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

Bakalářská práce

Doplnění mockup komponent pro simulační testování

Plzeň, 2013 Michal Říha

Prohlášení

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

V Plzni dne 24. července 2013

Michal Říha

Abstract

Creating component mockups for simulation testing

The goal of this bachelor's thesis is to explore possibilities in creating mockup components and implement such mockups of testing application. These shall be used in a SimCo simulation software for research and testing in the area of component application development. We are using OSGi™ - The Dynamic Module System for Java™, Spring Framework and Spring Dynamic Modules.

Abstrakt

Doplnění mockup komponent pro simulační testování

Cílem této bakalářské práce je prozkoumat možnosti v oblasti vytváření mockup komponent a implementace takových mockupů pro testovací aplikaci. Tyto pak budou použity společně se simulátorem komponentových aplikací SimCo při výzkumu a testování v oblasti komponentových aplikací. Použité technologie jsou OSGi™ - The Dynamic Module System for Java™, Spring Framework a Spring Dynamic Modules.

Obsah

1 Úvod

Cílem této práce je vytvořit sadu mockupů[1] ke komponentám reálné aplikace. Tyto mockupy budou následně použity při simulačním testováni dané aplikace v simulátoru SimCo.

Výstupem této práce jsou mockupy komponent testovací aplikace. Pro jejich generování vznikl nástroj, který tyto mockupy vytváří automaticky za běhu aplikace na základě scénáře popsaného v xml souboru. Díky tomu stačí pro testování vytvořit pouze scénář a už není třeba vytvářet fyzické bundly pro tyto mockupy. Pro scénáře vznikl popis syntaxe xml souborů a byla vytvořena komponenta pro načítání a ukládání scénářů.

Součástí práce je také vlastní implementace jednoduchého simulátoru, který přehrává simulaci opět popsanou scénářem uloženém v xml souboru. Pro tuto část stejně jako v předchozím případě byla popsána syntaxe xml souboru scénáře a vytvořena komponenta pro snadné načítání a ukládání scénářů. Tento přehrávač nevznikal jako náhrada simulátoru SimCo, ale pouze jako jednoduchý prostředek pro testování vznikajícího generátoru mockupů. Jelikož se však postupným vývojem ukázal jako docela schopná alternativa, přikládám ho jako součást práce.

[1] mockup - maketa pro účely studia, testování nebo zobrazení [18], zde komponenta která implementuje stejné rozhraní a navenek tedy poskytuje stejnou funkčnost, ale uvnitř vykonává předdefinovaný kód pro účely testování nebo prototypování

2 Komponentové programování

Snad každý vývojář se při tvorbě a údržbě větší aplikace dostane do situace, kdy je kód aplikace už příliš komplikovaný. Tak komplikovaný, že například oprava chyby může způsobit výskyt nových chyb v jiné části aplikace, která spoléhala na vadné chování původního kódu. Dalším problémem může být například nutnost pokaždé celou(!) aplikaci znovu zkompilovat, otestovat a nasadit do produkčního prostředí [10].

Řešením může být tvorba aplikace komponentovým způsobem. Aplikace je pak složena z většího množství kompaktních celků, tzv komponent. Tyto komponenty jsou navzájem propojeny pouze velice striktně definovaným způsobem, jenž brání tvorbě skrytých závislostí a vztahů, které by mohly komplikovat vývoj aplikace [4].

V komponentovém programování existuje několik odlišných postupů. První, bližší klasickému monolitickému programování, sice dělí aplikaci na komponenty, ale v okamžiku, kdy se aplikace spustí, dojde k jejich načtení a provázání a aplikace pak dále běží jako monolitická. To znamená, že údržba takové aplikace si vyžaduje její kompletní zastavení, vykonání potřebných prací a následné kompletní znovu spuštění [10]. Příkladem implementace takového postupu je Spring Framework [7].

Druhým způsobem je skládání aplikace za běhu. V takové případě je nutná existence určitého spouštěcího prostředí, které umožní správu načítání komponent, jejich provazování, spouštění a na konec také jejich zastavování a aktualizaci, to celé za běhu. Tento postup umožňuje provádět údržbu pouze části aplikace, zatímco zbylá část může dál normálně fungovat [10]. Stejně tak nasazení aktualizace nebo opravy spočívá v zastavení pouze nahrazované komponenty, načtení opravené komponenty a její spuštění. Přitom samotná aplikace se může skládat i z tisíců různých komponent. Příkladem tohoto postupu jsou různé implementace specifikace OSGi [2].

2.1 OSGi - specifikace modulárního systému

Specifikace OSGi je reakcí na nedostatky jazyka Java v oblasti modularity aplikací. V Javě například není možné definovat jakou verzi knihovny má aplikace používat - pokud je jich v classpath dostupných více, Java nijak negarantuje, kterou verzi systémový classloader vrátí [10].

Dále není možné podrobněji řídit přístupnost jednotlivých tříd. V Javě může být třída buď veřejná, pak je dostupná v rámci celého běžícího systému, a nebo může být bez označení. V takovém případě je dostupná pouze v rámci stejného balíku [5]. Takové dělení je však pro komplexnější aplikace často nedostatečné, protože mohou vznikat závislosti na neveřejných API mezi různými částmi aplikace, což znesnadňuje údržbu aplikace.

Specifikace OSGi je často mylně označována jako OSGi framework. OSGi je však pouze specifikací chování komponentového systému a až teprve její konkrétní implementace je OSGi frameworkem [10]. Mezi nejznámější implementace patří Eclipse Equinox, Apache Felix a Knopflerfish. Zřejmě nejpoužívanější je framework Equinox a to jednoduše proto, že je základem vývojového populárního prostředí Eclipse, který je mimo jiné základem nejrůznějších vývojových kitů(například Android SDK [20], Samsung SmartTV [21], PowerLinux [22]).

2.1.1 Moduly v prostředí OSGi

Základní jednotkou aplikace v prostředí OSGi je tzv. *bundle*. Český překlad *balík* bohužel koliduje s označením jmenného prostoru v Javě (*package*) a proto se nepoužívá [11]. Bundle je standardní `jar` archiv, který má ve svém manifest souboru uvedeny speciální hlavičky dané specifikací. Následuje ukázka těch nejčastějších [17].

Bundle-ManifestVersion: Verze hlaviček, obvykle má hodnotu 2.

Bundle-Name: Obecné pojmenování bundlu.

Bundle-SymbolicName: Jméno, pod kterým bundle vystupuje v systému[1].

Bundle-Version: Verze bundlu[2].

Import-Package: Seznam balíčků, které tento bundle používá.

Export-Package: Seznam balíčků, které bundle zveřejňuje k použití.

Bundle-Activator: Jméno třídy, která zajišťuje inicializaci bundlu při jeho aktivaci, pokud je potřebná.

Hlavičky **Import-Package** a **Export-Package** umožňují kromě jména balíku určit také konkrétní verzi. Pro import je možné určit jednu konkrétní verzi, rozsah verzí od-do, minimální a maximální verzi, kterou bundle vyžaduje. Díky tomu může být v systému přítomných více verzí a lze se spolehnout, že dostaneme vždy tu správnou. Výpis 2.1 ukazuje příklad manifest souboru OSGi bundlu.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: UniPlayerBindings
Bundle-SymbolicName: cz.zcu.kiv.bp.uniplayer.bindings
Bundle-Version: 2.0.1.qualifier
Import-Package: cz.zcu.kiv.bp.namespaces;version="[1.0, 2.0)",
 org.osgi.framework;version="1.3.0",
 org.osgi.service.event;version="1.3.0"
Export-Package: cz.zcu.kiv.bp.uniplayer.bindings,
 cz.zcu.kiv.bp.uniplayer.bindings.adapted,
 cz.zcu.kiv.bp.uniplayer.bindings.basics
```

Výpis 2.1: Ukázka manifestu OSGi bundlu

[1] jediná povinná hlavička

[2] ve formátu *major.minor.micro.kvalifikátor* (0.0.0.kval)

Autor bundlu při jeho tvorbě stojí před rozhodnutím, které balíčky exportovat. Doporučenou praxí je exportovat pouze rozhraní [1]. Díky tomu bude každá aplikace používající daný bundle nezávislá na konkrétní implementaci funkčnosti poskytované bundlem [1]. Navíc je možné daný bundle snadno zaměnit za jiný, který bude implementovat stejné rozhraní.

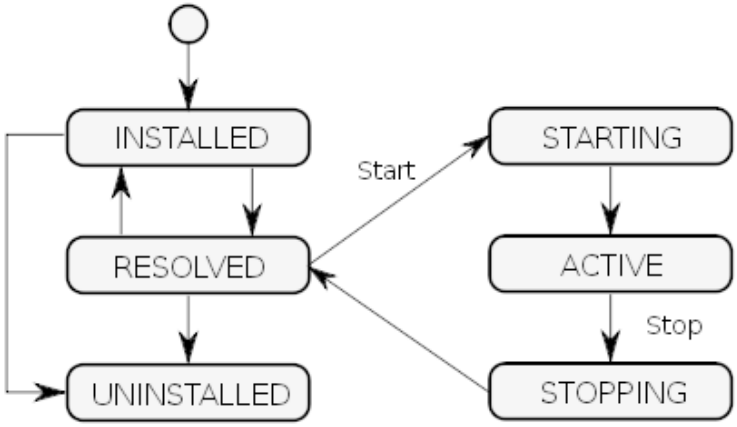
Samotná rozhraní lze exportovat jejich umístěním do balíčků pojmenovaných jinak než jsou balíčky, kde se nachází samotné implementace a pak exportovat pouze balíčky s rozhraními. Velmi často se volí rozlišení umístěním implementace do balíčku o jednu úroveň hlouběji pojmenovanou jako `impl`, `internal` a podobně [10]. Možnost určit, které balíčky budou viditelné vně bundlu souvisí s jednou velmi důležitou vlastností OSGi, která jde proti zvyklostem v Javě.

Java jako objektový jazyk s interpretovaným kódem musí implementovat mechanismy pro zavádění bytekódu tříd do paměti. Tento mechanismus je zajištěn prostřednictvím tzv. class loaderů. Tradičně se správa načítání tříd nechává přímo na virtuálním stroji Javy. Ten pracuje se třemi loadery, tzv. *boostrap* slouží pro načítání tříd tvořících jádro Javy (`<JAVA_HOME>/jre/lib`). Druhý loader (*extension* je zodpovědný za načítání tzv. rozšíření, tj. tříd z cesty `<JAVA_HOME>/jre/lib/ext`. Posledním je systémový loader (*system*), který načítá zdroje z cest určených systémovou proměnnou `java.class.path`. Tyto tři class loadery jsou sestaveny v hierarchii tak, že *bootstrap* je rodičem *extension* a ten rodičem systémového. Díky tomu jsou všechny balíčky a v nich obsažené veřejné třídy dostupné celé aplikaci [5].

OSGi však vytváří pro každý bundle vlastní class loader. Tyto class loadery mají společného předka, který jim zpřístupňuje balík `java.*` a balíky importované pomocí hlaviček v manifest souboru [23]. Takové chování však způsobuje problémy u technologií, které při své práci spoléhají na *TCCL - Thread Context Class Loader*, jako například Swing, JAXB atd. [9]. OSGi však obvykle používá ke svému běhu celou řadu vláken, ale zároveň nedefinuje podobu TCCL.

2.1.2 Životní cyklus bundlu

Základní vlastností OSGi je online modularita, bundly se do prostředí zavádí až za běhu. To je zvláště výhodné u komplexních aplikací, kdy je možné pozastavit jen část aplikace, provést údržbu a po té danou funkci opět aktivovat, aniž by muselo dojít k úplnému odstavení, tak jak je to u monolitických aplikací nebo aplikací využívajících Spring Framework (viz. kapitola 2.2) [10]. Na obrázku 2.1 je popsán životní cyklus každého OSGi bundlu.



Obrázek 2.1: Životní cyklus OSGi bundlu, zdroj [11]

UNINSTALED zavedený bundle není spravován prostředím

INSTALLED bundle je zaveden a spravován

RESOLVED bundle má splněny závislosti, viz. `Import-Package`

STARTING provádí se aktivační kód třídy uvedené jako `Bundle-Activator`

ACTIVE kód aktivační třídy byl dokončen

STOPPING provádí se deaktivační kód `Bundle-Activator` třídy

2.1.3 Služby

Samotné skrývání popř. zveřejňování balíčků však není pro tvorbu skutečně modulárních aplikací úplně dostatečné, protože pro sestavení takové aplikace musíme přesně znát jména všech balíčků, které chceme použít, což není vždy možné (například nám část funkčnosti bude dodávat třetí strana). V některých situacích také není vhodné takto vytvářet závislost na konkrétním baličku, protože nás nemusí zajímat konkrétní obsah, ale pouze to, že baliček implementuje námi požadované rozhraní - implementuje poskytovatele službu.

Služba není nic jiného než obyčejné veřejně přístupné java rozhraní [12]. Instanci třídy implementující takové rozhraní pak její autor zaregistruje do globálního registru jako poskytovatele služby. Případně může přidat seznam libovolných upřesňujících informací. Registraci služby je možné provést kdykoliv je bundle ve stavu STARTING nebo ACTIVE [12].

Pro správu služeb slouží globální registr `ServiceRegistry`, který je přístupný všem nainstalovaným bundlům. Registr udržuje informace o existujících službách a pro každého poskytovatele také počet tzv. referencí. Mechanismus referencí slouží k bezpečnému dynamickému nahrazování služeb. Pokud je počet referencí na poskytovatele, tj. komponent, které si vyžádaly poskytovatele služby, nenulový, nelze daného poskytovatele z registru odstranit [1]. Z tohoto důvodu je doporučovanou praxí uvolňovat referenci na poskytovatele ihned, jakmile už není potřeba.

Z registru je možné poskytovatele služby získat buď podle jména služby nebo přímo rozhraní definujícího službu. Požadovaného poskytovatele služby je možné upřesnit pomocí textové reprezentace vyhledávacího filtru LDAP (`http://www.ietf.org/rfc/rfc1960.txt`) [3]. Tento filtr je aplikován na pole vlastností, předávané při registraci poskytovatele [12].

Registr služeb nikdy nevrací přímo poskytovatele služby ale objekt `ServiceReference` a až pak tyto objekty nesou referenci na poskytovatele služby [3]. Postup získání poskytovatele je ukázán na výpisu 2.2.

```
// ziskej referenci na poskytovatel sluzby
ServiceReference<?> reference = context.getServiceReference(
    IMyService.class.getName()
);
//  z reference ziskej objekt poskytovatele
IMyService service = (IMyService)
    context.getService(serviceReference);

// alternativne
ServiceReference<IMyService> reference =
    context.getServiceReference(IMyService.class);
IMyService service = context.getService(serviceReference);
... // nejaka prace se sluzbou

// uvolneni reference na poskytovatele
context.ungetService(reference);
```

Výpis 2.2: Ukázka získání OSGi služby

2.1.4 Události

Mechanismus událostí je způsob jakým může libovolná komponenta informovat ostatní komponenty o změně svého stavu přičemž však nemusí vytvářet žádné vazby na tyto komponenty. Nemusí se dokonce ani starat, zda vůbec nějaké takové komponenty skutečně existují.

Systém událostí vyžaduje určitého prostředníka, který zajistí, aby komponenta mohla nějakým způsobem říci, že ji zajímají informace o určitých specifických událostech. Stejně tak je potřebný prostředek pro vytváření a publikování událostí. Takovým prostředníkem je v OSGi služba `org.osgi.service.event.EventAdmin` [3].

Komponenta, která je původcem události se nazývá *publisher* a komponenty, které na tuto událost chtějí reagovat se nazývají *subscribers*. Samotná událost je reprezentována instancí třídy `org.osgi.service.event.Event` [1]. Tato instance může také nést objekt, který je nějakým způsobem svázaný s konkrétní událostí. Například v grafických uživatelských rozhraních to bývá její původce, často to je podoba komponenty před a po změně stavu. OSGi definuje, že takovým objektem je instance třídy `java.util.Dictionary` [1], která nese libovolné množství párů klíč/objekt.

Specifikace OSGi popisuje dva způsoby publikování události - synchronní a asynchronní. Při synchronním publikování, čeká publikující komponenta na dokončení zpracování události všemi komponentami, které událost obsluhují. Při asynchronním způsobu komponenta událost pouze publikuje správci událostí a pokračuje dále v provádění svého kódu. [3]

Protože posluchače událostí zajímá většinou pouze úzká skupina konkrétních událostí, je nutná existence mechanismu jejich popisu a filtrování. Prostředí OSGi třídí události podle tzv. *topics* – jména událostí. Komponenty se pak registrují ke správci událostí pro naslouchání na události s konkrétními tématy. [16]

Komponenta, která chce naslouchat na události, musí být poskytovatelem služby `org.osgi.service.event.EventHandler` a při své registraci musí mít v předané mapě vlastností (viz. kapitola 2.1.3) pod klíčem `EventConstants.EVENT_TOPIC` pole řetězců se jmény událostí [3]. Správce události (služba `EventAdmin`) umožňuje ve

jméně události použít na poslední pozici znak '*'. Posluchač tím říká, že ho zajímají všechny události, jejichž jméno začíná řetězcem uvedeným před hvězdičkou [16]. Způsob registrace je ukázán ve výpisu 2.3.

```
String[] topics;
// bude naslouchat pouze na udalost foobar
topics = new String {"foo/bar"};
// nebo muze naslouchat na udalosti zacinajici na 'foo/bar/'
topics =  new String {"foo/bar/*"});

Hashtabl properties = new Hashtable();
properties.put(EventConstants.EVENT_TOPIC, topics);
// registrace posluchace
context.registerService(
    EventHandler.class.getName(),
    EventHandler instance,
    Hashtable properties
);
```

Výpis 2.3: Ukázka registrace k naslouchání na událostí z oboru foo/bar*

2.2 Spring framework

Spring Framework je open source aplikační framework, který si klade za cíl zbavit vývojáře potřeby zabývat se specifickými postupy konfigurace různých technologií často používaných při vývoji velkých enterprise aplikací. Každá z takových technologií často vyžaduje vlastní konfiguraci a postupy při implementaci. Problém je, že se tyto procesy většinou opakují a zároveň jejich výstupy jsou si velmi podobné. Úkolem Springu tedy je odstínit vývojáře od těchto věcí a zpřístupnit je pomocí abs-

traktních rozhraní a společných konfiguračních souborů. Díky tomu se pak aplikace stává nezávislou na konkrétních použitých technologiích. Příkladem jsou například techniky řešení persistence, transakcí nebo práce s databázemi. [7]

Základem Springu je balík modulů zvaný **Core Container**, který obsahuje moduly **Core**, **Beans**, **Context** a **Expression Language**. [7]

Moduly **Core** a **Beans** tvoří úplný základ frameworku v podobě implementace techniky zvané **Dependecy Injection** a **Inversion of Control** kontejneru (viz. kapitola 2.2.1).

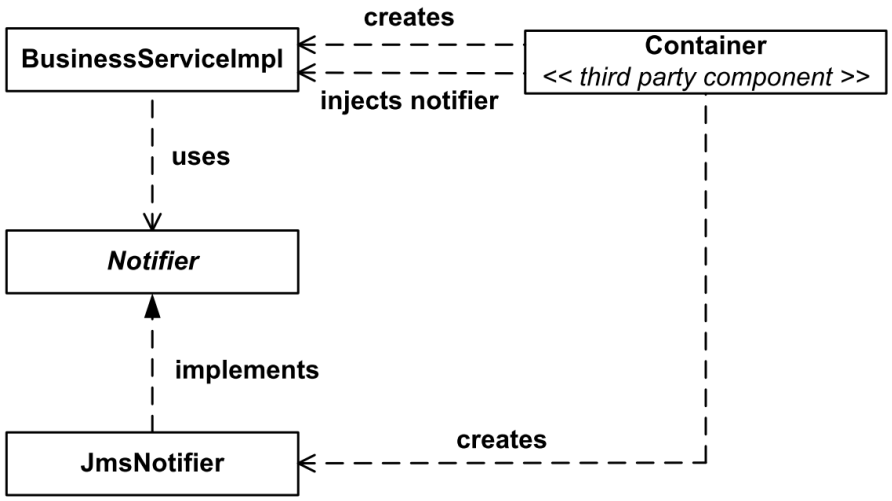
Modul **Context** staví na předchozích modulech a vytváří tzv. *aplikační kontext* poskytující nástroje pro podporu událostí, načítání zdrojů, internacionalizaci a jiné.

Expression Language implementuje jazyk a nástroje pro práci s kontejnerem a manipulaci s grafem projení objektů.

2.2.1 IoC – Inversion of Control

Základem Spring frameworku je IoC kontejner [7]. Jeho úkolem je převzít odpovědnost za vytváření objektů (instancí tříd), tzv. *beans*, a jejich provázání podle definovaných závislostí do grafu objektů. K vytváření grafu framework využívá techniku DI – *Dependency Injection*. Podporuje předávání závislostí prostřednictvím vlastností(property), konstruktorů a rozhraní [7]. Na obrázku 2.2 je znázorněn princip návrhového vzoru Dependency Injection.

Bez využít IoC musí každá komponenta řešit své závislosti vlastním způsobem, tj. součástí kódu komponenty je i nezbytná obsluha pro získávání požadovaných objektů, ošetření chybových stavů (potřebný objekt nemusí existovat, může být zrušen, ...) a reakce na ně. To znamená, že většina komponent, které ke svému běhu vyžadují



Obrázek 2.2: Princip DI, zdroj [10]

jiné komponenty, obsahuje vždy víceméně stejné úseky kódu lišících se jen v typech objektů, které hledají. Takové provázání se s růstem aplikace čím dál hůře udržuje a vytváří skryté závislosti. Přitom se však nejedná o kód vykonávající tzv. obchodní logiku aplikace. A zde nastupuje IoC kontejner, který na sebe převezme veškerou zodpovědnost za vytváření instancí a jejich předávání závislým objektům. [10]

Spring umožňuje konfiguraci závislostí prostřednictvím xml souboru nebo pomocí tzv. anotací. Použití anotací však váže implementaci aplikace na Spring, což je proti jeho filozofii – architektura nesmí předepisovat implementaci [7]. Výpis 2.4 ukazuje definici jednoduché Spring beany. Na výpisu 2.5 je ukázáno vytvoření a použití kontextu v aplikaci.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.sf.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:osgi="http://www.springframework.org/schema/osgi"
 xsi:schemaLocation="
```

```
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd">
<beans>
<!-- vytvor beanu notifierImpl -->
<bean id="notifierImpl" class="foo.bar.notify.JmsNotifier" />
<!-- vytvor beanu businessService -->
<bean id="businessService"
  class="foo.bar.service.impl.BusinessService">
<!-- vytvarena beana vyzaduje injektovani beany notifierImpl -->
    <property name="notifier" ref="notifierImpl" />
</bean>
</beans>
```

Výpis 2.4: Ukázka jednoduché konfigurace kontextu

```
ApplicationContext context = new ClassPathXmlApplicationContext(
  new String[] {"config.xml"}
);
// pozadej o nakonfigurovanou beanu businessService
BusinessService service = context.getBean(
  "businessService", foo.bar.service.impl.BusinessService.class
);
```

Výpis 2.5: Vytvoření a použití kontextu kontextu

2.2.2 Další moduly Springu

Spring framework dále obsahuje moduly pro abstrakci nad různými běžně používanými technologiemi. Například modul pro práci s relačními databázemi, modul pro transakční zpracování, modul bezpečnostního systému, modul pro tvorbu testů a moduly pro tvorbu webových/síťových aplikací. Důležitou součástí je také modul pro podporu aspektového programování(AOP). Popis technologií podporovaných těmito moduly však není tématem této práce a proto jim nebude věnován další prostor.

2.3 Spring Dynamic Modules

Ačkoliv Spring poskytuje robustní mechanismus konfigurace závislostí, s komplexní aplikací se i takováto konfigurace stává složitou, nepřehlednou a těžko udržovatelnou. Další nevýhodou je již zmíněná statičnost vytvořeného aplikačního kontextu, kde neexistuje mechanismus změn po jeho vytvoření. [10]

OSGi na druhou stranu neposkytuje žádnou podporu pro moderní programovací techniky jaké podporuje Spring. Silná orientace OSGi na dynamické služby je navíc velmi náchylná ke vzniku chyb při tvorbě aplikací. O této složitosti svědčí mimo jiné existence několika různých způsobů práce se službami. [10]

Spring DM si tedy klade za cíl umožnit tvorbu aplikací s pomocí moderních programovacích technik a zároveň jejich nasazení v OSGi prostředí [7].

Základní modulární jednotkou OSGi je *bundle*, který může exportovat balíky, importovat balíky a také poskytovat služby. Ve Springu je takovou základní jednotkou *aplikační kontext*, který obsahuje beany. Aplikační kontexty mohou být sestaveny do hierarchického stromu, kdy kontext potomka má přístup k beanám rodiče, ale už ne naopak. Tato vlastnost je využita pro export bean jako poskytovatelů OSGi

služeb [8].

Spring DM definuje tzv. *Spring powered bundly*, což jsou standardní OSGi bundly, které splňují jednu z následujících podmínek: [8]

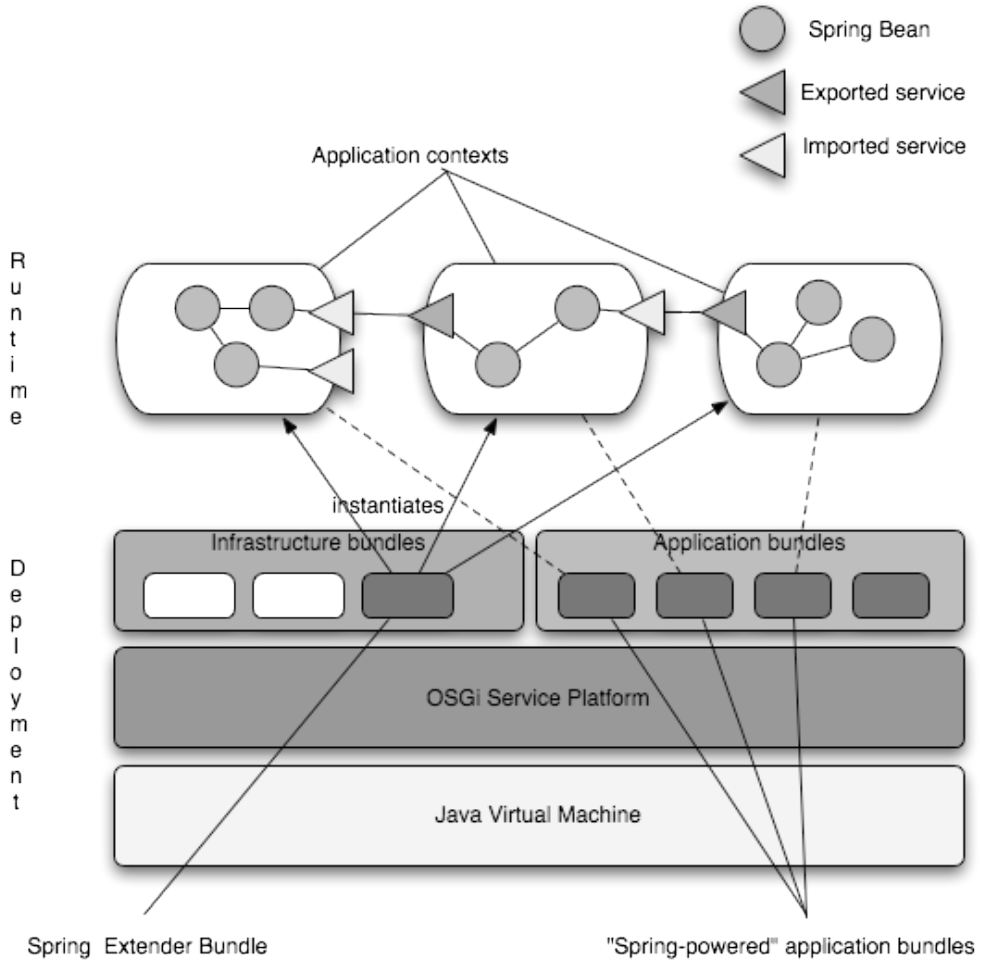
- Uvnitř bundlu se nachází adresář `META-INF/spring` obsahující jeden nebo více xml souborů.
- Manifestový soubor bundlu obsahuje hlavičku `Spring-Context`, jejíž hodnotou je seznam čárkami oddělených konfiguračních souborů.

Spring DM tedy vytváří pro každý bundle vlastní aplikační kontext a tyto pak skládá do hierarchického stromu [8]. Obrázek 2.3 ukazuje obecnou strukturu běžícího OSGi prostředí s využitím Spring DM.

2.3.1 Základ Spring DM

Základem Spring DM je bundle `org.springframework.osgi.extender`, který je odpovědný za vytváření aplikačních kontextů pro kompatibilní bundly. Jakmile je tento bundle aktivován v běžícím OSGi prostředí, vytvoří kontexty pro bundly ve stavu ACTIVE a dále naslouchá na události o aktivaci nových bundlů a i pro ně vytváří příslušné kontexty. [8]

Podstatné je, že pokud vytvářený kontext definuje nějakou závislost, snaží se extender tuto závislost vyřešit podle aktuální podoby kontextového stromu. Pokud však zjistí, že není možné takovou závislost uspokojit, například protože neexistuje žádný poskytovatel požadované služby, je vytváření nového kontextu zablokováno do doby, než je možné závislost uspokojit – například některý z následujících bundlů poskytuje požadovanou službu. Vytváření kontextu může být zablokováno maximálně po dobu pěti minut. Nevytvoření kontextu však nezmění stav bundlu a může tedy



Obrázek 2.3: OSGi, Spring Framework a Spring DM, zdroj [8]

dojít k existenci aktivního bundlu bez aplikačního kontextu. Pak je třeba ručně bundle zastavit a znovu spustit. [8]

Pozastavení vytváření kontextu je potřebné zejména z toho důvodu, že není možné spolehlivě určit pořadí aktivace bundlů a tedy garantovat, že nějaká služba bude v dané době již dostupná [17]. Toto chování lze změnit a vytváření kontextu pak může selhat okamžitě, jestliže nejsou splněny všechny závislosti [8].

Spring DM dále obsahuje bundly `org.springframework.osgi.core` – jádro

Spring DM - a `org.springframework.osgi.io` – podpora vstupně-výstupních operací [8].

2.3.2 Konfigurace aplikačního kontextu bundlu

Jak již bylo zmíněno v předchozím textu, Spring DM, respektive Spring framework, ke své funkci vyžadují popis, jak má vypadat daný aplikační kontext. Tento popis se provádí prostřednictvím xml souborů umístěných v adresáři `META-INF/spring`. Konfigurace může být uvedena v jednom souboru, ale může být pro přehlednost rozdělena i do více souborů. Spring DM rozšiřuje konfigurační soubory kontextu o jmenný prostor `http://www.springframework.org/schema/osgi`.Výpisu 2.6 ukazuje způsob vytvoření závislosti bundlu na službě `org.osgi.service.event.EventAdmin` a registraci beany `tm` jako poskytovatele služby `foo.bar.ITaskManager`. Při registraci poskytovatele je možné se odkazovat i na beany definované v jiném souboru. Často se tak vytváří jeden soubor pro definici bean a jeden soubor pro export a import OSGi služeb. [10]

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.sf.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:osgi="http://www.springframework.org/schema/osgi"
 xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/osgi
  http://www.springframework.org/schema/osgi/spring-osgi.xsd ">
<!-- zavilost na sluzbe org.osgi.service.event.EventAdmin -->
<osgi:reference id="EventAdmin"
 interface="org.osgi.service.event.EventAdmin" />
```

```
<!-- registruj tm jako poskytovatele sluzby foo.bar.ITaskManager
-->
<osgi:service id="TaskManager" ref="tm"
 interface="foo.bar.ITaskManager" />
<!-- zaregistruj do aplikacniho kontextu beanu tm -->
<bean name="tm" class="foo.bar.intern.TaskManagerImpl"
 init-method="start" destroy-method="stop">
    <constructor-arg name="eventAdmin" ref="EventAdmin"/>
</bean>
</beans>
```

Výpis 2.6: Konfigurace Spring-powerd bundlu

3 Simulování činnosti komponentové aplikace

3.1 Simulátor SimCo

SimCo je projekt, který vznikl na katedře informatiky v rámci diplomové práce v roce 2011. Jedná se o simulátor jehož účelem je provádět simulaci komponentové aplikace podle zadaného scénáře. Simulátor implementuje diskrétní událostní simulaci a metodu interpretace událostí. [13]

Vstupem simulátoru je množina reálných komponent aplikace, které se mají testovat. Dále je to množina simulovaných komponent, které představují jakési předdefinované testovací podmínky. Dále jsou to komponenty prostředníků, které zachytávají komunikaci mezi reálnými komponentami a umožňují tak sledovat průběh simulace i mimo simulované komponenty. Poslední součástí je pak scénář simulace, který popisuje počáteční události, se kterými simulátor začne simulaci. [13]

Simulátor definuje tyto druhy událostí: [13]

REAL_CALL Reálná metoda komunikuje s některou z jiných komponent SimCo aplikace.

REAL_RETURN Návrat z volané metody reálné komponenty.

SIMULATION_CALL Simulační metoda komunikuje s některou z jiných komponent Simco aplikace.

SIMULATION_RETURN Návrat z volané metody simulační komponenty.

REGULAR Jedná se o událost vyskytující se v pravidelných intervalech.

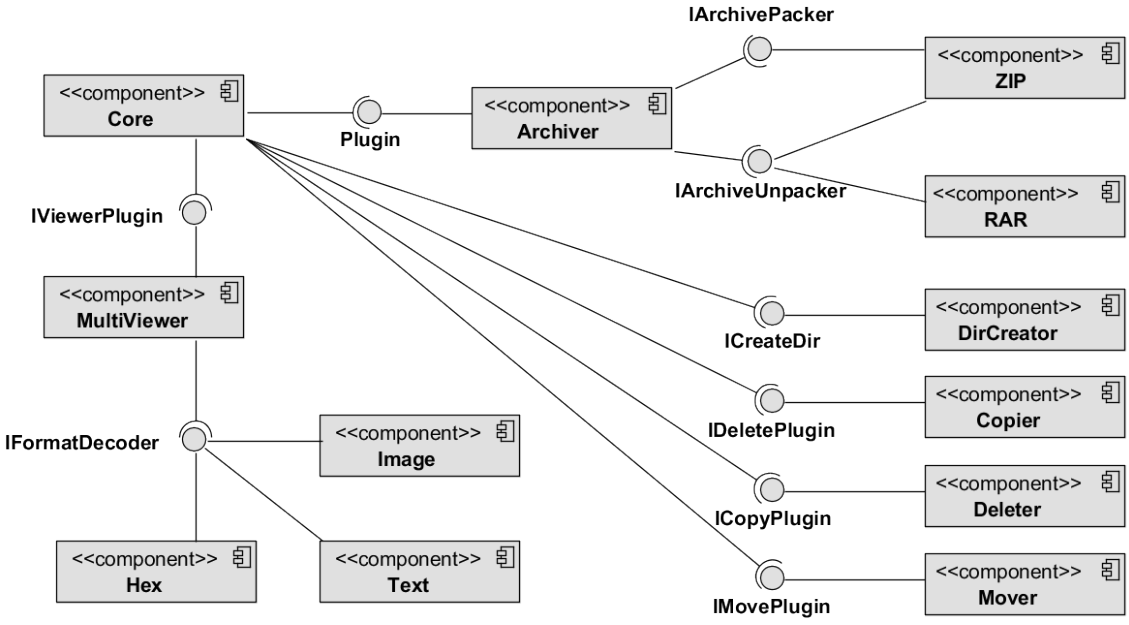
CASUAL Jedná se o událost, u které bude možno nadefinovat, s jakou pravděpodobností se bude vyskytovat.

Problémem simulátoru však je, že přenáší na uživatele povinnost vytvářet všechny části vstupu mimo reálných komponent. Cílem této práce je tedy nahradit ručně vytvářené simulované komponenty automaticky generovanými podle vstupního souboru.

3.2 Testovací aplikace – kivCommander

Jako testovací aplikace sloužila komponentová aplikace kivCommander vyvinutá v rámci semestrální práce z předmětu KIV/ZSWI v roce 2013 [14].

Jedná se správce souborů poskytujícího základní funkce. Aplikace byla vyvíjena přímo jako testovací aplikace pro potřeby vývoje simulačního software. Diagram komponent aplikace je zobrazen na obrazku 3.1



Obrázek 3.1: Diagram komponent testovaci aplikace kivCmd, zdroj [14]

Hlavní částí aplikace je komponenta `Core`, která je závislá na službách `ICopyPlugin`, `ICreateDirPlugin`, `IDeletePlugin`, `IMovePlugin` a `IViewPlugin`. Navíc využívá

všechny dostupné poskytovatele služby `Plugin`. Hlavní komponenta dále odebírá události `refresh`, `changeSourceDir`, `changeDestinationDir` a `swapFocus`.

3.2.1 MultiViewer

Služba `IViewPlugin` definuje rozhraní pro implementaci komponenty, která umožní v aplikaci zobrazit obsah vybraného souboru. Služba je poskytována komponentou `MultiViewer`, která je závislá na dostupnosti poskytovatelů služby `IFormatDecoder`.

`IFormatDecoder` umožňuje implementovat do prohlížeče podporu pro dosud nepodporované formáty. Komponenta `MultiViewer` obsahuje jednu implementaci `IFormatDecoder` a to komponentu `Hex`, která se použije pro zobrazení obsahu souboru v hexadecimálním režimu v případě, že není dostupný žádný poskytovatel podporující formát vybraného souboru.

3.2.2 Archiver

Komponenta `Archiver` implementuje poskytovatele nepovinné služby `Plugin`. Jejím cílem je rozšířit funkčnost správce o podporu práce s archivy. Komponenta definuje služby `IArchivePacker` a `IArchiveUnpacker`, které popisují obecné metody pro práci s archivy. Součástí aplikace jsou komponenty ZIP - poskytuje `IArchivePacker` i `IArchiveUnpacker` - a RAR - poskytuje pouze `IArchiveUnpacker`. Komponenta je pouze ručně vytvořeným mockupem pro účely vývoje.

3.2.3 DirCreator, Copier, Deleter, Mover

Komponenty `DirCreator`, `Copier`, `Deleter`, `Mover` implementují poskytovatele základních služeb

3.3 UniMocker

Cílem této práce je vytvořit sadu mockup komponent pro simulování testovací aplikace kivCommander prostřednictvím simulátoru SimCo. Ke splnění toho úkolu se nabízejí dvě možné cesty.

- Ručně vytvořit komponenty s rozhodovacím stromem.
- Automaticky generovat mockupy na základě souboru scénáře.

Ruční tvorba mockupů je do určitého počtu komponent rychlá a nenáročná. Nevýhodou je však jejich jednorázovost – pro každou změnu je potřeba znovu kód zkompilovat a nasadit. Automatická tvorba naopak umožňuje vytvářet mockupy přímo za běhu aplikace a ke změně jejich chování pouze stačí upravit soubor scénáře.

Pro svou práci jsem zvolil druhou variantu přičemž jsem využil mechanismu proxy tříd a objektů poskytovaný programovacím jazykem Java.

3.3.1 Proxy v Javě

Programovací jazyk Java nabízí prostředky pro vytváření tzv. zástupných tříd a objektů – tzv. *proxy* třídy a objekty. Proxy třída je dynamicky vytvořenou implementací sady rozhraní určených za běhu aplikace. Proxy objekt je pak instancí této třídy. Všechny proxy třídy jsou odvozeny od společného předka `java.lang.reflect.Proxy`. [6]

Takováto samotná třída by byla prakticky nepoužitelná, protože neimplementuje žádnou funkčnost. Proto mají všechny proxy třídy jeden konstruktor, který přijímá instanci rozhraní `java.lang.reflect.InvocationHandler`. Rozhraní definuje jedinou metodu `invoke(...)`, jenž slouží ke zpracování veškerých volání uskutečněných

na proxy objektu. Argumenty metody mají tento význam: [6]

`java.lang.Object` **proxy:** instance, na které došlo k volání metody
`java.lang.reflect.Method` **method:** metoda, která byla zavolána
`java.lang.Object[]` **args:** argumenty předané při volání metody

Hodnota vracená metodou `invoke` musí mít stejný typ jako má metoda `method` s výjimkou primitivních typů, pak se musí jednat o instanci odpovídajícího wrapperu [6]. Pro vytváření proxy tříd a objektů nabízí třída `Proxy` tovární metody `getProxyClass(...)` a `newProxyInstance(...)`. Příklad použití ukazuje výpis 3.1.

```
// objekt pro zpracovani volani na proxy objektu
InvocationHandler handler = new MyInvocationHandler(...);
// dynamicky vytvorena proxy trida implementujici rozhrani Foo
Class proxyClass = Proxy.getProxyClass(
    Foo.class.getClassLoader(),
    new Class[] { Foo.class }
);
// nova instance proxy tridy
Foo f = (Foo) proxyClass.getConstructor(
    new Class[] { InvocationHandler.class }
).newInstance(new Object[] { handler });

// nebo jednoduseji
Foo f = (Foo) Proxy.newProxyInstance(
    Foo.class.getClassLoader(),
    new Class[] { Foo.class },
    handler
);
```

```
// definice tridy implementujici InvocationHandler
class MyInvocationHandler implements InvocationHandler {
    public Object invoke(
        Object proxy, Method method, Object[] args
    ) throws Throwable {
    // nejaka cinnost
    }
}
```

Výpis 3.1: Ukázka práce s proxy

3.3.2 Využití Proxy ke tvorbě mockupů

Výše zmíněnou techniku využívá komponenta UniMocker ke generování mockupů poskytovatelů OSGi služeb. Po vytvoření proxy třídy je tedy třeba načíst třídu rozhraní, které definuje danou OSGi službu. Jak již však bylo zmíněno v kapitole o OSGi, jednotlivé bundly mají vlastní class loadery a nelze tedy jednoduše standardním způsobem tuto třídu načíst.

Na druhou stranu je každému bundlu dostupný seznam nainstalovaných bundlů. Každý takový bundl pak již dokáže zpřístupnit svůj class loader [3]. Stačí tedy znát symbolické jméno bundlu, který třídu rozhraní exportuje a pak projít seznam bundlů, najít bundl shodného jména, případně verze, a přes tento nalezený bundle již lze třídu načíst. Postup hledání ukazuje výpis 3.2.

```
// projdi existujici bundly
for (Bundle bundle : context.getBundles()) {
    // sestav identifikator bundlu pro porovnani
    Version v = bundle.getVersion();
```

```
    String key = String.format("%s:%s.%s.%s",
        bundle.getSymbolicName(),
        v.getMajor(), v.getMinor(), v.getMicro()
    );
    if (WANTED_BUNDLE.equalsIgnoreCase(key)) {
        // bundl byl nalezen, muzeme nacist hledanou tridu
        Class<?> clazz = bundle.loadClass(WANTED_CLASS);
        break;
    }
}
```

Výpis 3.2: Ukázka načtení třídy z bundlu

Aby bylo možné vytvořit instanci takto vyrobené proxy třídy, je třeba mít i již zmíněný `InvocationHandler`. Tento je zastoupen třídou `UniHandler`, která přijímá v konstruktoru scénář chování. Ten je reprezentovaný instancí rozhraní `Map<Method, Map<Object[], Object>>`, kde klíčem jsou metody rozhraní a hodnotou je opět mapa, jejíž klíčem je pole argumentů, které má handler pro danou metodu očekávat, a hodnotou je pak návratová hodnota příslušná těmto argumentům. Princip sestavení scénáře pro metodu `print(...)` je ukázán na výpisu 3.3.

```
// priprav mapu scenaree
Map<Method, Map<Object[], Object>> scenar = new HashMap<>();
// najdi pozadovanou metodu "print"
Method printMetoda = IPrinter.class.getMethod(
    "print", new Class<?>[] {String.class}
);
// priprav mapu navratovych hodnot pro danou metodu
Map<Object[], Object> navraty = new HashMap<>();
// pro volani metody s argumentem "Pozdrav" pouzij navratovou
```

```
   hodnotu "Ahoj"
navravraty.put(new String[] {"Pozdrav"}, "Ahoj");
// uloz mapu do scenare
scenar.put(printMetoda, navraty);
// vytvor handler pro scenar
InvocationHandler handler = new UniHandler(IPrinter.class,
   scenar);
// nasleduje jiz popsane vytvoreni proxy objektu a jeho
// registrace jako postytovatele OSGi sluzby
```

Výpis 3.3: Příprava scénáře pro mockup

Nejjednodušší je situace u metod, které nepřijímají žádné argumenty. Pro ty je třeba pouze vytvořit mapu, která instanci metody (třída `Method`) přiřadí vracenou hodnotu. Při řešení volání se pak pouze přímo vyvolá hodnota, na kterou v mapě ukazuje instance metody a řešení může skončit.

Mnohem častější je ale volání metod s argumenty. Pro výběr odpovídající návratové hodnoty je třeba správně určit rovnost argumentů. Nelze přitom pouze použít pole argumentů předané metodě `invoke(...)` jako klíč v mapě hodnot, protože pole je standardní objekt a tedy dvě proměnné obsahující referenci na pole jsou shodné právě tehdy, když ukazují na jedno a to samé pole [5]. Tato podmínka však nemůže být splněna, protože pole použité při vytváření scénáře nejsou v žádném vztahu s poli argumentů, která vytváří JVM pro předání metodě `invoke(...)`.

Z tohoto důvodu se vyhledávání návratových hodnot provádí jiným způsobem. UniHandler v konstruktoru vytvoří pro každou metodu z mockovaného rozhraní tabulku, kde sloupce odpovídají podpisu metody, tj. jeden sloupec pro jeden argument metody. Navíc je v tabulce jeden sloupec pro návratovou hodnotu. Řádky tabulky pak představují uložené scénáře volání. Vyhledávání návratové hodnoty pak probíhá po sloupcích, tj. nejprve se v tabulce vyhledají všechny řádky, které mají v prvním sloupci argument shodný s argumentem skutečně předaným. Všechny ostatní řádky pak mohou být označeny jako neshodné. V dalším průchodu se proces opakuje na následujícím sloupci přičemž neshodné řádky jsou rovnou přeskakovány. Tím dojde k vyznačení řádků, které odpovídají scénáři volání a první shodný řádek pak obsahuje hledanou návratovou hodnotu.

Tento postup má však jednu vážnou vadu. Scénář je absolutní a musí tedy dojít k úplné shodě argumentů, což je často nežádoucí. Proto je možné do scénáře volání místo očekávané hodnoty vložit instanci třídy `TAnyValue`, která říká, že u argumentu na aktuálním řádku tabulky není třeba testovat shodu hodnoty. Při určování návratové hodnoty se pak seznam řádků označených jako shodný seřadí vzestupně

podle počtu těchto pružných hodnot a použije se první řádek v seznamu. Díky tomu je pro každé volání vybrán ten nejlépe odpovídající řádek.

Pro určení dvou hodnot jako shodných musí být splněna alespoň jedna z následujících podmínek:

- oba argumenty mají hodnotu `null`
- oba argumenty jsou shodné podle zděděné metody `Object.equal()`
- pokud typ argumentu implementuje rozhraní `Comparable` a metoda `compareTo()` vrací hodnotu 0.
- pokud je typ argumentu pole, jsou argumenty shodné podle metody `Arrays.deepEquals()`
- pokud typ argumentu implementuje rozhraní `Collection`, jsou argumenty shodné podle metody `isEqualCollection(Object[], Object[])` třídy `org.apache.commons.collections.CollectionUtils`.

Pokud není v tabulce nalezen žádný shodný řádek, může handler podle nastavení buď vyhodit výjimku `UndefinedPossibilityException`, nebo situaci ignorovat a vrátit hodnotu `null`. Stejně tak pokud není ve scénáři nalezena žádná tabulka pro volanou metodu může situaci opět podle nastavení buď ignorovat a vrátit `null`, nebo vyhodit výjimku `UndefinedMethodInvocationException`.

3.3.3 Ovládání komponenty

Pro ovládání poskytuje bundle UniMocker poskytovatele služby `org.eclipse.osgi.framework.console.CommandProvider`. Díky tomu je možné komponentu ovládat z aktivní konzole OSGi frameworku.

Pro ovládání jsou dostupné následující příkazy:

um load jmeno_souboru.xml: načte scénář ze souboru jmeno_souboru.xml
um mock: vytvoří mockupy a zaregistruje je jako poskytovatele služeb
um diag: vypíše na konzoli diagnostické informace

Pro ukončení mockupů služeb je třeba standardními prostředky OSGi bundle zastavit. Při zastavení bundlu dojde automaticky ke zrušení registrace všech mockupů jako poskytovatelů služeb.

3.3.4 Formát souboru scénáře

Pro uložení scénářů jsem zvolil textový formát xml a jejich zpracování pak probíhá pomocí technologie **JAXB**(Java Architecture for XML Binding), která vytváří objektový strom automaticky naplněný daty podle obsahu souboru [24]. Veškeré třídy spojené se zpracováním jsou uloženy v balíku `cz.zcu.kiv.bp.unimocker.binding`, který je exportován bundlem `UniMockerBinding`.

Popis struktury souboru jsem implementoval v jazyce **XML Schema**, jehož výhodou je možnost určit datový typ a omezení hodnoty a její platnost datové struktury tak ověřit již při validaci. Díky tomu není třeba vytvářet vlastní prostředky pro parsování a ověřování. XML Schema navíc umožňuje vytvářet vlastní datové typy odvozené od základních vestavěných typů. [25]

Další výhodnou vlastností je podpora nástroje xjc, který na základě schématu dokáže vygenerovat kompletní balík tříd odpovídajících vlastním datovým typů a tedy ušetřit množství práce s ručním vytvářením. [26]

Kořen dokumentu tvoří element `project` s uvedeným jmenným prostorem `http://www.kiv.zcu.cz/component-testing/mocker`. Kořen má jediného potomka v podobě elementu `simulated-components` a ten pak množinu elementů `bundle`, ukázka viz. výpis 3.4.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/mocker">
  <simulated-components>
    <bundle symbolic-name="cz.zcu.kivcmd.core" version="1.0.0">
     ...
    </bundle>
  </simulated-components>
</project>
```

Výpis 3.4: Určení bundlu exportujího mockované rozhraní

Element `bundle` má jako potomky množinu elementů `service`, které se atributem `interface` odkazují na rozhraní konkrétní služby, viz výpis 3.5.

```
<bundle symbolic-name="cz.zcu.kivcmd.core" version="1.0.0">
  <service interface="cz.zcu.kiv.kc.interfaces.IViewPlugin">
   ...
  </service>
</bundle>
```

Výpis 3.5: Příklad vytvoření mockup poskytovatel služby `...IviewPlugin`

Tento element již slouží k popisu chování jednotlivých metod rozhraní a tedy obsahuje jako potomky elementy `method`. Příklad popisu chování metody `getName`, která nepřijímá žádný argument a vždy vrací řetězec `MockupPlugin2` je na výpisu 3.6.

```
<method name="getName">
    <invocation>
        <arguments />
        <return><String>MockupPlugin2</String></return>
    </invocation>
    ...
</method>
```

Výpis 3.6: Ukázka jednoho scénáře volání metody `getName`

Element `invocation` představuje jednu možnou kombinaci argumentů a návratové hodnoty a tedy odpovídá řádku ve vyhledávací tabulce zmíněné v předchozím textu. Element `arguments` má jako potomky elementy `argument`. Každý element musí mít atributem `ord-num` určeno pořadové číslo. Tato čísla slouží ke správnému seřazení argumentů, tak aby jejich pořadí odpovídalo pořadí v podpisu metody. Tyto elementy, stejně jako element `return`, pak již mají vždy jednoho potomka, který označuje datový typ argumentu. Pro metodu bez návratové hodnoty lze element `return` úplně vynechat. V elementu `argument` lze navíc použít element `AnyValue`, který značí univerzální argument. Přehled všech podporovaných typů je uveden v příloze A.4. V příloze A.1 je ukázán komplexní příklad scénáře .

3.4 UniPlayer

Součástí této práce je také komponenta UniPlayer. Tato komponenta vznikla jako prostředek pro testování implementovaného řešení komponenty UniMocker. Jejím úkolem je simulovat běh reálné komponentové aplikace podle načteného scénáře. Komponenta tedy dokáže částečně zastoupit funkci simulátoru SimCo. Na rozdíl od něj však neprovádí simulaci řízenou událostmi, ale pouze ve smyčce odebírá akce ze

seznamu a po jeho vyprázdnění končí. Scénář však není statický a umožňuje stejně jako SimCo definovat různé druhy opakování výskytu akce.

Scénáře je implementován jako iterovatelný objekt tj. třída `Scenario` implementuje rozhraní `Iterable`. Při každém zavolání metody `next()` poskytovaného iterátoru pak vždy dojde k aktualizaci scénáře. Aktuální akce získá nový čas výskytu a je vrácena zpět do scénáře, nebo vypršel její limit a je tedy ze scénáře odstraněna. UniPlayer umožňuje výskyt událostí se třema typy opakování:

rovnoměrné - akce se bude opakovat s konstantním krokem
exponenciální - výpočet času výskytu akce se řídí exponenciálním rozdělením
gaussovské - výpočet se řídí gaussovským rozdělením

Pro spolehlivé generování náhodných čísel s danými rozděleními jsem zvolil knihovnu `org.uncommons.math` (Apache 2.0, `http://maths.uncommons.org`), která implementuje celou řadu generátorů náhodných čísel, stejně jako náhodné veličiny s potřebným rozdělením pravděpodobnosti [15].

Scénář podporuje akce typu *událost* nebo *volání* metody služby. Pokud je akce typu událost, vyvolá UniPlayer asynchronní událost a pokračuje další akcí. Pokud je akce typu volání, pokusí se získat poskytovatele požadované služby, instanci volané metody a tuto metodu zavolá na získaném objektu poskytovatele. V případě, kdy se nepodaří službu získat, například proto, že neexistuje žádný poskytovatel, uspí své vlákno na dobu 30 vteřin a umožní tak v běžícím prostředí uživateli nainstalovat a aktivovat bundle, který bude danou službu poskytovat. Pokud ani po této době nebude žádný poskytovatel aktivní, je akce přeskočena.

3.4.1 Ovládání komponenty

Stejně jako UniMocker implementuje UniPlayer poskytovatele služby `org.eclipse.osgi.framework.console.CommandProvider`. Komponenta podporuje následující příkazy:

up load jmeno_souboru.xml - načte scénář ze souboru jmeno_souboru.xml

up play - spustí přehrávání načteného scénáře

up stop - pozastaví přehrávání aktuálního scénáře, další příkaz up start pokračuje v místě zastavení

up diag - vypíše na konzoli diagnostické informace o běžícím prostředí a také obsah souboru načteného scénáře

3.4.2 Formát souboru scénáře

Stejně jako UniMocker jsem pro scénáře simulace použil formát xml a technologii JAXB. Třídy pro zpracování souborů jsou uloženy v balíku `cz.zcu.kiv.bp.uniplayer.binding`, který je exportován bundlem `UniPlayerBinding`. Bundle opět obsahuje popis struktury souborů ve formátu XML schema.

Kořen dokumentu tvoří element project s uvedeným jmenným prostorem `http://www.kiv.zcu.cz/component-testing/player`. Kořenový element `project` má dva potomky v podobě nepovinného elementu `settings` a povinného elementu `actions`, viz. výpis 3.7.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/player">
    <settings>
        ...
    </settings>
    <actions>
        ...
    </actions>
</project>
```

Výpis 3.7: Základní struktura scénáře

Element `settings` slouží k nastavení maximální doby běhu simulace elementem `time-limit` a dále k nastavení časového zpoždění mezi jednotlivými výběry akcí ze scénáře elementem `simul-step-delay`. Toto nastavení slouží pro potřeby vizualizace běžící simulace.

Potomky elementu `actions` jsou pak popisy jednotlivých akcí scénáře pomocí elementů `action`. Element `action` má právě jeden atribut `time`, který označuje čas prvního výskytu. Každý element má právě dva potomky v podobě elementů `recurrence` a `command`. Element `recurrence` slouží k nastavení počtu opakování akce a nastavení použitého rozdělení pro výpočet času dalšího výskytu, viz. výpis 3.8.

```
<action time="...">
    <recurrence count="..." repeat-until="...">
<!-- potomkem je  prave jeden z nasledujicich elementu -->
<!-- opakovani s konstantnim krokem simulacniho casu -->
        <equidistant step="..."/>
<!-- opakovani s exponencialnim rozdelenim -->
```

```
        <exponential rate="..." />
<!-- opakovani s gaussovskym rozdelenim -->
        <gaussian mean="..." deviation="..."/>
    </recurrence>
    <command>
        ...
    </command>
</action>
```

Výpis 3.8: Popis akce

Element `command` má právě jednoho potomka, a to buď element `call` pro popis volání, nebo `event` pro popis události, viz. 3.9.

```
<!-- akce scenare bude volani metody sluzby -->
<call service="..." method="...">
    <arguments>
        ...
    </arguments>
</call>

<!-- akci bude vyvolani udalosti -->
<event topic="..." key="...">
    <argument> ... </argument>
</event>
```

Výpis 3.9: Typy akce

Pro element `argument(s)` platí stejná syntax jako u scénářů pro UniMocker vyjma elementů `Any`, které zde nemají smysl. V příloze A.2 je opět ukázán příklad komplexního scénáře.

3.5 Vzorové scénáře

3.5.1 Mockupy základních služeb - `basic_mocks.xml`

Protože testovací aplikace má umístěny všechny poskytovatele základních služeb v jednom bundlu, je třeba vytvořit mockupy všech těchto poskytovatelů. V příloze A.3.1 je ukázán začátek scénáře pro UniMocker. Na výpisu je ukázán popis pouze pro jednu službu, zbylé se vždy liší pouze jménem rozhraní služby. Mockovány jsou tedy tyto služby:

- `cz.zcu.kiv.kc.interfaces.ICopyPlugin`
- `cz.zcu.kiv.kc.interfaces.ICreateDirPlugin`
- `cz.zcu.kiv.kc.interfaces.IDeletePlugin`
- `cz.zcu.kiv.kc.interfaces.IMovePlugin`

3.5.2 Mockup archivačního formátu - `archiv_packer_mock.xml`

Tento scénář simuluje podporu vytváření archivů v novém archivního formátu. Jeho obsahem je tedy popis mockování služby `IArchivePacker`. Scénář je zobrazen na výpisu A.3.2.

3.5.3 Pohyb po souborovém systému - `filesys_navig.xml`

Tento scénář popisuje simulaci procházení systémem souborů, přechází mezi adresáři, mění aktivní panel. Simulace probíhá s využitím naslouchání hlavní komponenty aplikace na příslušné události. Scénář je zobrazen na výpisu A.3.3.

3.5.4 Simulace otevírání souborů - `opening_simul`

Další scénář popisuje simulaci otevíráni souborů ve vestavěném prohlížeči. od je zobrazen na výpisu A.3.4.

3.5.5 Kombinace mockupů a simulace - `complex_simul.xml`

Tento scénář simuluje používání základních funkcí správce souborů (kopírování, mazání. Kód scénáře je na výpisu A.3.5. Současně byl při simulaci využit scénář A.3.1 pro vytvořeni mockupů. Výstup simulátoru je zobrazen na výpisu 3.10. U akcí v časech 154, 169 a 700 je vidět, že voláni probíhala na instancích třídy `UniHandler`, což indikuje přítomnost mockupu. Na obrázku 3.2 je zobrazena podoba správce souborů s aktivními mockupy.

```
osgi> up play
100: event =>  changeSourceDir/dir[java.lang.String: d:/Obrázky/
    (wrapper: null)]
150: event =>  changeSourceDir/dir[java.lang.String: d:/ (
   wrapper: null)]
150: event =>  changeDestinationDir/dir[java.lang.String: e:/ (
   wrapper: null)]
150: call =>  cz.zcu.kiv.kc.interfaces.IViewPlugin.executeAction
    ([java.util.ArrayList: [d:\Stažené\mips2.asm] (wrapper:
   class cz.zcu.kiv.bp.uniplayer.bindings.TFileCollection), java
   .lang.String: d:/Stažené/ (wrapper: null), java.lang.String:
   d:/Stažené/ (wrapper: null)])cz.zcu.kiv.kc.viewer.
   Viewer@5c4086a8

154: call =>  cz.zcu.kiv.kc.interfaces.ICopyPlugin.executeAction
```

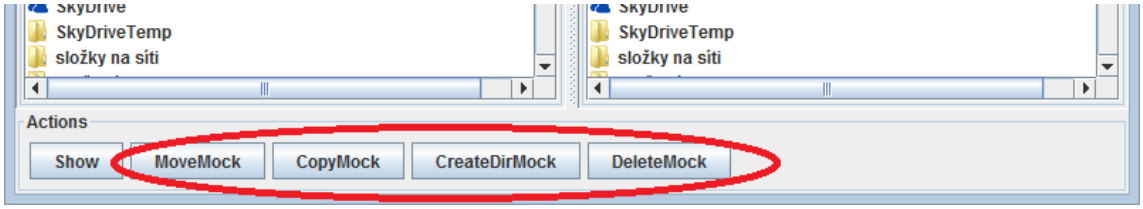
```
    ([java.util.ArrayList: [d:\676956.iso] (wrapper: class cz.
   zcu.kiv.bp.uniplayer.bindings.TFileCollection), java.lang.
   String: e:/tmp/nazdar (wrapper: null), java.lang.String: d:/
   (wrapper: null)])cz.zcu.kiv.bp.unimocker.UniHandler@1330b35b

157: event =>  changeDestinationDir/dir[java.lang.String: e:/ (
   wrapper: null)]
159: event =>  changeDestinationDir/dir[java.lang.String: e:/ (
   wrapper: null)]
169: call =>  cz.zcu.kiv.kc.interfaces.ICopyPlugin.executeAction
    ([java.util.ArrayList: [d:\676956.iso] (wrapper: class cz.
   zcu.kiv.bp.uniplayer.bindings.TFileCollection), java.lang.
   String: e:/tmp/nazdar (wrapper: null), java.lang.String: d:/
   (wrapper: null)])cz.zcu.kiv.bp.unimocker.UniHandler@1330b35b

200: event =>  changeDestinationDir/dir[java.lang.String: c:/ (
   wrapper: null)]
200: event =>  changeSourceDir/dir[java.lang.String: d:/Obrázky/
    (wrapper: null)]
211: event =>  changeDestinationDir/dir[java.lang.String: c:/ (
   wrapper: null)]
224: event =>  changeDestinationDir/dir[java.lang.String: e:/ (
   wrapper: null)]
250: event =>  changeSourceDir/dir[java.lang.String: d:/ (
   wrapper: null)]
272: event =>  changeDestinationDir/dir[java.lang.String: c:/ (
   wrapper: null)]
300: event =>  changeSourceDir/dir[java.lang.String: d:/Obrázky/
    (wrapper: null)]
309: event =>  changeDestinationDir/dir[java.lang.String: e:/ (
```

```
   wrapper: null)]
350: event =>  changeSourceDir/dir[java.lang.String: d:/ (
   wrapper: null)]
450: event =>  changeSourceDir/dir[java.lang.String: d:/ (
   wrapper: null)]
599: event =>  changeDestinationDir/dir[java.lang.String: c:/ (
   wrapper: null)]
658: event =>  changeDestinationDir/dir[java.lang.String: c:/ (
   wrapper: null)]
692: event =>  changeDestinationDir/dir[java.lang.String: c:/ (
   wrapper: null)]
700: call =>  cz.zcu.kiv.kc.interfaces.IDeletePlugin.
   executeAction ([java.util.ArrayList: [e:\tmp\nazdar] (
   wrapper: class cz.zcu.kiv.bp.uniplayer.bindings.
   TFileCollection), java.lang.String: d:/ (wrapper: null), java
   .lang.String: e:/tmp/ (wrapper: null)])cz.zcu.kiv.bp.
   unimocker.UniHandler@199574a6
```

Výpis 3.10: Výstup simulátoru UniPlayer pro scénář complex_simul.xml



Obrázek 3.2: Ukázka rozhraní aplikace s použitím mockupů

4 Závěr

Cílem této práce bylo vytvořit sadu mockup komponent a testovacích scénářů pro zadanou testovací aplikaci. Pro řešení jsem zvolil univerzálnější postup jehož výstupem je implementace komponenty UniMocker, která dokáže tyto mockup komponenty vytvářet automaticky za běhu a na základě předpisu ze souboru xml.

Jako vedlejší produkt této práce vznikla komponenta UniPlayer, která původně sloužila pouze jako zkušební nástroj pro vývoj UniMockeru. UniPlayer se však vyvinul ve schopnou alternativu původního simulátoru SimCo, a proto ho po konzultaci s vedoucím této práce přikládám.

Datové modely obou komponent jsou rozdělené do samostatných bundlů, což umožňuje jejich případné použití v dalších aplikacích jako jsou například editory scénářů apod.

Použité zdroje a literatura

[1] OSGi$^{\text{TM}}$ Alliance, *Guidelines for designing Java API for use in the OSGi environment*, `http://www.osgi.org/Design/Guidelines` [online], 2013.

[2] OSGi$^{\text{TM}}$ Alliance, *Benefits of Using OSGi*, `http://www.osgi.org/Technology/WhyOSGi` [online], 2013.

[3] OSGi$^{\text{TM}}$ Alliance, *OSGi^{TM} Service Platform Core Specification Release 4 Version 4.3* [online], `http://www.osgi.org/javadoc/r4v43/core/`, 2012.

[4] Adrian Colyer, *Springsource blog - Why should I care abour OSGi anyway?*, `http://blog.springsource.org/2008/05/15/why-should-i-care-about-osgi-anyway/` [online], 2008.

[5] Oracle corp., *The Java Tutorials*, `http://docs.oracle.com/javase/tutorial` [online], 2013.

[6] Oracle corp., *Java^{TM} Platform, Standard Edition 7 API Specification*, `http://docs.oracle.com/javase/7/docs/api` [online], 2013.

[7] kolektiv autorů, *Spring Framework Reference Documentation*, `http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html` [online], 2013.

[8] Adrian M Colyer, Hal Hildebrand, Costin Leau, Andy Piper, *Spring Dynamic Modules Reference Guide*, `http://static.springsource.org/osgi/docs/1.2.1/reference/html` [online], 2009.

[9] Neil Bartlett, *The Dreaded Thread Context Class Loader*, `http://njbartlett.name/2012/10/23/dreaded-thread-context-classloader.html` [online], 2012.

[10] Arnaud Cogoluègnes, Thierry Templier, Andy Piper, *Spring Dynamic Modules in Action*, ©Manning Publications Co., 2010. ISBN 1935182307

[11] Přemek Brada, *Úvod do komponent/OSGi*, `http://wiki.kiv.zcu.cz/UvodDoKomponent/OSGi` [online], 2013.

[12] Lars Vogel, *OSGi Services - Tutorial*, `http://www.vogella.com/articles/OSGiServices/article.html` [online], 2013.

[13] Tomáš Kabíček, *Diplomová práce - Simulační systém softwarových komponent založený na komponentách*, 2011.

[14] Luboš Petera, Tomáš Pospíšil, Michal Říha, Lukáš Vávra, Jiří Zákoucký, *Semestrální práce z předmětu KIV/ZSWI - Team kivCommander*, 2013.

[15] Daniel W. Dyer, *Uncommons Maths, Random number generators, probability distributions, combinatorics and statistics for Java*, `http://maths.uncommons.org/api/index.html` [online], 2012.

[16] Chris Aniszczyk, *OSGi EventAdmin*, `http://eclipsesource.com/blogs/2009/09/15/osgi-eventadmin/` [online], 2009.

[17] OSGi Community, *OSGi Community Wiki*, `http://wiki.osgi.org/wiki/` [online], 2011.

[18] Encyclopedia Britanica Company, *Merriam-Webster Dictionary*, `http://www.merriam-webster.com/dictionary/mock-up` [online], 2013.

[19] The Eclipse Foundation, *Eclipse Equinox OSGi*, `http://eclipse.org/equinox/` [online], 2013

[20] Google Inc., *Get the Android SDK*, `http://developer.android.com/sdk/index.html` [online], 2013

[21] Samsung Electronics Co., Ltd *SDK download* `http://www.samsungdforum.com/Devtools/Sdkdownload` [online], 2013.

[22] IBM, *Software Development Toolkit for PowerLinux*, `http://www-304.ibm.com/webapp/set2/sas/f/lopdiags/sdklop.html` [online], 2013.

[23] CodeScale Blog, *Basics about OSGi Classloading*, `http://codescale.wordpress.com/2009/05/22/basics-about-osgi-classloading/` [online], 2009.

[24] Lars Bogel, *JAXB Tutorial*, `http://www.vogella.com/articles/JAXB/article.html` [online], 2012.

[25] Jiří Kosek, *XML schémata*, `http://www.kosek.cz/xml/schema/` [online], 2007.

[26] Oracle corp., *xjc - Java^{TM} Architecture for XML Binding Binding Compiler*, `http://docs.oracle.com/javase/7/docs/technotes/tools/share/xjc.html` [online], 2013.

A Přílohy

A.1 Ukázka scénáře pro UniMocker

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/mocker">
  <simulated-components>
    <bundle symbolic-name="cz.zcu.kivcmd.core" version="1.0.0">
      <service interface="cz.zcu.kiv.kc.interfaces.IViewPlugin"
       ignore-undefined-methods="true"
       ignore-undefined-possibilities="true">
        <method name="executeAction">
          <invocation>
            <arguments>
              <argument ord-num="0">
                <Any base-type="FileArrayList" />
              </argument>
              <argument ord-num="1">
                <Any base-type="String" />
              </argument>
              <argument ord-num="2">
                <Any base-type="String" />
              </argument>
            </arguments>
            <return><Bolean>false</Boolean></return>
          </invocation>
          <invocation>
            <arguments>
              <argument ord-num="0">
                <Files type="LinkedList">
                  <item ord-num="0">
                    <File>d:/obr01.jpg</File>
```

```
                </item>
                <item ord-num="1">
                  <File>d:/obr02.jpg</File>
                </item>
                <item ord-num="2">
                  <File>d:/obr03.jpg</File>
                </item>
              </Files>
            </argument>
            <argument ord-num="1">
              <String>d:/</String>
            </argument>
            <argument ord-num="2">
              <String>e:/</String>
            </argument>
          </arguments>
          <return><Bolean>true</Boolean></return>
        </invocation>
      </method>
      <method name="getName">
        <invocation>
          <arguments />
          <return><String>MockupPlugin</String></return>
        </invocation>
      </method>
    </service>
  </bundle>
 </simulated-components>
</project>
```

A.2 Ukázka scénáře pro UniPlayer

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/player">
  <settings>
    <time-limit>75600</time-limit>
    <simul-step-delay>500</simul-step-delay>
  </settings>
  <actions>
    <action time="154">
      <recurrence count="2" repeat-until="300">
        <equidistant step="15"/>
      </recurrence>
      <command>
        <call service="foo.IBar" method="Render">
          <arguments>
            <argument ord-num="0">
              <File>d:/image.iso</File>
            </argument>
            <argument ord-num="1">
              <String>kokos</String>
            </argument>
          </arguments>
        </call>
      </command>
    </action>

    <action time="100">
      <recurrence count="10">
        <equidistant step="100"/>
```

```
      </recurrence>
      <command>
        <event topic="changeSourceDir" key="dir">
          <argument><String>d:/Obrázky/</String></argument>
        </event>
      </command>
    </action>
  </actions>
</project>
```

A.3 Vytvořené scénáře

A.3.1 basic_mocks.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/mocker"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <simulated-components>
    <bundle symbolic-name="cz.zcu.kivcmd.core" version="1.0.0">
      <service interface="cz.zcu.kiv.kc.interfaces.ICopyPlugin"
       ignore-undefined-methods="true"
       ignore-undefined-possibilities="true">
        <method name="getName">
          <invocation>
            <arguments />
            <return><String>CopyMock</String></return>
          </invocation>
        </method>
        <method name="executeAction">
          <invocation>
            <arguments>
              <argument ord-num="0">
                <AnyValue base-type="FileArrayList" />
              </argument>
              <argument ord-num="1">
                <AnyValue base-type="String" />
              </argument>
              <argument ord-num="2">
                <AnyValue base-type="String" />
```

```
            </argument>
          </arguments>
        </invocation>
      </method>
    </service>
    ...
  </bundle>
 </simulated-components>
</project>
```

Výpis A.1: Scénář mockupů základních služeb `getName`

A.3.2 archiv_packer_mock.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/mocker"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <simulated-components>
    <bundle symbolic-name="cz.zcu.kiv.kc.archiver" version="
       1.0.0">
      <service interface="cz.zcu.kiv.kc.archiver.IArchivePacker"
         >
        <method name="getExtension">
          <invocation>
            <arguments />
            <return><String>7z</String></return>
          </invocation>
        </method>
        <method name="getFormatName">
          <invocation>
           <arguments />
           <return><String>7-Zip</String></return>
          </invocation>
        </method>
        <method name="packFilesToArchive">
          <invocation>
            <arguments>
              <argument ord-num="0">
                <AnyValue base-type="FileArrayList" />
              </argument>
              <argument ord-num="1">
                <AnyValue base-type="File" />
```

```
            </argument>
            <argument ord-num="2">
              <AnyValue base-type="File" />
            </argument>
          </arguments>
          <return><Boolean>false</Boolean></return>
        </invocation>
      </method>
    </service>
  </bundle>
 </simulated-components>
</project>
```

Výpis A.2: Scénář mockupu služby archivního formátu

A.3.3 filesys_navig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/player"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <settings>
    <time-limit>75600</time-limit>
    <simul-step-delay>500</simul-step-delay>
  </settings>
  <actions>
    <action time="100">
      <recurrence count="15"><equidistant step="100"
         /></recurrence>
      <command>
        <event topic="changeSourceDir" key="dir">
          <argument><String>d:/Obrázky/</String></argument>
        </event>
      </command>
    </action>
    <action time="150">
      <recurrence count="15" repeat-until="92233720368547758O7">
        <exponential rate="5" />
      </recurrence>
      <command>
        <event topic="changeDestinationDir" key="dir">
          <argument><String>e:/</String></argument>
        </event>
      </command>
    </action>
    <action time="200">
```

```
      <recurrence count="2">
        <equidistant step="100"/>
      </recurrence>
      <command>
        <event topic="changeDestinationDir" key="dir">
          <argument><String>d:/Videa/</String></argument>
        </event>
      </command>
    </action>
  </actions>
</project>
```

Výpis A.3: Scénář simulace pohybu po souborovém systému

A.3.4 opening_simul.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/player"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <actions>
    <action time="150">
      <recurrence count="1">
        <equidistant step="15"/>
      </recurrence>
      <command>
        <call service="cz.zcu.kiv.kc.interfaces.IViewPlugin"
           method="executeAction">
          <arguments>
            <argument ord-num="0">
              <Files>
                <item ord-num="0">
                  <File>d:/Stažené/mips2.asm</File>
                </item>
              </Files>
            </argument>
            <argument ord-num="1">
              <String>d:/Stažené/</String>
            </argument>
            <argument ord-num="2">
              <String>d:/Stažené/</String>
            </argument>
          </arguments>
        </call>
      </command>
```

```
    </action>
    <action time="290">
      <recurrence count="1">
        <equidistant step="30"/>
      </recurrence>
      <command>
        <call service="cz.zcu.kiv.kc.interfaces.IViewPlugin"
           method="executeAction">
          <arguments>
            <argument ord-num="0">
              <Files>
                <item ord-num="0">
                  <File>d:/Obrázky/WP_000008.jpg</File>
                </item>
              </Files>
            </argument>
            <argument ord-num="1">
              <String>d:/Obrázky/</String>
            </argument>
            <argument ord-num="2">
              <String>d:/Stažené/</String>
            </argument>
          </arguments>
        </call>
      </command>
    </action>
  </actions>
</project>
```

Výpis A.4: Scénář simulace otevírání souborů v integrovaném prohlížeči

A.3.5 complex_simul.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://www.kiv.zcu.cz/component-testing/player"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <settings>
    <time-limit>75600</time-limit>
    <simul-step-delay>500</simul-step-delay>
  </settings>
  <actions>
    <action time="100">
      <recurrence count="3">
        <equidistant step="100"/>
      </recurrence>
      <command>
        <event topic="changeSourceDir" key="dir">
          <argument><String>d:/Obrázky/</String></argument>
        </event>
      </command>
    </action>
    <action time="150">
      <recurrence count="4">
        <equidistant step="100"/>
      </recurrence>
      <command>
        <event topic="changeSourceDir" key="dir">
          <argument><String>d:/</String></argument>
        </event>
      </command>
    </action>
```

```
<action time="150">
  <recurrence count="5">
    <exponential rate="5" time-span="900" />
  </recurrence>
  <command>
    <event topic="changeDestinationDir" key="dir">
      <argument><String>e:/</String></argument>
    </event>
  </command>
</action>
<action time="200">
  <recurrence count="6">
    <exponential rate="10" time-span="900" />
  </recurrence>
  <command>
    <event topic="changeDestinationDir" key="dir">
      <argument><String>c:/</String></argument>
    </event>
  </command>
</action>
<action time="154">
  <recurrence count="2">
    <equidistant step="15"/>
  </recurrence>
  <command>
    <call service="cz.zcu.kiv.kc.interfaces.ICopyPlugin"
       method="executeAction">
      <arguments>
        <argument ord-num="0">
          <Files>
```

```
            <item ord-num="0">
              <File>d:/676956.iso</File>
            </item>
          </Files>
        </argument>
        <argument ord-num="1">
          <String>e:/tmp/nazdar</String>
        </argument>
        <argument ord-num="2">
          <String>d:/</String>
        </argument>
      </arguments>
    </call>
  </command>
</action>
<action time="150">
  <recurrence count="1">
    <equidistant step="15"/>
  </recurrence>
  <command>
    <call service="cz.zcu.kiv.kc.interfaces.IViewPlugin"
       method="executeAction">
      <arguments>
        <argument ord-num="0">
          <Files>
            <item ord-num="0">
              <File>d:/Stažené/mips2.asm</File>
            </item>
          </Files>
        </argument>
```

```
          <argument ord-num="1">
            <String>d:/Stažené/</String>
          </argument>
          <argument ord-num="2">
            <String>d:/Stažené/</String>
          </argument>
        </arguments>
      </call>
    </command>
  </action>
  <action time="700">
    <recurrence count="1">
      <equidistant step="15"/>
    </recurrence>
    <command>
      <call service="cz.zcu.kiv.kc.interfaces.IDeletePlugin"
         method="executeAction">
        <arguments>
          <argument ord-num="0">
            <Files>
              <item ord-num="0">
                <File>e:/tmp/nazdar/</File>
              </item>
            </Files>
          </argument>
          <argument ord-num="1">
            <String>d:/</String>
          </argument>
          <argument ord-num="2">
            <String>e:/tmp/</String>
```

```
            </argument>
          </arguments>
        </call>
      </command>
    </action>
  </actions>
</project>
```

Výpis A.5: Scénář komplexní simulace s využitím mockupů

A.4 Přehled podporovaných datových typů

typ	**element**
java.lang.String	<String></String>
java.math.BigInteger	<BigInteger></BigInteger>
java.lang.Long	<Long></Long>
java.lang.Integer	<Integer></Integer>
java.lang.Short	<Short></Short>
java.lang.Byte	<Byte></Byte>
java.math.BigDecimal	<BigDecimal></BigDecimal>
java.lang.Double	<Double></Double>
java.lang.Float	<Float></Float>
java.lang.Boolean	<Boolean></Boolean>
java.io.File	<File></File>
null	<Null baseType="..."/>[1]
jakákoliv hodnota	<Any baseType="..."/>[1,2]
kolekce	jméno typu končící na 's' s výjimkou typů Null a Any (<Strings>, <Integers>, ...)

Tabulka A.1: Podporované druhy kolekcí

typ	**hodnota atributu `type`**
java.util.ArrayList	ArrayList[3]
java.util.LinkedList	LinkedList
Object[]	Array

[1] Přehled možných hodnot viz. dokumentace na přiloženém cd. Jedná se pouze o výčet jmen odpovídajících kombinaci zde popsaných typům a kolekcí.

[2] Lze použít pouze u scénářů pro UniMocker.

[3] Výchozí typ

A.4.1 Poznámka ke kolekcím typu pole

Hodnoty definované pomocí elementu kolekce atributem `type` s hodnotou `Array` mají runtime typ odpovídající datovému typu položek kolekce.

```
<Strings type="Array">
</Strings>
```

Vytvoří pole

```
String[] val = new String[0];
```

Pro kolekce typu seznam(`ArrayList` a `LinkedList`) však budou mít runtime typ pouze `ArrayList` a `LinkedList` tj. již nelze spolehlivě určit datový typ jejich komponent, resp. komponenty budou typu `Object`! Toto chování je způsobeno, tím že genericita je v Javě aplikována před kompilací a do zkompilovaného kódu se již nedostane informace o konkrétním typovém argumentu kolekcí `List` [6].