AFSwinx & AFRest

Uživatelská příručka

Verze: 1.0.0.

Martin Tomášek

martin@toms-cz.com

31.12.2014

Obsah

1. Úvod

AFSwinx a AFRest jsou frameworky umožňující generování uživatelského rozhraní. Generování je prováděno na serverové straně a interpretování na klientovi. K přenosu lze využít například služby typu REST. Referenční implementace je vytvořena pro Java SE aplikace založené na Swingu. Framework nabízí kromě generování rozhraní také moduly umožňující připojení na server a získání definicí a dat. Níže se seznámíte se základními koncepty použití a funkčností, kterými Framework disponuje.

1.1. Základní koncept

Základním konceptem je generování definicí dat na serverové straně. Lze takto mít pod kontrolou uživatelské rozhraní na klientovi. Definice také zohledňují aktuální datový či visuální model a dokáží pružně reagovat na změny datových typů či polí. Klient pouze specifikuje adresy zdrojů, na kterých jsou definice a adresy zdrojů, na kterých jsou data. Dále lze specifikovat adresy zdrojů pro aktualizace, mazání či vytváření dat. Klient tedy slouží pouze jako interpreter.

Serverová strana umožňuje na základě datových typů generovat různé typy widgetů. K dílčí generaci je využit framework AspectFaces.

2. Použití frameworku

2.1. Závislosti

Framework je koncipován jako Maven projekt. Následující závislosti musí být přidány na straně serveru do projektu, či jejich JAR jako knihovny. Dále je potřeba stáhnout přednastavené konfigurační soubory ke generování dat a vložit je do WEB-INF. Tyto soubory lze později upravit. Stáhnout je lze zde.

```
<dependency>
<groupId>com.tomscz.af</groupId>
<artifactId>AFRest</artifactId>
<version>0.0.1.-SNAPSHOT</version>
</dependency>
```

Obrázek 1 Maven závislosti serveru

Na klientské straně je potřeba přidat následující závislosti

```
<dependency>
<groupId>com.tomscz.af</groupId>
<artifactId>AFSwinx</artifactId>
<version>0.0.1.-SNAPSHOT</version>
</dependency>
```

Obrázek 2 Maven závislosti klienta

2.2. Serverová strana

Serverová strana vytváří definice dat a dodává data klientské aplikaci. Zároveň je potřeba aby serverová strana uměla i požadavek přijmout. K přijmutí požadavku lze využít REST, který lze velmi snadno implementovat za použití již stávajících knihoven. Zdroj, který generuje definice dat, pak vypadá například takto:

```
@javax.ws.rs.core.Context
HttpServletRequest request;

@GET
@Path("/definition")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_JSON})
public Response getResources() {
   try {
      AFRest afSwing = new AFRestGenerator(request.getSession().getServletContext());
      AFMetaModelPack data = afSwing.generateSkeleton(Country.class.getCanonicalName());
      return Response.status(Response.Status.OK).entity(data).build();
   } catch (MetamodelException | AFRestException e) {
      return Response.status(Response.Status.INTERNAL_SERVER_ERROR).build();
   }
}
```

Obrázek 3 Rest definice zdroje na serveru

Třída Country, na základě které bude vytvořena definice je zachycena na Obrázek 5 Objekt k inspekci na serveruPro jednoduchost nyní můžeme říci, že definice v sobě bude obsahovat pole pro id, jméno (name), zkratku (shortcut), a příznak aktivity (active). Důvodem tohoto zjednodušeného řešení je fakt, že v této příručce prozatím nebyly rozebrány možnosti generujícího frameworku, kterým je AspectFaces. Definice nespecifikuje, zdali se bude jednat o tabulku či o formulář celá tato záležitost je v plné kompetenci klienta.

Dalším důležitým zdrojem je zdroj s daty, či zdroj, který přijme data k vytvoření upravení či smazání. Příkladem je zdroj, který produkuje sadu dat do tabulky a je zachycen na Obrázek 4.

```
@GET
@Path("/list")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_JSON})
public Response getAllCountries() {
   CountryManager cm = new CountryManager();
   List<Country> countries = cm.findAllCountry();
   final GenericEntity<List<Country>> personGeneric =
         new GenericEntity<List<Country>>(countries) {};
   return Response.status(Response.Status.OK).entity(personGeneric).build();
}
```

Obrázek 4 zdroj, který vrátí sadu dat

```java
package com.tomscz.afserver.persistence.entity;

public class Country {

    private Long id;
    private String name;
    private String shortCut;
    private boolean active;

    public Country(){
        super();
    }

    public boolean isActive() {
        return active;
    }

    @UiLabel(value="country.isActive")
    @UiOrder(value=3)
    public void setActive(boolean active) {
        this.active = active;
    }

    @UiLabel(value="country.shortCut")
    @UiOrder(value=1)
    public String getShortCut() {
        return shortCut;
    }

    public void setShortCut(String shortCut) {
        this.shortCut = shortCut;
    }

    @UiLabel(value="country.name")
    @UiOrder(value=0)
    @UiRequired
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

Obrázek 5 Objekt k inspekci na serveru

V případě generování formuláře musí zdroj vracet právě jeden objekt, na základě kterého bude formulář sestaven. Příklad takového zdroje lze najít na Obrázek 6. Formulář může být naplněn daty pouze z konkrétní instance třídy.

```
@GET
@Path("/{id}")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_JSON})
public Response getCountry(@PathParam("id") int id) {
   try {
      CountryManager cm = new CountryManager();
      Country country = cm.findById(id);
      return Response.status(Response.Status.OK).entity(country).build();
   } catch (BusinessException e) {
      return Response.status(Response.Status.BAD_REQUEST).build();
   }
}
```

Obrázek 6 zdroj, který vrací instanci země

2.2.1. Podmíněné generování

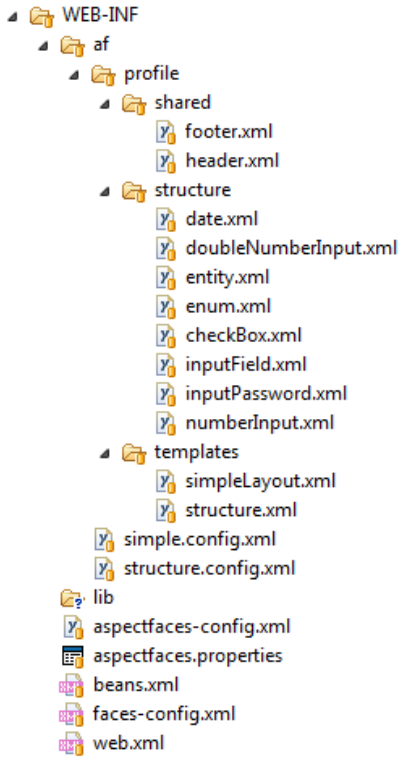
V následující sekci bude zobrazena ukázka, která nastavuje Framework pro generování dat a po vygenerování ještě specifikuje některé vlastnosti. Ukázka je na Obrázek 7 - Podmíněné generování dat a provedení dodatečného nastavení po generování. Nejprve je vytvořena instance generátoru. Poté jsou nastaveny mapovací soubory, které budou použity pro složené datové typy. Pro kořenovou třídu bude využit absenceInstanceManagement.config.xml pro proměnou affectedPerson, absenceType a country je využito mapování absence.instance.inner.simple.xml. Následně je vygenerována meta definice objektu. Objekt má proměnou status, kterou chceme dále upřesnit. Vytvoříme možnosti, kterými může tato proměnná disponovat. Počet proměnných je závislý na uživatelské roli. Možnosti nastavíme pomocí metody setOptions(možnosti, proměnná).
V případě, že se jedná o proměnnou uvnitř jiné třídy, lze použít standardní tečkovou notaci například. affectedPerson.name. Toto nastaví možnosti do třídy, kterou reprezentuje proměnná affectedPerson uvnitř kořenového objekty a možnosti budou nastaveny pro proměnnou name.

```
@GET
@Path("/definitionManaged/{userName}")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_JSON})
@RolesAllowed({"admin", "user"})
public Response getDefinitionToManageAbsenceInstance(
      @javax.ws.rs.core.Context HttpServletRequest request,
      @PathParam("userName") String userName) {
      try {
            AFRest afSwing = new
                  AFRestGenerator(request.getSession().getServletContext());
            HashMap<String, String> customMapping = new HashMap<String, String>();
            customMapping.put(AbsenceInstance.class.getCanonicalName(),
                   "absenceInstanceManagement.config.xml");
            customMapping.put("affectedPerson",
                  "absence.instance.inner.simple.xml");
            customMapping.put("absenceType", "absence.instance.inner.simple.xml");
            customMapping.put("country", "absence.instance.inner.simple.xml");
            AFMetaModelPack data =
                afSwing.generateSkeleton(AbsenceInstance.class.getCanonicalName(),
                             customMapping, "");
            AFSecurityContext securityContex =
                   (AFSecurityContext)
                     request.getAttribute(AFServerConstants.SECURITY_CONTEXT);
            if (securityContex.isUserInRole(UserRoles.ADMIN)) {
                HashMap<String, String> stateOptions;
                stateOptions =
                        AFRestUtils.getDataInEnumClass(AbsenceInstanceState.class
                                .getCanonicalName());
                data.setOptionsToFields(stateOptions, "status");
            } else {
                HashMap<String, String> stateOptions =
                     new HashMap<String,String>();
                stateOptions.put(AbsenceInstanceState.CANCELLED.name(),
                   AbsenceInstanceState.CANCELLED.name());
                stateOptions.put(AbsenceInstanceState.REQUESTED.name(),
                   AbsenceInstanceState.REQUESTED.name());
                data.setOptionsToFields(stateOptions, "status");
            }
            return Response.status(Response.Status.OK).entity(data).build();
        } catch (MetamodelException e) {
            return Response.status(Response.Status.INTERNAL_SERVER_ERROR).build();
        }
}
```

Obrázek 7 - Podmíněné generování dat a provedení dodatečného nastavení po generování

2.3. Generování definic s využitím AspectFaces

Jak již bylo zmíněno, k hlavní inspekci dat je využit Framework AspectFaces. Při prvním seznámení s projektem bylo potřeba zkopírovat do WEB-INF obsah vzorového nastavení pro generování. Struktura je zobrazená na obrázku Obrázek 8. Důležitým souborem je structure.config.xml. Tento soubor specifikuje mapování datových typů na komponenty.



Obrázek 8 struktura projektu

V případě, že datový typ v této specifikaci chybí tak mu nebude přiřazena žádná komponenta a tento prvek se tedy ve formuláři neobjeví. Jednotlivé komponenty jsou ve složce structure. Důvodem je pouze specifikace v structure.config.xml, která na tuto složku odkazuje. Komponenty tak mohou být umístěny v libovolné složce. Další důležitou součástí jsou šablony, které jsou umístěny ve složce templates. Tyto šablony specifikují, jak se budou komponenty vytvářet, dále lze mezi ně při vytváření vkládat tagy. Pro účely AFSwin postačí, že kořenová třída by měla používat structure.xml a odvozené třídy simpleLayout.xml. Detaili xml si můžete prohlídnout ve vzorovém projektu.

2.4. Klient

Klientem se v našem frameworku rozumí jakákoliv Swingová aplikace. Framework generuje panely, které lze přidat do dalších panelů. Tímto způsobem můžeme docílit toho, že pouze část aplikace bude využívat generované uživatelské rozhraní. Zbývající část aplikace může být napsána k libovůli uživatele.

2.4.1. Základní použití

Základním použitím jsou formuláře nebo tabulky. Budeme vycházet ze zdrojů uvedených výše. Důležité jsou tyto faktory

- Endpoint pro definice dat
- Endpoint pro data
- Endpoint pro akce (vytvoření, aktualizace, mazání)
- Komponenta – zvolení zdali půjde o formulář či tabulku
- Skin – základní nebo rozšířený

- Lokalizace – Framework podporuje lokalizace dat i validačních zpráv
- Identifikátor formuláře.

Hlavní správu zajišťuje třída AFSwinx. Jedná se o singleton třídu pomocí, které by měli být vytvářeny komponenty. Skrze tuto třídu lze získat komponentu kdekoliv v rámci aplikaci a nad komponentou provádět akce. Příklad použití AFSwinx a vytvoření tabulky je na Obrázek 9. Ukázka vytvoření formuláře je na Obrázek 10 vytvoření formuláře. Nejprve je potřeba získat instanci AFSwinx. Poté builder, který postaví tabulku. Tento builder lze vyměnit za formulářový builder. Nejprve builder inicializujeme a to tak, že nastavíme identifikátor tabulky, předáme mu inputsteam se specifikací zdrojů a identifikátor zdroje, který chceme použít. První dvě řádky načítají soubor connection.xml jako input steam. Soubor může být načten mnoha způsoby toto je pouze jeden z nich. Identifikátor formuláře je současně klíč, pod kterým ho lze získat z třídy AFSwinx. Na následujících ukázkách jsou identifikátory country a countryForm.

```
InputStream connectionStream =
getClass().getClassLoader().getResourceAsStream("connection.xml");
AFSwinxTable table = AFSwinx.getInstance().getTableBuilder()
                     .initBuilder("country", connectionStream,
                                                    "tableCountryPublic")
                     .buildComponent();
```

Obrázek 9 vytvoření tabulky v AFSwinx

```
InputStream connectionStream =
getClass().getClassLoader().getResourceAsStream("connection.xml");
AFSwinxForm form = AFSwinx.getInstance().getFormBuilder()
                     .initBuilder("countryForm", connectionStream,
                                                    "countryAdd")
                     .buildComponent();
```

Obrázek 10 vytvoření formuláře

K vytvoření tabulky či formuláře je potřeba specifikovat zdroje. V předchozím příkladu jsme provedli specifikaci na základě XML. Na Obrázek 11 je definice zdrojů. V kořenovém elementu connectionRoot může být 0 až N elementů typu connection s identifikátorem, na základě kterého je rozhodnuto jaký zdroj se použije. Element metaModel v elementu connection je povinný neboť specifikuje zdroj s definicemi z Obrázek 3. Element data specifikuje konkrétní data z Obrázek 4 a element post specifikuje zdroj, na který budou data odeslána. Kromě textového zápisu lze využít i expression language a to v její standardní formě například: #{id}. Zdroje zobrazené na Obrázek 11 definice zdrojů získávají data o metamodelu z nezabezpečeného zdroje na adrese http://localhost:8080/AFServer/rest/country ve formátu JSON. Data pro metamodel jsou na adrese http://localhost:8080/AFServer/rest/country/list a data budou zpětně zaslána na adresu http://localhost:8080/AFServer/rest/country metodou post. Podporované metody jsou get, post, put a delete.

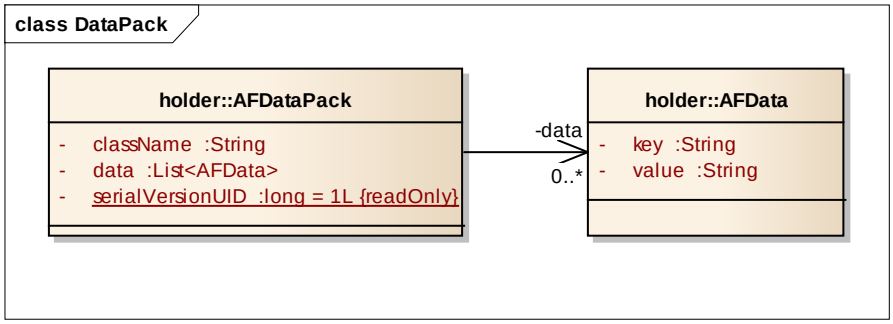
```xml
<?xml version="1.0" encoding="UTF-8"?>
<connectionRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
	<connection id="tableCountryPublic">
		<metaModel>
			<endPoint>localhost</endPoint>
			<endPointParameters>
				/AFServer/rest/country/definition
			</endPointParameters>
			<protocol>http</protocol>
			<port>8080</port>
			<header-type>
				<param>content-type</param>
				<value>Application/Json</value>
			</header-type>
		</metaModel>
		<data>
			<endPoint>localhost</endPoint>
			<endPointParameters>
				/AFServer/rest/country/list
			</endPointParameters>
			<protocol>http</protocol>
			<port>8080</port>
		</data>
	</connection>
		<connection id="countryAdd">
		<metaModel>
			<endPoint>localhost</endPoint>
			<endPointParameters>
				/AFServer/rest/country/definition
			 </endPointParameters>
			<protocol>http</protocol>
			<port>8080</port>
			<header-param>
				<param>content-type</param>
				<value>Application/Json</value>
			</header-param>
			<security-params>
				<security-method>basic</security-method>
				<userName>#{username}</userName>
				<password>#{password}</password>
			</security-params>
		</metaModel>
		<send>
			<endPoint>localhost</endPoint>
			<endPointParameters>/AFServer/rest/country</endPointParameters>
			<protocol>http</protocol>
			<method>post</method>
			<port>8080</port>
			<security-params>
				<security-method>basic</security-method>
				<userName>#{username}</userName>
				<password>#{password}</password>
			</security-params>
		</send>
	</connection>

</connectionRoot>
```

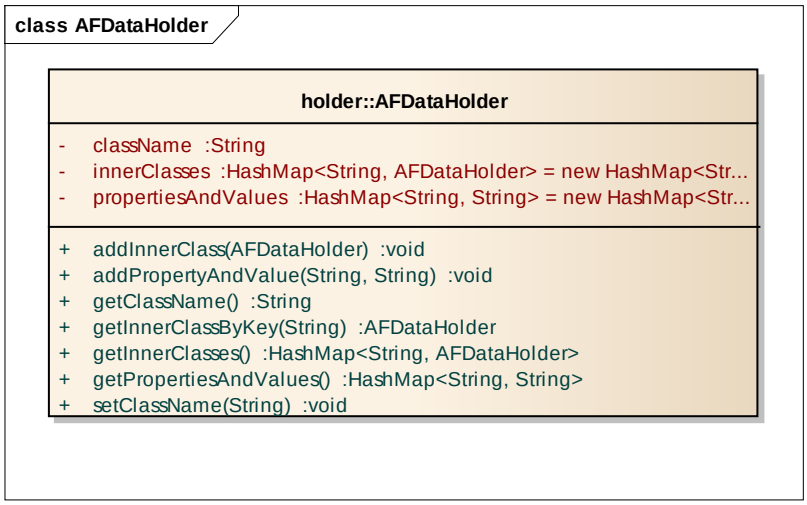
Obrázek 11 definice zdrojů

2.4.2. Získání dat

V případě potřeby lze z komponent získat data. Pokud se jedná o tabulku, můžeme získat aktuálně vybraná data a v případě formuláře aktuální data, která jsou ve formuláři zobrazena. Datový objekt nesoucí data je v případě formuláře AFDataHolder, zobrazený na Obrázek 13 - objekty, které reprezentují aktuální data ve formuláři a v případě tabulky je to AFDataPack, zobrazený na Obrázek 12 - objekty, které reprezentují aktuální data v tabulce. Oba tyto objekt drží konkrétní proměnné a hodnoty k nim. Objekt AFDataHolder udržuje informace o hodnotách a konkrétních proměnných v hash mapě innerProperties a neprimitivní proměnné jsou v hash mapě innerClasses. V případě AFDataPack jsou proměnné a jejich hodnoty v listu, který drží objekt AFData. Pro detailní informace lze využít JavaDocu či zdrojových souborů.



Obrázek 12 - objekty, které reprezentují aktuální data v tabulce



Obrázek 13 - objekty, které reprezentují aktuální data ve formuláři

2.4.2.1. Formulář

Způsob, jak získat data z formuláře je na Obrázek 14 získání dat z formuláře. Nejprve je potřeba získat konkrétní instanci formuláře, pomocí klíče, pod kterým byl vytvořen. A poté aktuální data resealizovat.

2.4.2.2. Tabulka

Z tabulky lze získat pouze data odpovídající aktuálnímu vybranému záznamu. Ukázka je na Obrázek 15 získání vybraných dat z formuláře a vložení je do tabulky. Nejprve je potřeba získat konkrétní instanci tabulky a poté data získat pomocí metody getSelectedData(). Pokud nejsou žádná data vybrána je vyhozena IndexOutOfBoundException. Tutu výjimku je vhodné odchytit a adekvátně ji zpracovat. Vybraná data z tabulky lze ihned nastavit do formuláře, pokud byl vytvořen na základě stejné definice. V případě, že nebyl, pak budou vyplněné pouze ty sloupečky, které se shodují. Ostatní zůstanou prázdné. Ukázka je taktéž na Obrázek 15 získání vybraných dat z formuláře a vložení je do tabulky.

```
AFDataHolder data =
AFSwinx.getInstance().getExistedComponent("loginForm").resealize();
```

Obrázek 14 získání dat z formuláře

```
protected void chooseDataInTableAndSetToForm(String tableId, String
formId){
      AFSwinxTable table =
                  (AFSwinxTable)
                  AFSwinx.getInstance().getExistedComponent(
                      tableId);
         try{
            List<AFDataPack> datas = table.getSelectedData();
            AFSwinxForm form =
                  (AFSwinxForm)
                  AFSwinx.getInstance().getExistedComponent(
                      formId);
            form.fillData(datas);
         }
         catch(IndexOutOfBoundsException exception){
                  view.getDialogs().failed(
                        "afswinx.choose.table.choosed",
                        "afswinx.choose.table.outOfIndex",
                         exception.getMessage());
         }
    }
```

Obrázek 15 získání vybraných dat z formuláře a vložení je do tabulky

3. Možnosti

3.1. Mapování

Jak již bylo předesláno. Framework využívá k inspekci data framework AspectFaces. Součástí tohoto frameworku je mapování datových typů na komponenty. Komponenty jsou definovány pomocí XML. Mapování datových typů na komponenty je také ve formátu XML. Ukázka mapování je na Obrázek 16 Mapování datových typů na komponenty. Objekt typu String se bude mapovat na komponentu, jejíž definice je popsána v souboru inputField.xml. Do této komponenty budou propagovány proměnné minLength a maxLength. V případě že se jedná o heslo tak se bude objekt mapovat na komponentu inputPassword.xml. Objekty typu int a long se budou mapovat na komponentu numberInput.xml. Objekt typu Address se bude mapovat na entity.xml, neboť se jedná o složitý datový objekt. Cesty ke komponentám odpovídají struktuře na Obrázek 8 struktura projektu.

Podporované vlastnosti, kterými komponenta disponuje, jsou v Tabulka 1 podporované vlastnosti komponent. Tato tabulka obsahuje kromě názvu elementu, pod kterým ho lze vložit do XML, také popis co daný element značí a jaký je jeho význam.

```
<mapping>
	<type>String</type>
	<default tag="structure/inputField.xml" maxLength="255" minLength="4" />
	<condition expression="${type == 'password'}"
	tag="structure/inputPassword.xml"/>
</mapping>

<mapping>
	<type>int</type>
	<type>long</type>
	<default tag="structure/numberInput.xml" />
</mapping>

<mapping>
	<type>Address</type>
	<default tag="structure/entity.xml" />
</mapping>
```

Obrázek 16 Mapování datových typů na komponenty

Na základě mapování se použije definice komponenty. Komponenta inputField.xml je zobrazena na Obrázek 17 definice komponenty structure/inputField.xml. Komponenta musí začínat kořenovým elementem widget.

```
<widget>
	<widgetType>textField</widgetType>
	<fieldName>$field$</fieldName>
	<label>$label$</label>
	<readonly>false</readonly>
	<visible>true</visible>
	<validations>
		<required>$required$</required>
		<minLength>$minLength$</minLength>
		<maxLength>$maxLength$</maxLength>
	</validations>
	<fieldLayout>
		<layoutOrientation>$layoutOrientation$</layoutOrientation>
		<labelPossition>$labelPossition$</labelPossition>
		<layout>$layout$</layout>
	</fieldLayout>
</widget>
```

Obrázek 17 definice komponenty structure/inputField.xml

Tabulka 1 podporované vlastnosti komponent

Element	Popis
widgetType	Typ komponenty. Specifikuje, zdali se jedná o textové pole, číselné pole, drodown pole a podobně. Výčet možností bude specifikován v další kapitole
fieldName	Název pole. Například v objektu country z druhé kapitoly by se mohlo jednat například o name, shortCut a active.
Label	Popis, který bude zobrazen klientovi u políčka s touto hodnotou. Lze použít i lokalizační texty například country.name a framework si z resource bundle již hodnotu pro country.name dohledá sám.

Validations	Validace, které bude vstupní pole podporovat. Výčet validací bude rozebrán dále
fieldLayout	Popis layoutu. Layout vstupního pole na základě této definice budou prvky skládány v rámci této komponenty.
Readonly	Specifikuje, zdali je objekt pouze pro čtení.
Visible	Specifikuje, zdali je objekt viditelný.

V případě, že se jedná o složitý datový typ je využitá komponenta entity.xml. Tuto komponentu nelze měnit a v případě změny, může tato změna způsobit nefunkční inspekci složitých datových typů. Této komponentě se předá název třídy, nad kterou bude provedena inspekce a název proměnné v aktuální entitě. Ukázka je na Obrázek 18 entity.xml – komponenta popisující neprimitivní datový typ.

```
<entityClass>
	<entityFieldType>$DataTypeFullClassName$</entityFieldType>
	<fieldName>$fieldName$</fieldName>
</entityClass>
```

Obrázek 18 entity.xml – komponenta popisující neprimitivní datový typ

3.2. Komponenty

V Tabulka 2 Podporované komponenty je seznam podporovaných komponent včetně jejich názvu a identifikace v XML.

Tabulka 2 Podporované komponenty

Komponenty	Identifikátor	Popis
Vstupní textové pole	textField	Standardní vstupní pole pro text
Pole textu	Label	Standardní výstupní pole. Needitovatelné. Zobrazování jako text.
Vstupní číselné pole integer	numberField	Vstupní pole pro čísla typu int.
Vstupní číselné pole double	numberDoubleField	Vstupní pole pro čísla typu double.
Vstupní číselné pole long	numberLongField	Vstupní pole pro čísla typu long.
Výběrové menu	dropDownMenu	Výběrové menu. Lze vybrat jednu z několika možností.
Zaškrtávací pole	checkBox	Zaškrtávací políčko, Lze zaškrtnout jednu z možností.
Vstupní textová area	textArea	Vstupní textové pole pro velké texty.
Výběrové menu	option	Výběrové menu lze vybrat jednu z možností.
Datové pole s kalendářem	calendar	Vstupní pole, do kterého se vloží data pomocí date pickeru.
Vstupní pole pro heslo	Password	Vstupní pole pro heslo.

3.2.1. Validace

Komponenty mohou mít různé validace. Validace nespecifikují vizuální stav komponenty, ale ověřují její vnitřní stav, předtím než je komponenta využita ke zpětnému sestavení objektu. Podporované validace jsou v Tabulka 3 Podporované validace. Tato tabulka obsahuje jejich název, defaultní lokalizační klíč a popis funkčnosti.

Tabulka 3 Podporované validace

Název	Lokalizační klíč	Popis
MinValue	validation.number.toSmall	Validuje minimální číselnou hodnotu pole.
MaxValue	validation.number.toBig	Validuje maximální číselnou hodnotu pole.
MinLength	validation.length.toSmall	Validuje minimální počet znaků v poli.
MaxLength	validation.length.toBig	Validuje maximální počet znaků v poli.
Required	validation.required	Validuje, zdali je pole vyplněno.
Number	validation.number	Validuje, zdali je v poli číselná hodnota. Typ číselné hodnoty je závislé na číselné.
Retype	validation.retype	Validuje, zdali bylo pole správně opsáno.
Contains	validation.contain	Validuje, zdali je v poli obsažena určitá hodnota.

3.2.2. Layouty

Uvnitř komponenty lze určovat, jakým způsobem bude komponenta postavena a kde bude umístěn její popis (label). Layouty jsou určeny podle os. Rozeznáváme osu X a Y a jednosloupcový layout a dvousloupcový layout. Zároveň lze určit pozice labelu. Zdali bude jako první či jako poslední. Tabulka 4 Určování layoutu tabulka specifikuje prvky layoutu.

Tabulka 4 Určování layoutu

Název	Popis
LabelPossition (before, after, none)	Pozice labelu. Před komponentou, po komponentě, label nebude uveden.
LayoutDefinition (TwoColumnsLayout, OneColumnLayout)	Definice layoutu dvousloupcový, jednosloupcový.
LayoutOrientation(AxisX, AxisY)	Orientace layoutu (podle osy X, Y)

V případě dvousloupcového layoutu podle osy Y budou komponenty vykreslovány nejprve dvě vedle sebe a další pod ně. V případě dvousloupcového layoutu podle osy X. Budou komponenty vykreslovány nejprve jako dvě pod sebe a vedle nich další. Počet sloupců určují počet komponent na jednom patře a orientace určuje orientaci všech pater.

4. Závěr

Tato uživatelská příručka slouží k prvnímu seznámení s generováním uživatelského rozhraní pomocí webových služeb. Pro detailní použití odkazuji na ukázkový projekt, který je přiložen

a zdrojové kódy tohoto ukázkového projektu. Součástí je také již předpřipravené mapování souborů frameworku AspectFaces na jehož základě je vytvářena definice dat.