České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Martin Hofman**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Ukládání geografických dat v NoSQL databázi**

Pokyny pro vypracování:

1. Seznamte se s jazykem GML a jeho aplikačním profilem AIXM5.1
2. Navrhněte způsob efektivní analýzy AIXM5.1 instančních souborů. Vezměte v úvahu jejich potenciálně velmi velký rozsah.
3. Navrhněte způsob efektivního ukládání AIXM 5.1 dat v databázi. Protože se množina zobrazovaných vlastností (features) může během životnosti databáze měnit, zaměřte se na vhodnou NoSQL databázi.
4. Při volbě databáze a tvorbě datového modelu zohledněte potřebu prostorové filtrace uložených dat.
5. Uložiště geografických dat implementujte.
6. Proveďte základní vyhodnocení rychlosti ukládání, modifikace a vyhledávání uložených dat.

Seznam odborné literatury:

Lake R., Burggraf D., Trninic M., Rae L.: "Geography Mark-Up Language - Foundation for the Geo-Web", John Wiley & Sons, 2004

Chodorow K.: "MongoDB: The Definitive Guide", O'Reilly, 2013

Vedoucí: doc.Ing. Zdeněk Kouba, CSc.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

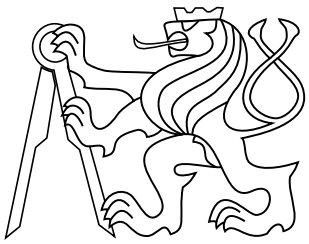
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 26. 3. 2015

bakalářská práce

Ukládání geografických dat v NoSQL databázi

Martin Hofman



Květen 2015

Doc. Ing. Zdeněk Kouba, CSc.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra počítačů

Poděkování

Na tomto místě bych rád poděkoval vedoucímu bakalářské práce Doc. Ing. Zdeňku Koubovi, CSc., Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych rád poděkoval své rodině za podporu během celého studia. Nakonec bych rád poděkoval spolužáků za rady v průběhu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt

Tato práce se zabývá syntaktickou analýzou AIXM 5.1 instančních souborů a jejich efektivním uložením do NoSQL databáze. V práci jsou nejprve rozebrány použitelné technologie a následně je proveden výběr nejvhodnějších. Pro syntaktickou analýzu byla zvolena kombinace technologií JAXB a StAXU. Pro trvalé uložení AIXM dat byla zvolena databáze MongoDB. Jako prostředek objektově-dokumentového mapování geografických objektů do databáze byla zvolena knihovna Morphia. Dále je v práci popsán návrh řešení a jeho implementace. Nakonec jsou uvedeny výsledky základních měření odezvy prostorových dotazů.

Klíčová slova

Geografická informace; aeronautická informace; AIXM 5.1; GML 3.2; NoSQL; JAXB; StAX; MongoDB

Abstrakt

This thesis deals with parsing of AIXM 5.1 files and their effective storage in a NoSQL database. In this thesis, we analyze appropriate technologies and subsequently the most suitable ones for implementation of individual tasks are selected. The combination of technologies JAXB and StaX was chosen for parsing AIXM 5.1 data files. MongoDB database was chosen as a means for building the AIXM 5.1 data storage. Morphia library was selected as a high level API offering object-document mapping for efficient storing/retrieving geographical features objects into/from the database. Further the thesis describes the design and implementation of the system that is subject of this thesis. Finally, the results of basic experiments aimed at the assessment of geographical queries response times.

Keywords

Geographical information; aeronautical information; AIXM 5.1; GML 3.2; NoSQL; JAXB; StAX; MongoDB

Obsah

Seznam obrázků

Seznam tabulek

Zkratky

API	Application Programming Interface
BJSON	Binary-JSON
CRUD	Create, Read, Update, Delete
CRS	Coordinate Reference System
EUROCONTROL	European Organisation for the Safety of Air Navigation
FAA	Federal Aviation Administration
GML	Geography Markup Language
Hibernate OGM	Hibernate Object/Grid Mapper
JAXB	Java Architecture for XML Binding
JDK	Java Development Kit
JSON	JavaScript Object Notation
JPA	Java Persistence API
JPQL	Java Persistence Query Language
NoSQL	Not only SQL
SQL	Structured Query Language
UUID	Universal Unique Identifier
XJC	Java Architecture for XML Binding Binding Compiler
XML	Extensible Markup Language

1 Úvod

Aeronautická data jsou data popisující polohu letišť, přistávacích, vzletových a pojížděcích drah, překážek leteckého provozu, heliportů a dalších objektů spjatých s leteckou dopravou. Pro výměnu těchto dat mezi institucemi podílejícími se na letovém provozu vyvinuly společně organizace EuroControl a FAA standard AIXM 5.1 (Aeronautical Information Exchange Model) postavený nad standardem GML 3.2. GML (Geography Mark-up Language) 3.2 slouží k popisu geografické informace a umožňuje odvozování vlastních geografických objektů. Oba standardy využívají pro reprezentaci dat značkovací jazyk XML (Extensible Markup Language) a k validaci těchto dat XML Schema. Data vyjádřená pomocí standardu AIXM 5.1 umožňují popsat geografické objekty (tzv. features) a jejich vlastnosti (tzv. properties). Příkladem takového geografického objektu je např. vzletová dráha (runway), příkladem její vlastnosti je její délka. AIXM 5.1 předpokládá vývoj dat v čase a proto zakódovanou indormaci vždy vztahuje k určitému časovému okamžiku (time slice). Pomocí systému časových řezů lze tedy postihnout životní cyklus geografických objektů a vývoj jejich vlastností v čase.

Standard AIXM 5.1 je zpracován velmi komplexně a je schopen poskytnout detailní popis. V současnosti se pro uchovávání aeronautické informace v praxi používají hlavně relační databáze. Je obtížné vytvořit obecný relační model odpovídající standardu AIXM 5.1 a zejména do něj převést data ze starších verzí. Tyto problémy způsobují, že obtížně proniká do praxe, přestože platná legislativa ukládá přechod na AIXM 5.1 zúčastněným institucím za povinnost.

Běžná praxe při použití relační databáze je vytvořit pro každý typ geografického objektu(feature) samostatnou tabulku. Protože lze očekávat další vývoj standardu, který přinese další typy geografických objektů, bude to znamenat změnu struktury všech relačních úložišť AIXM dat, což je časově i finančně náročná operace.

Cílem práce je namísto použití relačních databází navrhnout nové řešení, spočívající ve využití NoSQL databáze. NoSQL databází existuje mnoho druhů např. objektové, grafové, dokumentové a databáze založené na ukládání dvojic klíč-hodnota (key-value databases). V práci jsou popsány tři databáze a následně je vybrána nejvhodnější z nich. Dále je navržen způsob uložení aeronautických dat do NoSQL databáze a následné vyhledání geografických objektů podle jejich geografických souřadnic. NoSQL databáze, nabízejí díky své volnější struktuře potenciál pro výrazné snížení nákladů při změnách standardu, které by v případě relačního modelu vyžadovaly změnu struktury databáze.

V současnosti neexistují žádné volně dostupné technologie umožňující práci se standardem AIXM 5.1. V práci je navržena metoda syntaktické analýzy (parsování) AIXM 5.1 a její implementace. Dále se práce zabývá výběrem vhodné databáze, způsobem uložení aeronautických dat a vytvořením geografických indexů pro vyhledávání podle geografické polohy. Následuje ověření, zda lze efektivně uložit aeronautická data do NoSQL databáze a následně je rychle vyhledat. Nakonec jsou uvedeny výsledky měření, jak dlouho trvá vyhledávání podle geografické polohy v NoSQL databázi.

2 AIXM 5.1 a GML 3.2

2.1 XML - Extensible Markup Language

XML (Extensible Markup Language) je značkovací jazyk určený pro zápis dat ve formě strukturovaných dokumentů. Umožňuje definovat vlastní sadu značek. XML dokument se skládá z elementů tvořených značkami (tagy) a hodnotou, která je uzavřena mezi otevírací a ukončovací značkou příslušného elementu:

```
<element>hodnota</element>
```

nebo dalším elementem mezi nimi:

```
<element><element2>hodnota</element2></element>
```

Element může být prázdný a neobsahovat žádnou hodnotu:

```
<element></element> nebo ekvivalentně pomocí zkrácené syntaxe <element/>
```

Každý element může mít nula až n atributů:

```
<element atribut="hodnota"/>
```

Na začátku XML dokumentu je XML deklarace uvádějící verzi XML a kódování:

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML dokument může vypadat takto:

```
1 <?xml version="1.0"?>
2 <a att="a">
3         <b>
4                 <c>text1</c>
5         </b>
6 </a>
```

Aby byl dokument **správně strukturovaný** (well formed) musí splňovat následující podmínky:

- XML dokument musí mít jeden kořenový element.
- Všechny elementy musí mít počáteční a uzavírací značku, popřípadě prázdný element může používat zkrácenou syntaxi prázdného elementu (viz výše).
- XML značky rozlišují velká a malá písmena (jsou case sensitive). Tedy

```
<T> a <t>
```

jsou dvě různé značky.

- Všechny XML elementy musí být správně vnořené. To znamená, pokud je otevírací značka jistého vnitřního elementu umístěna uvnitř jiného (vnějšího) elementu, musí být i příslušná zavírací značka daného vnitřního elementu umístěna uvnitř vnějšího elementu.
- Hodnoty XML atributů musí být uzavřeny v uvozovkách.

2.1.1 XPath

XPath je jazyk sloužící k identifikaci uzlů a atributů XML dokumentů. Pracuje s XML dokumentem jako se stromem. Způsob zápisu je podobný zápisu cesty k souboru používané operačními systémy.

2.2 XML Schema

XML Schema slouží k popisu struktury XML dokumentů. Každé XML Schema je samo o sobě XML dokumentem a nevyžaduje tedy speciální syntaxi. Je možné ho vytvářet v obyčejném XML editoru. Obsahuje rozsáhlou sadu předem definovaných datových typů a umožňuje odvozování vlastních datových typů. Příklad XML Schema (xsd) souboru:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="student">
4      <xs:complexType>
5          <xs:sequence>
6              <xs:element name="jmeno" type="xs:string"/>
7              <xs:element name="prijmeni" type="xs:string"/>
8          </xs:sequence>
9          <xs:attribute name="id" type="xs:string"/>
10     </xs:complexType>
11  </xs:element>
12 </xs:schema>
```

Instanční XML soubor se v kořenovém elementu odkazuje na soubor obsahující XML schema.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:noNamespaceSchemaLocation="student.xsd"
    id="1">
    <jmeno>Martin</jmeno>
    <prijmeni>Hofman</prijmeni>
</student>
```

2.3 GML 3.2

GML 3.2 je značkovací jazyk definující sadu značek pomocí několika XML Schemat, určený k zápisu geografické informace. Každý geografický objekt je popsán svými vlastnostmi a jejich hodnotami. Hodnoty vlastností mohou být uvedeny přímo v hodnotě elementu (inline); (příklad převzat z [1]):

```
<app:Bridge gml:id="B1">
        <gml:centerOf>
                <gml:Point gml:id="P1" srsName="..">
                        <gml:pos>10 20</gml:pos>
                </gml:Point>
        </gml:centerOf>
</app:Bridge>
```

nebo odkazem (remote); (příklad převzat z [1]):

```
<app:Tower gml:id="B1">
        <gml:centerOf xlink="#P1"/>
</app:Tower>
```

Mezi jednotlivými geografickými objekty mohou být vztahy. Například most (bridge) vedoucí přes rokli (gorge); (příklad převzat z [1]):

```
<app:Bridge gml:id="B1">
        <gml:centerOf>
                <gml:Point gml:id="P1" srsName="..">
                        <gml:pos>10 20</gml:pos>
                </gml:Point>
        </gml:centerOf>
        <app:spans>
                <app:Gorge gml:id="G1">
                        <app:windth>250</app:windth>
                </app:Gorge>
        </app:spans>
</app:Bridge>
```

Jak je z příkladu patrné, most (bridge) ani rokle (gorge) nejsou definovány GML standardem. GML však umožňuje odvozovat vlastní geografické objekty. Tyto objekty musí být odvozeny od datového typu *AbstractFeatureType*. Každý geografický objekt může mít volitelně několik geometrických popisů (geometry) zpravidla v dvourozměrném nebo trojrozměrném prostoru. GML poskytuje následující geometrické vlastnosti definované ve schematech *feature.xsd* a *geometryAggregates.xsd*:

- *centerOf*
- *position*
- *extentOf*
- *centerLineOf*
- násobné varianty výše uvedených (*multiCenterOf*, *multiPosition*, *multiExtentOf*, *multiCenterLineOf*)
- *location*

Souřadnice mohou být zapsány v následujících formátech:

- Element *pos* se používá k zapsání přímé pozice jedné souřadnice

```
<gml:pos dimension="2" srsName="..">20 10</gml:pos>
```

  Atribut *dimension* je volitelný a jeho hodnota určuje dimenzi geografických souřadnic, atribut *srsName* je volitelný a obsahuje odkaz na použitý CRS (Coordinate Reference Systems).
- Element *coordinates* se používá k kompaktnímu zápisu souřadnic. Použitý CRS musí referencován z rodičovského elementu.

```
<gml:coordinates>0,3 6,3 6,0</gml:coordinates >
```

- Element *posList* slouží k zapsání seznamu souřadnic. Vzhledem k tomu, že souřadnice nemají separátor, je nutno uvést dimenzi.

```
<gml:posList dimension="2">20 10 10 24 13 26</gml:pos>
```

- Element *pointRep* slouží k odkázání na souřadnice definované jinde.

2.4 AIXM 5.1

Standard AIXM 5.1 rozšiřuje GML 3.2 o nově odvozené geografické objekty a jejich vlastnosti. Každý geografický objekt má lokálně unikátní id a identifier, který globálně určuje obsah geografického objektu. Důležitou součástí AIXM je časový model rozlišující následující typy časových řezů:

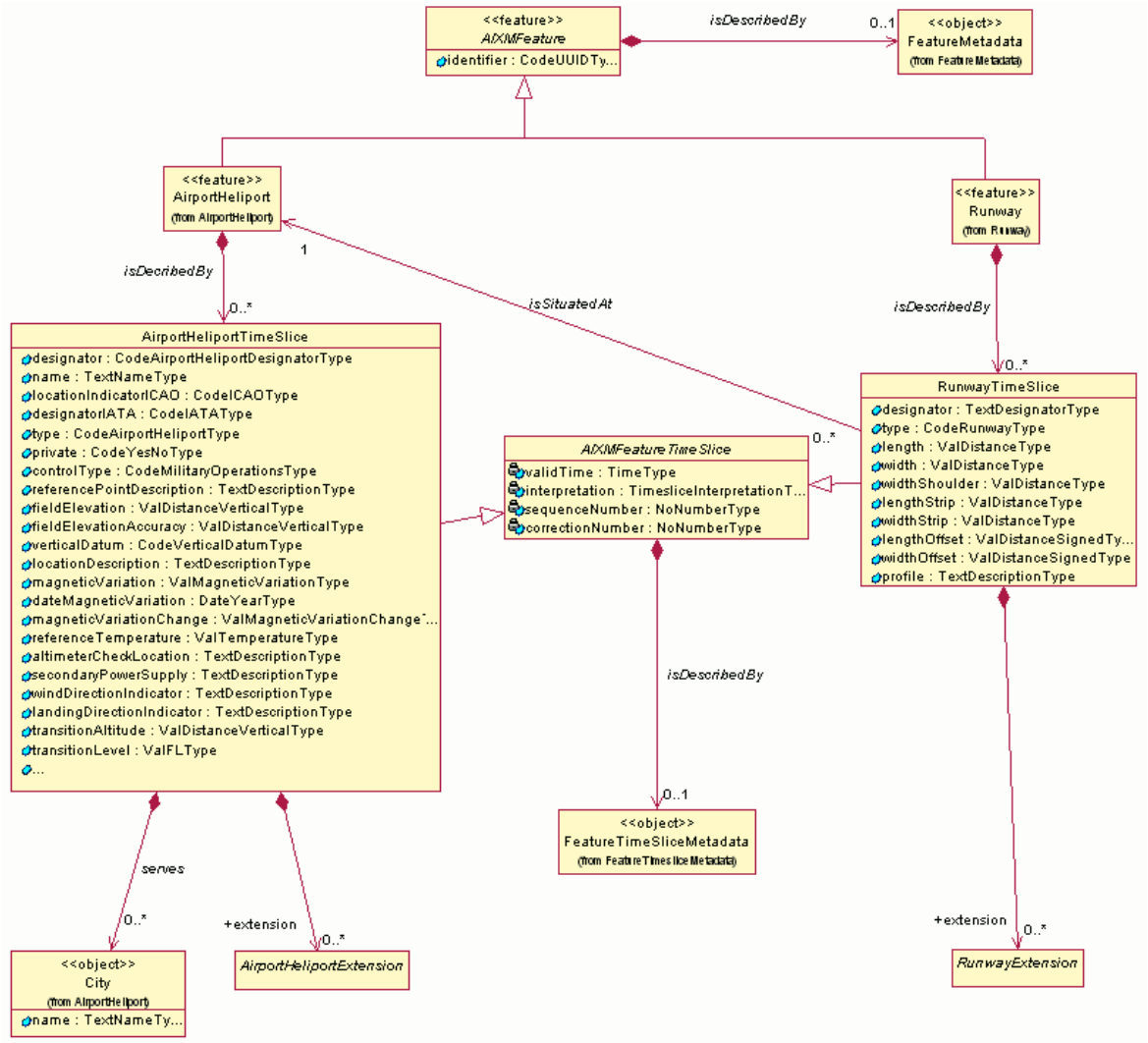
- BASELINE popisuje **stav** geografického objektu jako výsledek **trvalé** změny [2]
- PERMDELTA popisuje **změnu** geografického objektu v důsledku **trvalé** změny [2]
- TEMPDELTA popisuje **změnu** stavu po dobu trvání **dočasné** události - např. uzavření vzletové dráhy z důvodu její údržby. [2]
- SNAPSHOT popisuje **stav** k určitému času po aplikaci všech TEMPDELTA na odpovídající BASELINE. [2]

Dále časový řez obsahuje:

- *sequenceNumber* unikátní identifikátor časového úseku daného geografického objektu [2]

- *correctionNumber* slouží k úpravě již odeslaného časového řezu (má stejné sequenceNumber). Jako validní se bere časový úsek s nejvyšším correctionNumber. [2]

Struktura geografického objektu (v tomto případě se jedná o heliport) je znázorněna na následujícím obrázku:



Obrázek 1 Struktura geografického objektu AirportHeliport (převzato z [2])

Diagram tříd na obrázku 1 ukazuje, že každý geografický objekt (feature) dědí od abstrakní třídy *AIXMFeature*. Konkrétní geografické objekty jsou popsány hodnotami svých vlastností v příslušných časových řezech. Dále je z diagramu patrné, že každý geografický objekt může mít metadata a navíc může každý geografický objekt obsahovat rozšíření (extension) definované uživatelem.

Následující příklad ukazuje, jak může vypadat záznam heliportu v AIXM 5.1:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <aixm:AirportHeliport xmlns:xlink="http://www.w3.org/1999/xlink"
   ↪  xmlns:aixm="http://www.aixm.aero/schema/5.1"
   ↪  xmlns:gml="http://www.opengis.net/gml/3.2"
   ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ↪  gml:id="urn.uuid.dd062d88-3e64-4a5d-bebd-89476db9ebea"
   ↪  xsi:schemaLocation="http://www.aixm.aero/schema/5.1
   ↪  http://www.aixm.aero/schema/5.1/AIXM_Features.xsd">
3  <gml:identifier
   ↪  codeSpace="urn:uuid:">dd062d88-3e64-4a5d-bebd-89476db9ebea</gml:identifier>
4  <aixm:timeSlice>
5      <aixm:AirportHeliportTimeSlice gml:id="ahts1EADH">
6          <gml:validTime>
7              <gml:TimePeriod gml:id="vtnull0">
8                <gml:beginPosition>2009-01-01T00:00:00.000</gml:beginPosition>
9                <gml:endPosition indeterminatePosition="unknown"/>
10             </gml:TimePeriod>
11         </gml:validTime>
12         <aixm:interpretation>BASELINE</aixm:interpretation>
13         <aixm:sequenceNumber>1</aixm:sequenceNumber>
14         <aixm:correctionNumber>0</aixm:correctionNumber>
15         <aixm:featureLifetime>
16             <gml:TimePeriod gml:id="ltnull0">
17               <gml:beginPosition>2009-01-01T00:00:00.000</gml:beginPosition>
18               <gml:endPosition indeterminatePosition="unknown"/>
19             </gml:TimePeriod>
20         </aixm:featureLifetime>
21         <aixm:designator>EADH</aixm:designator>
22         <aixm:name>DONLON/DOWNTOWN HELIPORT</aixm:name>
23         <aixm:magneticVariation>-3</aixm:magneticVariation>
24         <aixm:dateMagneticVariation>1990</aixm:dateMagneticVariation>
25         <aixm:magneticVariationChange>0.03</aixm:magneticVariationChange>
26         <aixm:servedCity xsi:nil="true" nilReason="missing"/>
27         <aixm:responsibleOrganisation>
28             <aixm:AirportHeliportResponsibilityOrganisation gml:id="ID01">
```

```
29                  <aixm:role>OPERATE</aixm:role>
30                  <aixm:theOrganisationAuthority
                    ↪ xlink:title="DONLON_HELIPORT_AUTHORITY"
                    ↪ xlink:href="urn:uuid:74efb6ba-a52a-46c0-a16b-03860d356882"/>
31              </aixm:AirportHeliportResponsibilityOrganisation>
32          </aixm:responsibleOrganisation>
33          <aixm:ARP>
34              <aixm:ElevatedPoint gml:id="elpoint1EADH"
                ↪ srsName="urn:ogc:def:crs:EPSG::4326">
35                  <gml:pos>52.288888888888884 -32.035</gml:pos>
36                  <aixm:elevation uom="M">18.0</aixm:elevation>
37                  <aixm:geoidUndulation xsi:nil="true" nilReason="unknown"/>
38              </aixm:ElevatedPoint>
39          </aixm:ARP>
40          <aixm:availability xsi:nil="true" nilReason="withheld"/>
41          <aixm:annotation>
42              <aixm:Note gml:id="ID03">
43                  <aixm:propertyName xsi:nil="true" nilReason="missing"/>
44                  <aixm:translatedNote>
45                      <aixm:LinguisticNote gml:id="ID04">
46                          <aixm:note lang="fr-fr">Note en Français</aixm:note>
47                      </aixm:LinguisticNote>
48                  </aixm:translatedNote>
49              </aixm:Note>
50          </aixm:annotation>
51      </aixm:AirportHeliportTimeSlice>
52  </aixm:timeSlice>
53  </aixm:AirportHeliport>
```

Z příkladu patrné, že údaje o daném geografickém objektu jsou uložené v časovém řezu (time slice). Časový řez obsahuje element *validTime*, určující interval platnosti tohoto řezu. Element *featureLifeTime* říká, kdy byl daný geografický objekt vytvořen a kdy skončí jeho platnost. Dále je v tomto příkladu zapsána pozice heliportu jako souřadnice bodu reprezentujícího tento heliport na mapě. Atribut *gml:id* je lokální identifikátor určující identitu tohoto konkrétního geografického objektu v rámci daného souboru. Naproti tomu *gml:identifier* je globální identifikátor daného geografického objektu bez ohledu na hranice souboru určuje jeho obsah. K dosažení zaručené jednoznačnosti globálního identifikátoru, se používá algoritmu pro generování UUID (Universal Unique Identifier). Specifikace AIMX 5.1 doporučuje pro lokální identifikátor geografických objektů *gml:id* používat rovněž UUID.

3 Analýza použitelných technologií

3.1 Parsery xml

3.1.1 SAX

Princip: SAX (Simple API for XML) je rozhraní založené na událostech, které nastávají při sekvenčním čtení zpracovávaného XML dokumentu. V průběhu zpracování souboru jsou generovány následující události:

- počátek dokumentu
- konec dokumentu
- načtena počáteční značka elementu
- načtena koncová značka elementu
- načtena (textová) hodnota elementu
- načtena instrukce pro zpracování

SAX parser je prostředí jazyka Java používán tak, že vytvoříme třídu dědící od třídy *DefaultHandler* a následně překryjeme (override) metody, jež odpovídají událostem, na které chceme reagovat (název metody odpovídá požadované události). Tyto metody implementují požadovanou logiku, obsluhující příslušnou událost.

Příklad: Pokud máme následující XML soubor:

```
1 <?xml version="1.0"?>
2 <a att="a">
3   <b>
4     <c>text1</c>
5   </b>
6 </a>
```

Postupně budou vyvolány následující metody (u metod nejsou zapsány parametry):

1. startDocument() - začátek dokumentu
2. startElement() - začátek elementu a
3. startElement() - začátek elementu b
4. startElement() - začátek elementu c
5. characters() - zavolá se pro znaková data text1
6. endElement() - konec elementu c
7. endElement() - konec elementu b
8. endElement() - konec elementu a
9. endDocument() - konec dokumentu

Výhody a nevýhody

+ jednoduché použití
+ rychlost
+ nenáročnost na paměť
− pouze jednosměrné sekvenční procházení vstupního souboru
− z hlediska programátora pracné

3.1.2 StAX - Streaming API for XML

Princip: Na rozdíl od SAXu řídí sekvenční pohyb po dokumentu programátor pomocí metod *next* a *hasNext*. StAX definuje rozhraní *XMLStreamReader* (jsou uvedeny pouze nejpodstatnější metody):

```
1 public interface XMLStreamReader {
2     public int next() throws XMLStreamException;
3     public boolean hasNext() throws XMLStreamException;
4     public String getText();
5     public String getLocalName();
6     public String getNamespaceURI();
7 }
```

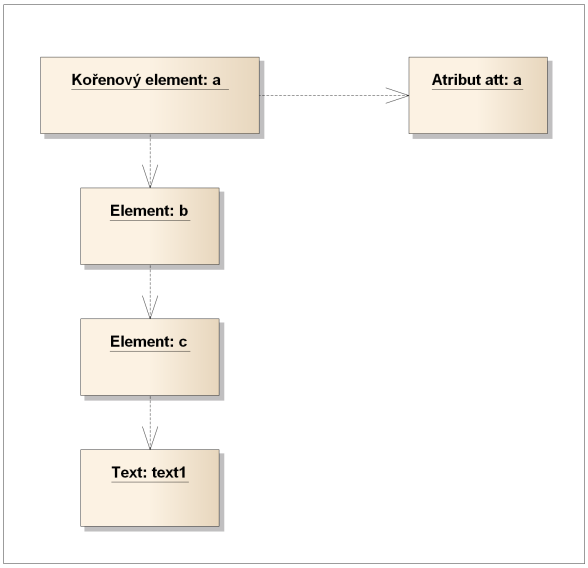
Výhody a nevýhody

+ programátor řídí zpracování
+ rychlost
+ nenáročnost na paměť
+ nepotřebné elementy mohou být ignorovány
− pouze jednosměrné sekvenční procházení vstupního souboru
− z hlediska programátora pracné

3.1.3 DOM

Princip: DOM (Document Object Model) je objektová reprezentace XML dokumentu ve formě stromu. DOM umožňuje přistupovat k uzlům stromu kdykoliv, v jakémkoliv pořadí a vyhledávat uzly stromu na základě jejich *id*.

Příklad: Na následujícím je obrázku je vidět DOM reprezentace XML dokumentu z příkladu SAX parseru:



Obrázek 2 Dom reprezentace XML dokumentu

Jak je vidět na obrázku 2, struktura vytvořeného stromu je zcela intuitivní. Danému elementu XML souboru odpovídá uzel *Element* mající následníky reprezentující vnořené elementy. Případný jeho předchůdce ve stromě reprezentuje element, do něhož je daný element zanořen. Atribut daného elementu je reprezentován uzlem speciálního typu *Atribut*, který je následníkem uzlu reprezentujícího daný element. Podobně textová hodnota elementu je reprezentována následníkem speciálního typu *Text*.

Výhody a nevýhody

+ užitečné pro dynamickou modifikaci a přistup ke stromu
+ možnost dotazování nad daty
− rychlost (je třeba stavět celý strom)
− paměťová náročnost

3.1.4 JAXB

Princip: JAXB (Java Architecture for XML Binding) je mechanizmus popsaný standardem JSR-222 pro serializaci (marshalling) javovských objektů do XML dokumentů a naopak pro jejich deserializaci (umarshalling). Toto mapování využívá tříd majících speciální javovské anotace, jejichž přehled je uveden v tabulce 1.

Anotace	Význam	Umístění
XmlRootElement	kořenový element XML dokumentu	nad definicí třídy
XmlElement	element XML dokumentu	nad definicí instanční proměnné
XmlElementRef	používá se pro smíšený obsah a substituční skupiny	nad definicí instanční proměnné
XmlAttribute	atribut elementu odpovídající třídě	nad definicí instanční proměnné
XmlValue	reprezentuje textovou hodnotu elementu, reprezentovaného danou třídou	nad definicí instanční proměnné

XmlJavaTypeAdapter	určuje třídu na níž se při marshallingu (resp. unmarshallingu) zavolá metoda marshall (resp. unmarshall) s upravenou (resp. původní) třídou jako parametrem a neupravenou (resp. upravenou) třídou jako návratovým typem	nad definicí třídy
XmlType	poskytuje informace z XML Schematu, které nevyplývají z Javovského kódu	nad definicí třídy
XmlRegistry	factory s metodami pro tvorbu všech druhů objektů	nad definicí třídy
XmlElementDecl	slouží jako anotace k factory metodě vracející JAXBElement	nad factory metodu
XmlAccessorType	slouží k modifikaci toho, jak JAXB přistupuje k instančním proměnným	nad definicí třídy
XmlSeeAlso	instruuje JAXB k zahrnutí vybraných tříd (např. u potomků třídy)	nad definicí třídy

Tabulka 1 Anotace použitelné v JAXB

Generování tříd

Máme-li k dispozici XML Schema definující strukturu XML souboru obsahujícího serializované javovské objekty, JAXB umožňuje vygenerovat pomocí generátoru XJC zdrojový kód javovských tříd (včetně příslušných JAXB anotací) odpovídajících těmto objektům. XJC (Java Architecture for XML Binding Binding Compiler) je součástí JDK (Java Development Kit). Generátor XJC je možné spouštět z příkazového řádku, ale mnohem pohodlnější je jeho použití prostřednictvím IDE (Integrated Development Environment), s nímž bývá integrován. Pokud pro překlad a sestavování vyvíjeného projektu používáme nástroj maven [3], můžeme do procesu sestavování projektu snadno zařadit k tomu určený vestavný modul. V takovém případě mavenovský konfigurační soubor pom.xml (project object model) obsahuje následující XML element:

```
1  <plugin>
2    <groupId>org.jvnet.jaxb2.maven2</groupId>
3      <artifactId>maven-jaxb2-plugin</artifactId>
4      <version>číslo verze</version>
5      <configuration>
6        <schemaDirectory>lokace xsd souborů</schemaDirectory>
7        <bindingDirectory>lokace binding souborů</bindingDirectory>
8        <generatePackage>výsledný package</generatePackage>
9        <strict>false</strict>
10       <extension>true</extension>
11       <plugins>
12         <plugin>
13           <groupId>org.jvnet.jaxb2_commons</groupId>
```

```
14            <artifactId>jaxb2-basics</artifactId>
15            <version>číslo verze</version>
16          </plugin>
17          <plugin>
18            <groupId>org.jvnet.jaxb2_commons</groupId>
19            <artifactId>jaxb2-basics-annotate</artifactId>
20            <version>číslo verze</version>
21          </plugin>
22        </plugins>
23        <args>
24          <arg>-Xannotate</arg>
25          <arg>-XtoString</arg>
26        </args>
27      </configuration>
28      <executions>
29        <execution>
30          <id>generate</id>
31          <goals>
32            <goal>generate</goal>
33          </goals>
34        </execution>
35      </executions>
36  </plugin>
```

Zdrojová kód javovských tříd je generován z datových typů definovaných v příslušném XML Schematu.

Přizpůsobení generování tříd - existují situace, ve kterých je možné/nutné proces generování tříd přizpůsobit:

- konflikt ve jménech vygenerovaných tříd a jejich továrních metod (factory method),
- potřeba použít již existující třídu místo generování třídy nové,
- potřeba smysluplně pojmenovat (namísto použití implicitního jména) třídu reprezentující anonymní datový typ,
- poskytnutí mapování do výčtového typu, který není standardní součástí Javy.

Toto přizpůsobení může být provedeno buď přímým (inline) vložením anotace do odpovídajícího XML Schematu:

```
1  <xs:annotation>
2     <xs:appinfo>
3        binding declaration
4     </xs:appinfo>
5  </xs:annotation>
```

nebo použitím externího tzv. ”binding”souboru:

```
<jxb:bindings schemaLocation = "lokace xsd souboru">
   <jxb:bindings
   node = "XPath výraz určující umístění datového typu nebo elementu">
      binding declaration
   <jxb:bindings>
</jxb:bindings>
```

JAXBElement

V případě, že pro atribut generované třídy není možné jednoznačně určit datový typ na jehož základě by při následné serializaci bylo možné určit jméno a jmenný prostor příslušného elementu, je takovému atributu přiřazen datový typ JAXBElement, který kromě reprezentace hodnoty odpovídajícího atributu obsahuje potřebnou informaci (jméno elementu, jmenný prostor) pro potřeby jednoznačné serializace. Taková situace nastává v následujících případech:

- Definice nějakého elementu v XML Schematu obsahuje*mioccurs=0* a zároveň *nillable=”true”*. Pokud se element daném místě v instančním XML souboru nevyskytuje, je při serializaci odpovídající javovský atribut nastaven na *null*. Pokud se element v daném místě vyskytuje, ale má nastaven XML atribut *nill=true*, bude při serializaci příslušnému javovskému atributu přiřazena instance třídy *JAXBElement* obsahující atribut *value=null*.
- Na dané pozici se může vyskytnout více elementů stejného datového typu (XML Schema zde definuje např. substituční skupinu nebo *choice* element).
  Příkladem může být požití substituční skupiny:

```
<xs:element name="root">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="A"/>
       </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="A" type="xs:string"/>

<xs:element name="B" type="xs:string" substitutionGroup="A"/>
```

Tomu odpovídá následující konstrukce ve vygenerovaném zdrojovém kódu:

```
@XmlElementRef(name = "A",  type = JAXBElement.class)
    protected JAXBElement<String> A;
```

Jak je z příkladu zřejmé, v kořenovém elementu *root* se může vyskytnout buď element *A* nebo *B*, oba téhož datového typu *String*. Kdyby XJC vygeneroval atribut *A* typu *String*, nebyl by JAXB při serializaci (marshall) do XML souboru schopen určit, o který z elementů *A*/*B* se jedná. Proto generuje atribut *A* typu *JAXBElement<String>*, což umožňuje při deserializaci (unmarshall) zachovat informaci o skutečném jménu elementu a následně ji využít při serializaci.

Adaptéry

Adaptéry se používají, pokud je třeba po načtení, resp. před zápisem, elementu jeho hodnotu upravit. V takovém případě je potřeba

- nadefinovat třídu příslušného adapteru, jehož metody *marshall*/*unmarshall* implementují potřebnou konverzi a
- odpovídající javovskou třídu opatřit anotací *XmlJavaTypeAdapter*, která předepisuje při serializaci/deserializaci zavolat metodu *marshall*/*unmarshall* příslušného adapteru.

Příklad:

```
1  public class Adapter extends XmlAdapter<Adapted, Clazz>
2  {
3      @Override
4      public Clazz unmarshal(Adapted v) throws Exception {
5      //úprava
6      }
7
8      @Override
9      public Adapted marshal(Clazz v) throws Exception {
10     //úprava
11     }
12 }
```

V tomto příkladu třída *Adapted* reprezentuje třídu, která by danému elementu odpovídala bez adaptace, zatímco třída *Clazz* reprezentuje upravenou třídu.

Výhody a nevýhody

+ třídy lze vygenerovat
+ funguje obousměrně
+ snadná udržitelnost kódu
− paměťově náročné

3.2 Technologie pro parsování geografických dat

3.2.1 GeoTools

GeoTools je knihovna sloužící k práci se geografickými daty. Knihovna umožňuje parsování GML 3.2 dokumentů. Pokrývá pouze základ standardu. Je použitelná hlavně pro parsování

geometrií. Bohužel není schopna zpracovat všechna data z AIXM souboru, proto je tato knihovna pro potřeby této práce nepoužitelná.

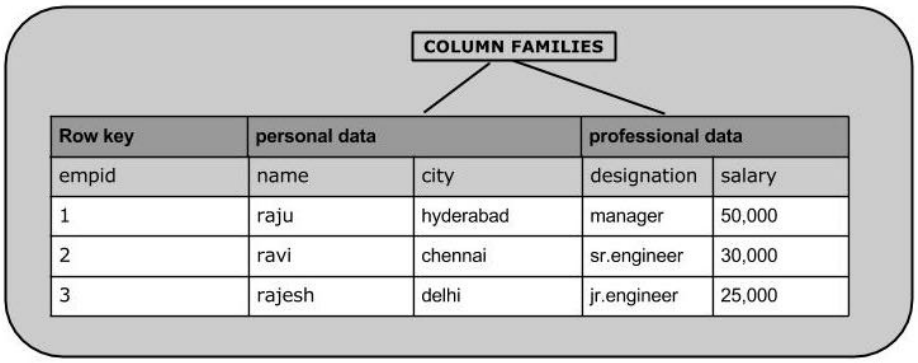
3.2.2 AIXM-J

AIXM-J je nedokončený projekt AIXM parseru, pokrývající pouze základní AIXM objekty. Je založený na technologii DOM. Jeho vývoj byl ukončen v roce 2013. Během tvorby této práce byly zdrojové kódy a dokumentace ze stránek projektu smazány.

3.3 NOSQL databáze

3.3.1 HBase

HBase je sloupcově orientovaná databáze, která je určena pro ukládání velkého množství dat. Databáze nepodporuje transakce. Datovou strukturu tvoří tabulka, skládající se z rodin sloupců, složených ze sloupců, reprezentujících kolekce key-value hodnot.



Obrázek 3 Příklad uložení dat v databázi HBase
(převzato z [4])

3.3.2 Redis

Redis je key-value databáze. Je určena k vykonávání velkého množství požadavků, proto primárně pracuje v paměti. Umožňuje ukládat řetězce, jejich seznamy, množiny, asociativní pole (hashmap) a jim odpovídající klíč.

3.3.3 MongoDB

MongoDB je dokumentově orientovaná databáze. Tvoří jí kolekce, které sestávají z dokumentů. Dokumenty nemají pevně danou strukturu a jsou reprezentovány ve formátu BJSON (Binary-JSON). Jak název napovídá, jedná se o notaci JSON (JavaScript Object Notation) v binárním tvaru.

JSON - JavaScript Object Notation

JSON je platformě nezávislý datový formát určený pro výměnu dat. JSON tvoří neuspořádaná množina páru název/hodnota. Hodnotou může být řetězec, číslo, pole hodnot nebo jiný JSON. JSON vypadá následovně:

```
{ "_id": "b", "c": ["d", "e"] }
```

Reference mezi dokumenty

Každý dokument musí mít identifikátor (dále *id*), který ho v rámci kolekce jednoznačně určuje. Mezi dokumenty různých kolekcí je možno zavést následující reference:

- **Manuální** - z jeden dokument se může pomocí *id* odkazovat na jiné dokumenty v téže nebo jiných kolekcích s tím omezením, že všechny odkazy musí směrovat do téže kolekce.
- **DBRef** - součástí odkazu je kromě identifikátoru *id* také jméno kolekce v níž odkazovaný dokument leží.

Geografické souřadnice

MongoDB, což je pro tuto práci zásadní, umožňuje ukládat geometrii geografických objektů a následně vyhledávat geografické objekty na základě prostorových dotazů. Souřadnice jsou brány jako 2D a třetí rozměr je ignorován. Souřadnice by měly být ve formátu GeoJSON. Existuje sice možnost nepoužít GeoJSON, ale ta funguje pouze pro bod 2D prostoru (jedná se o zastaralou konstrukci, pocházející ze starších verzí MongoDB).

GeoJSON je standard pro popis geografických souřadnic. Má tři základní souřadnicové typy:

- Point - určuje jeden bod

```
{ "type": "Point", "coordinates": [10.0, 10.0] }
```

- LineString - reprezentuje po částech lineární interpolaci křivky

```
1 { "type": "LineString",
2   "coordinates": [ [10.0, 0.0], [10.0, 1.0] ]}
```

- Polygon - určuje n-úhelník, přičemž první LineString určuje vnější obal (exterior) a další geometrie typu LineString určují ”díry”(interior) v n-úhelníku

```
1 { "type": "Polygon",
2   "coordinates": [
3     [ [10.0, 0.0], [11.0, 1.0], [10.0, 1.0], [10.0, 0.0] ],
4     [ [10.2, 0.2], [10.8, 0.2], [10.2, 0.8], [10.2, 0.2] ]
5     ]
6 }
```

GeoJSON dále umožňuje vytvářet kolekce těchto datových typů: kolekce bodů - *MultiPoint*, kolekce geometrií typu *LineString* - *MultiLineString*, kolekce geometrií typu *Polygon* - *MultiPolygon*, kolekce jakýchkoliv výše popsaných geometrií - *GeometryCollection*.

Geografické indexy: Pro zpracování prostorových dotazů se používají geografické indexy dvou druhů:

- 2d - určuje projekci bodu do dvourozměrné roviny
- 2dshphere - určuje geografický objekt určený sférickými souřadnicemi (zeměpisná šířka a délka) ve formátu GeoJSON

Geografický index se vytvoří se následujícím způsobem:

```
1 db.collection.createIndex(
2    { lokace geografického indexu : "2dsphere" } )
```

Geografické dotazy:
MongoDB umožňuje použití čtyř geografických dotazů viz. tabulka 2 níže.

Dotaz	Význam	Podporované indexy
geoWithin	vrací objekty uvnitř daného geometrického objektu	2dsphere a 2d
geoIntersects	vrací objekty protínající daný geometrický objekt	2dsphere
near	vrací nejbližší objekty k danému geometrickému objektu	2dsphere a 2d
nearSphere	vrací objekty v blízkosti bodu na sféře	2dsphere a 2d

Tabulka 2 Geografické dotazy v MongoDB

3.4 Software pro ukládání do databáze

3.4.1 MongoDB Java Driver

MongoDB Java Driver umožňuje základní operace s databází. Poskytuje javovské API umožňující ovládat databázi z javovského programu. Manipulace s daty probíhá pomocí javovské třídy *Document*, jejíž instance reprezentuje dokument ve formátu JSON.

3.4.2 Hibernate OGM (Object/Grid Mapper)

Jedná se o framework implementující nadmnožinu JPA (Java Persistence API), který je však možno používat jako poskytovatele JPA (JPA provider). Umožňuje ukládání do:

- key/value databází
- dokumentových databází
- grafových databází

Rozdíly oproti relační variantě

Hibernate OGM se repertoárem javovských anotací příliš neliší od své relační varianty, podporuje však jedinou strategii dědění. Každý typ je reprezentován samostatnou kolekcí, nejsou tedy podporovány anotace *Inheritance*, *DiscriminatorColumn* ani *SecondaryTables*, která slouží v relační variantě k mapování jedné entity do více tabulek.

Dotazování

je možno pomocí:
- JPQL (Java Persistence Query Language)
- nativního dotazu

Geografické souřadnice

Hibernate OGM neumožňuje používání geografických souřadnic, proto není pro naši práci použitelný.

3.4.3 Spring data

Spring data je framework umožňující ukládat data do MongoDB a následně na ně klást dotazy. Mapování do databáze je předepsáno pomocí javovských anotací.

Příklad:

```
1  @Document(collection="coll") //dokument se uloží do kolekce coll
2  public class C{
3
4    @Id //id dokumentu
5    private String id;
6
7    @Field("fName") //přejmenování
8    private String s;
9
10   @Transient//atribut ignorovám při ukládání/načítání
11   private Integer i2;
12
13   @DBRef //reference bude uložena pomocí com.mongodb.DBRef
14   private List<A> a;
15
16   @PersistenceConstructor // tento konstruktor bude volán
17                           // při načítání dokumentu z databáze
18   public C(String id, String s, Integer i2, A a) {
19     this.i = i;
20     this.s = s;
21     this.i2 = i2;
22     this.a = a;
23   }
24   }
```

Přístup k databázi

Existují dvě možnosti přístupu k databázi:

- Mongooperations - jedná se programátorské rozhraní (API) podobné javovskému driveru pro MongoDB.
- Pomocí tzv. repositářů, což je rozhraní, které definuje aplikační programátor. Repozitář dědí od jednoho z frameworkem (např. *CrudRepository*) poskytnutých rozhraní a poté definujeme metody pro operaci s daty v databázi, jejichž jména musí vyhovovat předepsané konvenci. Nevytváříme žádné instance tohoto rozhraní, o to se postará framework Spring Data.
  Příklad:

```
1  public interface UserRepository
2    extends CrudRepository<User, Long> {
3    Long deleteByLastname(String lastname);
4    List<User> findByEmailAddressAndLastname
5    (EmailAddress emailAddress, String lastname);
6  }
```

  V tomto příkladu *UserRepository* dědí od rozhraní *CrudRepository*, který umožňuje CRUD (Create, Read, Update, Delete) operace. Dále jsou přidány metody dvě metody:
  - *deleteByLastname* vymaže uživatele podle příjmení
  - *findByEmailAddressAndLastname* vyhledá uživatele podle emailu a příjmení, výsledek vrátí je v seznamu.

Geografické souřadnice: Co se týká geografických objektů, Spring Data umožňuje pracovat pouze s geometriemi typu bod. Podporuje prostorové dotazy *nearSphere*, *within* a *near*. Pro toto omezení jsme vyloučili použití frameworku Spring Data pro potřeby této bakalářské práce.

3.4.4 Morphia

Morphia je knihovna umožňující ukládání dat do MongoDB včetně dotazování. K popisu, jak má probíhat mapování, používá Morphia rovněž javovské anotace.

Příklad dokumentu:

```
1  @Entity("A") // uloží se do kolekce A
2  class E {
3    // id dokumentu, pokud není nastaveno je automaticky vygenerováno
4    @Id
5    ObjectId id;
6
7    // hodnota je automaticky uložena
8    String a;
9
10   // proměnná s hodnotou null není uložena
11   Long b = null;
12
```

```
13    // defaultně jsou tato data brána jako @Embedded
14    // neboli převedena na JSON a uložena
15    Address c;
16
17    //refrence uložená bez automatického načítání
18    Key<Q> d;
19
20    //reference s automatickým načítáním
21    @Reference
22    List<B> u = new ArrayList<B>();
23
24    //přejmenování
25    @Property("started")
26    Date s;
27
28    //atribut může být indexován pro větší výkon
29    @Indexed
30    boolean x = false;
31
32    //atribut může být načten, ale ne uložen
33    @NotSaved
34    String m;
35
36    //atribut ignorovám při ukládání/načítání
37    @Transient
38    int p;
39
40    //metoda prováděná během změn životního cyklu dat
41    // Pre/PostLoad, Pre/PostPersist, PreSave
42    @PostLoad
43    void postLoad() {}
44 }
```

Geografické indexy

Morphia umožňuje používání geografických indexů následující způsoby:

- Souřadnice bodu pomocí anotace:

```
1 @Indexed(IndexDirection.GEO2D)
2 protected double[] loc = null;
```

- Mapováním objektů přímo do GeoJSONu pomocí třídy GeoJson (v tomto případě je třeba po uložení objektu s geografickou informací nastavit index pomocí ovladače)

Dotazování

Morphia umožňuje dotazování pomocí:

- filtrování skládající se z *filter("field operátor", volitelně hodnota)* např. tedy:

```
filter("f >", 12)
```

  vyfiltruje všechny dokumenty, které mají v položce f hodnotu větší než 12.
- voláním metod: *exists*, *doesNotExist*, *greaterThan*, *greaterThanOrEq*, *lessThan*, *lessThanOrEq*,*equal*, *notEqual*, *hasThisOne*, *hasAllOf*, *hasAnyOf*, *hasNoneOf*, *hasThisElement*, *sizeEq*
- geografické dotazy *near* a *within*

4 Návrh řešení

4.1 Parsování aeronautických dat

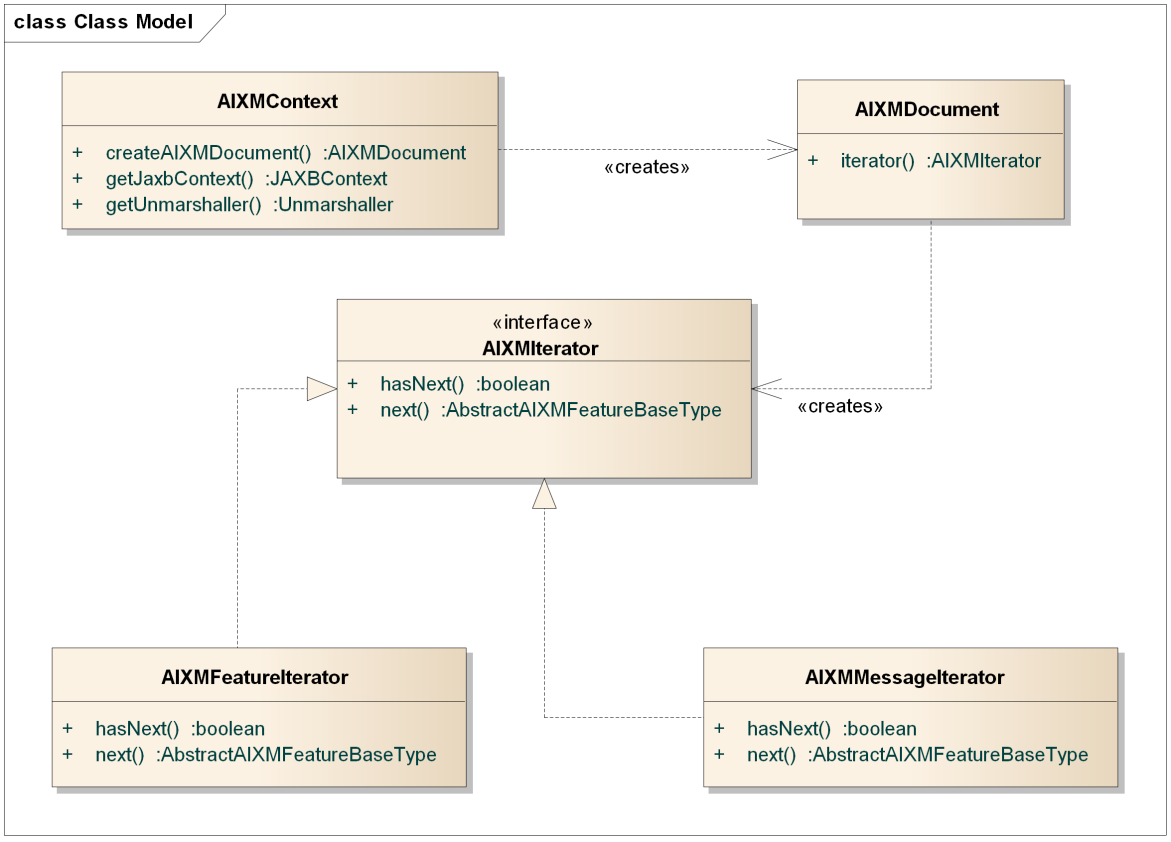
Vzhledem k tomu, že neexistuje žádný bezplatně dostupný parser pokrývající standard AIXM 5.1, bylo nutné napsat vlastní.

4.1.1 Výběr technologie parseru

Pro parser byla vybrána kombinace JAXB a StAXu. JAXB bylo vybráno, protože umožňuje třídy přímo vygenerovat z XML Schematu. Dále umožňuje jak parsovaní XML, tak převod objektů do XML dokumentů. Hlavní nevýhodou JAXB je nepoužitelnost pro velké soubory, tuto nevýhodu jsme překonali tím, že rozsáhlý XML soubor dekomponujeme pomocí StAXu na jednotlivé geografické objekty, které následně zpracujeme pomocí technologie JAXB.

4.1.2 Návrh parseru pro velké soubory

Struktura navrženého parseru je repzentována diagramem tříd zobrazeným na obrázku 4.



Obrázek 4 Návrh parseru AIXM dokumentů

Tento parser je navržen tak, že ze souboru obsahujícího aeronatická data parsuje postupně jednotlivé geografické objekty. K těmto objektům se přistupuje pomocí iterátoru. Existují 2 druhy iterárorů:

- *AIXMFeatureIterator* se použije, pokud soubor obsahuje jeden geografický objekt. K načtení geografického objektu je použito JAXB.
- *AIXMMessageIterator* se použije, pokud soubor obsahuje více než jeden geografický objekt. K přeskočení XML tagů AIXM kolekce je použit StAX a na načtení geografického objektu je použito JAXB.

Třída *AIXMContext* slouží k vytváření *AIXMDocumentů* reprezentující jednotlivé dokumenty obsahující aeronautická data. Třída *AIXMContext* obsahuje *JAXBContext* balíčku ve kterém se nachází vygenerované třídy, tvorba tohoto objektu trvá dlouho (řádově jednotky až desítky vteřin). *JAXBContext* je volně přístupný, proto je možné využít veškerou funkcionalitu JAXB.

4.2 Ukládání do databáze

4.2.1 Výběr databáze

Pro realizaci úložiště AIXM dat byla vybrána databáze MongoDB. Důvody pro výběr této databáze byly následující:

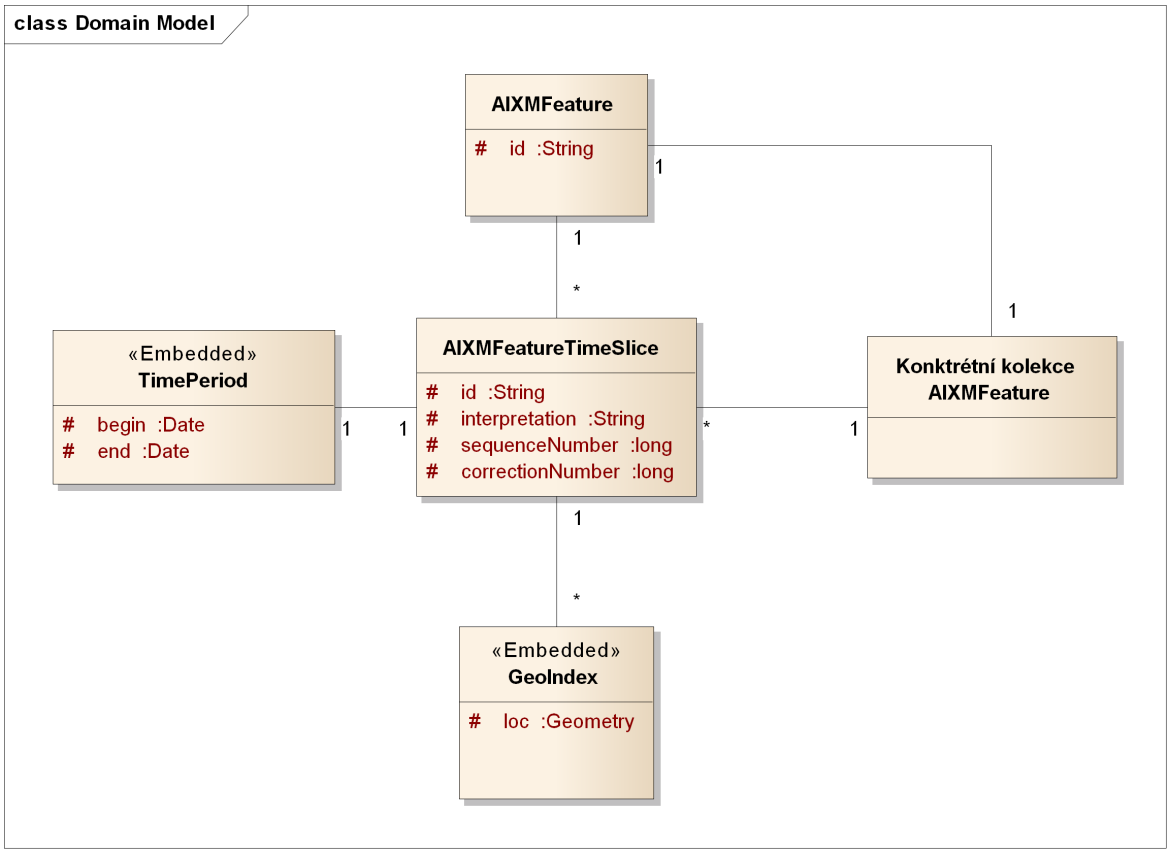
- otevřenost vůči změnám - dokumenty nemají pevně danou strukturu
- prostorová indexace a filtrování - umožňuje ukládat geografická data v GeoJSONu a provádět nad nimi prostorové dotazy
- malá paměťová náročnost - je nutno zpracovávat velké soubory
- nejvíce používaná NoSQL databáze současnosti

4.2.2 Výběr frameworku pro ukládání do databáze

Vybrána byla Morphia, neboť jako jediná umožňuje mapování všech geometrií podporovaných databází MongoDB do notace GeoJSON.

4.2.3 Návrh ukládání do databáze

Ukládání bylo navrženo následujícím způsobem:



Obrázek 5 Návrh ukládání geografických objektů do databáze

Jak je z obrázku 5 patrné, každý geografický objekt má svou kolekci. Aby bylo možno vyhledávat podle geografické pozice, je zde kolekce *AIXMFeatureTimeSlice*. Tato kolekce nese polohu ve formátu GeoJSON, informace ohledně platnosti časového řezu a informace o interpretaci časového řezu. Kolekce *AIXMFeature* umožňuje vyhledávat geometrické objekty napříč kolekcemi podle *id*. *Embedded* kolekce *GeoIndex* slouží k zabalení geometrií, protože na *GeometryCollection* se nedá dotazovat.

5 Implementace

5.1 Implementace parseru

5.1.1 Generování tříd

Podkapitola JAXB 3.1.4 popisuje, jak probíhá generování tříd reprezentujících jednotlivé typy geografických objektů. Během generování bylo nutné řešit v ”binding”souboru (viz. 3.1.4 - Přizpůsobení generování tříd) následující problémy:

- kolize názvu datových typů z různých jmenných prostorů:

```
<bindings schemaLocation="lokace schématu" version="1.0">
        <bindings node="XPath výraz">
            <property name="název třídy"/>
        </bindings>
</bindings>
```

- kolize názvů factory metod třídy object factory

```
<bindings schemaLocation="lokace schématu" version="1.0">
        <bindings node="XPath výraz">
            <factoryMethod name="název metody"/>
        </bindings>
</bindings>
```

5.1.2 Úprava tříd

V některých případech knihovna Morphia neumožňovala serializaci instancí vygenerovaných tříd do databáze. Tyto třídy bylo nutné upravit, aby je Morphia byla schopná zpracovat. Problém byl v tom, že některé vygenerované třídy neobsahovaly defaultní konstruktor a Morphia nebyla schopna je načíst z databáze. Úpravy lze rozdělit do dvou kategorií:

- Datový typ *BigInteger* bylo nutné nahradit datovým typem *long* a *BigDecimal* datovým typem *double*. Pro reprezentaci datového typu *UnlimitedIntegerType* byl použit *String*.
- Pro datový typ JAXBElement lze úpravy rozdělit do těchto kategorií:
  - JAXBElement, jehož atribut *value* obsahuje objekt, který nemá žádné potomky. Řešením bylo odstranit JAXBElement a XmlElementRef změnit na XmlElement.
    Příklad:
    Před úpravou:

```
@XmlElementRef(name = "primeMeridian",
 namespace = "http://www.opengis.net/gml/3.2",
```

```
3    type = JAXBElement.class)
4  protected JAXBElement<PrimeMeridianPropertyType> primeMeridian;
```

Protože vygenerovaná třída *PrimeMeridianPropertyType* nemá žádné potomky, je aplikovatelná výše popsaná úprava. Po úpravě:

```
1  @XmlElement(name = "primeMeridian",
2  namespace = "http://www.opengis.net/gml/3.2")
3  protected PrimeMeridianPropertyType primeMeridian;
```

– JAXBElement, jehož atribut *value* obsahuje objekt mající potomky (ve smyslu dědičnosti), ale každému z nich odpovídá v XML Schematu jediný element. Řešením bylo odstranit JAXBElement a objektu v něm obsaženému včetně jeho potomků přidat anotaci XmlRootElement se jmény odpovídajícími dané substituční skupině, aby ji bylo možné referencovat.

Příklad:
Před úpravou:

```
1  @XmlElementRef(name = "Envelope",
2   namespace = "http://www.opengis.net/gml/3.2",
3   type = JAXBElement.class, required = false)
4      protected JAXBElement<? extends EnvelopeType> envelope;
```

Vygenerovaná třída EnvelopeType odpovídá v XML Schematu jedinému elementu a rovněž každý z jejích potomků odpovídá v XML Schematu jednomu elementu. Je tedy aplikovatelná výše popsaná úprava. Po úpravě:

```
1  @XmlElementRef(name = "Envelope",
2   namespace = "http://www.opengis.net/gml/3.2",
3   required = false)
4      protected EnvelopeType envelope;
```

Třída opatřená anotací tak, aby bylo možné na ní referencovat:

```
1  @XmlRootElement(name="Envelope",
2   namespace = "http://www.opengis.net/gml/3.2")
3  public class EnvelopeType implements ToString
```

– JAXBElement, jehož atribut *value* může obsahovat instance jedné javovské třídy, jež však je v XML Schematu reprezentována několika různými elementy téže substituční skupiny. Řešením tohoto problému je převést JAXBElement pomocí adaptéru na vlastní třídu s definovaným defaultním konstruktorem, jež obsahuje všechny atributy původního JAXBElementu.
Příklad:
Před úpravou:

```
@XmlElementRef(name = "AbstractObject", namespace =
↪ "http://www.opengis.net/gml/3.2", type = JAXBElement.class,
↪ required = false)
    protected List<JAXBElement<?>> abstractObject;
```

Po úpravě:

```
@XmlJavaTypeAdapter(ArrayAssociationTypeAdapter.class)
public class ArrayAssociationType
    implements ToString
{
    protected List<MyJAXBElement> abstractObject;
```

XML adaptér:

```
public class ArrayAssociationTypeAdapter extends
↪ XmlAdapter<ArrayAssociationTypeAdapter.ArrayAssociationTypeAdapted,
↪ ArrayAssociationType>
{

    @Override
    public ArrayAssociationType
    ↪ unmarshal(ArrayAssociationTypeAdapted v) throws Exception {
        ArrayAssociationType arrayAssociationType=new
        ↪ ArrayAssociationType();

        arrayAssociationType.setOwns(v.owns);

        List<MyJAXBElement> abstractObject=new
        ↪ LinkedList<MyJAXBElement>();

        for(JAXBElement<?> e:v.abstractObject)
        {
            MyJAXBElement myJAXBElement=new MyJAXBElement();
            myJAXBElement.value=e.getValue();
            myJAXBElement.localPart=e.getName().getLocalPart();

            ↪ myJAXBElement.namespaceURI=e.getName().getNamespaceURI();
            myJAXBElement.prefix=e.getName().getPrefix();

            abstractObject.add(myJAXBElement);
        }
```

```
22
23          arrayAssociationType.setAbstractObject(abstractObject);
24
25          return arrayAssociationType;
26      }
27
28      @Override
29      public ArrayAssociationTypeAdapted marshal(ArrayAssociationType
        ↪  v) throws Exception {
30
31          ArrayAssociationTypeAdapted arrayAssociationTypeAdapted=new
            ↪  ArrayAssociationTypeAdapted();
32
33          List<JAXBElement<?>>abstractObject=new LinkedList<>();
34
35          for(MyJAXBElement myJAXBElement:v.getAbstractObject())
36          {
37              abstractObject.add(new JAXBElement(new
                ↪  QName(myJAXBElement.namespaceURI,
                ↪  myJAXBElement.localPart, myJAXBElement.prefix),
                ↪  myJAXBElement.value.getClass(),myJAXBElement.value));
38          }
39
40          arrayAssociationTypeAdapted.abstractObject=abstractObject;
41          arrayAssociationTypeAdapted.owns=v.isOwns();
42
43          return arrayAssociationTypeAdapted;
44      }
45
46      public static class ArrayAssociationTypeAdapted
47      {
48          @XmlElementRef(name = "AbstractObject", namespace =
            ↪  "http://www.opengis.net/gml/3.2", type =
            ↪  JAXBElement.class, required = false)
49          protected List<JAXBElement<?>> abstractObject;
50          @XmlAttribute(name = "owns")
51          protected java.lang.Boolean owns;
52      }
53  }
```

Provedené úpravy navíc zjednodušují přístup k objektům a jejich atributům. Generátor *XJC* má tendenci kvůli zachování pořadí elementů dávat je všechny do jednoho seznamu. Problém je naznačen na následujícím (zkráceném) příkladu uvádějícím atribut *rest*, který obsahuje

seznam objektů typu JAXBElement s nespecifikovaným obsahem:

```
@XmlElementRefs({
        @XmlElementRef(name = "oppositeTrack", namespace =
        ↪ "http://www.aixm.aero/schema/5.1", type = JAXBElement.class,
        ↪ required = false),
        @XmlElementRef(name = "designatorSuffix", namespace =
        ↪ "http://www.aixm.aero/schema/5.1", type = JAXBElement.class,
        ↪ required = false),
        @XmlElementRef(name = "type", namespace =
        ↪ "http://www.aixm.aero/schema/5.1", type = JAXBElement.class,
        ↪ required = false),
    })
    protected List<JAXBElement<?>> rest;
```

5.2 Implementace ukládání dat do databáze

5.2.1 Převod geometrií do GeoJSONu

Převod do GeoJSONu je realizován rekurzivní metodou *createGeoJson* objektu *AIXMFeature* v balíčku *cz.cvut.fel.bp.aixmparser.geojson*, která převede geografické souřadnice geografického objektu do GeoJSONu. Převod funguje do všech datových typů GeoJSONu, kromě třídy *GeometryCollection*, u níž nefungují geografické dotazy. Toto mapování není úplně dokonalé, neboť vyjadřovací schopnosti GeoJSONu jsou menší než AIXM 5.1. Problém může nastat například u oblouku, který bude po převodu do GeoJSONu brán jako spojnice bodů (*LineString*). Dále program umí převádět geografické souřadnice pouze v souřadnicových referenčních systémech *CRS:84* a *EPSG:4326*. Formáty *CRS:84* a *EPSG:4326* slouží k zápisu 2D geografických souřadnic. Liší se v tom, že *CRS:84* zapisuje souřadnice ve formátu "zeměpisná délka zeměpisná šířka"a *EPSG:4326* zapisuje souřadnice obráceně ve formátu "zeměpisná šířka, zeměpisná délka".

5.2.2 Způsob uložení do databáze

Pro uložení do databáze je vytvořena statická metoda *persistAIXMFeature* objektu *AIXMFeature* v balíčku *cz.cvut.fel.bp.aixmparser.geojson*. Tato metoda provede přemapování geometrií do GeoJSONu a následně geografický objekt uloží. Poté vytvoří dokumenty do kolekcí *AIXMFeature* a *AIXMFeatureTimeSlice*. Nakonec vytvoří odpovídající geografické indexy v databázi. Metoda vypadá následujícím způsobem:

```
public static void persistAIXMFeature
(AbstractAIXMFeatureBaseType feature, String type, DB db, Datastore ds)
```

AbstractAIXMFeatureBaseType feature je geografický objekt určený k uložení do databáze. *String type* určuje typ indexu v našem případě "2dsphere". *DB db* je konkrétní databáze potřebná pro tvorbu indexu. *Datastore ds* slouží k volání funkcí knihovny Morphia. Alternativou k této metodě byla možnost použít anotace volané při změně životního cyklu dat, ale

to by znamenalo nutnost předávat parametry *db* a *ds* již v konstruktoru, navíc by uložení jedno dokumentu mělo za následek vytvoření několika dalších dokumentů. Tak byl v zájmu přehlednosti zvolen tento přístup.

6 Ověření

6.1 Použitá data

Pro ověření bylo použita data volně dostupná na stránkách instituce Eurocontrol. Tato data obsahují 33 687 geografických objektů. Ne každý geografický objekt obsahuje geografické souřadnice.

6.2 Výsledná měření

Měření byla provedena na všech dostupných datech. Průměrná doba uložení jednoho geografického objektu včetně vytvoření dokumentů AIXMFeature a AIXMFeatureTimeSlice a převedení souřadnic do GeoJSONU trvalo průměrně 1.08 ms.

Geografické dotazy: Výsledky u *near* dotazu jsou všechny geografické objekty obsahující geografické souřadnice seřazené podle blízkosti k danému bodu. Sloupec *Trvání dotazu v databázi* udává délku trvání samotného zpracování dotazu databází MongoDB. Sloupec *Výsledný čas* udává délku trvání zpracování dotazu včetně režie knihovny Morphia. Sloupec *Počet výsledků* obsahuje počet geografických objektů vyhovujících danému dotazu. Každý dotaz byl proveden 100 krát a výsledek byl zprůměrován, přičemž byly prohledávány pouze platné časové řezy geografických objektů.

Dotaz	Trvání dotazu v databázi [ms]	Výsledný čas [ms]	Počet výsledků
within((90,90),(-90,90), (-90,-90),(90,-90),(90,90))	148	8018	18533
within((7,47),(8,47),(8,48), (7,48),(7,47)))	419	483	32
within((0,0),(100,0),(100,100), (0,100),(0,0))	320	1828	4726
within((51,-31),(53,-31),(53,-33),(51,-33),(51,-31))	421	487	0
near(57, 56)	195	9698	32590
near(41, 12)	191	9288	32590

Tabulka 3 Výsledné časy dotazů

6.3 Testování

6.3.1 Testování parseru

Testování správnosti anotací

Vzhledem ke změnám ve vygenerovaných třídách bylo nutné ověřit, zda nedošlo k narušení jejich správnosti. Ověření probíhalo tak, že nejprve byl dokument načten pomocí JAXB (ne pomocí AIXM parseru) a následně opět uložen do XML dokumentu. Výsledně vygenerovaný soubor byl následně srovnán s původním souborem. K ověření se nabízelo použít XMLUnit [5], ale vzhledem k odlišnostem souborů, které nebyly chybami (např. v původním souboru byl obsah elementu "-3.0"a ve vygenerovaném "-3"), byl použit online dostupný nástroj Pretty Diff[1] určený k porovnávání XML souborů. Tento nástroj umožňuje zobrazit rozdíly mezi soubory. Vzhledem k tomu, že rozdílů nebylo mnoho a nejednalo o chyby, bylo ověření snadné.

Testování parsování

Cílem tohoto testování bylo ověřit správnou funkčnost parseru včetně chybových stavů. Testování se zaměřilo hlavně na správnou funkčnost iterátorů. Ověření probíhalo tak, že se nejprve daný dokument načetl pomocí JAXB a následně pomocí parseru. Nakonec bylo provedeno ověření, jestli obsahují stejné geografické objekty.

6.3.2 Testování ukládání do databáze

Testování mapování do GeoJSONU

Cílem tohoto testování bylo ověřit správnost převodu souřadnic do GeoJSONu. Převod byl proveden na dostupných datech pro všechny základní typy GeoJSONu (Point, LineString, Polygon).

Testování uložení

Cílem tohoto testování bylo ověřit, že dojde ke korektnímu vytvoření dokumentů a geografických indexů. To bylo provedeno tak, že byl geografický objekt uložen do databáze a následně vyhledán v kolekci AIXMFeature dle id, resp. v kolekci AIXMFeatureTimeSlice dle souřadnic.

6.4 Shrnutí

Z naměřených výsledků v tabulce 3 je jasně patrné, že dotaz v databázi trvá poměrně krátkou dobu, ale výsledný čas po provedení dotazu a vrácení výsledku Morphiií je relativně vysoký. Tento problém je částečně způsoben tím, že Morphia se musí během načítání objektu z databáze ještě dotazovat do kolekce odpovídající danému geografickému objektu a načíst jeho hodnotu. Hlavním faktorem, který ovlivňuje délku trvání dotazu, je počet nalezených objektů v databázi, neboť s jejich rostoucím počtem roste i doba převodu výsledku z JSONu do javovských objektů.

[1] http://prettydiff.com/

7 Závěr

Cílem této práce bylo navrhnout a implementovat parsování pro velké AIXM 5.1 soubory a jejich ukládání do NoSQL databáze. Práce se tedy dá rozdělit na dvě relativně nezávislé části - parsování a ukládání.

V první části bylo nutno navrhnout a implementovat efektivní způsob parsování velkých AIXM 5.1 souborů. Vzhledem k neexistenci vhodných volně dostupných nástrojů bylo nutno tento parser implementovat vlastnoručně. Pro tuto implementaci byla zvolena kombinace JAXB a StAXu. Výsledný parser funguje jako iterátor po jednotlivých geografických objektech v AIXM 5.1 dokumentu, které jsou následně deserializovány technologií JAXB.

V druhé části bylo nutno navrhnout a implementovat ukládání do NoSQL databáze. Jako databáze byla vybrána MongoDB, hlavně díky možnosti ukládat a dotazovat se nad geografickými daty uloženými ve formátu GeoJSON. Vybranou knihovnou pro mapování objektů do databáze byla vybrána Morphia, neboť umožňuje mapování objektů do formátu GeoJSON. Návrh uložení byl koncipován tak, aby bylo možno pro všechny typy geografických objektů vyhledávat podle jejich geografické pozice. Toto uložení bere v úvahu časový model standardu AIXM 5.1 .

K dosažení spolupráce mezi vyvinutým parserem a použitou metodou ukládání do databáze byla navržena sada pravidel pro úpravu javovských tříd vygenerovaných generátorem XJC z XML Schemat definujících formáty AIXM 5.1 a GML 3.2 .

Vyjadřovací schopnosti formátu GeoJSON neumožňují stejně komplexní popis jako standard AIXM 5.1. Příkladem je geometrie typu oblouk (arc), která je v AIXM 5.1 standardně reprezentována třemi body. Prosté převedení do formátu GeoJSON ji nahradí geometrií LineString, t.j. dvěma na sebe navazujícími úsečkami. Jedním z rozšíření, které by mohlo navázat na předloženou bakalářskou práci, by mohla být přesnější aproximace oblouku geometrií LineString o více bodech.

Příloha A

Obsah přiloženého CD

Zde se nachází obsah přiloženého CD. Seznam souboru a adresářů. Včetně informací o tom, co v nich hledat.

- Implementace/ - Zdrojové kódy včetně testovacích dat.
- Text/ - Tento dokument v PDF i se zdrojovým kódem systému LaTeX.
- readme.txt - Textový soubor s informacemi o disku.

Literatura

[1] Ron Lake; David S. Burggraf; Milan Trninic; Laourie Rae. *Geography Mark-Up Language*. 2004.

[2] *AIXM 5 Temporality Model*. URL: http://www.aixm.aero/gallery/content/public/AIXM51/AIXM%20Temporality%201.0.pdf (cit. 13. 15. 2015).

[3] *What is maven*. URL: https://maven.apache.org/what-is-maven.html (cit. 13. 15. 2015).

[4] *Apache HBase Reference Guide*. URL: http://hbase.apache.org/book.html#_getting_started (cit. 13. 15. 2015).

[5] *About XMLUnit*. URL: http://www.xmlunit.org/l (cit. 13. 15. 2015).

[6] I. Mlynkova; M. Necasky; J. Pokorny; K. Rychta; K. Toman; V. Toman. *XML technologie Pricipy a aplikace v praxi*. 2008.

[7] Mark Pollack; Thomas Risberg; Oliver Gierke; Costin Leau; Jon Brisbin; Thomas Darimont; Christoph Strobl. *Spring Data MongoDB - Reference Documentation*. URL: http://docs.spring.io/spring-data/mongodb/docs/current/reference/html/ (cit. 13. 15. 2015).

[8] Emmanuel Bernard; Sanne Grinovero; Gunnar Morling; Davide D'Alto. *Hibernate OGM Reference Guide*. URL: http://docs.jboss.org/hibernate/ogm/4.1/reference/en-US/html/ (cit. 13. 15. 2015).

[9] *Introducing JSON*. URL: http://www.json.org/ (cit. 13. 15. 2015).

[10] *aixm-j*. URL: http://sourceforge.net/projects/aixm-j/i (cit. 13. 15. 2015).

[11] *Morphial*. URL: https://github.com/mongodb/morphia/wiki (cit. 13. 15. 2015).

[12] *Quickstart*. URL: http://www.saxproject.org/quickstart.html (cit. 13. 15. 2015).

[13] *XML DOM Tutorial*. URL: http://www.w3schools.com/dom/ (cit. 13. 15. 2015).

[14] Wolfgang Laun. *A JAXB Tutorial*. URL: https://jaxb.java.net/tutorial/ (cit. 13. 15. 2015).

[15] *Customizing JAXB Bindings*. URL: http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/JAXBUsing4.html (cit. 13. 15. 2015).

[16] *Documentation*. URL: http://redis.io/documentation (cit. 13. 15. 2015).

[17] *XML Schema Tutorial*. URL: http://www.w3schools.com/schema/default.asp (cit. 13. 15. 2015).

[18] *XML Tutorial*. URL: http://www.w3schools.com/xml/ (cit. 13. 15. 2015).

[19] *XmlAdapter - JAXB's Secret Weapon*. URL: http://blog.bdoughan.com/2010/07/xmladapter-jaxbs-secret-weapon.html (cit. 13. 15. 2015).

[20] Blaise Doughan. *Using JAXB's @XmlAccessorType to Configure Field or Property Access*. URL: http://blog.bdoughan.com/2011/06/using-jaxbs-xmlaccessortype-to.htmll (cit. 13. 15. 2015).

[21] Blaise Doughan. *JAXB and Inheritance - Using Substitution Groups*. URL: http://blog.bdoughan.com/2010/11/jaxb-and-inheritance-using-substitution.htmll (cit. 13. 15. 2015).

[22] *User Guide*. URL: http://docs.geotools.org/latest/userguide/ (cit. 13. 15. 2015).

[23] *AIXM Application Schema Generation*. URL: http://www.aixm.aero/gallery/content/public/AIXM51/AIXM_Application_Schema_Generation-1.1.pdf (cit. 13. 15. 2015).

[24] *AIXM UML to XML Schema Mapping*. URL: http://www.aixm.aero/gallery/content/public/AIXM51/AIXM_UML_to_AIXM_XSD_Mapping-1.1.pdf (cit. 13. 15. 2015).

[25] *AIXM 5 Feature Identification and Reference*. URL: http://www.aixm.aero/gallery/content/public/AIXM51/AIXM_Feature_Identification_and_Reference-1.0.pdf (cit. 13. 15. 2015).