Dokumentace k projektu SIN 2016/2017

Dokumentace popisuje vše potřebné k vypracování projektu na téma Inteligentní budovy. našeho týmu. První sekce je Návod ke zprovoznění. Je umístěna jako první, aby bylo možné rychle zjistit jakým způsobem lze aplikaci spustit. Další sekce pak popisuje účastníky projektu a jaké problémy se rozhodli jednotlivci řešit. Následuje podrobnější zadání založené na již dříve odevzdané anotaci, návrh - tím i původní myšlenka celého projektu - a popis stavu vypracování. Na konci je zhodnocení výsledků naší práce.

Návod ke zprovoznění

Odevzdaný archiv by měl obsahovat soubor *makefile* pomocí něhož je možné stáhnout zdrojové soubory a již kompilovaný projekt do adresáře kde se makefile nachází a byl spuštěn. Příkaz *make install* provede stažení výše zmíněných souborů. Příkaz *make run* spustí demo aplikace. Je zahrnut i příkaz *make compile*, ale ten pouze vypíše hlášení o tom, že makefile sám neumí zdrojové soubory zkompilovat a je nutné tak učinit ručně pomocí vývojového prostředí IDE.

Pokud je nutné manuálně zkompilovat zdrojové soubory, vytvoříme nový projekt v libovolném IDE (například Netbeans, Eclipse) v libovolném umístění na disku. Tato projektová složka vytvořená IDE vývojovým prostředím obvykle obsahuje podsložku (podadresář) *src*. Do ní stačí nakopírovat obsah stejnojmenné složky z repositáře. Kompilace a spuštění jsou už pak závislé na konkrétním IDE.

Účastníci projektu a rozdělení práce

Účastníky na projektu jsou:
Roman Vais - xvaisr00 (vedoucí)
Tomáš Repík - xrepik00
Ondřej Blucha - xrepik00

Během návrhové fáze jsme si rozdělili práci následujícím způsobem. Roman Vais se zavázal vypracovat model a simulační prostředí. Tomáš Repík a Ondřej Blucha se dohodli mezi sebou na vypracování ostatních součástí projektu. Finálně si rozdělili práci tak, že Tomáš vypracuje Grafické rozhraní a Ondřej si veze na starost tvorbu řídícího systému. Více o jednotlivých částech v sekci zadání níže.

Zadání

Jako zadání projektu jsme si zvolili simulaci řízení uklízecích robotů v prostředí IT firmy. Takto jsme specifikovali obecné oficiální zadání "Model inteligentní budovy" - zadání číslo 2. Náš projekt se zabývá simulací části prostředí inteligentní budovy, která je sídlem IT firmy. Projekt by měl implementovat řídící systém a simulaci prostředí. Systém byl během návrhové fáze rozdělen na 2 až 3 části popsané níže. Součástí mělo být uživatelské rozhraní, umožňující měnit parametry modelu. Simulace bude samostatná od řídícího systému a proto je simulace by tak měla být provozu schopná i bez zásahu řídícího systému.

Řídící systém
Měl by umožňovat plánování a řízení úklidu pomocí robotů a fungovat jako “centrální server” sbírající data ze senzorů a od samotných robotů přítomných v budově. Tato data budou následně využitá pro optimalizaci úklidu a řízení samotných robotů.

Simulace
Simulované prostředí psané v čisté Javě případně v prostředí frameworku Jade.
Mělo by simulovat na abstraktní úrovni chování senzorů, jednotlivých robotů a osob pohybujících se po budově (například vytváření nepořádku). Reprezentace prostředí (opět na abstraktní úrovni) měla být zadávána parametricky. Pro komunikaci s řídícím systémem mělo vzniknout dedikované rozhraní.

Generátor událostí
Část systému kterou je možné implementovat samostatně, nebo jako součást (libovolné z / každé z) obou předcházejících částí. Generátor by měl systematiky vytvářet události, které budou měnit reprezentaci prostředí, jinými slovy tvořit samotné chování jednotlivých entit. Například vytváření nepořádku, přeskupení nábytku jako jsou židle nebo stolky, pohyb osob v prostorách budovy apod. Do práce generátoru samozřejmě patří i vytváření událostí, které modelují příkaz řídícího systému některým entitám.

Návrh
Simulační prostředí a model byly tvořeny od začátku tak aby byly co nejobecnější a mohli poskytovat výsledky simulace samy o sobě pomocí tzv. logování. Logy událostí mohou být zaznamenány na standardní výstup, standardní chybový výstup, nebo v případě potřeby (nutná odpovídající implementace) do libovolného souboru. (Příklad jak by takový soubor mohl vypadat lze najít v repositáři - soubor *buildingTemplate.xml*.)

Celý projekt byl postavený na myšlence, že díky parametrizaci simulace bude možné testovat chování řídícího systému v různých situacích , např. různé množství robotů, místností, více robotů než dokovacích stanic, roboti s větší kapacitou baterie, různý počet místností apod. Jednotlivé entity by si pak implementovaly své vlastní chování - reakce na události v okolí a reakci na ně v podobě změny svého stavu a plánováním dalších událostí. Toto mělo probíhat buď podle předem připraveného scénáře nebo pomocí nějakého pravděpodobnostního rozložení. Vykonávání Akcí jako takových bylo z hlediska modelu zcela zanedbáno a počítalo se pouze s diskrétními změnami stavu.

Řídící systém měl reagovat na události a hlášení o změnách stavu jednotlivých kontrolovaných entit a tyto pak na základě nějakého vhodného algoritmu řídit. Mělo docházet k interakci *inteligence* řídícího algoritmu a událostí generovaných entitami.

Pro takovýto komplexní systém je potřeba zajistit aby veškeré údaje byly uložené na jednom místě a nedocházelo tak k nekonzistencím. Například budova by měla mít pouze jedinou reprezentaci - jedinou instaci, stejně tak kalendář událostí, který celou simulaci řídí. K tomu se obvykle v objektově orientovaném programování využívá návrhový vzor *Singleton* (jedináček). V čisté Javě ovšem bývají s tímto návrhovým vzorem jisté komplikace. Přístup k

instancím singletonů není příliš elegantní. Některá rozhraní, například entity nebo místnosti, mají obvykle společné rozhraní ale lišící se implementace. Jak volit správnou implementaci aby řešení (kód) bylo znovupoužitelné a nebylo nutné jej měnit na mnoha místech ? Tyto výše zmíněné problémy řeší princip takzvané *Dependency injection*. Jedním z nejznámějších frameworků který toto řeší je Google Guice. Je to však mohutný framevork, který přesahuje potřeby tohoto projektu. Navrhli jsme proto vlastní zjednodušený sytstém dependency injection.

Od grafického rozhraní jsme očekávali pouze jednoduchost a přehlednost. Neměli jsme na něj jiné požadavky než schopnost demostrovat co se v modelu během simulace děje jinou metodou než textovým výpisem.

Implementace a její průběh

Vytvořili jsme repositář na serveru Github, kde je možné zpětně dohledan jak vývoj projektu postupně probíhal. Odkaz na repozitář je zde: https://github.com/xvaisr/SINproject/.

Při vypracování bohužel docházelo k nedorozuměním při vzájemné komunikaci návrhu což mělo za následek implementační problémy při propojování jednotlivých čáastí projektu a při vyhotovení řídícího systému. Úspěšně se nám podařilo dokončit model a grafické rozhraní. Řídící systém se bohužel dokončit nepodařilo. Člen týmu, který si na počátku zpracování projektu vzal na starost konkrétní návrh a implementaci řídícího systému bohužel vypracoval několik návrhů na řešení. Ovšem tyto návrhy měli značné chyby jako například přímou manipulaci s modelem, což by změnilo nezávislou simulaci, která měla s řídícím systémem pouze interagovat, na model plně ovládaný řídícím systémem a v zásadě bychom tak získaly jeden pevně daný scénář.

Ani po obhajobě projektu zbytek týmu neobdržel použitelnou ...
(In progress odevzdano pro jistotu)