zobrazení otázky, informace o časovém omezení (*Duration*), informace vztahující se k ohodnocení (*Response processing*) a korespondující nápovědy (*Hint*), řešení (*Solution*) a reakce (*Feedback*). Některé tyto informace je možné vyplnit i u sekce nebo celého testu. V následujícím textu si je přiblížíme.

4.3 Typy podporovaných otázek

QTI je navrženo tak, aby nebylo svázáno s konkrétní vyučovací metodou. Místo toho zpřístupňuje množinu obecně používaných testových otázek. Podporuje 5 možných typů odpovědi na otázku:

1. výběr správné odpovědi (*Logical identifier*),
2. vyplnění textu (*String*),
3. vyplnění čísla (*Number*),
4. výběr správné souřadnice (*X-Y co-ordinate*),
5. výběr správné skupiny (*Logical group*).

To lze navíc kombinovat s několika různými styly zobrazení, které říkají, jakým způsobem je otázka zkoušenému prezentována: zda má být označení správné odpovědi docíleno pomocí přepínacího tlačítka, pomocí zaškrtávacích políček, či posunutím jezdce nebo zda má zkoušený správnou odpověď zapsat do textového políčka, či raději označit správnou část textu pomocí myši. Přesná pravidla, jak QTI dovoluje kombinovat typ odpovědi se stylem zobrazení, jsou popsána v [1].

Díky tomu jsou podporovány tyto základní druhy testových otázek. Jejich názvy jsou ponechány v angličtině, protože takto jsou v oblasti elektronické podpory výuky standardně používány.

Multiple choice – výběr jedné správné odpovědi z více možných.

Multiple response – výběr více správných odpovědí.

True/False – prakticky typ Multiple choice omezený na výběr ze dvou otázek.

Image hot spot – označení správné odpovědi kliknutím na aktivní bod v obrázku.

Fill-in-blank – vyplnění správné odpovědi do příslušného textového políčka. Počet textových políček v otázce není omezen.

Select text – identifikace správné části textu označením myší.

Slider – výběr správného celého nebo reálného čísla z daného intervalu pomocí posuvného jezdce.

Drag object – přesun objektů myší na předdefinovaná místa.

Drag target – přesun objektů myší na cílový objekt.

Ordering objects – srovnání objektů podle nějakého, předem daného, klíče.

Matching objects – rozdělení objektů do správných skupin.

Connect the points – spojení množiny bodů podle daného klíče.

QTI dovoluje i vytvoření složených otázek libovolnou kombinací otázek základních. Vedle možnosti použití základních a složených typů otázek, je ve specifikaci QTI ponecháno také místo pro rozšíření o nové typy. Toho se dá využít ve chvíli, kdy vyvíjíme systém podle této specifikace a potřebujeme do něho zařadit nějakou nestandardní otázku. Samozřejmě je velká pravděpodobnost, že ostatní systémy s touto otázkou nebudou schopny pracovat.

4.4 Možný obsah otázek

Jedním z požadavků při tvorbě testových otázek je, abychom mohli pracovat nejen s textem, ale měli i možnost prokládat ho například obrázky. QTI toto dovoluje a vedle textových a obrazových prvků umí pracovat i se zvukovými soubory, videi a dokonce i s interaktivními aplikacemi nebo applety. Není tedy problém vytvořit multimediální test. V QTI jsou prvky možného obsahu souhrnně nazývány materiál (*Material*).

S materiálem umožňuje QTI provádět některé typografické úkony. Text může být například formátován do odstavců, určitá část textu může být zvýrazněna. Lze si také nastavit pozici prvku a jeho velikost na obrazovce. Obrázky, zvukové klipy, videa, aplikace a applety se dají v otázce odkazovat prostřednictvím URI (Universal Resource Identifier) adresy, na které se nacházejí. Nebo se mohou, za pomoci vhodného kódování, vložit přímo do XML souboru.

QTI dovoluje i vytvoření tzv. alternativního materiálu (*Altmaterial*), který umožňuje použití dalšího možného obsahu otázky. Tímto způsobem lze mimo jiné vytvořit otázku pro více jazyků najednou. Počet alternativních materiálů není nijak omezen.

Konkrétní zobrazení otázek, potažmo celého testu, na obrazovce závisí na zobrazovacím systému, který bude interpretovat informace obsažené v XML souboru odpovídajícímu specifikaci. QTI styl zobrazení nedefinuje, pouze poskytuje doporučení zobrazovacímu systému.

4.5 Systém ohodnocování

Velkým přínosem použití počítačů při testování znalostí je, že nám dovolí automatické vyhodnocení otázek. Alespoň do určité míry – těžko můžeme po počítači chtít, aby vyhodnotil například esej. Ale u otázek, kde stačí správnou odpověď vybrat z nabízených nebo kde se má odpověď shodovat s přesně daným textem či číslem, automaticky vyhodnocovat lze.

QTI obsahuje poměrně propracovaný systém pro vyhodnocování otázek. U každé otázky si je možné nadefinovat proměnné, do kterých se zaznamenává ohodnocení za odpověď na otázku. Pokud si žádné vlastní proměnné nenadefinujeme, pracuje se se standardně generovanou proměnnou. Dalším krokem je vytvoření logických podmínek, při jejichž splnění se změní hodnota námi určené proměnné.

Příklad: Mějme otázku se čtyřmi možnými odpověďmi:

1. Správná odpověď
2. Správná odpověď
3. Neúplná správná odpověď
4. Špatná odpověď

A chceme, aby zkoušený dostal za označení každé správné odpovědi +3 body, za označení neúplné +1 bod, ale jakmile označí špatnou, aby dostal -4 body. Dále chceme počet správně a nesprávně označených odpovědí. Nadefinujeme si proměnné `PocetBodu`, `Spravne` a `Spatne` a vytvoříme následující pravidla:

- pokud je zvolena odpověď č. 1 nebo č. 2, pak `PocetBodu += 3`, `Spravne++`
- pokud je zvolena odpověď č. 3, pak `PocetBodu += 1`
- pokud je zvolena odpověď č. 4, pak `PocetBodu = -4`, `Spatne++`

QTI poskytuje širokou paletu možných operátorů v podmínkách. K dispozici jsou mimo jiné logické AND, OR a NOT a také relační operátory pro testování rovnosti nebo velikosti, kterých se dá využít při testu na konkrétní hodnotu. Proměnné nejsou omezeny pouze na číselné typy. Je možné si nadefinovat například proměnnou booleovského typu a do té pak uložit informaci o správnosti/nesprávnosti odpovědi. Více informací o proměnných a operátorech lze najít v [2] a [3].

To se týkalo otázek, ale co když chceme automaticky vyhodnotit celý test? K tomu používá QTI vyhodnocovací algoritmus (*Scoring algorithm*). Proměnné z jednotlivých otázek se sesbírají a podle zvoleného vyhodnocovacího algoritmu se vyhodnotí. Pro výslednou hodnotu, kterou nám algoritmus vrátí, je opět možno definovat proměnnou. QTI obsahuje sadu předdefinovaných vyhodnocovacích algoritmů, které pracují se standardně generovanými proměnnými. Jsou to:

Algoritmus NumberCorrect vracející počet správně zodpovězených otázek, které byly vybrány a zobrazeny zkoušenému.

Algoritmus WeightedNumberCorrect vracející vážený počet správně zodpovězených otázek, které byly vybrány a zobrazeny zkoušenému. Jednotlivé otázky mají přiřazenu váhu, která odráží, jak moc se má otázka podílet na výsledném skóre.

Algoritmus ParameterWeightedNumberCorrect vracející parametrizovaný vážený počet správně zodpovězených otázek, které byly vybrány a zobrazeny zkoušenému. Je shodný s algoritmem WeightedNumberCorrect, ale hodnota váhy se nastavuje globálně, ne pro každou otázku zvlášť.

Algoritmus SumofScores vracející součet bodů otázek, které byly vybrány a zobrazeny uživateli.

Algoritmus WeightedSumofScores vracející vážený součet bodů otázek, které byly vybrány a zobrazeny uživateli. Jednotlivé otázky mají přiřazenu váhu, která odráží, jak moc se má otázka podílet na výsledném skóre.

Algoritmus ParameterWeightedSumofScores vracející parametrizovaný vážený součet bodů otázek, které byly vybrány a zobrazeny uživateli. Je stejný jako algoritmus WeightedSumofScores, ale hodnota váhy se nastavuje globálně, ne pro každou otázku zvlášť.

Algoritmus BestKofN vybírající „K“ otázek z „N“ možných podle nejvyššího počtu bodů a vracející součet jejich bodů.

Algoritmus GuessingPenalty vracející počet správně a nesprávně zodpovězených otázek, včetně postihu za „hádání“ správných odpovědí.

Všechny uvedené algoritmy mimo GuessingPenalty neberou ohled na to, zda se zkoušený pokusil na otázku odpovědět nebo zda mu byla otázka pouze zobrazena. Pokud hodláme toto brát v potaz, musíme použít jejich Attempted verze (tj. například NumberCorrectAttempted nebo ParameterWeightedSumofScoresAttempted). Attempted verze není k dispozici u algoritmu BestKofN.

Samozřejmě existuje možnost vytvořit si vlastní vyhodnocovací algoritmus, který bude pracovat s námi definovanými proměnnými. Detailní informace jak toho docílit a doplňující informace o předdefinovaných vyhodnocovacích algoritmech jsou uvedeny v [3].

4.6 Reakce, nápověda, řešení

Reakce, nápověda a řešení představují zpětnou vazbu k otázce. Reakce také umožňuje definovat zpětnou vazbu k celému testu.

Reakce úzce souvisí s vyhodnocováním otázek a testů, popsaném v předchozím textu. V momentě, kdy dojde ke splnění podmínky pro ohodnocení otázky a proměnné se nastaví hodnota, můžeme zkoušeného informovat, jak moc se „strefil“ do správné odpovědi. Zkoušený tak rovnou bude mít k dispozici údaj o tom, na kolik otázek odpověděl špatně. V kontextu celého testu se dá toto ještě více rozšířit. Poté co proběhne zvolený vyhodnocovací algoritmus a vrátí nám výsledek, je možné vytvořit pro testování hodnoty výsledku sadu podmínek. Pro každou podmínku pak lze definovat odpovídající reakci.

Příklad: Mějme v testu 10 otázek a zvolený vyhodnocovací algoritmus NumberCorrect, který nám vrátí počet správných odpovědí – ten si uložíme do proměnné `PocetSpravnych`. Vytvoříme následující pravidla:

- `PocetSpravnych > 5`, pak zobraz reakci: Absolvoval.
- `PocetSpravnych == 5`, pak zobraz reakci: Nutná ústní zkouška.
- `PocetSpravnych < 5`, pak zobraz reakci: Neabsolvoval.

Při tvorbě těchto podmínek je opět možné použít celou škálu operátorů. Pravidla pro jejich tvorbu jsou uvedena v [3].

Nápověda a řešení tvoří jiný typ zpětné vazby než reakce. Pomocí nápovědy přiblížíme zkoušenému správnou odpověď na otázku. Můžeme si zvolit, zda mu ji zobrazíme po částech nebo celou najednou. Pomocí řešení pak správnou odpověď zkoušenému prozradíme.

Možný obsah reakce, nápovědy a řešení je znovu nazýván materiál a platí pro něj ta samá pravidla, jako pro materiál otázky. Není tedy problém použít například pro řešení zvukový klip, který zkoušenému přímo „řekne“, jak měla znít správná odpověď.

Kdy a jak se reakce, nápověda či řešení objeví na obrazovce, opět závisí na zobrazovacím systému.

4.7 Výběr a uspořádání

Mechanismus výběru a uspořádání (*Selection & Ordering*) slouží k vymezení pravidel pro konečné zobrazení testu zkoušenému. Uplatnění najde v situacích, kdy potřebujeme nějakým způsobem zvolit podmnožinu otázek, kterou pak prezentujeme zkoušenému. Jako modelová situace poslouží toto: Vytvořili jsme si velké množství otázek, ze kterých ale chceme pro konkrétní test použít pouze určitý počet. K tomu ještě navíc požadujeme, aby se každému zkoušenému otázky zobrazily v různém pořadí.

Vlastní mechanismus uplatněný v QTI se skládá ze 3 nezávislých, ale navzájem souvisejících, kroků – výběru, uspořádání a sekvence.

Výběr je prvním krokem. Spočívá v aplikaci výběrových pravidel na otázky. Ty říkají, jakým způsobem chceme otázky vybírat: zda se mají vybrat všechny dostupné otázky, či zda mají mít konkrétní vlastnost, zda chceme všechny s touto vlastností nebo z nich jen náhodně vybrat stanovený počet. Dokonce je možné vytvořit i pravidla, která berou v potaz logické asociace, tedy například omezit výběr otázek v závislosti na výběru jiných.

Uspořádání definuje pořadí zobrazení vybraných otázek tomu kterému zkoušenému. Určuje, jestli mají být zobrazeny v pořadí, v jakém byly vytvořeny nebo zda se mají zobrazit v pořadí náhodném. Lze si také stanovit, aby některé otázky měly fixní polohu při každém zobrazení.

A sekvence říká, zda-li má být otázka zahrnuta do konečného výběru pouze jednou nebo vícekrát.

Všechny tyto kroky se na test aplikují až v době jeho zobrazení. Detailnější informace o mechanismu výběru a uspořádání jsou uvedeny v [4].

4.8 QTI Lite

V některých případech není nutné implementovat kompletní verzi QTI. IMS proto přišla s „ořezanou“ verzí QTI pojmenovanou QTI Lite. V této verzi je původní ASI model redukován pouze na položky a je podporován pouze jeden typ otázek – Multiple choice. Není tedy možné vytvářet testy, sdružovat otázky do sekcí nebo vytvářet repositáře otázek. Mezi další změny QTI Lite, oproti původní verzi, patří:

- Zjednodušený systém ohodnocování otázek, je možné vytvářet pouze celočíselné proměnné a nelze použít vyhodnocovací algoritmy.
- Nelze provádět výběr a uspořádání otázek.
- Nelze vytvářet nápovědy a řešení.
- Materiál otázky je omezen – není možné vytvářet formátovaný text a používat zvukové a video prvky.
- Otázky není možné časově omezit.

Specifikace QTI Lite je podmnožinou specifikace QTI, čili všechny položky odpovídající QTI Lite odpovídají i QTI. Z tohoto důvodu tedy není nijak dotčena interoperabilita položek.

4.9 Verze 2.0

V současné době je na stránkách IMS k dispozici QTI verze 2.0. Tato verze již nepoužívá ASI model a místo něho byla vytvořena složitější hierarchická struktura. Specifikace nyní přesně definuje objektově orientovaný model pro testová data. Pro jednotlivé objekty modelu jsou k dispozici abstraktní třídy obsahující deklarace atributů objektů.

Dále byl přepracován model interakce otázek se zkoušeným. Původní tvorba otázek pomocí kombinace typu odpovědi se stylem zobrazení byla zrušena. Důraz byl také kladen na zvýšení kontroly prezentace otázek uživateli. To bylo umožněno povolením prvků jazyka XHTML při tvorbě otázek. Otázky lze nyní vytvářet i pouhou parametrizací předem připravených šablon. Zjednodušená verze QTI Lite byla z této verze odstraněna. Také ohodnocování otázek a s ním sdružené vytváření ohodnocovacích podmínek doznalo rozšíření.

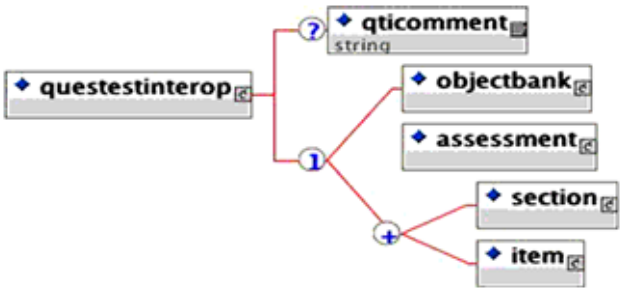
Velký prostor byl věnován zasazení QTI mezi další specifikace, které IMS vyvíjí. Jedná se o Content Packaging, Simple Sequencing a Learning Design. QTI se tak stala velmi obsáhlou specifikací, která umožňuje sdílení a přenos komplexních testovacích dat, stejně jako výsledků, kterých v testech dosáhli jednotliví zkoušení.

Pro snazší přechod na novou verzi vydala organizace IMS sadu návodů, jak konkrétně postupovat při konverzi dat z nějaké předchozí verze na verzi 2.0. K dispozici je v [6].

Kapitola 5

Návrh a implementace API pro IMS QTI

Tato kapitola se věnuje popisu API pro práci se specifikací IMS QTI. V minulé kapitole bylo řečeno, že QTI je realizováno v XML. Základním XML elementem specifikace je element `questestinterop`, který rozděluje testová data do částí ASI modelu. Prezentované obrázky byly převzaty z [2].



Obrázek 5.1: Znázornění struktury elementu *questestinterop*

API bylo naprogramováno v programovacím jazyce Java s využitím knihovny `dom4j` pro práci s XML stromem specifikace QTI. Nachází se v balíku `cz.muni.fi.ims`. Má 2 způsoby použití. Jednak dovoluje vytvoření prázdného XML stromu, do kterého budou postupně přidávány nové elementy a tento strom bude pak moci být uložen do souboru. Nebo je možné načíst již existující XML strom a provádět v něm úpravy. API také provádí kontrolu správnosti XML stromu, aby bylo zaručeno, že odpovídá specifikaci QTI.

API bylo vytvořeno pro QTI verze 1.2.1 a kromě ní plně spolupracuje také s verzí 1.2. Dřívější verze jsou také podporovány, ale pouze do jisté míry, neboť jejich XML formát se v některých částech liší.

5.1 Objektový model pro API

Prvním krokem při vytváření API bylo zvolení vhodného objektového modelu, který bude reprezentovat strukturu XML elementů specifikace QTI. Samo XML už hierarchii poskytuje a proto jí bylo využito. Pro každý element, který není v XML stromu listem, byla vytvořena třída nesoucí korespondující jméno se jménem elementu. Tedy například pro element `assessment` byla vytvořena třída `Assessment`, nebo element `objectbank` dostal třídu

`ObjectBank`. Toto pravidlo bylo porušeno pouze u elementu `questestinterop`, pro který byla vytvořena třída pojmenovaná `QTI` (bude popsána v části Třída QTI).

Dalším krokem bylo doplnění vytvořených tříd vhodnými metodami. U jejich tvorby byl použit následující postup:

1. Každá třída má definován konstruktor sloužící k naplnění jejích atributů, které odpovídají atributům elementu. Výjimku tvoří atribut jménem `ident`, který je generován automaticky (2 třídy umožňují i jeho naplnění konstruktorem, jak bude popsáno v části Atribut ident) a atributy z prostoru názvů `xml`, kterým je ponechána jejich standardní hodnota. Všechny parametry konstruktoru jsou pro jednoduchost typu `String`.

2. Pro každý podelement, který má vlastní třídu, byla vytvořena metoda `create<název odpovídající třídy>`, která danou třídu vrací. Parametry metody odpovídají atributům podelementu.

3. Pro každý podelement, který vlastní třídu nemá a může se pod daným elementem vyskytovat vícenásobně, byla vytvořena metoda `add<název odpovídajícího podelementu>`. Parametry metody odpovídají atributům podelementu. Pokud může podelement uchovávat textovou hodnotu, je metodě přidán parametr `newValue` typu `String`.

4. Pro každý podelement, který vlastní třídu nemá a může se pod daným elementem vyskytovat pouze jednou, byla vytvořena metoda `set<název odpovídajícího podelementu>`. Parametry metody odpovídají atributům podelementu. Pokud může mít podelement textovou hodnotu, je metodě opět přidán parametr `newValue` typu `String`.

Ilustrujme si to na příkladu elementu `assessment`, jehož struktura je zachycena na obrázku 5.2. Třída `Assessment` bude mít následující metody:

- Konstruktor `Assessment(String title)`,
- `void setComment(String newValue)`,
- `void setDuration(String newValue)`,
- `QTIMetadata createQTIMetadata()`,
- `Objectives createObjectives(String view)`,
- `Control createAssessmentControl(String feedbacks, String hints, String solutions, String view)`,
- `Rubric createRubric(String view)`,

Obrázek 5.2: Znázornění struktury elementu *assessment*

- `PresentationMaterial createPresentationMaterial(),`
- `OutComesProcessing createOutComesProcessing(String scoremodel),`
- `void setAssessprocExtension(),`
- `Feedback createAssessFeedback(String title, String view, String ident),`
- `SelectionOrdering createSelectionOrdering(String sequencetype),`
- `Reference createReference(),`
- `void addSectionRef(String linkrefid),`
- `Section createSection(String title).`

Takto zkonstruované metody slouží pouze pro vytváření nových elementů v XML stromu. Potřebné jsou však i metody, které zpřístupní element již existující. Ty jsou vytvořeny podle následujícího schéma:

1. Pro každý podelement, který má vlastní třídu, byla vytvořena metoda `get<název odpovídající třídy>`, která danou třídu vrací.
2. Pro každý podelement, který vlastní třídu nemá, byla vytvořena metoda `get<název odpovídajícího podelementu>`, která vrací třídu `Params` (třída `Params` bude popsána v části Třída Params).
3. Pokud se podelement může v daném elementu vyskytovat vícekrát, je metoda `get` rozšířena o parametr `int num` říkající, který výskyt podelementu se má zpřístupnit. Výskyt elementů je číslován od 1.
4. Pokud má podelement atribut `ident`, je k dispozici i metoda `get` s parametrem `String ident`, která zpřístupní podelement s odpovídajícím hodnotou atributu.

Jestliže daný podelement v XML stromu neexistuje, vrací metoda `get` ve všech svých variantách hodnotu `null`. V případě již zmiňované třídy `Assessment` je situace tato:

- `Params getComment(),`
- `Params getDuration(),`
- `QTIMetadata getQTIMetadata(int num),`
- `Objectives getObjectives(int num),`
- `Control getAssessmentControl(int num),`
- `Rubric getRubric(int num),`
- `PresentationMaterial getPresentationMaterial(),`
- `OutComesProcessing getOutComesProcessing(int num),`
- `Params getAssessprocExtension(),`
- `Feedback getAssessFeedback(int num),`
- `Feedback getAssessFeedback(String ident),`
- `SelectionOrdering getSelectionOrdering(),`
- `Reference getReference(),`
- `Params getSectionRef(),`

- `Section getSection(int num)`,
- `Section getSection(String ident)`.

Některé třídy by při dodržení popsaného postupu měly stejný obsah. Proto byly sdruženy do jedné, které byla pojmenována obecnějším jménem. V uvedených příkladech je to případ tříd `Control` zastupující třídy `AssessmentControl`, `SectionControl` a `ItemControl` a také `Feedback`, která zastupuje třídy `AssessFeedback` a `SectionFeedback`.

Použití takto vytvořených tříd si demonstrujeme v následující části programového kódu:

```
Section section ...

Item item = section.createItem(”itemTest”, ”label1”, ”6”);
item.createItemControl(”Yes”, ”No”, ”No”, ”Candidate”);

Presentation present = item.createPresentation(”item1”);

Response resp = present.createResponseLID(”Single”, ”No”);
resp.createRenderChoice(”No”, ”0”, ”10”);
present.getResponseLID(1).createRenderChoice(”Yes”, ”5”, ”7”);

ResProcessing rproc = item.createResProcessing(”NumberCorrect”);

OutComes outcom = rproc.createOutComes();
outcom.addDecvar(”PocetBodu”, ”Integer”, ”0”, ”0”);
outcom.addDecvar(”Spravne”, ”Boolean”, ”False”);
rproc.getOutComes().addDecvar(”Spatne”, ”Boolean”, ”True”);
```

To nám vygeneruje tento XML kód:

```
<section>
  <item title=”itemTest” ident=”I001” label=”label1” maxattempts=”6”>
    <presentation label=”item1”>
      <response_lid ident=”LID01” rcardinality=”Single” rtiming=”No”>
        <render_choice shuffle=”No” minnumber=”0” maxnumber=”10”/>
        <render_choice shuffle=”Yes” minnumber=”5” maxnumber=”7”/>
      </response_lid>
    </presentation>
    <resprocessing scoremodel=”NumberCorrect”>
      <outcomes>
        <decvar varname=”PocetBodu” vartype=”Integer” defaultval=”0”/>
        <decvar varname=”Spravne” vartype=”Boolean” defaultval=”False”/>
        <decvar varname=”Spatne” vartype=”Boolean” defaultval=”True”/>
      </outcomes>
    </resprocessing>
  </item>
</section>
```

Několik tříd má speciální použití a proto si je detailněji popíšeme. Kompletní seznam tříd je uveden v Příloze A.

5.2 Třída AbstractIMSTag

Všechny třídy vytvořené podle postupu popsaného v předchozí podkapitole jsou potomky abstraktní třídy `AbstractIMSTag`. Jádro této třídy tvoří `Element` knihovny `dom4j` reprezentující rodičovský element třídy, tedy element, pro který je ta která třída vytvořena. Třída `AbstractIMSTag` poskytuje 2 metody:

- `getChildInOrder`, která zpřístupňuje *n-tý* podelement rodičovského elementu. Číslování potomků začíná číslem 1. Vrací třídu `Params` příslušnou podelementu. Pokud takový podelement neexistuje, vrací se hodnota `null`.
- `getParams`, která vrací třídu `Params` příslušnou rodičovského elementu.

5.3 Třída Params

Třída `Params` slouží pro práci s konkrétním elementem XML stromu. Zpřístupňuje jeho obsah a atributy a umožňuje s nimi manipulovat. Obsahuje tyto metody:

- `getName`, která vrací název elementu. Vrácená hodnota je typu `String`.
- `getValue`, která vrací textový obsah elementu. Vrácená hodnota je typu `String`.
- `setValue` nastavující novou hodnotu textového obsahu elementu.
- `getAttValue` vracející hodnotu zadaného atributu. Vrácená hodnota je typu `String` pokud atribut existuje. V opačném případě je vrácena hodnota `null`.
- `setAttValue`, která slouží k nastavení nové hodnoty pro atribut. Pokud atribut pro element neexistuje, je vytvořen nový a nastaví se mu požadovaná hodnota.

Pro práci s atributy elementu lze také využít veřejně přístupný seznam `attributes` této třídy. Skládá se z prvků `Attribute` knihovny `dom4j`.

5.4 Třída QTI

Základní třídou v balíčku `cz.muni.fi.ims` je třída `QTI`. Zastupuje nejzevnější element specifikace QTI – `questestinterop`, jehož struktura je uvedena na obrázku 5.1. Základem třídy je `Document` knihovny `dom4j`, uchovávající souhrnné informace o XML stromu. U této třídy vždy začíná generování XML stromu a tedy poskytuje metody pro tvorbu a zpřístupnění částí ASI modelu:

- `createAssessment`, `getAssessment`,
- `createObjectBank`, `getObjectBank`,

- `createSection`, `getSection`,
- `createItem`, `getItem`.

Kromě toho poskytuje třída `QTI` také nástroje pro ukládání XML stromu do souboru a načtení již uloženého XML stromu. Jsou to:

Konstruktor `QTI(String name)` sloužící k vytvoření nového prázdného XML stromu. Parametr `name` je použit při tvorbě `ident` atributu (o tom bude pojednáno v části Atribut ident).

Konstruktor `QTI(File file)` slouží k načtení existujícího XML stromu ze souboru.

Metoda `saveXML(File file)` ukládá aktuální XML strom do zvoleného souboru.

Metoda `setXMLEncoding(String charset)` nastavuje kódování, ve kterém bude XML strom uložen do souboru. Standardně je nastaveno kódování UTF-8.

Metoda `getXMLEncoding()` vracející nastavené kódování pro uložení souboru nebo vrátí kódování, ve kterém byl uložen právě načtený soubor.

Tato třída může vyvolat některé výjimky. Budou popsány v části Výjimky.

5.5 Atribut ident

Jednotlivé části ASI modelu je nutné celosvětově jednoznačně identifikovat. Díky tomu se při importování dá spolehlivě určit, z jakého zdroje ta která část pochází. Identifikace se provádí pomocí atributu `ident`, který obsahuje identifikační řetězec. Organizace IMS vydala doporučení, jak by měl tento identifikační řetězec zvaný PLIRID (Persistent, Location-Independent Resource Identifier), vypadat. Jeho struktura je popsána v [5].

Atribut `ident` je u tříd reprezentujících části ASI modelu generován automaticky. Jako PLIRID byl zvolen řetězec:

`URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:<object>_<jméno><číslo>`

kde

- `<objekt>` je jedno ze čtyř písmen A, B, S nebo I, podle toho zda se jedná o třídy `Assessment`, `ObjectBank`, `Section` nebo `Item`,
- `<jméno>` se nahradí řetězcem, zadaným v konstruktoru třídy `QTI`,

- `<číslo>` označuje třímístný čítač objektů.

Tedy konkrétně například:

```
URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:I_MATEMATIKA003
```

Atribut `ident` mohou mít i jiné elementy, než jen ty, které tvoří části ASI modelu. U těchto elementů je také vyžadována jednoznačná identifikace, ale pouze v rámci XML stromu nebo určité části XML stromu. Pro třídy těchto elementů opět platí, že se atribut `ident` generuje automaticky. Jak konkrétně pro jednotlivé třídy vypadá uvádí Příloha A.

Třídy, které automaticky generují atribut `ident`, mají k dispozici metodu `getIdent`, která vygenerovaný identifikátor vrací. Třídy `Feedback` a `ItemFeedback` mají navíc i metodu `reserveIdent`, která rezervuje identifikátor pro pozdější užití. Tyto třídy rovněž umožňují specifikovat hodnotu atributu `ident` pomocí svých konstruktorů.

5.6 Výjimky

Metody třídy příslušející nějakému elementu mohou vyvolat 2 typy výjimek. Oba přímo souvisí s hodnotami atributů elementu. Většina atributů je nepovinná. To znamená, že místo uvedení jejich hodnoty, můžeme použít hodnotu `null`. V XML stromu se pak tento atribut nevyskytuje, nebo je použita jeho standardní hodnota, má-li ji určenu. Přesto některé atributy deklaraci hodnoty vyžadují. Pokud pro ně není hodnota specifikována, je vyvolána výjimka `RequiredAttributeMissingException` s doplňující informací, který atribut je nutné vyplnit. Druhým typem je výjimka `WrongAttributeValueException`, která je vyvolána, pokud chceme atributu přiřadit jinou hodnotu, než má deklarováno. Také ona obsahuje doplňující informaci, která hodnota je špatně zadaná.

Metody třídy `QTI` mohou vyvolávat i jiné typy výjimek, které se vztahují k načítání a ukládání XML souboru. Pokud se při načítání souboru zjistí, že jeho formát neodpovídá specifikaci QTI, je vyvolána výjimka `NonIMSXMLException` spolu s doplněním, na kterém místě souboru došlo k porušení formátu QTI. Pokud nelze soubor pro načtení nalézt, vyvolá se `FileNotFoundException`. Při ukládání XML stromu se nejprve ověří, zda tento strom odpovídá specifikaci QTI. Pokud tomu tak není, je vyvolána výjimka `BadIMSConstructionException`, obsahující opět důvod, proč daný strom specifikaci odporuje. Dojde-li při vlastním zapisování souboru na médium k nějaké chybě, je vyvolána výjimka `IOException`. Je-li pro uložení souboru zvoleno neznámé kódování, vyvolá se výjimka `UnsupportedEncodingException`.

5.7 Třída Global

Třída `Global` obsahuje statické metody používané ostatními třídami. Jedná se o tyto metody pro přímou práci s XML stromem, pro kterou využívá knihovnu `dom4j`:

- `setElement` pro vytvoření nového elementu v XML stromu.

- `setElementValue`, `setElementCDATA` pro nastavení nového textového obsahu pro element. Pokud element neexistuje, je nejprve vytvořen.
- `addElementValue`, `addElementCDATA` pro vytvoření nového elementu s daným textovým obsahem.
- `getObjectByNum` pro zpřístupnění podelementu podle jeho výskytu. Výskyt podelementů je číslován od 1.
- `getObjectByIdent` pro zpřístupnění podelementu podle jeho atributu `ident`.
- `getAttributes` pro zpřístupnění seznamu atributů elementu.
- `checkAttributeList` pro kontrolu hodnoty atributu.

Vedle statických metod uchovává třída `Global` také veřejně přístupné konstanty se jmény všech elementů a atributů, které jsou vymezeny pro specifikaci QTI. Dále je v ní definován atribut `pathToDTD`, obsahující adresu k DTD popisující XML formát specifikace QTI. Tato cesta je zapsána v každém uloženém XML souboru v sekci `DOCTYPE`. Rovněž obsahuje atribut `ID` uchovávající zvolený identifikační řetězec PLIRID.

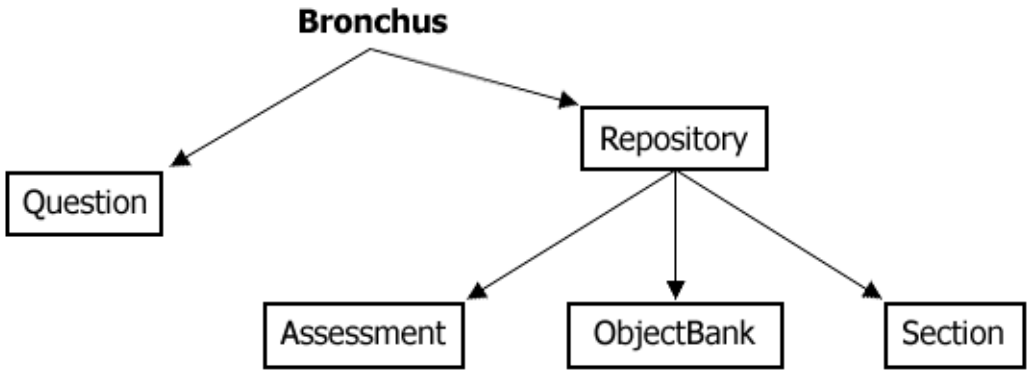
Kapitola 6

Implementace vstupních a výstupních operací

Bronchus je editor otázek a testů, který vznikl v rámci diplomové práce kolegy Tomáše Udržala. Bronchus byl navrhnut tak, aby umožňoval vytvářet a modifikovat otázky a testy odpovídající specifikaci QTI. Ale sám o sobě slouží pouze jako grafické uživatelské prostředí a editované otázky a testy ukládá pouze do svých vnitřních tříd. Proto bylo API pro QTI, popsané v předchozí kapitole, použito pro implementaci vstupních a výstupních operací pro tento editor. Díky tomu může Bronchus ukládat vytvořené otázky a testy do XML souboru nebo je z něho načítat. Také může importovat otázky vytvořené jiným nástrojem, ale odpovídající specifikaci QTI. Další informace o editoru Bronchus jsou obsaženy v [7].

V této kapitole bude popsán postup transformace dat z vnitřních tříd editoru Bronchus do jim odpovídajících tříd, které reprezentují části ASI modelu specifikace QTI, a naopak. O tuto transformaci se starají třídy vytvořené opět v programovacím jazyce Java, které jsou umístěné v balíku `cz.muni.fi.bronchusio`.

6.1 Třídy editoru Bronchus



Obrázek 6.1: Hlavní třídy editoru Bronchus

Bronchus používá pro uložení testových dat 4 hlavní třídy: `Assessment`, `ObjectBank`, `Section` a `Question`. Třídy `Assessment`, `ObjectBank` a `Section` mají navíc společnou

rodičovskou třídu `Repository`. Jelikož je Bronchus navržen pro QTI, obsahují tyto třídy data spojená s nějakou částí ASI modelu. Například tedy třída `Question` obsahuje data relevantní pro položku specifikace QTI a třída `Section` data vztahující se k sekci.

Protože se názvy některých tříd editoru Bronchus kryjí s názvy tříd v API pro QTI, bude v dalším textu třída editoru Bronchus označována termínem vnitřní třída.

6.2 Balík cz.muni.fi.bronchusio.saver

Tento balík obsahuje třídy pro transformaci dat z vnitřních tříd editoru do XML souboru. Ke komunikaci s editorem slouží třída `Saver`. Její konstruktor `Saver(String codeTable)` umožňuje nastavit kódování, které bude použito pro výstupní XML soubor. Pokud není specifikováno, použije se standardně kódování UTF-8.

Hlavní metodou v třídě `Saver` je metoda `save`, která ukládá transformovaný XML strom. Její parametry jsou:

- Vnitřní třída editoru typu `Repository`, která se má přetransformovat. Podporovány jsou pouze třídy `Assessment` a `ObjectBank`.
- Soubor, do kterého se bude ukládat.
- Informace, zda přepisovat existující soubory.

Metoda vrací třídu implementující rozhraní `Map`. Ta je výstupem metody `getErrorList` třídy `FileWork` a bude popsána v části Třída FileWork.

Pokud v průběhu transformace dojde k chybě, je metodou `save` vyvolána výjimka `SaveErrorException` doplněná o informaci, o jakou chybu se jedná.

6.2.1 Rozhraní SaverProcesor a třída SaverProcesorFactory

Pro samotný proces transformace konkrétní vnitřní třídy editoru na odpovídající třídu reprezentující část ASI modelu, bylo vytvořeno rozhraní `SaverProcesor`. Rozhraní disponuje jedinou metodou:

`process` – metoda spouštějící transformaci. Vnitřní třídu editoru přemění na odpovídající třídu ASI modelu, která je specifikována jako parametr této metody.

Každou vnitřní třídu editoru lze transformovat vhodnou implementací tohoto rozhraní. Smyslem rozhraní je vytvořit nástroj, který by dovoloval, při případném doplnění další vnitřní třídy do editoru, jednoduché rozšíření procesu transformace. Výhodou tohoto přístupu je, že se takovéto rozšíření nijak neprojeví na funkčnosti již existujících implementací.

Implementovány jsou tyto 4 třídy, které odpovídají jednotlivým vnitřním třídám editoru:

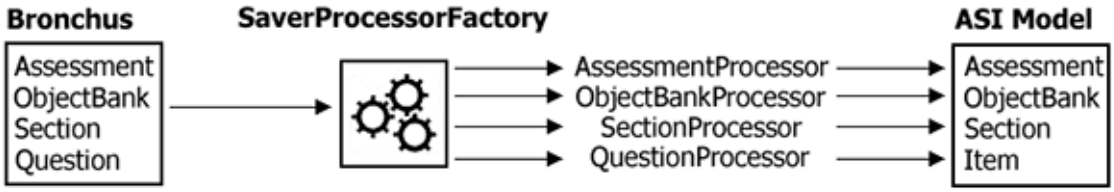
- `AssessmentProcessor`,

- `ObjectBankProcessor`,
- `SectionProcessor`,
- `QuestionProcessor`.

Se kterou konkrétní instancí vnitřní třídy mají pracovat, je specifikováno parametrem jejich konstruktoru. Jelikož konstruktory a některé metody jsou všem těmto třídám společné, byla pro ně vytvořena abstraktní rodičovská třída `AbstractSaverProcessor`.

Vlastní proces transformace vnitřní třídy editoru se tedy skládá z následujících kroků:

1. zjistit konkrétní typ instance vnitřní třídy,
2. vytvořit implementaci rozhraní `SaverProcesor` pro zjištěný typ vnitřní třídy,
3. spustit metodu `process` této implementace.



Obrázek 6.2: Schéma práce třídy SaverProcessorFactory

Mnohem efektivnější by však bylo, kdyby existoval nástroj, který by detekoval typ instance vnitřní třídy a přiřadil k ní odpovídající implementaci rozhraní. Tím by se kroky 1 a 2 sdružily v jeden. K tomuto účelu slouží třída `SaverProcessorFactory` a její statická metoda `getSaverProcessor`, která jako svůj parametr přebírá obecnou instanci vnitřní třídy a vrací k ní příslušnou třídu implementující rozhraní `SaverProcessor`. Pokud se typ vnitřní třídy nerozezná, je vrácena hodnota `null`. Schéma práce třídy `SaverProcessorFactory` lze ilustrovat obrázkem 6.2.

Rozhraní `SaverProcessor` je umístěno v balíku `cz.muni.fi.bronchusio.saver.processor`, stejně jako všechny zmíněné třídy. Princip práce jednotlivých tříd bude popsán v dalším textu. Balík navíc obsahuje také třídu `ScoreModel`, kterou ostatní třídy využívají. Její funkce bude objasněna v části Třída ScoreModel.

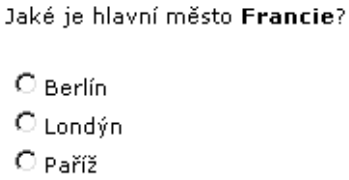
6.2.2 Třída QuestionProcessor

Třída `QuestionProcessor` prochází data vnitřní třídy `Question`, interpretuje jejich význam a ukládá je do správných částí třídy `Item` tak, aby odpovídaly specifikaci QTI. Třída

`Question` nese obecné informace o otázce: její název a znění, časové omezení otázky, podmínkovou část, nápovědu a řešení otázky. Také poskytuje údaj o váze otázky, pokud je potřeba ve vyhodnocovacím algoritmu. Časové omezení otázky je ukládáno ve speciálním formátu, o němž bude zmínka v části Třída Utility. Podmínková část bude rozpracována dále.

Prvním krokem transformace je uložení názvu, znění otázky a její váhy, pokud je specifikována. To je následováno uložením těla otázky, což jsou například možné odpovědi na otázku nebo údaje o textovém políčku otázky. Bronchus podporuje tvorbu osmi typů otázek, každá je potomkem třídy `Question`. Každá má své zvláštnosti a z tohoto důvodu je nutné před vlastním ukládáním těla otázky zjistit nejprve její typ.

Otázka Multiple choice má přiřazenu třídu `MultipleChoiceQuestion`. Znění možných odpovědí je uloženo v seznamu a je přítomen příznak, zda lze pořadí odpovědí přehazovat (*shuffle*).



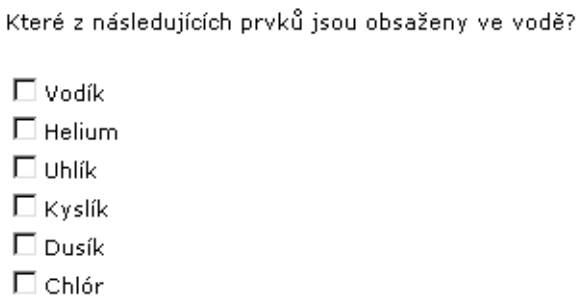
Obrázek 6.3: Otázka Multiple choice

XML kód pro Multiple choice otázku uvedenou na obrázku 6.3 je tento:

```
<item title="geo" ident="ITEM001">
  <presentation label="MC01">
    <flow class="Block">
      <material>
        <mattext>Jaké je hlavní město </mattext>
        <matemtext>Francie?</matemtext>
      </material>
      <response_lid ident="LID01" rcardinality="Single">
        <render_choice shuffle="Yes">
          <response_label ident="LID01_A" rshuffle="Yes">
            <flow_mat class="Block">
              <material>
                <mattext>Berlín</mattext>
              </material>
            </flow_mat>
          </response_label>
          <response_label ident="LID01_B" rshuffle="Yes">
            <flow_mat class="Block">
              <material>
```

```
              <mattext>Londýn</mattext>
            </material>
          </flow_mat>
        </response_label>
        <response_label ident="LID01_C" rshuffle="Yes">
          <flow_mat class="Block">
            <material>
              <mattext>Paříž</mattext>
            </material>
          </flow_mat>
        </response_label>
      </render_choice>
    </response_lid>
  </flow>
 </presentation>
</item>
```

Otázka Multiple response je uložena v třídě `MultipleResponseQuestion`. V zásadě se mnoho neliší od otázky Multiple choice, možné odpovědi jsou opět uloženy v seznamu a znovu je zde přítomen příznak shuffle. Rozdíl je především v podmínkové části.



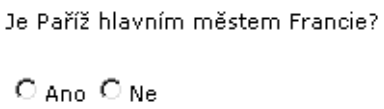
Obrázek 6.4: Otázka Multiple response

Otázce z obrázku 6.4 odpovídá následující XML kód:

```
<item title="chem" ident="ITEM002">
  <presentation label="MR01">
    <flow class="Block">
      <material>
        <mattext>Které z následujících prvků jsou obsaženy
                 ve vodě?</mattext>
      </material>
      <response_lid ident="LID02" rcardinality="Multiple">
        <render_choice shuffle="Yes">
          <response_label ident="LID02_A" rshuffle="Yes">
            <flow_mat class="Block">
```

```
            <material>
              <mattext>Vodík</mattext>
            </material>
          </flow_mat>
        </response_label>
        <response_label ident="LID02_B" rshuffle="Yes">
          <flow_mat class="Block">
            <material>
              <mattext>Helium</mattext>
            </material>
          </flow_mat>
        </response_label>
        ...
        <response_label ident="LID02_F" rshuffle="Yes">
          <flow_mat class="Block">
            <material>
              <mattext>Chlór</mattext>
            </material>
          </flow_mat>
        </response_label>
      </render_choice>
    </response_lid>
  </flow>
 </presentation>
</item>
```

Otázka True/False. Pro tuto otázku slouží třída `TrueFalseQuestion`. Má pouze 2 možné odpovědi, které jsou uloženy v atributech třídy. Má také příznak shuffle a oproti minulým dvěma otázkám má ještě příznak, zda mají být možné odpovědi zobrazeny pod sebou nebo vedle sebe.



Obrázek 6.5: Otázka True/False

Příslušný XML kód pro True/False otázku z obrázku 6.5 je:

```
<item title="geo" ident="ITEM003">
  <presentation label="TF01">
    <flow class="Block">
      <material>
        <mattext>Je Paříž hlavním městem Francie?</mattext>
      </material>
      <response_lid ident="LID03" rcardinality="Single">
```

```
        <render_choice shuffle="Yes">
          <flow_label class="Block">
            <response_label ident="LID03_A" rshuffle="Yes">
              <material>
                <mattext>Ano</mattext>
              </material>
            </response_label>
            <response_label ident="LID03_B" rshuffle="Yes">
              <material>
                <mattext>Ne</mattext>
              </material>
            </response_label>
          </flow_label>
        </render_choice>
      </response_lid>
    </flow>
  </presentation>
</item>
```

Fill-in-blank otázka je uložena v třídě `FillInBlankQuestion`. Tato otázka mísí materiál otázky s instrukcí pro uložení informací o textovém políčku. Proto je potřeba od sebe materiál a instrukce oddělit.

Doplňte chybějící slova do textu Richarda III.

Now is the [ ] of our discontent made glorious [ ] by these sons of [ ].

Obrázek 6.6: Otázka Fill-in-blank

Znění otázky je tedy uloženo v tomto formátu:

```
... material1 <img src="..." velikost="5" data="String" typ="input">
material2 ...
```

ze kterého je transformováno na odpovídající XML reprezentaci:

```
...
<material>
  material1
</material>
<response_str>
  <render_fib maxchars="5" fibtype="String">
    <response_label ident="A"/>
  </render_fib>
</response_str>
```

```
<material>
  material2
</material>
...
```

XML kód pro otázku z obrázku 6.6 je:

```
<item title="eng" ident="ITEM004">
  <presentation label="FB01">
    <flow class="Block">
      <material>
        <mattext>Doplňte chybějící slova do textu Richarda III.
        </mattext>
      </material>
      <flow class="Block">
        <material>
          <mattext>Now is the</mattext>
        </material>
        <response_str ident="STR04" rcardinality="Single">
          <render_fib maxchars="20" prompt="Box" fibtype="String">
            <response_label ident="STR04_A" rshuffle="No"/>
          </render_fib>
        </response_str>
        <material>
          <mattext>of our discontent made glorious</mattext>
        </material>
        <response_str ident="STR05" rcardinality="Single">
          <render_fib maxchars="25" prompt="Box" fibtype="String">
            <response_label ident="STR05_B" rshuffle="No"/>
          </render_fib>
        </response_str>
        <material>
          <mattext>by these sons of</mattext>
        </material>
        <response_str ident="STR06" rcardinality="Single">
          <render_fib maxchars="25" prompt="Box" fibtype="String">
            <response_label ident="STR06_C" rshuffle="No"/>
          </render_fib>
        </response_str>
        <material>
          <mattext>.</mattext>
        </material>
      </flow>
    </flow>
  </presentation>
</item>
```

Otázka Short answer a její data jsou uloženy v třídě `ShortAnswerQuestion` a omezují se

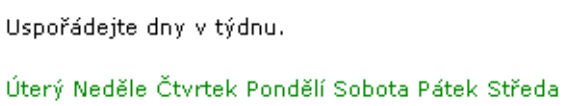
pouze na definici velikosti zobrazeného textového políčka.

Obrázek 6.7: Otázka Short answer

XML kód pro otázku z obrázku 6.7 je:

```
<item title="var" ident="ITEM005">
  <presentation label="SA01">
    <flow class="Block">
      <material>
        <mattext>Popište jak startuje auto</mattext>
      </material>
      <response_str ident="STR07" rcardinality="Ordered">
        <render_fib fibtype="String" rows="10" columns="20">
          <response_label ident="STR07_A" rshuffle="No"/>
        </render_fib>
      </response_str>
    </flow>
  </presentation>
</item>
```

Otázka Ordering objects. Třída pro tuto otázku má jméno `OrderingObjectsQuestion`. Stejně jako True/False otázka má k dispozici seznam možných odpovědí a dále příznaky o orientaci odpovědí a shuffle. Vlastní informace o správném pořadí jednotlivých odpovědí je uložena v podmínkové části.



Obrázek 6.8: Otázka Ordering objects

Obrázku 6.8 je příslušný tento XML kód:

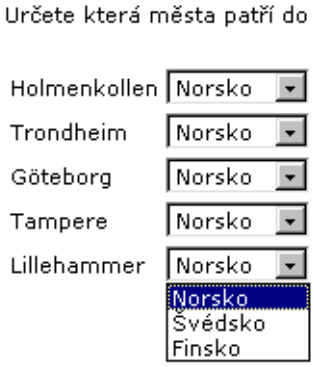
```
<item title="var" ident="ITEM006">
  <presentation label="OO01">
```

```
      <flow class="Block">
        <material>
          <mattext>Uspořádejte dny v týdnu</mattext>
        </material>
        <response_lid ident="LID08" rcardinality="Ordered">
          <render_choice shuffle="Yes">
            <response_label ident="LID08_A" rshuffle="Yes">
              <flow_mat class="Block">
                <material>
                  <mattext>Úterý</mattext>
                </material>
              </flow_mat>
            </response_label>
            <response_label ident="LID08_B" rshuffle="Yes">
              <flow_mat class="Block">
                <material>
                  <mattext>Neděle</mattext>
                </material>
              </flow_mat>
            </response_label>
            ...
            <response_label ident="LID08_G" rshuffle="Yes">
              <flow_mat class="Block">
                <material>
                  <mattext>Středa</mattext>
                </material>
              </flow_mat>
            </response_label>
          </render_choice>
        </response_lid>
      </flow>
    </presentation>
  </item>
```

Otázka Matching objects má přidruženu třídu `ConnectingObjectsQuestion`. Obsahuje seznam odpovědí a seznam možností, kam odpověď zařadit. Jak správně přiřadit odpověď konkrétní možnosti, je opět uloženo v podmínkové části.

XML kód pro otázku Matching objects z obrázku 6.9 je:

```
<item title="geo" ident="ITEM007">
  <presentation label="CO01">
    <flow class="Block">
      <material>
        <mattext>Určete která města patří do jakých států.</mattext>
      </material>
      <response_lid ident="LID09" rcardinality="Single">
        <material>
```

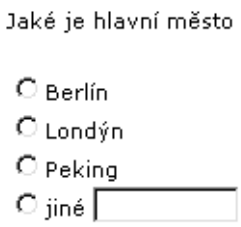


Obrázek 6.9: Otázka Matching objects

```
    <mattext>Holmenkollen</mattext>
  </material>
  <render_choice shuffle="Yes">
    <response_label ident="LID09_A" rshuffle="Yes">
      <material>
        <mattext>Norsko</mattext>
      </material>
    </response_label>
    <response_label ident="LID09_B" rshuffle="Yes">
      <material>
        <mattext>Švédsko</mattext>
      </material>
    </response_label>
    <response_label ident="LID09_C" rshuffle="Yes">
      <material>
        <mattext>Finsko</mattext>
      </material>
    </response_label>
  </render_choice>
</response_lid>
<response_lid ident="LID10" rcardinality="Single">
  <material>
    <mattext>Trondheim</mattext>
  </material>
  <render_choice shuffle="Yes">
    <response_label ident="LID10_A" rshuffle="Yes">
      <material>
        <mattext>Norsko</mattext>
      </material>
    </response_label>
    ...
  </render_choice>
```

```
      </response_lid>
      ...
      <response_lid ident="LID13" rcardinality="Single">
        <material>
          <mattext>Lillehammer</mattext>
        </material>
        <render_choice shuffle="Yes">
        ...
        </render_choice>
      </response_lid>
    </flow>
  </presentation>
</item>
```

Otázka Combined. Tato otázka je vlastně otázka Multiple choice doplněná o textové políčko. Je uložena v třídě `CombinedQuestion`. Oproti Multiple choice otázce má tedy navíc informaci o velikosti textového políčka a také o materiálu, kterým je textové políčko uvozeno.



Obrázek 6.10: Otázka Combined

Odpovídající XML kód pro obrázek 6.10 je:

```
<item title="geo" ident="ITEM008">
  <presentation label="CB01">
    <flow class="Block">
      <material>
        <mattext>Jaké je hlavní město </mattext>
        <matemtext>Francie?</matemtext>
      </material>
      <response_lid ident="LID14" rcardinality="Single">
        <render_choice shuffle="Yes">
        ...
        </render_choice>
      </response_lid>
      <response_str ident="STR15" rcardinality="Single">
        <render_fib maxchars="10" prompt="Box" fibtype="String">
          <material>
```

```
            <mattext>jiné</mattext>
          </material>
          <response_label ident="STR15_D" rshuffle="No"/>
        </render_fib>
      </response_str>
    </flow>
  </presentation>
</item>
```

Dalším krokem transformace je uložení podmínkové části otázky – Bronchus umožňuje vytvořit pro každou otázku sérii podmínek, které slouží k obodování otázky a zobrazení odpovídající reakce. Tyto podmínky je nutné překonvertovat tak, aby odpovídali specifikaci QTI. Možné jsou tyto 3 typy podmínek:

1. podmínky pro zachycení pravdivosti zvolené odpovědi

   Například:

   ```
   (A==true && B==true) || C==false
   ```

2. podmínky pro testování odpovědi na konkrétní hodnoty

   Například:

   ```
   A=="Aphex" || (B>7 && B<=10)
   ```

3. podmínky pro zachycení pořadí odpovědi nebo příslušnosti do určité skupiny

   Například:

   ```
   A==#3 && B==#2 && C==#1
   ```

Každý typ podmínky se hodí pro jinou otázku – zatímco první typ je určen pro otázky Multiple choice, Multiple response nebo True/False, druhý se hodí spíše pro Fill-in-blank otázku. O třetí typ se dělí otázky Ordering objects a Matching objects, i když u každé z nich je interpretace podmínky rozdílná. Combined otázka může mít kombinaci prvního a druhého typu.

Jak je vidět z ukázek, Bronchus pro jednoduchost používá v podmínkách pro označení odpovědi velké písmeno abecedy A–Z. Nicméně ve specifikaci QTI je pro identifikaci určitého XML elementu použit jeho atribut `ident`. Proto je prvním krokem konverze podmínky přemapování identifikátorů odpovědí na odpovídající hodnoty `ident` atributů, které byly zjištěny v průběhu ukládání těla otázky. Pokud toto vztáhneme na `ident` atributy z XML kódu uvedeného u otázky Multiple response, bude prvně uvedená podmínka přemapována na:

```
(LID02==LID02_A && LID02==LID02_B) || LID02!=LID02_C
```

Druhá podmínka při použití `ident` atributů z XML kódu u otázky Fill-in-blank na:

```
STR04=="Aphex" || (STR05>7 && STR05<=10)
```

A třetí při `ident` atributech z otázky Ordering objects na:

```
LID08==#3#LID08_A && LID08==#2#LID08_B && LID08==#1#LID08_C
```

A při `ident` atributech z otázky Matching objects na:

```
LID09==LID09_C && LID10==LID10_B && LID11==LID11_A
```

Dalším krokem je pak vlastní konverze do QTI formátu. K této konverzi byl použit nástroj `ANTLR`. Gramatika pro konverzi je uložena v balíku `cz.muni.fi.bronchusio.antlrparsers`. Finální konverze první podmínky do XML kódu tedy bude následující:

```
<conditionvar>
  <or>
    <and>
      <varequal respident="LID02" case="Yes">LID02_A</varequal>
      <varequal respident="LID02" case="Yes">LID02_B</varequal>
    </and>
    <not>
      <varequal respident="LID02" case="Yes">LID02_C</varequal>
    </not>
  </or>
</conditionvar>
```

Pro druhou podmínku bude výsledný XML kód tento:

```
<conditionvar>
  <or>
    <varequal respident="STR04" case="Yes">Aphex</vargt>
    <and>
      <vargt respident="STR05">7</vargt>
      <varlte respident="STR05">10</varlte>
    </and>
  </or>
</conditionvar>
```

A třetí podmínka bude mít v případě otázky Ordering objects tento XML kód:

```
<conditionvar>
  <and>
    <varequal respident="LID08" case="Yes" index="3">LID08_A</varequal>
    <varequal respident="LID08" case="Yes" index="2">LID08_B</varequal>
    <varequal respident="LID08" case="Yes" index="1">LID08_C</varequal>
  </and>
</conditionvar>
```

V případě otázky Matching objects tento XML kód:

```
<conditionvar>
  <and>
    <varequal respident="LID09" case="Yes">LID09_C</varequal>
```

```
    <varequal respident="LID10" case="Yes">LID10_B</varequal>
    <varequal respident="LID11" case="Yes">LID11_A</varequal>
  </and>
</conditionvar>
```

Spolu s konvertovanými podmínkami jsou uloženy i jim příslušné body a reakce. Zaznamenání bodů se řídí vyhodnocovacím algoritmem zvoleným pro test, jehož je otázka součástí. Které proměnné jsou deklarovány bude podrobněji popsáno v části Třída ScoreModel. V posledním kroku dojde k zápisu nápovědy a řešení otázky a také se zaznamenají příslušná metadata o otázce. Jak vypadá kompletní XML kód po transformaci třídy `Question`, uvádí Příloha B.

6.2.3 Třída SectionProcessor

Třída `SectionProcessor` slouží k převodu dat z vnitřní třídy `Section` do sekce specifikace QTI, která je reprezentována třídou `Section` z API pro QTI. Bronchus používá sekce pro implementaci mechanismu výběru a uspořádání, kdy pravidla pro tento mechanismus specifikovaná pro sekci automaticky platí pro všechny otázky, které sekce obsahuje. Proto se ve vnitřní třídě `Section` nacházejí informace vztahující se právě k mechanismu výběru a uspořádání: typ výběru, počet vybraných otázek ze sekce, zvolené uspořádání atd. Dále tato třída poskytuje seznam otázek a podsekcí, které se v dané sekci nacházejí.

Transformace tedy spočívá v uložení informací o mechanismu výběru a uspořádání a spuštění transformace pro příslušné otázky a podsekce. Zapsána jsou rovněž metadata o sekci. Příkladem transformovaného XML kódu je:

```
<section ident="SEC001">
  <selection_ordering>
     <selection>
       <selection_number>4</selection_number>
     </selection>
   <order order_type="Random"/>
  </selection_ordering>
  <item>...</item>
  <item>...</item>
  ...
  <item>...</item>
</section>
```

Pro kompletní XML kód po transformaci třídy `Section` viz. Příloha B.

6.2.4 Třída AssessmentProcessor

Transformaci vnitřní třídy `Assessment` provádí třída `AssessmentProcessor`. Transformuje ji na odpovídající třídu `Assessment` z API pro QTI. Vnitřní třída `Assessment` nese údaje o vyhodnocovacím algoritmu vybraném pro test: název zvoleného algoritmu, reakce

na výsledek algoritmu a podmínky pro zobrazení reakcí. Dále uchovává informaci o názvu testu a jeho časovém omezení. K dispozici je rovněž seznam sekcí, ze kterých je test složen.

Transformační proces lze rozdělit na dvě části. V té první dochází k zaznamenání údajů o vyhodnocovacím algoritmu. Opětovně je nutné překonvertovat podmínky pro zobrazení reakcí do podoby souhlasící se specifikací QTI. Formát, který používá Bronchus lze ilustrovat takto:

```
A>=3 && A<7
```

kde velké písmeno A slouží pro označení proměnné, se kterou pracuje vyhodnocovací algoritmus. Jména možných proměnných jsou uvedena v popisu třídy `ScoreModel`. Po konverzi podmínky do požadované formy je výsledek tento:

```
<test_variable>
  <and_test>
    <variable_test varname="A" testoperator="GTE">3</variable_test>
    <variable_test varname="A" testoperator="LT">7</variable_test>
  </and_test>
<test_variable>
```

Dále je podmínka uložena a spolu s ní také přidružená reakce. V druhé části konverze dojde k zapsání zbylých údajů o testu a příslušných metadat. Poté je spuštěna transformace pro sekce, které test obsahuje. Kompletní XML kód je opět možné najít v Příloze B.

6.2.5 Třída ObjectBankProcessor

Vnitřní třída `ObjectBank` obsahuje pouze seznam otázek, které mají byt zahrnuty do repositáře otázek. Činnost třídy `ObjectBankProcessor`, která ji převádí na třídu `ObjectBank` z API pro QTI, tedy spočívá pouze ve spuštění transformace otázek z tohoto seznamu. Následně jsou opět zapsána metadata o vytvořeném repositáři otázek.

6.2.6 Třída ScoreModel

V průběhu transformace jednotlivých částí ASI modelu je nutné mezi jednotlivými transformačními třídami sdílet informaci o zvoleném vyhodnocovacím algoritmu, především o proměnné s ním spojené. K tomu slouží třída `ScoreModel`, která je transformačním třídám předávána v jejich konstruktoru a má tyto metody:

- `setModelName`, `getModelName` – metody pro nastavení a zjištění jména vyhodnocovacího algoritmu.
- `setVarName`, `getVarName` – metody pro nastavení a zjištění jména přidružené proměnné.
- `setVarType`, `getVarType` – metody pro nastavení a zjištění typu proměnné.

- `setDefaultValue`, `getDefaultValue` – metody pro nastavení a zjištění standardní hodnoty pro proměnnou.
- `setMinValue`, `getMinValue` – metody pro nastavení a zjištění minimální možné hodnoty pro proměnnou.
- `setMaxValue`, `getMaxValue` – metody pro nastavení a zjištění maximální možné hodnoty pro proměnnou.
- `setOperation`, `getOperation` – název operace, která se má s proměnnou provést při ohodnocování otázky (například *SET* nebo *ADD*).

Bronchus používá pouze interní vyhodnocovací algoritmy specifikace QTI. Ty jsou spřaženy se těmito proměnnými:

Algoritmy NumberCorrect a WeightedNumberCorrect používají pro uložení svého výsledku proměnnou *COUNT* typu Integer, pro ohodnocení otázek proměnnou *CORRECT* typu Boolean.

Algoritmy SumofScores, WeightedSumofScores a BestKofN používají jak pro výsledek, tak pro obodování otázek, proměnnou *SCORE* typu Integer.

Stejné proměnné používají také Attempted verze vyhodnocovacích algoritmů.

6.3 Balík cz.muni.fi.bronchusio.loader

Balík `cz.muni.fi.bronchusio.loader` obsahuje třídy pro transformaci dat z XML souboru odpovídajícímu specifikaci QTI do vnitřních tříd editoru Bronchus. Spojení s editorem zde zajišťuje třída `Loader` pomocí dvou metod:

Metoda `load(File file)` slouží k načtení XML souboru, který byl v minulosti uložen editorem Bronchus. Vrací vnitřní třídu typu `Repository`, konkrétně typ `Assessment` nebo `ObjectBank`. Pokud se při načítání XML souboru zjistí, že nebyl uložen editorem Bronchus, je vyvolána výjimka `CreatedByOtherToolException`.

Metoda `importQuestions(File file)` také slouží k načtení XML souboru. Hlavním rozdílem však je, že tento soubor nemusel být uložen editorem Bronchus. Tímto způsobem lze tedy načíst jakýkoliv XML soubor, který odpovídá specifikaci QTI. Ze souboru jsou vybrány pouze ty otázky, se kterými umí Bronchus pracovat a jejich seznam je metodou vrácen.

Pokud soubor pro načtení není možné nalézt, je zmíněnými metodami vyvolána výjimka `FileNotFoundException`. V případě, že soubor neodpovídá specifikaci QTI, je vyvolána výjimka `NonIMSXMLException`.

6.3.1 Rozhraní LoaderProcessor a třída LoaderProcessorFactory

Stejně, jako v případě transformace opačným směrem, bylo i zde pro transformační proces vytvořeno rozhraní. Jeho název je `LoaderProcessor` a opět disponuje jedinou metodou:

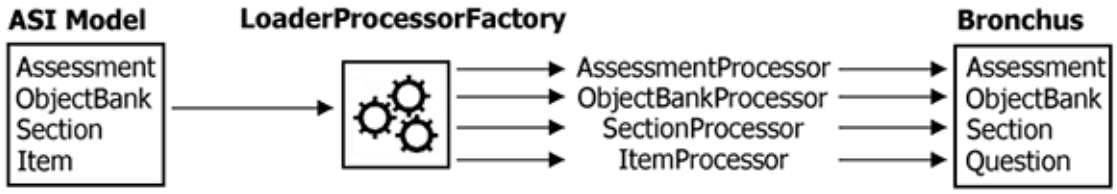
`process` – metoda spouštějící transformaci. Třídu ASI modelu přemění na odpovídající vnitřní třídu editoru, kterou pak metoda vrátí.

Každá třída ASI modelu může být transformována vhodnou implementací tohoto rozhraní. Důvody pro zavedení tohoto rozhraní jsou identické s důvody popsanými v části Rozhraní SaverProcesor a třída SaverProcesorFactory.

Toto rozhraní implementuje abstraktní třída `AbstractLoaderProcessor`, která obsahuje společné konstruktory a metody pro její 4 potomky – třídy odpovídající jednotlivým třídám ASI modelu specifikace QTI:

- `AssessmentProcessor`,
- `ObjectBankProcessor`,
- `SectionProcessor`,
- `ItemProcessor`.

Konkrétní instance třídy ASI modelu, se kterou mají pracovat, jim je předána v jejich konstruktoru.



Obrázek 6.11: Schéma práce třídy LoaderProcessorFactory

Podobně jako pro rozhraní `SaverProcessor` a ze stejných důvodů, které byly uvedeny v části Rozhraní SaverProcesor a třída SaverProcesorFactory, byla i zde zavedena třída `LoaderProcessorFactory`. Její statická metoda `getLoaderProcessor` převezme jako svůj parametr obecnou instanci třídy ASI modelu a vrátí k ní náležející třídu implementující

rozhraní `LoaderProcessor`. Pokud se typ třídy ASI modelu nerozezná, vrací metoda hodnotu `null`.

Uvedené třídy společně s rozhraním `LoaderProcessor` jsou umístěny v balíku `cz.muni.fi.bronchusio.loader.processor`. Protože se jedná o obrácený postup již popsané transformace vnitřních tříd na třídy ASI modelu, bude popis činnosti jednotlivých tříd omezen pouze na třídu `ItemProcessor`. V balíku se navíc nachází ještě třída `GlobalInfo`, jejíž funkce bude osvětlena v části Třída GlobalInfo.

6.3.2 Třída ItemProcessor

Třída `ItemProcessor` převádí data z třídy `Item` do vnitřní třídy `Question`. Důležitým krokem této transformace je rozpoznání typu otázky, která je v třídě `Item` uložena. To se děje dvěma způsoby:

1. Nejprve se provádí zjištění typu otázky z uložených metadat, pokud jsou přítomna.
2. Pokud metadata nejsou nalezena, provede se heuristická analýza obsahu třídy `Item`. Na základě zjištěných znaků obsahu se pak rozhodne o typu otázky. Pokud jsou o typu nějaké pochybnosti, je otázka vrácena jako neidentifikovaná.

Na základě zjištěného typu otázky je vytvořena instance potomka třídy `Question`. Další postup je roven reverznímu postupu transformace popsanému v části Třída QuestionProcessor. Pokud při transformaci těla otázky dojde k nesrovnalostem v jeho struktuře, například vinou špatného detekování typu otázky, je vyvolána interní výjimka a otázka se dále nezpracovává.

6.3.3 Třída GlobalInfo

Při načítání testu je nutné editoru Bronchus sdělit, zda v jsou něm přítomny reakce, nápovědy, řešení a časová omezení. Tyto informace se musí „sesbírat“ při procesu transformace a opět se k tomu využívá sdílená instance třídy. Tou je třída `GlobalInfo` a transformačním třídám se předává v jejich konstruktoru. Poskytuje tyto metody:

- `setFeedbackPresented`, `isFeedbackPresented` – metody pro nastavení a zjištění příznaku přítomnosti reakce.
- `setHintPresented`, `isHintPresented` – metody pro nastavení a zjištění příznaku přítomnosti nápovědy.
- `setSolutionPresented`, `isSolutionPresented` – metody pro nastavení a zjištění příznaku přítomnosti řešení.
- `setDurationPresented`, `isDurationPresented` – metody pro nastavení a zjištění příznaku přítomnosti časového omezení.

Podle nastavených příznaků se pak nastaví příslušné atributy vnitřní třídy `Assessment`.

6.4 Balík cz.muni.fi.bronchusio.common

V balíku `cz.muni.fi.bronchusio.common` se nalézají třídy, které jsou využívány všemi transformačními třídami při transformačním procesu.

6.4.1 Třída Utility

Třída `Utility` obsahuje statické metody, používané především ke konverzím.

Metoda `saveMaterialTag` slouží k uložení materiálu pod příslušný element. Bronchus předává materiálovou informaci ve formátu, který mísí text s instrukcemi o změně materiálu. Instrukce mohou být dvojího typu:

1. změna zvýraznění textu, která má formát:
   `text1 <i>zvýrazněný text</i> text2`
2. změna typu materiálu s formátem:
   `text1 <img src="..." alt="soubor" typ="typ materiálu"> text2`

V prvním případě se provede konverze do tohoto formátu:

```
<material>
  <mattext>text1 </mattext>
  <matemtext>zvyrazněný text</matemtext>
  <mattext> text2</mattext>
</material>
```

V případě druhém je konverze tato:

```
<material>
  <mattext>text1 </mattext>
  element pro typ materiálu
  <mattext> text2</mattext>
</material>
```

kde *element pro typ materiálu* odpovídá

- `<mataudio uri="soubor"/>`, pokud je typ materiálu roven *audio*
- `<matimage uri="soubor"/>`, pokud je typ materiálu roven *image*
- `<matvideo uri="soubor"/>`, pokud je typ materiálu roven *video*

Tato konverze úzce spolupracuje s třídou `FileWork`, která bude popsána dále. Pro opačnou konverzi je určena metoda `getTextFromMat`.

Metoda `getDurationInISO` provádí další typ konverze, tentokrát konverzi časového omezení. Bronchus předává údaj o časovém omezení jako časový údaj v minutách. Pro specifikaci QTI je nutné tento údaj překonvertovat do formátu podle normy ISO8601[1]. Formát pro časový úsek odpovídá tomuto řetězci:

`PTnHnMnS`

kde *n* je číslo, které udává korespondující časové složky – *H* odpovídá hodinám, *M* minutám a *S* sekundám. Tedy například 123minut je překonvertováno na řetězec:

`PT2H3M0S`

Opačnou konverzi je možné provést metodou `getDurationFromISO`.

Kromě konverzních metod poskytuje třída `Utility` ještě další metody:

- `saveMetadata`, `loadMetadata` – metody pro uložení a zjištění metadat pro určenou část ASI modelu.
- `saveScoreModel` – metoda pro uložení informací o vyhodnocovacím algoritmu získaných z instance třídy `ScoreModel`.

6.4.2 Třída FileWork

Materiál může zahrnovat odkazy na externí soubory, které je nutno překopírovat k ukládanému XML souboru. Při transformačním procesu je tedy potřeba každý odkazovaný soubor zaznamenat. Pro tento účel byla vytvořena třída `FileWork`, jejíž instance je opět sdílena mezi transformačními třídami, pomocí jejich konstruktoru. Třída `FileWork` disponuje následujícími třídami:

Obrázek 6.12: Struktura vytvořeného adresáře

1. <http://www.iso.org/iso/en/prods-services/popstds/datesandtime.html>

- `setPath`, `getPath` – metody pro nastavení a zjištění cesty, kam se bude XML soubor ukládat.
- `createPath` – metoda, která zjistí, zda cílová cesta existuje a pokud ne, pak ji vytvoří.
- `addToQueue` – metoda pro přidání souboru a jeho typu do seznamu. Tato metoda vrací URI podle RFC1630[2] reprezentující novou cestu po překopírování souboru.
- `startCopy` – metoda spouštějící proces kopírování souborů uložených v seznamu. V nastavené cestě se vytvoří adresářová struktura podle typu souboru, jak je ilustrováno obrázkem 6.12 (nastavená cesta je `testdir` a ukládaným souborem je `text.xml`).
- `getErrorList` – metoda, vracející třídu implementující rozhraní `Map`. Třída obsahuje seznam souborů, které nešly zkopírovat spolu s důvodem, proč tomu tak bylo.
- `clearErrorList` – metoda, která smaže chybový seznam.
- `correctPath` – metoda převádějící cestu z formátu URI na obecně používaný tvar.
- `setKeepExistingFiles`, `isKeepExistingFiles` – metody na nastavení a zjištění příznaku nepřepisování existujících souborů při kopírování.

2. <http://rfc.net/rfc1630.html>

Kapitola 7

Závěr

Při návrhu aplikace, která bude sloužit pro tvorbu a zpracování testových dat, je důležitým krokem výběr vhodného formátu reprezentující tato data. Tato práce popisovala tvorbu API pro existující formát testových dat podle specifikace IMS QTI verze 1.2.1 (viz. kapitola 5). API bylo tvořeno s ohledem na možné budoucí inovace tohoto formátu. Bohužel verze 2.0 specifikace QTI přinesla tak zásadní změny tohoto formátu, že rozšíření navrhnutého API na tuto verzi není možné. Nicméně lze API použít při vytváření nástroje pro konverzi dat mezi těmito verzemi.

Díky implementovaným vstupním a výstupním operacím pomocí tohoto API (viz. kapitola 6), byla zprostředkována editoru Bronchus plná podpora specifikaci QTI verze 1.2.1. Navrhnutý systém ukládání a načítání je samozřejmě možné dále rozšířit. Například o práci s „balíčky“ podle další specifikace organizace IMS, kterou je Content Packaging. Pro vytvoření takového „balíčku“ je možné okamžitě převzít uložená data spolu s vytvořenou adresářovou strukturou, která vznikla při jejich ukládání. Naopak pro modifikaci dat v „balíčku“ stačí pouze zpřístupnit jeho obsah.

Literatura

[1] Smythe, C., Shepherd, E., Brewer, L., Lay, S.: *IMS Question & Test Interoperability: ASI Information Model, Final Specification, Version 1.2*, 2002, Dokument dostupný na URL <http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_infov1p2.html> (květen 2005). 4.3

[2] Smythe, C., Shepherd, E., Brewer, L., Lay, S.: *IMS Question & Test Interoperability: ASI XML Binding Specification, Final Specification, Version 1.2*, 2002, Dokument dostupný na URL <http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_bindv1p2.html> (květen 2005). 4.5, 5

[3] Smythe, C., Shepherd, E., Brewer, L., Lay, S.: *IMS Question & Test Interoperability: ASI Outcomes Processing Specification, Final Specification, Version 1.2*, 2002, Dokument dostupný na URL <http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_outv1p2.html> (květen 2005). 4.5, 4.6

[4] Smythe, C., Shepherd, E., Brewer, L., Lay, S.: *IMS Question & Test Interoperability: ASI Selection & Ordering Specification, Final Specification, Version 1.2*, 2002, Dokument dostupný na URL <http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_saov1p2.html> (květen 2005). 4.7

[5] Smythe, C., Shepherd, E., Brewer, L., Lay, S.: *IMS Question & Test Interoperability: ASI Best Practice & Implementation Guide, Final Specification, Version 1.2*, 2002, Dokument dostupný na URL <http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_bestv1p2.html> (květen 2005). 5.5

[6] Lay, S.: *IMS Question & Test Interoperability: Migration Guide, Final Specification, Version 2.0*, 2005, Dokument dostupný na URL <http://www.imsglobal.org/question/qti_v2p0/imsqti_migrv2p0.html> (květen 2005). 4.9

[7] Udržal, T.: *Aplikace pro tvorbu otázek a testů [Diplomová práce]*, Fakulta informatiky Masarykovy univerzity v Brně, 2005. 6

[8] Drášil, P., Bažant, I., Šimák, B., Pitner, T.: *Relevantní standardy v oblasti e-Learningu*, CESNET, 2004, Dokument dostupný na URL <http://www.cesnet.cz/doc/techzpravy/2004/elearning/elearning24.pdf> (květen 2005).

Rejstřík

Příloha A

Seznam tříd API

AbstractIMSTag

AltMaterial

Assessment – atribut `ident` tvořen automaticky:

```
URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:A_SAMPLE001
```

BadIMSConstructionException

Condition

ConditionVar

Control

Feedback – atribut `ident` tvořen automaticky:

```
A_SAMPLE001_AFBK01, S_SAMPLE001_SFBK01
```

Flow

FlowLabel

FlowMat

Global

Hint

Interpretvar

Item – atribut `ident` tvořen automaticky:

```
URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:S_SAMPLE001
```

ItemFeedback – atribut `ident` tvořen automaticky:

```
I_SAMPLE001_IFBK01
```

ItemMetadata

ItemRubric

Material

NonIMSXMLException

ObjectBank – atribut `ident` tvořen automaticky:

```
URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:B_SAMPLE001
```

Objectives

Objects

ObjectsCondition

Order

OutComes

OutComesFeedbackTest

OutComesProcessing

Params

Presentation

PresentationMaterial

QTI

QTIMetadata

QTIMetadataField

Reference

Render

RequiredAttributeMissingException

RespCondition

Response – atribut `ident` tvořen automaticky:

```
LID01, XY01, STR01, NUM01, GRP01
```

ResponseLabel – atribut `ident` tvořen automaticky:

```
LID01_A, LID01_B, STR01_A, NUM01_C, GRP01_A ...
```

ResProcessing

Rubric

Section – atribut `ident` tvořen automaticky:

```
URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:S_SAMPLE001
```

Selection

SelectionOperator

SelectionOrdering

SHMaterial

Solution

Test

WrongAttributeValueException

Příloha B

Ukázka transformovaného XML kódu

```
<?xml version=”1.0” encoding=”utf-8”?>
<!DOCTYPE questestinterop SYSTEM ”http://www.imsglobal.org/question/qtiv1p2p1/
XMLSchemav1p2p1/xmla/ims_qtiasiv1p2p1schema/dtds/qtiasifullncdtd/ims_qtiasiv1p2p1
.dtd”>

<questestinterop>
 <assessment ident=”URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:A_SAMPLE001”
 title=”sample”>
  <qtimetadata>
   <qtimetadatafield>
    <fieldlabel>qmd_toolvendor</fieldlabel>
    <fieldentry>Bronchus</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_feedbackspermitted</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_hintspermitted</fieldlabel>
    <fieldentry>No</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_solutionspermitted</fieldlabel>
    <fieldentry>No</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_itemselection</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_itemsequence</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_sectionselection</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
```

```
  <fieldlabel>qmd_sectionsequence</fieldlabel>
  <fieldentry>Yes</fieldentry>
 </qtimetadatafield>
</qtimetadata>
<assessmentcontrol feedbackswitch="Yes" hintswitch="No" solutionswitch="No"
view="Candidate"/>
<outcomes_processing scoremodel="SumOfScores">
 <outcomes>
  <decvar varname="SCORE" vartype="Integer" defaultval="0"/>
 </outcomes>
 <outcomes_feedback_test>
  <test_variable>
   <variable_test varname="SCORE" testoperator="EQ">3</variable_test>
  </test_variable>
  <displayfeedback feedbacktype="Response" linkrefid="A_SAMPLE001_AFBK01"/>
 </outcomes_feedback_test>
 <outcomes_feedback_test>
  <test_variable>
   <variable_test varname="SCORE" testoperator="LT">3</variable_test>
  </test_variable>
  <displayfeedback feedbacktype="Response" linkrefid="A_SAMPLE001_AFBK02"/>
 </outcomes_feedback_test>
</outcomes_processing>
<assessfeedback ident="A_SAMPLE001_AFBK01" view="Candidate">
 <flow_mat class="Block">
  <material>
   <mattext charset="utf-8" texttype="text/plain" xml:space="default">
   Postupuješ dál</mattext>
  </material>
 </flow_mat>
</assessfeedback>
<assessfeedback ident="A_SAMPLE001_AFBK02" view="Candidate">
 <flow_mat class="Block">
  <material>
   <mattext charset="utf-8" texttype="text/plain" xml:space="default">
   Neprošel</mattext>
  </material>
 </flow_mat>
</assessfeedback>
<section ident="URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:S_SAMPLE001">
 <qtimetadata>
  <qtimetadatafield>
   <fieldlabel>qmd_feedbackspermitted</fieldlabel>
   <fieldentry>No</fieldentry>
  </qtimetadatafield>
  <qtimetadatafield>
   <fieldlabel>qmd_hintspermitted</fieldlabel>
   <fieldentry>No</fieldentry>
```

```
 </qtimetadatafield>
 <qtimetadatafield>
  <fieldlabel>qmd_solutionspermitted</fieldlabel>
  <fieldentry>No</fieldentry>
 </qtimetadatafield>
 <qtimetadatafield>
  <fieldlabel>qmd_numberofitems</fieldlabel>
  <fieldentry>1</fieldentry>
 </qtimetadatafield>
 <qtimetadatafield>
  <fieldlabel>qmd_sectionsincluded</fieldlabel>
  <fieldentry>No</fieldentry>
 </qtimetadatafield>
</qtimetadata>
<sectioncontrol feedbackswitch="No" hintswitch="No" solutionswitch="No"
view="Candidate"/>
<outcomes_processing scoremodel="SumOfScores">
 <outcomes>
  <decvar varname="SCORE" vartype="Integer" defaultval="0"/>
 </outcomes>
</outcomes_processing>
<selection_ordering>
 <selection/>
 <order order_type="Sequential"/>
</selection_ordering>
<item title="chem" ident="URN:IMS-PLIRID-V1:FIMUNI:60200:qtiv1p2p1:I_SAMPLE001">
 <itemmetadata>
  <qtimetadata>
   <qtimetadatafield>
    <fieldlabel>qmd_toolvendor</fieldlabel>
    <fieldentry>Bronchus</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_timedependence</fieldlabel>
    <fieldentry>No</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_feedbackspermitted</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_solutionspermitted</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_hintspermitted</fieldlabel>
    <fieldentry>Yes</fieldentry>
   </qtimetadatafield>
```

```
   <qtimetadatafield>
    <fieldlabel>qmd_itemtype</fieldlabel>
    <fieldentry>Logical identifier</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_questiontype</fieldlabel>
    <fieldentry>Multiple-response</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_renderingtype</fieldlabel>
    <fieldentry>Choice</fieldentry>
   </qtimetadatafield>
   <qtimetadatafield>
    <fieldlabel>qmd_responsetype</fieldlabel>
    <fieldentry>Multiple</fieldentry>
   </qtimetadatafield>
  </qtimetadata>
</itemmetadata>
<itemcontrol feedbackswitch="Yes" hintswitch="Yes" solutionswitch="Yes"
view="Candidate"/>
<presentation label="MR01">
 <flow class="Block">
  <material>
   <mattext charset="utf-8" texttype="text/plain" xml:space="default">
   Které z následujících prvků jsou obsaženy ve vodě?</mattext>
  </material>
  <response_lid ident="LID01" rcardinality="Multiple" rtiming="No">
   <render_choice shuffle="Yes">
    <response_label ident="LID01_A" rshuffle="Yes" rarea="Rectangle">
     <flow_mat class="Block">
      <material>
       <mattext charset="utf-8" texttype="text/plain" xml:space="default">
       Vodík</mattext>
      </material>
     </flow_mat>
    </response_label>
    <response_label ident="LID01_B" rshuffle="Yes" rarea="Rectangle">
     <flow_mat class="Block">
      <material>
       <mattext charset="utf-8" texttype="text/plain" xml:space="default">
       Helium</mattext>
      </material>
     </flow_mat>
    </response_label>
    <response_label ident="LID01_C" rshuffle="Yes" rarea="Rectangle">
     <flow_mat class="Block">
      <material>
       <mattext charset="utf-8" texttype="text/plain" xml:space="default">
```

```
        Uhlík</mattext>
       </material>
      </flow_mat>
     </response_label>
     <response_label ident="LID01_D" rshuffle="Yes" rarea="Rectangle">
      <flow_mat class="Block">
       <material>
        <mattext charset="utf-8" texttype="text/plain" xml:space="default">
        Kyslík</mattext>
       </material>
      </flow_mat>
     </response_label>
     <response_label ident="LID01_E" rshuffle="Yes" rarea="Rectangle">
      <flow_mat class="Block">
       <material>
        <mattext charset="utf-8" texttype="text/plain" xml:space="default">
        Dusík</mattext>
       </material>
      </flow_mat>
     </response_label>
     <response_label ident="LID01_F" rshuffle="Yes" rarea="Rectangle">
      <flow_mat class="Block">
       <material>
        <mattext charset="utf-8" texttype="text/plain" xml:space="default">
        Chlór</mattext>
       </material>
      </flow_mat>
     </response_label>
    </render_choice>
   </response_lid>
  </flow>
</presentation>
<resprocessing scoremodel="SumOfScores">
 <outcomes>
  <decvar varname="SCORE" vartype="Integer" defaultval="0"/>
 </outcomes>
 <respcondition title="Autogenerated" continue="No">
  <conditionvar>
   <and>
    <varequal respident="LID01" case="Yes">LID01_A</varequal>
    <not>
     <varequal respident="LID01" case="Yes">LID01_B</varequal>
    </not>
    <not>
     <varequal respident="LID01" case="Yes">LID01_C</varequal>
    </not>
    <varequal respident="LID01" case="Yes">LID01_D</varequal>
    <not>
```

```
      <varequal respident="LID01" case="Yes">LID01_E</varequal>
     </not>
     <not>
      <varequal respident="LID01" case="Yes">LID01_F</varequal>
     </not>
    </and>
   </conditionvar>
   <setvar varname="SCORE" action="Add">3</setvar>
   <displayfeedback feedbacktype="Response" linkrefid="I_SAMPLE001_IFBK01"/>
  </respcondition>
  <respcondition continue="No">
   <conditionvar>
    <or>
     <not>
      <varequal respident="LID01" case="Yes">LID01_A</varequal>
     </not>
     <not>
      <varequal respident="LID01" case="Yes">LID01_D</varequal>
     </not>
    </or>
   </conditionvar>
   <setvar varname="SCORE" action="Add">-1</setvar>
   <displayfeedback feedbacktype="Response" linkrefid="I_SAMPLE001_IFBK02"/>
  </respcondition>
 </resprocessing>
 <itemfeedback ident="I_SAMPLE001_IFBK01" view="Candidate">
  <flow_mat class="Block">
   <material>
    <mattext charset="utf-8" texttype="text/plain" xml:space="default">
    Dobře</mattext>
   </material>
  </flow_mat>
 </itemfeedback>
 <itemfeedback ident="I_SAMPLE001_IFBK02" view="Candidate">
  <flow_mat class="Block">
   <material>
    <mattext charset="utf-8" texttype="text/plain" xml:space="default">
    Špatně</mattext>
   </material>
  </flow_mat>
 </itemfeedback>
 <itemfeedback ident="I_SAMPLE001_IFBK03" view="Candidate">
  <solution feedbackstyle="Complete">
   <solutionmaterial>
    <flow_mat class="Block">
     <material>
      <mattext charset="utf-8" texttype="text/plain" xml:space="default">
      Voda se skládá z vodíku a kyslíku.</mattext>
```

```
        </material>
       </flow_mat>
      </solutionmaterial>
     </solution>
    </itemfeedback>
    <itemfeedback ident="I_SAMPLE001_IFBK04" view="Candidate">
     <hint feedbackstyle="Complete">
      <hintmaterial>
       <flow_mat class="Block">
        <material>
         <mattext charset="utf-8" texttype="text/plain" xml:space="default">
         Voda se skládá ze 2 prvků.</mattext>
        </material>
       </flow_mat>
      </hintmaterial>
     </hint>
    </itemfeedback>
   </item>
  </section>
 </assessment>
</questestinterop>
```

Příloha C

Licenční ujednání knihovny dom4j

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name "DOM4J" must not be used to endorse or promote products derived from this Software without prior written permission of MetaStuff, Ltd. For written permission, please contact <dom4j-info@metastuff.com>.

4. Products derived from this Software may not be called "DOM4J" nor may "DOM4J" appear in their names without prior written permission of MetaStuff, Ltd. DOM4J is a registered trademark of MetaStuff, Ltd.

5. Due credit should be given to the DOM4J Project - <http://www.dom4j.org>

THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved.

Příloha D

Obsah přiloženého CD

`/dp.pdf`	diplomová práce ve formátu PDF
`/text/`	zdrojový text diplomové práce ve formátech DocBook a LaTeX
`/dist/`	zkompilované tvary API a balíku bronchusio
`/lib/`	knihovny pro API a balík bronchusio
`/doc/`	JavaDoc dokumentace API a balíku bronchusio
`/src/`	zdrojové kódy API a balíku bronchusio