VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

GENERÁTOR KLIENTŮ PRO LANGUAGE SERVER PROTOCOL
CLIENT GENERATOR FOR LANGUAGE SERVER PROTOCOL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Dominik Jelínek**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Číka, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Dominik Jelínek ***ID:*** 177264
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Generátor klientů pro Language Server Protocol

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a implementovat nástroj pro vytváření klientů Language Server Protocol (LSP) [1] pro různá vývojová prostředí (Eclipse, Eclipse Che, VS Code, ...). Vstupem nástroje bude určitý server LSP. Výstupem budou klienti LSP pro zadaný server na vstupu, včetně instrukcí pro integraci klienta do vývojového prostředí. Výsledný generátor bude schopen generovat klienty pro server Apache Camel LSP [2] a bude jej možné lehce rozšířit o podporu dalších serverů LSP. Generátor by měl být uživatelsky přívětivý a postaven nad moderními technologiemi a principy. Body zadání:

1. Nastudovat Language Server Protocol,
2. nastudovat architekturu klientů LSP pro různá vývojová prostředí,
3. popsat vytváření klientů LSP v různých vývojových prostředích. Konkrétně: popis implementace LSP v daném vývojovém prostředí, popis nezbytných kroků pro vytvoření nového klienta LSP, popis sestavení, instalace a distribuce klienta LSP,
4. návrh a implementace nástroje pro automatizované vytváření klientů LSP postaveného na základě znalostí získaných v předchozích bodech.

DOPORUČENÁ LITERATURA:

[1] Language Server Protocol. Official page for Language Server Protocol [online]. 2018 [cit. 2018-09-05]. Dostupné z: https://microsoft.github.io/language-server-protocol/

[2] Apache Camel LSP. The Apache Camel LSP server implementation [online]. 2018 [cit. 2018-09-05]. Dostupné z: https://github.com/camel-tooling/camel-language-server/

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: doc. Ing. Petr Číka, Ph.D.
Konzultant: Mgr. Tomáš Sedmík (Red Hat Czech s.r.o.)

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práce se zabývá protokolem *Microsoft Language Server Protocol*. V teoretické části práce je popsána architektura a vlastnosti protokolu, popis implementace protokolu *LSP* uvnitř vývojových prostředí *Eclipse* a *VS Code* a postup pro vytvoření nového zásuvného modulu do vývojových prostředí *Eclipse* a *VS Code*. Dále se teoretická část práce věnuje seznámení s projektem *Apache Camel* a implementací *Camel Language* serveru a *Camel Language* klientů pro rozšířenou jazykovou podporu. Poslední zmínka teoretické části je věnována nástroji pro usnadnění vytváření nových projektů *Yeoman*. Popis praktické části práce se zabývá vlastnostmi a implementací vytvořeného generátoru klientů pro *Language Server Protocol*.

KLÍČOVÁ SLOVA

Apache Camel, Eclipse, JSON-RPC, Language Server Protocol, LSP, VS Code, Yeoman

ABSTRACT

The Diploma thesis deals with the *Microsoft Language Server Protocol*. The theoretical part describes the architecture and protocol properties, the *LSP* implementation within the *Eclipse* and *VS Code* development environments and a procedure for creating a new plug-in in the *Eclipse* and *VS Code*. In addition, the theoretical part familiarizes with *Apache Camel* project and implementation of *Camel Language* server and *Camel Language* clients for extended language support. The last mention in the theory is about the *Yeoman* tool for scaffolding a new projects. The description of the practical part deals with properties and implementation of the created *LSP* clients generator for *Language Server Protocol*.

KEYWORDS

Apache Camel, Eclipse, JSON-RPC, Language Server Protocol, LSP, VS Code, Yeoman

JELÍNEK, Dominik. *Generátor klientů pro Language Server Protocol*. Brno, 2019, 59 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Petr Číka, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Generátor klientů pro Language Server Protocol“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Petru Číkovi, Ph.D. a externímu odbornému konzultantovi z firmy Red Hat panu Mgr. Tomáši Sedmíkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

V dnešní době velkého výběru pokročilých textových editorů a vývojových prostředí je z pohledu uživatelů (programátorů) kladen velký důraz na podporu širokého množství programovacích jazyků a podporu rozšířených funkcí editoru pro daný programovací jazyk, např. automatické dokončování příkazů, přechod na definici, formátování kódu, atd. Pro vývoj a podporu i pouze jedné ze zmíněných funkcí editoru pro jeden programovací jazyk je zapotřebí vyvinout velké úsilí napříč všemi dostupnými vývojovými nástroji. Důvodem značné časové náročnosti je zejména rozdílná architektura a implementace jednotlivých prostředí.

Proto byl vyvinut *Language Server Protocol*, který umožňuje soustředit úsilí vývojářské komunity programovacích jazyků do práce na jednom jazykovém serveru (*Language Server*) pro každý programovací jazyk, který bude poskytovat všechny zmíněné funkce a mnoho dalších. Současně se komunita kolem vývojových nástrojů může soustředit na vývoj specifického jazykového klienta (*Language Client*) pro dané vývojové prostředí nebo textový editor, který bude navíc schopen komunikovat s jakýmkoliv z dostupných jazykových serverů a poskytnout tak požadovanou plnou jazykovou podporu.

Hlavním cílem diplomové práce je navrhnout a implementovat nástroj pro automatizované vytváření jazykových klientů komunikujících přes *Language Server Protocol* pro různá vývojová prostředí.

První kapitola teoretické části diplomové práce je věnována protokolu *Microsoft Language Server Protocol* a seznámení s důvody vzniku protokolu a jeho následné využití v praxi. Dále následuje popis architektury protokolu a vysvětlení principu komunikace. Následující kapitola se zaměřuje na popis možností rozšiřování vývojových prostředí *Eclipse* a *VS Code* o nové zásuvné moduly. Dále jsou zde probrány způsoby instalace nových zásuvných modulů do vývojových prostředí *Eclipse* a *VS Code*.

Třetí kapitola teoretické části popisuje implementaci protokolu *Language Server Protocol* pro vývojová prostředí *Eclipse* a *VS Code*. Součástí této kapitoly je popis nezbytných kroků pro vytvoření nového jazykového klienta, popis sestavení klienta, instalace a distribuce jazykového klienta pro *Language Server Protocol*.

Navazující kapitola seznamuje s projektem *Apache Camel*. V této kapitole je detailně popsána již existující implementace jazykového serveru *Camel Language Server* a momentálně dostupných jazykových klientů *Camel Language Client*.

Pátá kapitola teoretické části práce je věnována nástroji pro usnadnění práce při vytváření struktury nových projektů – *Yeoman*.

V poslední kapitole je popsán vytvořený **generátor klientů LSP**. Popis vytvoření generátoru se dělí do několika částí. První dvě části jsou věnovány popisu

přípravy prostředí pro vývoj a popisu jednotlivých kroků pro přípravu projektu generátoru. Následující třetí a čtvrtá část jsou zaměřeny na vlastnosti vytvořeného generátoru a použitou interní logiku generování klientů. V poslední části kapitoly je demonstrováno ověření funkčnosti generátoru pro různá vývojová prostředí a rozdílné jazykové servery.

1 Microsoft Language Server Protocol

Language Server Protocol [1], nebol-li *LSP*, byl původně vyvinut společností *Microsoft*, která v roce 2016 oznámila oficiální spolupráci se společnostmi Red Hat, Codenvy a Sourcegraph za účelem větší podpory, rychlejšího růstu a standardizace protokolu.

Jedná se o otevřený protokol, který je využíván pro komunikaci mezi textovým editorem nebo vývojovým prostředím a serverem podporujícím určitý programovací jazyk za účelem rozšíření vlastností editoru pro podporu daného programovacího jazyka.

Vývoj a podpora funkcí editoru, jako jsou např. automatické dokončování příkazů, přechod na definici nebo formátování kódu pro každý programovací jazyk, si žádá značné množství času a úsilí. Je zapotřebí vytvořit každou ze zmíněných funkcí zvlášť pro každé vývojové prostředí nebo textový editor, jelikož ty se liší svou architekturou a implementací.

Hlavní myšlenkou protokolu *LSP* je usnadnit vývoj podpory programovacích jazyků ve vývojových prostředích přesunem veškeré logiky na stranu serveru, která je dále zpřístupněna prostřednictvím protokolu *LSP*. Tímto způsobem lze uživateli poskytnout rozšířenou podporu pro daný programovací jazyk bez závislosti na editoru nebo vývojovém prostředí.

1.1 Protokol JSON-RPC

JSON-RPC [2] je bezstavový protokol vzdáleného volání procedur (*Remote Procedure Call*, *RPC*). Cílem bylo navrhnout jednoduchý protokol, a proto definuje pouze několik datových struktur a pravidel pro jejich zpracování.

Jednoduchost protokolu *JSON-RPC* umožňuje přenos zpráv v rámci jednoho stejného procesu, mezi procesy a přes internet. Datovým formátem protokolu je *JSON* (*RFC 4627*) [3].

Vlastnosti protokolu

Protokol pracuje na principu odesílání požadavků, odpovědí, událostí a chyb [4]. Všechny zmíněné typy přenosu jsou považovány za jednotlivé objekty, serializované formátem protokolu *JSON*.

Každý z objektů obsahuje parametry, které mohou, ale nemusí, být povinné a jsou specifické pro daný typ zprávy.

Objekt požadavku

Parametry, které lze nalézt v objektu požadavku jsou:

- **jsonrpc**
  - specifikuje verzi použitého *JSON-RPC* protokolu,
  - aktuální verze protokolu je „2.0“.
- **method**
  - text, obsahující název požadované metody protokolu,
  - názvy začínající předponou „rpc“ jsou protokolem vyhrazeny pro interní metody a rozšíření a nemohou být jinak použity.
- **params** (*nepovinný údaj*)
  - pokud existuje, musí se jednat o strukturovanou hodnotu:
    - **pozice v poli** – parametr musí být pole, obsahující prvky v očekávaném pořadí druhou stranou komunikace,
    - **jméno objektu** – musí být objekt, obsahující názvy všech členů odpovídající očekávaným názvům druhé strany komunikace, názvy musí přesně odpovídat.
- **id**
  - identifikátor vytvořený klientem, který musí obsahovat hodnotu `String` (text), `Number` (celé číslo) nebo být „`NULL`“ (prázdná hodnota),
  - parametr je určen pro synchronizaci objektů (požadavku a odpovědi).

Objekt události

Jedná se o typově stejný objekt jako je objekt požadavku, avšak s tím rozdílem, že parametr `id` je roven hodnotě `NULL`. Strana, která odeslala objekt události nečeká na odpověď a díky tomu nedochází k žádným problémům.

Objekt odpovědi

Posílá se jako reakce na obdržený objekt požadavku. Parametry, které lze nalézt v objektu odpovědi jsou:

- **jsonrpc**
  - specifikuje verzi použitého *JSON-RPC* protokolu,
  - aktuální verze protokolu je „2.0“.
- **result**
  - povinný – při úspěšném dokončení požadované metody požadavku,
  - nepovinný – po obdržení chyby při provádění požadované metody, požadavku,
  - výstupní hodnota odpovídá návratové hodnotě vykonávané metody požadavku.

- **error**
  - povinný – po obdržení chyby při provádění požadované metody požadavku,
  - nepovinný – při úspěšném dokončení požadované metody požadavku,
  - výstupní hodnota odpovídá příslušnému objektu chyby definovanému v tab. 1.1.
- **id** (*povinný údaj*)
  - hodnota musí být totožná s `id` obdrženého objektu požadavku.

Objekt chyby

Parametry, které lze nalézt v objektu chyby jsou:

- **code**
  - číslo, které signalizuje typ vyskytnuté chyby (datový typ `integer`).
- **message**
  - textová hodnota obsahující chybovou hlášku.
- **data** (*nepovinný údaj*)
  - hodnota, která obsahuje podrobnější popis vyskytnuté chyby.

Kód	**Zpráva**	**Popis**
-32700	Chyba parseru	Server obdržel nevalidní *JSON*, chyba nastala při parsování *JSON* textu.
-32600	Neplatná žádost	Odeslaný *JSON* není validní.
-32601	Metoda nebyla nalezena	Metoda neexistuje/není dostupná.
-32602	Neplatné parametry	Neplatné parametry metody.
-32603	Interní chyba	Interní *JSON-RPC* chyba.
-32000 až -32099	Chyba serveru	Rezervováno pro chyby serveru definované implemetací.

Tab. 1.1: Přehled chybových kódů a jejich význam.

Dávkové volání objektů

Další vlastností protokolu je podpora odesílání více objektů najednou. Klient odešle pole, které obsahuje více objektů požadavků. Server postupně zpracuje všechny objekty požadavků a odpoví polem, které obsahuje odpovídající objekty odpovědí viz výpis 1.1.

Server může zpracovat dávkové volání objektů jako soubor souběžných úloh a zpracovávat je v libovolném pořadí. Objekty odpovědí vrácené z dávkového volání v poli, mohou být rovněž v libovolném pořadí. Klient musí následně propojit pole objektů požadavků s polem objektů odpovědí na základě parametru id v rámci každého objektu.

```
požadavek --> [
  {"jsonrpc": "2.0", "method": "sum",
      "params": [1,2,4], "id": "1"},
  {"jsonrpc": "2.0", "method": "foo.get",
      "params": {"name": "myself"}, "id": "5"},
  {"jsonrpc": "2.0", "method": "get_data", "id": "9"}
]
<-- odpověď [
  {"jsonrpc": "2.0", "result": 7, "id": "1"},
  {"jsonrpc": "2.0", "error": {"code": -32601,
      "message": "Method not found"}, "id": "5"},
  {"jsonrpc": "2.0", "result": ["hello", 5], "id": "9"}
]
```

Výpis 1.1: Ukázka objektu požadavku a odpovědi dávkového zpracování objektů.

1.2 Architektura a komunikace JSON-RPC

Základními prvky komunikace mezi vývojovými nástroji a jazykovými servery jsou:

- **Jazykový server** (*Language Server*) je určen k poskytnutí specifické jazykové podpory a komunikace s vývojovými nástroji prostřednictvím protokolu, který umožňuje komunikaci mezi procesy.
- **Jazykový klient** (*Language Client*) je určen k zprostředkování komunikace vývojových nástrojů s jazykovými servery.
- **Language Server Protocol** slouží pro přenos zpráv a řízení komunikace mezi klientem a příslušným serverem.
- **JSON-RPC** je bezstavový protokol vzdáleného volání procedur (*RPC*), který je využíván pro přenos zpráv protokolem LSP.

Jazykový server [5] lze popsat jako samostatně běžící proces, který je spuštěn na pozadí operačního systému. Vývojové nástroje komunikují se serverem prostřednictvím jazykového protokolu (*LSP*) přes *JSON-RPC*.

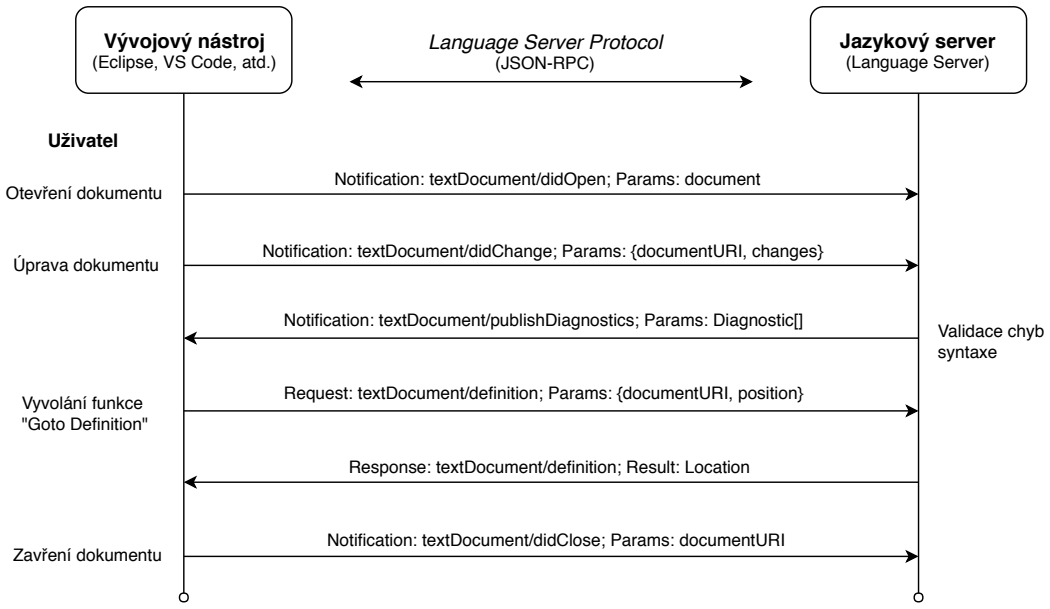
Obrázek 1.1 znázorňuje běžnou komunikaci mezi vývojovým prostředím a jazykovým serverem při jednoduché editaci souboru. Pokud uživatel upravuje zdrojový kód

za použití nástroje s implementovanou podporou protokolu jazykového serveru, nástroj funguje jako klient, který zpracovává jazykové služby poskytované jazykovým serverem. Vývojový nástroj může být textový editor nebo *IDE* a jazykové služby mohou být např. refaktorování zdrojového kódu, dokončování klíčových slov atd.

Popis komunikace

Klient informuje server o interakci uživatele, např. otevření souboru, vkládání znaku na konkrétní pozici v textu atd. Klient může také požádat server o provedení jazykově specifické funkce, např. formátovat zadaný rozsah v textovém dokumentu.

Následně server odpovídá na každý požadavek klienta příslušnou odpovědí. Například požadavek na formátování je zodpovězen odpovědí, která přenese formátovaný text klientovi, nebo chybovou odpovědí obsahující podrobnosti o chybě.



Obr. 1.1: Ukázka komunikace vývojového nástroje a jazykového serveru.

Typy zpráv definované pro komunikaci prostřednictvím protokolu *Language Server Protocol* mezi jazykovým klientem a jazykovým serverem:

1. **Požadavek** (*Request*) je zasílán ve formě *JSON-RPC* zprávy klientem nebo serverem. Na každý požadavek musí vždy existovat příslušná reakce ve formě odpovědi.
2. **Odpověď** (*Response*) obsahuje informace pro klientem/serverem obdržený požadavek.
3. **Událost** (*Notification*) na rozdíl od předchozích dvou zpráv pouze informuje druhou stranu a neočekává žádnou reakci ve formě odpovědi.

```
"jsonrpc": "2.0",
"id": "7",
"method": "textDocument/completion",
"params": {
  "textDocument": {
    "uri": "file://${path}/asdf/src/camel-context.xml"
  },
  "uri": "file://${path}/asdf/src/camel-context.xml",
  "position": {
    "line": 5,
    "character": 16
  }
}
```

Výpis 1.2: Ukázka obsahu zprávy přijatého požadavku protokolu *LSP*.

Všechny zmíněné zprávy mohou být inicializovány klientem nebo serverem, není pravidlem, že požadavky existují pouze ze strany klienta a server zasílá jen odpovědi. Protokol neurčuje požadavky pro přenos zpráv mezi serverem a klientem, tzn. pro komunikaci lze využít různé varianty spojení:

- **Volání metod stejného procesu** – klient a server komunikují prostřednictvím výměny řetězců ve formátu *JSON* v rámci jednoho stejného procesu.
- **Meziprocesová komunikace** – více procesů na jednom nebo více lokálních strojích komunikujících mezi sebou na úrovni běžícího OS.
- **Síťová komunikace** – mezi klientem a serverem probíhá komunikace prostřednictvím síťové adresy a portu (*socket*).

Komunikace protokolu a jazykového serveru probíhá na úrovni odkazů na soubory (`URI`) a **pozice kurzoru** uvnitř odkazovaného souboru [6]. Jedná se o neutrální datové typy, které lze aplikovat a využít na každý programovací jazyk. Použité datové typy nejsou na úrovni doménového modelu programovacího jazyka, který by obvykle představoval abstraktní syntaxní stromy a symboly překladačů (např. datové typy, jmenné prostory, ...).

Jednoduchost a neutrálnost použitých datových typů protokol *LSP* velmi zjednodušuje, protože standardizovat cestu k souboru (*URI*) a pozici kurzoru ve zdrojovém kódu je mnohem jednodušší než se pokusit standardizovat abstraktní syntaxe a symboly kompilátoru napříč různými programovacími jazyky.

```
"jsonrpc": "2.0",
"id": "7",
"result": [{
   "label": "timer:timerName",
   "documentation": "The timer component is used for...",
   "deprecated": false,
   "textEdit": {
     "range": {
       "start": {
         "line": 5,
         "character": 13
       },
       "end": {
         "line": 5,
         "character": 16
       }
     },
     "newText": "timer:timerName"
   }
 }]
```

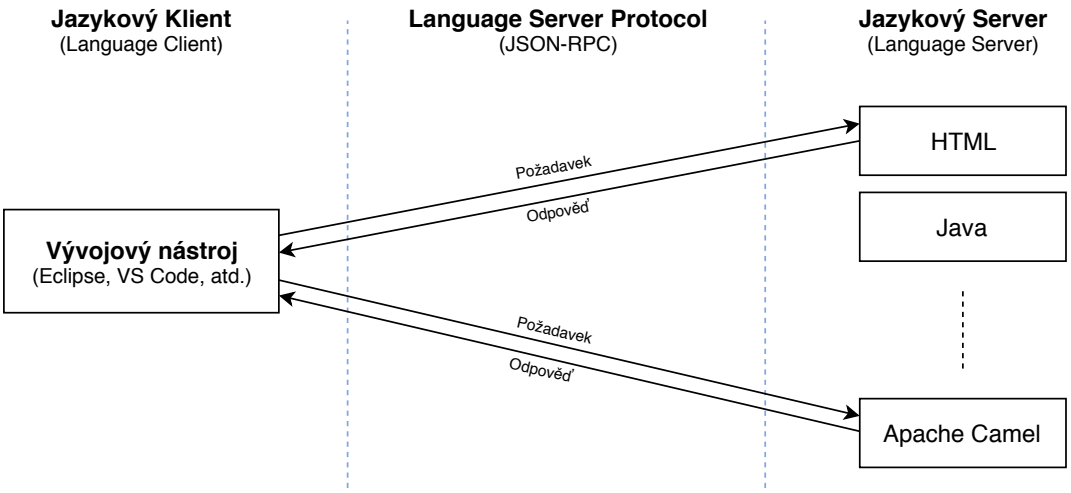
Výpis 1.3: Ukázka obsahu zprávy odeslané odpovědi protokolu *LSP*.

Aktuální verze protokolu 3.0 je přizpůsobena pro práci s dokumenty v textové podobě (`String`), momentálně neexistuje podpora pro binární soubory. Umístění (pozice kurzoru) uvnitř souboru je vyjádřeno jako offsetová linie tzv. „zero-based line“ a odpovídající posun znaků, který využívá kódování `UTF-16`.

Protokol *LSP* podporuje komunikaci s více jazykovými servery zároveň, tzn. uživatelem používaný vývojový nástroj umožňuje využívat pokročilé funkce editoru pro více programovacích jazyků současně, viz obr. 1.2.

Language Server Protocol definuje mnoho funkcí, ovšem ne každý vytvořený jazykový server je schopen podporovat všechny protokolem dostupné funkce. Pro ujasnění serverem podporovaných funkcí existuje v *LSP* vlastnost **`capabilities`** nebo-li „**schopnosti**“. Schopnosti jazykového serveru seskupují sadu jazykových funkcí, které jsou na straně serveru, ale i klienta podporované.

Například server oznamuje, že je schopen zpracovat požadavek „A“, ale nemusí být schopen zpracovat požadavek „B“. Podobně vývojový nástroj (klient) oznamuje serveru své podporované schopnosti.



Obr. 1.2: Ukázka podpory více jazykových serverů současně.

Výsledná implementace jazykového serveru do konkrétního vývojového nástroje není definována protokolem *LSP*, ale je plně ponechána k řešení programátorům daného vývojového nástroje.

2 Rozšiřování vývojových prostředí

Účelem kapitoly je popsat základní architekturu vývojových prostředí *Eclipse* a *VS Code*. Popsat způsoby rozšiřování o nové zásuvné moduly a možnosti instalace (distribuce) nově vytvořených zásuvných modulů do zmíněných *IDE*.

Pro bližší popis byly do teoretické části diplomové práce záměrně vybrány pouze dvě vývojové prostředí – *Eclipse* a *VS Code*. Jedná se o jedny z populárních *IDE* současnosti.

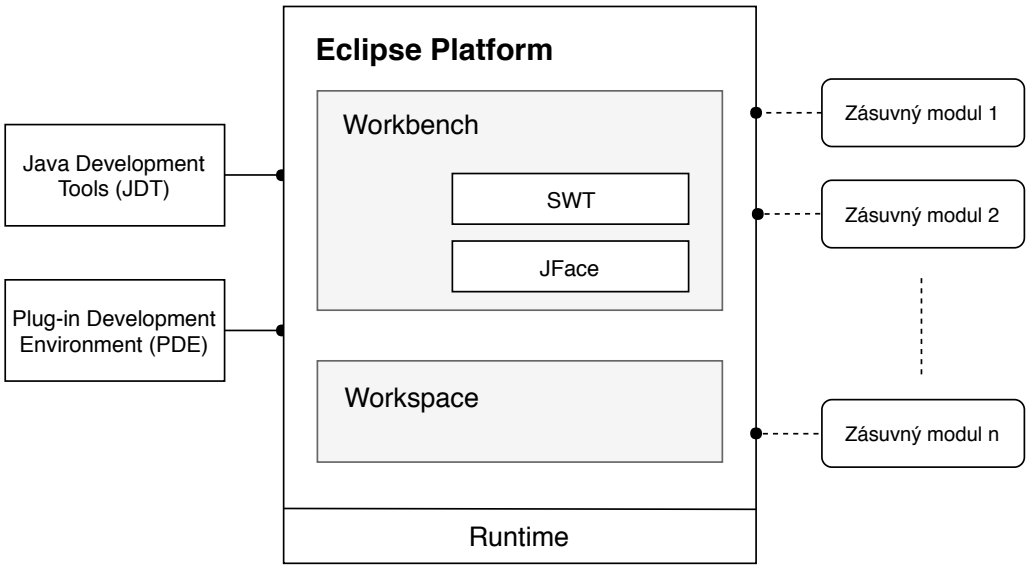
2.1 Vývojové prostředí Eclipse

Architektura vývojového prostředí

Vývojové prostředí *Eclipse* je postaveno na konceptu zásuvných modulů (*plug-inů*), které mohou, ale nemusí, být vzájemně závislé.

Zásuvný modul (*plug-in*) je nejmenší samostatná jednotka v *Eclipse*. O architektuře prostředí *Eclipse* [7] lze hovořit jako o „plug-in architektuře“. Z obrázku 2.1 je patrné rozdělení vývojového prostředí *Eclipse* na tři základní části:

- platforma *Eclipse* (*Eclipse Platform*),
- nástroje pro vývoj v jazyce *Java* (*Java Development Tools*, *JDT*),
- prostředí pro vývoj zásuvných modulů (*Plug-in Development Environment*, *PDE*).



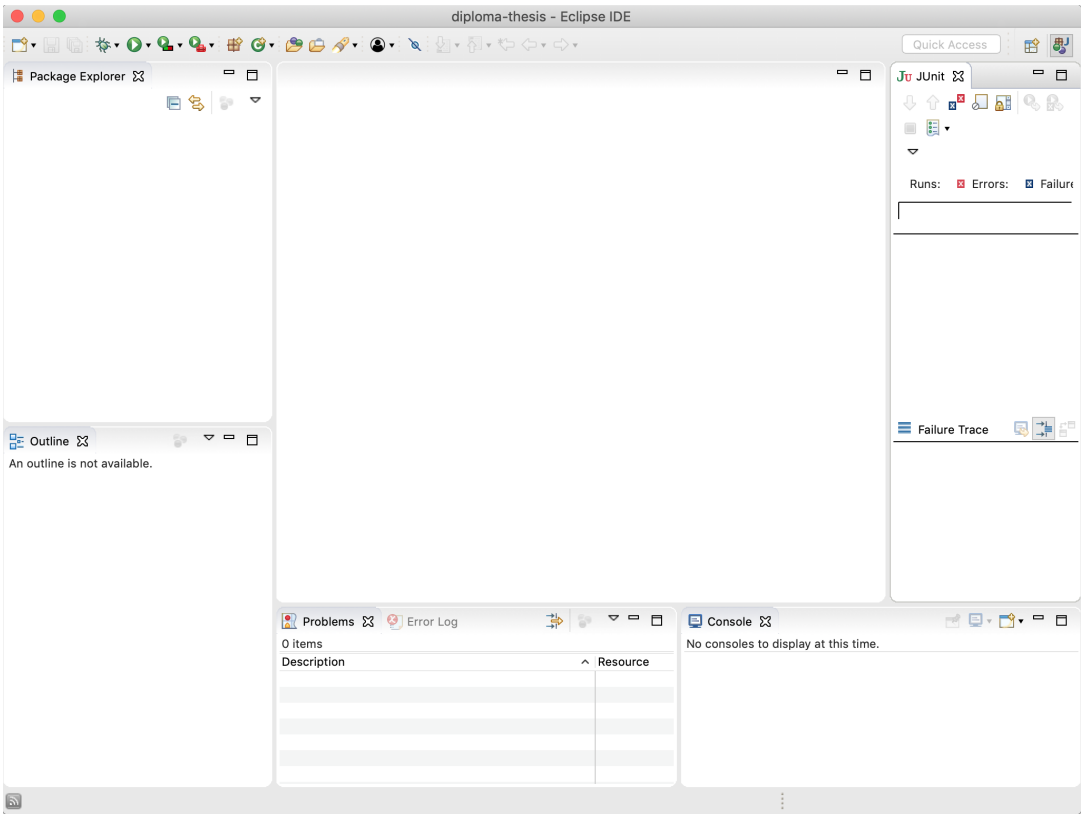
Obr. 2.1: Blokové schéma architektury projektu *Eclipse*.

Tyto tři části tvoří základ projektu *Eclipse*, ve kterém je ***Eclipse Platform*** [8] hlavním prvkem celého vývojového prostředí.

Platforma *Eclipse* je dále tvořena třemi komponentami:

- uživatelská pracovní plocha (*Workbench*),
- pracovní prostor (*Workspace*),
- běhové prostředí *Eclipse* (*Runtime*).

Běhové prostředí *Eclipse* je postaveno na specifikaci frameworku *OSGi* [9]. Jedná se o specifikaci modulárního dynamického systému pro programovací jazyk *Java*, který dovoluje v *Eclipse IDE* přidávání a odebírání zásuvných modulů za běhu aplikace.



Obr. 2.2: Okno aplikace vývojového prostředí *Eclipse*.

Základním pojmem v celé logice *OSGi* je ***OSGi balíček*** (tzv. *OSGi bundle*) [10]. Bundle je běžný *JAR* archiv obsahující několik speciálních hlaviček, které definují všechny závislosti, „životní cyklus“ a status přidaného zásuvného modulu. Každý nově přidaný zásuvný modul je chápán jako nový *OSGi bundle* v *OSGi* logice.

Pro práci a vývoj aplikací založených na *OSGi* existuje projekt ***Eclipse Equinox*** [11]. *Eclipse Equinox* je implementací základního frameworku *OSGi* a pro běh aplikací poskytuje tzv. ***OSGi-based runtime***.

Každou část *Eclipse IDE* tedy tvoří *OSGi bundle* (např. *JDT*, *PDE*, *SWT*, *JFace* a další) a při inicializaci vývojového prostředí jsou všechny části poskládány do výsledného běhového prostředí *Eclipse IDE*.

Java Development Tools (*JDT*) je zásuvný modul, který umožňuje používat dva důležité nástroje vývojového prostředí – **debugger** a **kompilátor**.

Plug-in Development Environment (*PDE*) je zásuvný modul, který poskytuje rozšířené nástroje do prostředí *Eclipse* pro tvorbu nových zásuvných modulů.

Vytváření zásuvných modulů

Pro vytváření vlastních zásuvných modulů [12] je v *Eclipse IDE* dostupný zásuvný modul ***PDE*** (*Plug-in Development Environment*). Součástí *PDE* jsou pro tvorbu nového plug-inu předem definované šablony, které mají implementovány základní grafické ovládací prvky z knihoven *SWT* a *JFace*. Šablon je několik typů a ve vývojovém prostředí *Eclipse* jsou dostupné v záložce **`File > New > Project > Plug-in Development > Plug-in Project`**. Mezi základní šablony patří:

- plug-in pro tvorbu textového editoru,
- plug-in s náhledem (*view*)
- a ukázkový „*Hello, world plug-in*“.

Vlastnosti zásuvného modulu definuje několik souborů. Hlavním je soubor *XML* s názvem **`plugin.xml`** (výpis 2.1). Soubor definuje vlastnosti vyvíjeného zásuvného modulu a popisuje způsob implementace funkcionalit pro vývojové prostředí *Eclipse*.

```
1  <plugin>
2   <extension point="org.eclipse.ui.views">
3    <category name="Sample␣Category"
4       id="org.vutbr.diploma.example">
5    </category>
6    <view id="org.vutbr.diploma.example.views.SampleView"
7       name="Sample␣View"
8       icon="icons/sample.png"
9       class="org.vutbr.diploma.example.views.SampleView"
10      category="org.vutbr.diploma.example"
11      inject="true">
12   </view>
13  </extension>
14  <extension point="org.eclipse.help.contexts">
15      <contexts file="contexts.xml"/>
16  </extension>
17 </plugin>
```

Výpis 2.1: Struktura souboru `plugin.xml`.

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: Example
4  Bundle-SymbolicName: org.vutbr.example;singleton:=true
5  Bundle-Version: 1.0.0.qualifier
6  Bundle-Activator: org.vutbr.example.Activator
7  Bundle-Vendor: VUT Brno by djelinek
8  Require-Bundle: org.eclipse.ui,
9   org.eclipse.core.runtime
10 Bundle-RequiredExecutionEnvironment: JavaSE-1.8
11 Automatic-Module-Name: org.vutbr.example
12 Bundle-ActivationPolicy: lazy
```

Výpis 2.2: Ukázka souboru `MANIFEST.MF`.

Atributy značky `<plugin>` a reference závislých zásuvných modulů jsou pro zpřehlednění umístěny do souboru **`MANIFEST.MF`** (výpis 2.2).

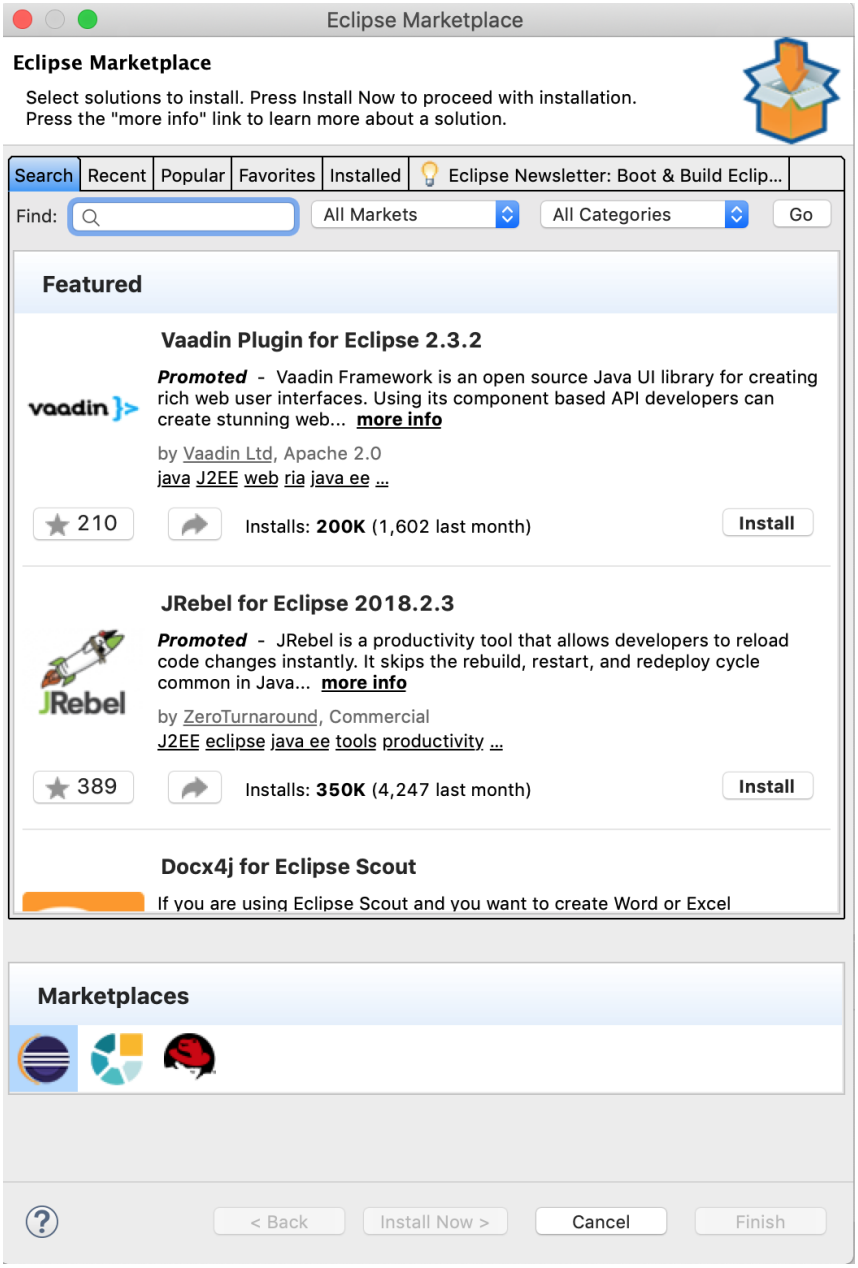
Během vývoje nového zásuvného modulu je praktické průběžně testovat funkčnost. V zásadě je nutné plug-in opakovaně kompilovat, nainstalovat do aktuální instance *Eclipse IDE* a až poté otestovat. zda-li všechny implementované vlastnosti fungují správně.

Postup opětovné instalace při každé kontrole funkčnosti plug-inu je velmi neefektivní, proto *Eclipse* umožňuje za běhu spustit nové okno vývojového prostředí nad aktuálně spuštěnou instancí *Eclipse IDE*. Nově vytvořená instance vývojového prostředí již obsahuje nejnovější verzi vyvíjeného plug-inu a je možné ihned otestovat správnost nových funkcí.

Ke spuštění nového běhového prostředí je v *Eclipse IDE* připravena položka menu **`Run > Run As > Eclipse Application`** nebo **`Run > Run Configurations...`**, kde je uživateli dovoleno podrobněji specifikovat běhové prostředí, které bude pro testování vytvořeno.

Instalace a distribuce zásuvných modulů

Nedílnou součástí nového zásuvného modulu je následná distribuce uživatelům a instalace do vývojového prostředí *Eclipse* [13]. Prioritou by vždy měla být jednoduchost a pohodlí instalace z pohledu běžného uživatele.



Obr. 2.3: Okno zásuvného modulu *Eclipse Marketplace Client*.

Prvním krokem je sestavení vytvořeného projektu nového zásuvného modulu do publikovatelné podoby, kterou lze jednoduše instalovat do vývojového prostředí.

V tomto ohledu *Eclipse IDE* nabízí čtyři varianty distribuce:

1. exportovat zásuvný modul přímo do aktuálně spuštěného *Eclipse IDE*,
2. exportovat zásuvný modul do souboru (`.jar`), který lze následně jednoduše přidat do běhového prostředí *Eclipse*,
3. vytvořit instalační odkaz (*Update site*) a využít integrovaného instalačního průvodce **Eclipse manažer** (*Eclipse udpate manager*) a plug-in nainstalovat
4. nebo umístit exportovaný zásuvný modul na tzv. ***Eclipse Marketplace*** [14], kde jsou dostupné všechny oficiální zásuvné moduly a rozšíření pro vývojové prostředí *Eclipse*.

První tři varianty jsou jednoduché a časově nenáročné, v *Eclipse IDE* jsou přímo integrováni průvodci, prostřednictvím kterých není složité výsledný projekt sestavit a exportovat do požadované podoby. Čtvrtá varianta je více časově náročná a složitější, je zapotřebí být zaregistrován na doméně `eclipse.org` a po přidání obsahu do databáze *Eclipse Marketplace* trvá 24 hodin, než bude nově přidaný obsah zobrazen.

Pro běžného uživatele je z pohledu dostupnosti a pohodlí umístění nového klienta na stránku *Eclipse Marketplace* vhodné řešení, protože *Eclipse IDE* má integrován zásuvný modul (obr. 2.3 – ***Eclipse Marketplace Client*** [15], který zpřístupňuje veškerý dostupný a instalovatelný obsah pro danou verzi *Eclipse*.

2.2 Vývojové prostředí Visual Studio Code

Architektura vývojového prostředí

Vývojové prostředí *Visual Studio Code* [16] (zkráceně ***VS Code***) bylo roku 2015 představeno společností *Microsoft* jako editor zdrojových kódů, který je volně dostupný a s otevřeným zdrojovým kódem (*open-source*) pro všechny vývojáře.

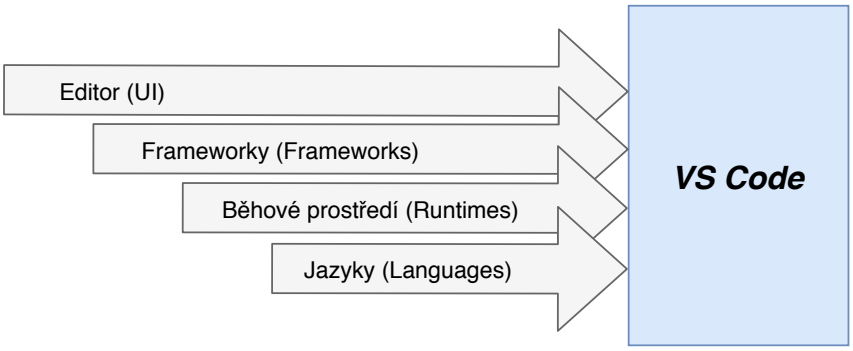
VS Code je postaven na frameworku *Electron* [17], který je používán k vytváření aplikací s běhovým prostředím *Node.js* [18] spuštěných na ploše operačního systému a to prostřednictvím nástroje (*enginu*) rozvržení *Blink* [19].

Pro pokročilou práci se soubory je do vývojového prostředí integrován editor *Monaco* [20].

Electron byl vyvinut firmou *GitHub* pro vytváření multiplatformních desktopových aplikací s použitím technologií *HTML*, *CSS* a *JavaScript*. Své vlastnosti a výhody získává prostřednictvím kombinace technologií *Chromium* [21] a *Node.js* do jednoho jediného běhového prostředí.

Chromium je open-source prohlížečový projekt, jehož cílem je vybudovat bezpečnější, rychlejší a stabilnější způsob prohlížení internetu pro všechny uživatele.

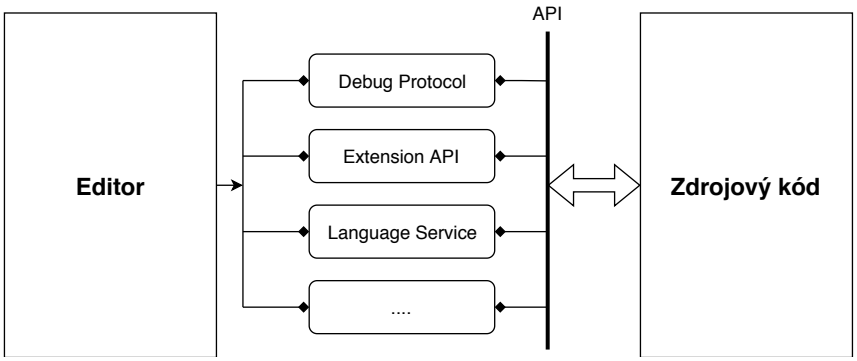
Node.js je v podstatě asynchronní proces řídící běhové prostředí *JavaScript*. *Node* je navržen zejména pro vytváření škálovatelných síťových aplikací.



Obr. 2.4: Architektura vývojového prostředí *VS Code* [22].

Celé *IDE* je vysoce uživatelsky přizpůsobitelné a umožňuje rozšiřování o nové funkcionality prostřednictvím přidání nových rozšíření – ***extensions***. Rozšíření lze přidat z množství již existujících nebo vytvořit nové vlastní. *VS Code* lze popsat jako entitu, která se skládá z několika různých komponent (obr. 2.4). Základní rozdělení komponent vývojového prostředí je do čtyř kategorií dle obr. 2.5:

1. ***Editor*** – jedná se o editor *Monaco*, který je určen pro pokročilou práci se soubory (pokročilá funkce *IntelliSense*, validace zdrojových kódů, základní zbarvení syntaxe, atd.).
2. ***Debug Protocol*** – umožňuje vývoj, integraci a spuštění rozšíření určených pro ladění zdrojových kódů.
3. ***Extension API*** – je rozhraní sloužící pro úpravy a tvorbu rozšíření vývojového prostředí.
4. ***Language Service*** – je rozhraní pro komunikaci s jazykovým serverem za účelem poskytnutí pokročilé jazykové podpory.



Obr. 2.5: Komunikace mezi editorem a zdrojovými kódy [22].

Vytváření nových rozšíření

Před zahájením samotného vytváření vlastních rozšíření do vývojového prostředí *VS Code* [23] je nejprve nutné připravit (nainstalovat) všechny nezbytné nástroje:

1. ***Node.js***,
2. ***Git*** a
3. ***Yeoman***.

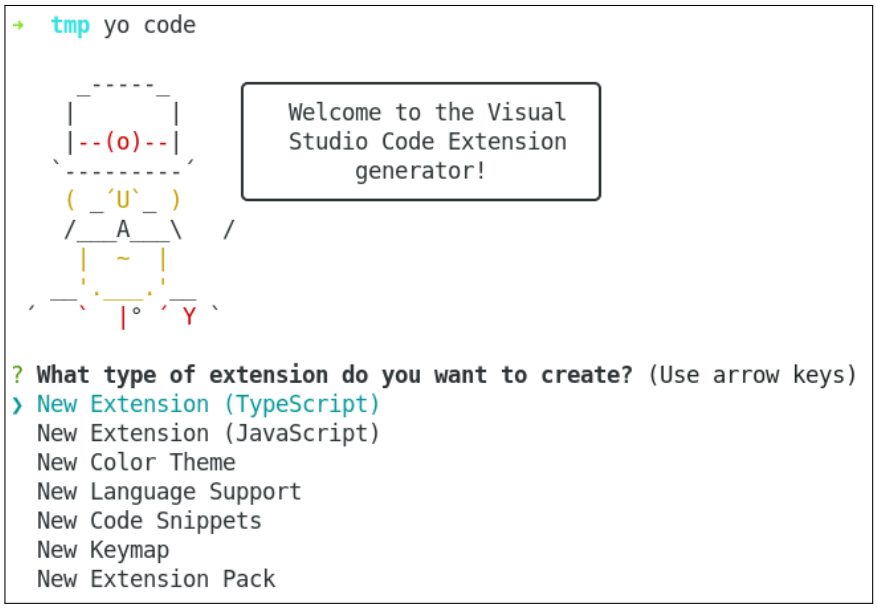
Pro jednoduché založení základního projektu nového rozšíření, které obsahuje všechny potřebné soubory a základní závislosti, je dostupný generátor projektů – ***VS Code Extension Generator*** [24].

Zaprvé je nutné výše zmíněný generátor šablon nainstalovat zadáním příkazu do příkazové řádky (výpis 2.3).

```
npm install -g yo generator-code
```

Výpis 2.3: Příkaz pro instalaci generátoru šablon rozšíření.

Jakmile je generátor šablon nainstalován, stačí generátor spustit příkazem „`yo code`“ a vybrat možnost „*New Extension (TypeScript)*“ viz obr. 2.6.



Obr. 2.6: Hlavní menu generátoru šablon - *VS Code Extension Generator*.

Následně je zobrazen průvodce pro vytvoření nového rozšíření (obr. 2.7), kde je vyžadováno zadat pouze několik základních parametrů popisujících připravované rozšíření.

```
? What type of extension do you want to create? New Extension (TypeScript)
? What's the name of your extension? AhojSvete
? What's the identifier of your extension? ahojsvete
? What's the description of your extension?
? Initialize a git repository? Yes
? Which package manager to use? npm
```

Obr. 2.7: Průvodce pro vytvoření nového rozšíření.

Jakmile je generování nového projektu šablony rozšíření dokončeno, bude struktura projektu vypadat přibližně jako na obr. 2.8.

```
nove_rozsireni
├── .vscode
│   ├── launch.json ..... Konfigurace pro spouštění a ladění rozšíření.
│   └── tasks.json ..... Konfigurace pro sestavení úloh.
├── .gitignore ..... Seznam ignorovaných částí při sestavení rozšíření.
├── README.md ..... Popis vlastností nového rozšíření.
├── src
│   └── extensions.ts ..... Zdrojový kód rozšíření.
├── package.json ..... Manifest soubor rozšíření.
└── tsconfig.json ..... TypeScript konfigurace.
```

Obr. 2.8: Souborová struktura nového rozšíření.

Důležité soubory každého rozšíření vývojového prostředí *VS Code* jsou soubory **`package.json`** a **`extensions.ts`**. Další úpravy a vývoj rozšíření lze provádět již uvnitř samotného prostředí *VS Code*.

Soubor `package.json` je tzv. manifest soubor nového rozšíření, obsahuje vlastnosti rozdělené do sekcí, které definují schopnosti rozšíření.

Soubor `extensions.ts` je hlavní soubor nového rozšíření. Obsahuje celou základní logiku nového rozšíření a rovněž důležitou aktivační funkci, která se vykoná vždy při spuštění rozšíření.

Nově vytvořené rozšíření lze spustit a testovat manuálně v nově vytvořené instanci editoru v aktuálním běhovém prostředí, a to prostřednictvím položky menu **`Debug > Start Without Debugging`** nebo využitím klávesové zkratky **`Ctrl + F5`**.

Instalace a distribuce rozšíření

Jakmile je nové rozšíření funkční a připraveno k distribuci, existují dva hlavní způsoby, kterými lze rozšíření pro *Visual Studio Code* dále nabízet:

1. ***VS Code Extension MarketPlace***,
2. **Balíček** – instalační formát *VSIX*.

VS Code Extension MarketPlace [25] je veřejně přístupná knihovna všech dostupných rozšíření pro vývojové prostředí *VS Code* – uživatelé jsou sami schopni balíčky vyhledávat, stahovat a instalovat do svého prostředí.

Balíček ve formátu *VSIX* lze využít pro instalaci do lokálního vývojového prostředí nebo pro přímou distribuci jiným uživatelům bez oficiálního zveřejnění. Balíčky se vytváří a instalují prostřednictvím nástroje **`vsce`** (*Visual Studio Code Extensions*), nástroj je určen pro práci v příkazovém řádku.

Existuje ještě jedna varianta, a to instalace (načtení) rozšíření do lokální instalace vývojového prostředí *VS Code*. Jedná se pouze o vložení projektu rozšíření do adresáře obsahujícího všechny dostupné rozšíření aktuální instance *VS Code* (**`.vscode/extensions`**) a následné znovunačtení okna editoru příkazem **`Reload Window`**.

3 Implementace klientů pro Language Server Protocol

Pro ulehčení tvorby jazykového klienta komunikujícího s jazykovým serverem jsou již v mnoha vývojových prostředích (programovacích jazycích) k dispozici **sady vývojových nástrojů** (*Software Development Kit*, *SDK*) – knihovny, které lze pro vývoj klienta použít.

Např. pro zjednodušení práce při implementaci jazykových klientů a serverů v programovacím jazyce *Java* existují knihovny ***LSP4J*** a ***LSP4E***.

3.1 Dostupné knihovny

Knihovna LSP4J

Eclipse LSP4J [26] je knihovna jazyku *Java* určená pro práci s protokolem *LSP*. Knihovna je uzpůsobena pro jazykové servery a klienty implementované v jazyce *Java*.

Knihovna LSP4E

Knihovna *Eclipse LSP4E* [27] byla vytvořena jako nadstavba knihovny *LSP4J*. Obsahuje vše nezbytné pro integraci jakéhokoliv jazykového serveru, který splňuje podmínky specifikace protokolu *LSP* do vývojového prostředí *Eclipse*.

Prostřednictvím knihovny *LSP4E* lze řídit komunikaci mezi serverem a vývojovým prostředím (klientem), tzn. odesílat požadavky a zpracovávat přijaté odpovědi v *Eclipse*. Výchozí integrace v *Eclipse IDE* je poskytovat rozšiřující funkce do editorů *Generic* a *Extensible*, ale některé funkce umožňují integraci s dalšími dostupnými editory.

Příklady aktuálně dostupné integrace LSP v Eclipse IDE

- Zobrazit chyby jako záznamy problémů v náhledu chyb uvnitř vývojového prostředí (*Problems view*).
- Zobrazit návrhy dokončování kódu do textového editoru (*Generic Editor*).
- Zobrazit dokumentaci elementu jako nápovědu po najetí ukazatelem myši.
- Přejít na deklaraci ve zdrojovém kódu.
- Najít odkazy (*references*) ve zdrojovém kódu.
- Pojmenovat (přejmenovat) elementy zdrojového kódu.
- Provést náhled struktury prvků zdrojového kódu (*Outline view*).

Node SDK

Knihovna *Node SDK* [28] obsahuje moduly, které jsou určené pro práci s protokolem *LSP* (moduly pro klienta i server) pro vytváření nových rozšíření do vývojového prostředí *VS Code*.

K interakci klienta s vývojovým prostředím je využíváno vysoce komplexní aplikační rozhraní – *Extensions API*. Seznam dostupných modulů knihovny *Node SDK*:

- **vscode-languageclient** – určen pro komunikaci mezi jazykovým serverem a *VS Code* rozšířením.
- **vscode-languageserver** – implementace jazykového serveru s běhovým prostředím *Node.js*.
- **vscode-languageserver-protocol** – aktuální definice protokolu jazykového serveru v jazyce *TypeScript*.
- **vscode-languageserver-types** – datové typy používané klientem a serverem.
- **vscode-jsonrpc** – základní protokol zpráv pro komunikaci mezi klientem a serverem.

Příklady aktuálně dostupné integrace LSP v editoru VS Code

- Zvýrazňování všech stejných elementů v souboru – např. proměnných (*Document Highlights*).
- Zobrazení dokumentace elementu jako nápovědu po najetí kurzorem myši (*Hover*).
- Poskytnutí nápovědy pro specifický element souboru (*Signature Help*).
- Přechod na deklaraci elementu ve zdrojovém kódu (*Goto Definition*).
- Přechod na definici typu elementu pro specifický element zdrojového kódu (*Goto Type*).
- Přechod na implementaci specifického elementu zdrojového kódu (*Goto Implementation*).
- Nalezení všech odkazů na vybraný element v celém souboru zdrojového kódu (*Find References*).
- Seznam všech elementů definovaných v dokumentu (*List Document Symbols*).
- Seznam všech symbolů v celém projektu (*List Workspace Symbols*).
- Formátování celého dokumentu nebo vybraných částí, založené na typu dokumentu (*Document Formatting*).
- Přejmenování elementu napříč celým dokumentem (*Rename*).
- Výpočet a řešení závislostí elementů uvnitř celého dokumentu (*Document Links*).

3.2 Kroky pro vytvoření nového jazykového klienta

Vývojové prostředí Eclipse

Pro vytvoření nového jazykového klienta do vývojového prostředí *Eclipse* je zapotřebí nejprve připravit vše nezbytné pro začátek vývoje klienta:

1. Znalost umístění odpovídajícího *LSP* serveru – spustitelná podoba serveru (pro jazyk *Java* např. soubor s koncovkou `.jar`).
2. Znalost způsobu nastartování serveru
3. Vývojové prostředí s nainstalovanými zásuvnými moduly pro vývoj nových plug-inů (např. *Eclipse IDE* s nainstalovaným *PDE* viz dostupná distribuce *Eclipse IDE for Eclipse Committers*).
4. Znalost knihoven pro usnadnění implementace komunikace klienta a příslušného serveru prostřednictvím protokolu *LSP* v jazyce *Java* – *LSP4J*, *LSPE*.

Jakmile je výše zmíněná příprava kompletní, lze přejít k postupné implementaci jazykového klienta pro *Eclipse IDE* a navázání komunikace s nastartovaným serverem. Obecné kroky pro vytvoření nového klienta do *Eclipse IDE* jsou:

1. Vytvoření nového zásuvného modulu viz kapitola 2.1.
2. Přidání závislostí plug-inu nutných pro implementaci klienta (definice uvnitř souboru `MANIFEST.MF`):
   - **`org.eclipse.lsp4e`**,
   - **`org.eclipse.lsp4j`**,
   - **`org.eclipse.wst.sse.ui`**,
   - **`org.eclipse.wst.xml.ui`**.
3. Přidání nezbytných rozšíření do souboru **`plugin.xml`**, zejména:
   - **`org.eclipse.lsp4e.languageServer`** – identifikace spuštěného jazykového serveru prostřednictvím identifikátoru `languageId`,
   - **`org.eclipse.wst.sse.ui.completionProposal`**.
4. Stažení a umístění jazykového serveru do dostupných zdrojů zásuvného modulu (např. `${path}/jméno_projektu/resources/`).
5. Implementace třídy pro nastartování jazykového serveru a navázání spojení s využitím dostupných knihoven *LSP4J* a *LSP4E*
   – **`org.eclipse.lsp4e.server.ProcessStreamConnectionProvider`**.
6. Implementace tříd pro zajištění požadované funkce jazykové podpory **`org.eclipse.lsp4e.operations.completion.LSContentAssistProcessor`**.

Vývojové prostředí Visual Studio Code

Pro vytvoření nového jazykového klienta do vývojového prostředí *Visual Studio Code* je zapotřebí nejprve připravit vše nezbytné pro začátek vývoje klienta:

1. Znalost umístění odpovídajícího *LSP* serveru – spustitelná podoba serveru (pro server implementovaný v jazyce *Java* např. soubor s koncovkou `.jar`).
2. Znalost způsobu nastartování serveru.
3. Vývojové prostředí *VS Code*.
4. Znalost knihoven a aplikačních rozhraní pro usnadnění implementace komunikace klienta a příslušného serveru prostřednictvím protokolu *LSP* v jazyce *TypeScript* – ***Node SDK*** a ***Extension API***.

Jakmile je výše zmíněná příprava kompletní, lze přejít k postupné implementaci jazykového klienta a navázání komunikace s nastartovaným serverem. Obecné kroky pro vytvoření nového klienta pro *VS Code* jsou:

1. Vytvoření nového rozšíření viz kapitola 2.1.
2. Přidání nezbytných schopností klienta (definice jednotlivých sekcí uvnitř manifest souboru **`package.json`**):
   - `activationEvents` – sekce obsahuje definice typu souboru, pro který bude klient naslouchat (např. typ *XML* – koncovka souboru `.xml`).
   - `contributes` – bližší specifikace konfigurace požadovaného jazyka,
   - `dependencies` – závislosti jazykového klienta na ostatních rozšířeních (vždy zejména moduly – `vscode` a `vscode-languageclient`).
3. Stažení a umístění jazykového serveru do dostupných zdrojů rozšíření (např. `${path}/jméno_projektu/jars/`).
4. Implementace hlavního souboru rozšíření klienta – **`extensions.ts`**, pro nastartování jazykového serveru, navázání spojení a následnou komunikaci s využitím knihovny ***Node SDK*** a aplikačního rozhraní ***Extension API***.

Pro urychlení vývoje, lze použít šablonu projektu pro vytváření nového rozšíření jazykové podpory do vývojového prostředí *VS Code* – **`lsp-sample`**. Šablona je velmi dobře dokumentovaná, obsahuje celou základní strukturu jazykového klienta a je volně dostupná ke stažení z repozitáře šablon[1] pro vývojové prostředí *VS Code*.

[1] https://github.com/Microsoft/vscode-extension-samples/tree/master/lsp-sample

4 Projekt Apache Camel

Apache Camel [29] je výkonný *open source middleware* framework sloužící pro vytváření integračních systémových toků, orientovaných na posílání zpráv (*message-oriented*), založený na pravidlech (*rule-based*) a na známých modelech ***Enterprise Integration Patterns*** (*EIP*) [30].

Využívá programovací rozhraní nebo doménově specifický jazyk (*domain-specific-language*, *DSL*) *Java*, který umožňuje prostřednictvím frameworku ***Spring*** [31], ***Blueprint*** [32] nebo ***Scala DSL*** [33] definovat konfiguraci soubory *XML*.

Apache Camel dále podporuje inteligentní dokončování pravidel směrování s použitím běžného kódu *Java* – ***Java DSL*** [34], a to i bez velkého množství konfiguračních *XML* souborů.

Camel se často používá s dalšími projekty – *Apache ServiceMix* [35], *Apache ActiveMQ* [36], *Apache CXF* [37] a *Apache Karaf* [38] pro projekty orientované na poskytování služeb (*service-oriented*).

Framework používá identifikátory *URI*, aby bylo možné snadno pracovat s jakýmkoliv druhem transportního nebo „*messagingového*“ modelu – *HTTP*, *ActiveMQ*, *CXF* atd. a pracovat s volitelnými datovými formáty.

Jedná se o malou knihovnu, s minimálním počtem závislostí pro snadné vkládání do jakékoliv *Java* aplikace. Lze pracovat se stejným aplikačním rozhraním (*API*) bez ohledu na použitém druhu přenosu – jednou vytvořené *API* zajistí podporu komunikace se všemi komponentami, které jsou poskytovány i mimo framework.

Podporuje množství artefaktů jako např. **komponenty** (*Components*), **datové formáty** (*Data Formats*) a **jazyky** (*Languages*).

Nepochybnou výhodou frameworku je rozsáhlá podpora pro testování, která umožní snadno vyzkoušet vytvořené **trasy** (tzv. *routy*).

Základní prvky architektury Camel

Nejvyšší úrověň architektury *Camel* [39] je jednoduchá. *CamelContext* (výpis 4.1) představuje běhový systém *Camel runtime system*, propojující prvky architektury – **trasy** (*Routes*), **komponenty** (*Components*) nebo **koncové body** (*Endpoints*).

Dále, **procesory** (*Processors*) zpracovávají **směrování** a **transformace** mezi koncovými body, zatímco koncové body integrují různé externí systémy.

- **Zpráva** (*Message*) – obsahuje data, která jsou přenášena po trase. Každá zpráva má jedinečný identifikátor a je složena z těla, záhlaví a příloh.
- **Kontejner** (*Exchange*) – je vytvořen při obdržení zprávy příjemcem během procesu směrování. Umožňuje různé typy interakcí mezi systémy (např. může definovat jednosměrnou zprávu nebo zprávu s požadavkem na odpověď).

- **Koncový bod** (*Endpoint*) – je kanál, kterým může systém přijímat nebo odesílat zprávy. Může odkazovat na *URI* webové služby, *URI* fronty, soubor, e-mailovou adresu apod.
- **Komponenta** (*Component*) – funguje jako nabídka koncových bodů. Jednoduše řečeno, komponenty nabízejí rozhraní pro různé technologie používající stejný přístup a syntaxi. *Camel* již podporuje řadu komponent v *DSL* pro velké množství dalších technologií (např. *FTP*, *Twitter*, ...), ale také umožňuje psát vlastní komponenty.
- **Procesor** (*Processor*) – je jednoduché *Java* rozhraní, které se používá k přidání vlastní logiky integrace do trasy.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <camelContext
3        id="example-context"
4        xmlns="http://camel.apache.org/schema/spring">
5
6    <route id="example-route">
7      <from
8          id="_from"
9          uri="direct:idName" />
10
11     <to
12         id="_to"
13         uri="direct:idName"/>
14   </route>
15 </camelContext>
```

Výpis 4.1: Ukázka syntaxe *CamelContext* pro *Spring DSL*.

4.1 Jazykový server

Implementace serveru [40] od společnosti Red Hat – ***Camel Language Server***, která poskytuje chytrou jazykovou podporu pro *Camel DSL*. *Camel Language Server* implementuje protokol *LSP* a může být použit s libovolným editorem, který protokol *LSP* podporuje a existuje pro něj odpovídající jazykový klient.

Podporované funkce

Camel Language Server je implementován v programovacím jazyce *Java* prostřednictvím knihoven *LSP4J* a *LSP4E*, které slouží pro usnadnění vývoje a komunikace mezi klientem a serverem přes protokol *LSP* ve vývojovém prostředí *Eclipse*. Mezi podporované a funkční vlastnosti poslední verze serveru „`1.0.0-SNAPSHOT`“ (ze dne 4. Červen 2018) patří:

- Dokončování kódu pro *Camel URI* s *XML DSL*
  - pro *Camel* komponenty,
  - pro atributy *Camel* komponent
  - a pro hodnoty atributů *Camel* komponent.
- Zobrazení dokumentace najetím kurzoru myši na *Camel URI* komponenty.
- Kontrola syntaxe zápisu *Camel URI*.

Plánované funkce

- Hlášení o chybách při analýze a kompilaci.
- Pokročilejší dokončování kódu (*Advanced code completion*).
- Zobrazení struktury zdrojového kódu (*Code outline*).
- Navigace ve zdrojovém kódu (*Code navigation*).
- Reference (*References*).
- Zvýrazňování (*Highlights*).
- Formátování kódu.

4.2 Jazykový klient

V současné době existují první implementace jazykových klientů [41] do často používaných vývojových prostředí. Klienti pro *Camel Language Server* přidávají do *XML* editoru daného vývojového prostředí chytrou jazykovou podporu pro *Camel DSL*. Momentálně dostupní klienti *Camel Language Server* existují pro:

- *Eclipse IDE*,
- *Visual Studio Code*,
- *Eclipse Che*.

Eclipse IDE

Podporuje funkce dokončování kódu a zobrazení dokumentace *URI* pro *Camel* komponenty po najetí kurzoru myši při použití textového editoru *Generic*. Při použití základního editoru *XML* nebo editoru *Java* je k dispozici pouze dokončování zdrojového kódu.

Visual Studio Code

Poslední dostupná verze jazykového klienta pro vývojové prostředí *VS Code* přidává funkce jazykové podpory *Apache Camel*:

- Jazyková podpora pro *Apache Camel URI*:
  - Automatické dokončování kódu pro komponenty, atributy a seznamy hodnot atributů.
  - Zobrazení dokumentace najetím ukazatele myši na *URI Camel* komponenty.
- Navigace do *Camel* tras pro soubory *XML*.
- Kontrola syntaxe *URI Camel* komponent s *XML DSL* při ukládání souborů.
- Nalezení reference pro `direct` a `direct-vm` komponenty.

Eclipse Che

Klient pro vývojové prostředí *Eclipse Che* přináší podporu *Apache Camel LSP* pro soubory *XML*, které používají jmenný prostor *Apache Camel*.

5 Nástroj Yeoman

Yeoman [42] je terminálový nástroj určený pro usnadnění práce při zahajování nových projektů a zefektivňuje údržbu stávajících projektů. Implementuje osvědčené postupy a nástroje sloužící pro zachování produktivity práce.

Výhodou nástroje je, že není jazykově omezený – projekty lze generovat pro libovolný programovací jazyk.

Z uvedeného důvodu *Yeoman* poskytuje ekosystém generátorů šablon rozmanitých projektů. Každý jeden generátor je v podstatě zásuvný modul (plug-in) v prostředí nástroje *Yeoman*, který lze samostatně spustit za účelem vytvoření projektu nebo některé z dílčích částí projektu.

Prostřednictvím oficiální sady generátorů je uživatelům poskytnuto tzv. *Yeoman workflow*. Zmíněný pracovní postup (*workflow*) je v podstatě obsáhlý zásobník na straně klienta, obsahující nástroje a frameworky, které mohou vývojářům pomoci rychle a poměrně jednoduše vytvářet nové aplikace.

Díky modulární architektuře založené na dílčích zásuvných modulech je zajištěna integrita zásobníku a možnost velmi jednoduché rozšiřitelnosti o nové generátory.

Yeoman je koncipován, aby poskytl vše, co je zapotřebí k zahájení vývoje bez jakýchkoliv běžných problémů spojených s prvotním manuálním nastavováním nového projektu.

Sada nástrojů

Pracovní postup (*Yeoman workflow*) zahrnuje tři typy nástrojů pro zvýšení produktivity a spokojenosti při vytváření nové aplikace:

- prostředí pro vytvoření kostry projektu – ***yo***,
- nástroj pro sestavení projektu (*Build System*) – např. *Gulp*, *Grunt*, atd.
- a správce balíčků (*Package Manager*) – např. ***npm*** nebo *Bower*.

Prostředí pro vytvoření kostry projektu řídí parametry nové aplikaci, definice konfigurace sestavení projektu (např. *Gulpfile*), vybírá relevantní úkoly sestavování aplikace a potřebné závislosti pro správce balíčků (např. *npm*).

Nástroj pro sestavení projektu se používá k vytváření, prohlížení a testování projektu.

Správce balíčků zajišťuje automatizovanou správu závislostí projektu.

Všechny tři nástroje jsou vyvíjeny a udržovány samostatně, ale dohromady fungují jako součást předepsaného efektivního pracovního postupu.

Běžné příklady použití

- Rychlé vytvoření nového projektu.
- Vytvoření nové části projektu (např. zásuvný modul s jednotkovými testy).
- Vytváření modulů nebo balíčků.
- Zavádění nových služeb.
- Propagace nového projektu umožněním uživateli využít ukázkovou aplikací.

Základní terminálové příkazy

Pro jednoduché ovládání nástroje *Yeoman* je k dispozici několik základních příkazů (tab. 5.1), které je doporučeno znát.

Příkaz	**Popis**
`yo`	Otevře kompletní menu prostředí nástroje *Yeoman*.
`yo „název generátoru"`	Spustí definovaný generátor.
`yo --help`	Zobrazí nápovědu pro nástroj *Yeoman*.
`yo --generators`	Vypíše seznam všech instalovaných generátorů.
`yo --version`	Zobrazí verzi nástroje *Yeoman*.
`yo doctor`	Spustí diagnostiku a poskytne kroky k vyřešení nejběžnějších známých problémů.

Tab. 5.1: Přehled základních příkazů nástroje *Yeoman*.

6 Generátor klientů pro Language Server Protocol

Hlavním cílem diplomové práce byl návrh a implementace nástroje pro automatizované vytváření jazykových klientů *LSP*, jehož účelem je usnadnění práce při vývoji nových klientů pro libovolný jazykový server *LSP*. Po anylýze složitosti a struktury zásuvných modulů pro jednotlivá vývojová prostředí byl pro jednodušší a efektivnější implementaci zvolen nástroj *Yeoman*, prostřednictvím kterého byl vytvořen nový generátor klientů *LSP*, nazvaný „***LSP Clients Generator***“.

Kapitola nejprve popisuje nezbytné přípravné kroky pro implementaci generátoru prostřednictvím nástroje *Yeoman*. Dále se věnuje vlastnostem vytvořeného generátoru, interní logice použité pro generování nových klientů a poslední podkapitola se zabývá ověřením funkčnosti generátoru a generovaných klientů pro různá vývojová prostředí.

6.1 Příprava pro vývoj

Pro vytvoření nového generátoru prostřednictvím nástroje *Yeoman* je nejprve zapotřebí připravit lokální prostředí pro vývoj a mít k dispozici:

1. terminál (příkazový řádek),
2. nainstalované běhové prostředí *Node.js*,
3. manažer balíčků *npm*,
4. verzovací nástroj *Git*
5. a libovolný editor, který podporuje jazyk *JavaScript* (např. *VS Code*).

Posledním krokem před zahájením vývoje nového generátoru je instalace nástroje příkazového řádku *Yeoman* umožňující vytváření projektů využívajících tzv. šablon.

```
npm install -g yo
```

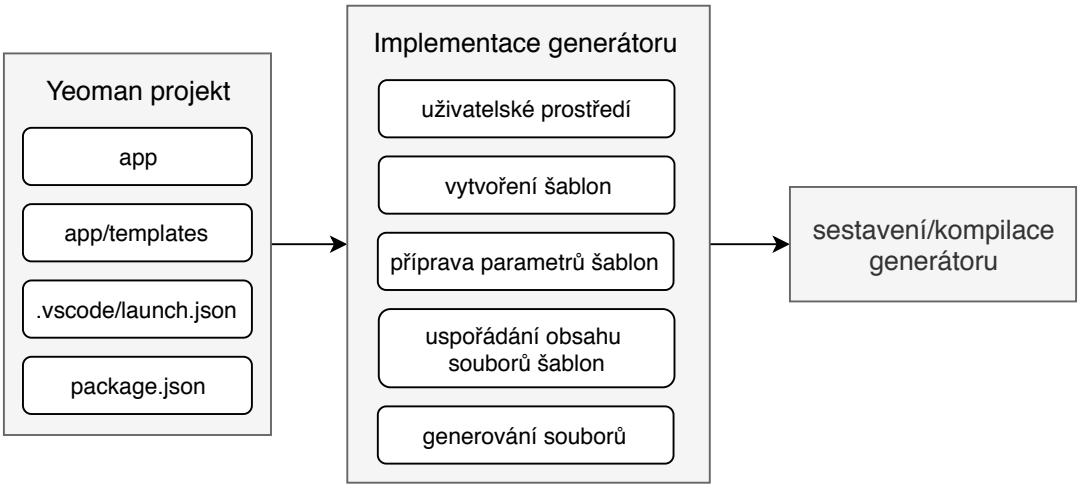
Výpis 6.1: Příkaz pro instalaci nástroje příkazového řádku *Yeoman*.

6.2 Postup implementace

Postup vytvoření generátoru klientů *LSP* lze rozdělit do tří částí diagramu, který je znázorněn na obrázku 6.1:

1. založení nového *Yeoman* projektu,
2. vytvoření nového generátoru,
3. sestavení/kompilace generátoru.

Prvním krokem vytvoření generátoru je založení nového projektu, ve kterém je nejprve zapotřebí manuálně připravit strukturu, která by se měla podobat struktuře *Yeoman* projektu viz obr. 6.2.



Obr. 6.1: Diagram vývoje generátoru klientů.

Připravený projekt slouží jako základ pro implementaci dalších částí generátoru klientů *LSP*, znázorněných v digramu na obrázku 6.1. Následující krok vývoje již popisuje implementaci jednotlivých částí generátoru a jeho vlastností.

Vlastnosti jednotlivých částí implementace projektu jsou více popsány v kapitolách „*6.3 Vlastnosti generátoru*“ a „*6.4 Interní logika generování klientů*“.

```
LICENSE
README.md
app
node_modules
package-lock.json
package.json
tests
```

Obr. 6.2: Základní struktura projektu *Yeoman*.

Posledním krokem je sestavení výsledného projektu (výpis 6.2), které zajistí aktualizaci všech závislostí generátoru a instalaci do nástroje příkazového řádku `yo`.

```
npm link
```

Výpis 6.2: Příkaz pro kompilaci vytvořeného generátoru.

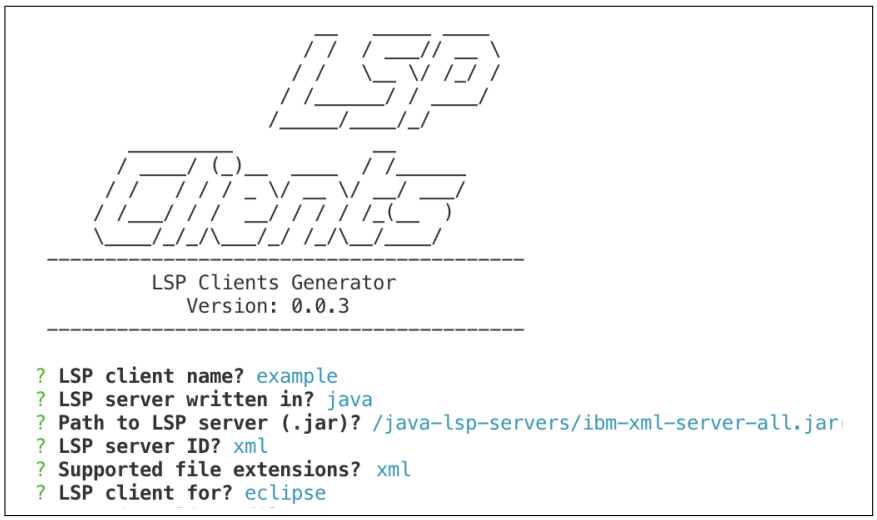
6.3 Vlastnosti generátoru

Uživatelské rozhraní

Generátor klientů *LSP* disponuje jednoduchým uživatelským rozhraním, které je prostřednictvím terminálu (příkazového řádku) plně interaktivní viz obr. 6.3. Hlavním způsobem interakce generátoru s uživatelem jsou výzvy (*Prompts*). Modul výzev je v nástroji *Yeoman* zprostředkován knihovnou *Inquirer.js* [43] – jedná se o kolekci běžných uživatelských rozhraní interaktivního příkazového řádku.

Uživatelské rozhraní generátoru využívá několik základních typů z široké nabídky dostupných výzev:

- vstupní pole (*Input*),
- seznam (*List*),
- zaškrtávací pole (*Checkbox*).



Obr. 6.3: Ukázka interaktivního uživatelského rozhraní.

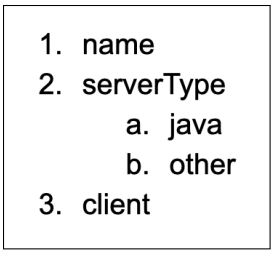
Pro urychlení procesu generování klientů a umožnění uživatelům použití generátoru klientů např. jako součást skriptu, generátor rovněž disponuje schopností spuštění jedním příkazem přímo v terminálu (příkazovém řádku), obr. 6.4.

```
yo lsp-clients \
    appname=MyClient \
    serverType=java \
    jarPath=${home}/lsp-server.jar \
    serverID=LANGUAGE_ID_APACHE_CAMEL \
    fileType=xml,java \
    client=eclipse
```

Obr. 6.4: Ukázka spuštění generátoru z příkazového řádku.

Vstupní argumenty

Prostřednictvím uživatelského rozhraní jsou od uživatele získána základní vstupní data, která jsou nezbytná pro vytvoření nového plnohodnotného klienta. Požadované parametry jsou rozděleny do dvou hlavních úrovní viz obr. 6.5. Skrze zmíněné úrovně se generátor dále rozhoduje na dalších potřebných informacích nutných k dokončení nového klienta.



Obr. 6.5: Základní úrovně vstupních argumentů generátoru.

Prvním argumentem je **`name`**. Jedná se o textový argument, který je automaticky předvyplněn na základě názvu adresáře, ve kterém je generátor klientů spuštěn (může být uživatelem libovolně upraven).

Další argument **`serverType`** určuje typ serveru, který má uživatel k dispozici a zná jeho další parametry. V této chvíli generátor nabízí na vstupu dvě možnosti – **`java`** a **`other`**. Typ serveru `java` znamená, že server, který bude vstupem pro generátor je implementován v jazyce *Java*. Možnost `other`, prozatím označuje všechny ostatní implementace serverů *LSP*. Při výběru varianty typu serveru `java`, je uživatel následně vyzván k zadání několika dalších rozšiřujících parametrů, které blíže popisují vstupující jazykový server *LSP*:

- **`jarPath`**,
- **`serverID`**,
- **`fileTypes`**.

Tyto tři parametry jsou základní obecné vlastnosti všech serverů *LSP* vytvořených v jazyce *Java* a každý uživatel je před použitím generátoru musí nejprve zjistit. Argument `jarPath` je textový parametr, určující cestu ke zkompilovanému balíčku serveru *LSP* ve formátu archivu `.jar`. Argument `serverID` slouží k identifikaci procesu spuštěného serveru a navázání komunikace s klientem, jedná se o textový identifikátor. Poslední argument `fileTypes` určuje typy souborů, pro které server implementuje rozšířenou jazykovou podporu.

Poslední argument **`client`** vybírá vývojové prostředí, pro které bude nový modul jazykového klienta vygenerován.

Šablony

Vytvořený generátor klientů *LSP* funguje na principu dostupných šablon, které jsou prostřednictvím další logiky a vstupních parametrů správně nakonfigurovány a následně vygenerovány do výsledného projektu.

V případě generátoru klientů určuje každá šablona jedno z podporovaných vývojových prostředí, pro které může být jazykový klient vytvořen. Momentálně patří mezi podporované klienty (tzn. implementované šablony) neboli vývojová prostředí – ***Eclipse***, ***Eclipse Che***, ***Theia*** a ***VS Code***. Šablony jsou rozděleny do dvou základních skupin. Skupiny jsou rozlišeny unikátní předponou v názvu viz obr 6.6:

1. `default`,
2. `java`.

```
default-che-client
default-eclipse-client
default-theia-client
default-vscode-client
java-che-client
java-eclipse-client
java-theia-client
java-vscode-client
```

Obr. 6.6: Seznam dostupných šablon generátoru klientů *LSP*.

První skupina šablon s předponou `default` je využívána jako výstupní pro všechny servery, které nejsou implementovány v jazyce *Java*. Jedná se pouze o základní obecnou šablonu, která obsahuje pouze soubor `README.md`, ve kterém jsou bližší informace o obecném vytvoření klienta pro požadované vývojové prostředí a další potřebné informace pro uživatele.

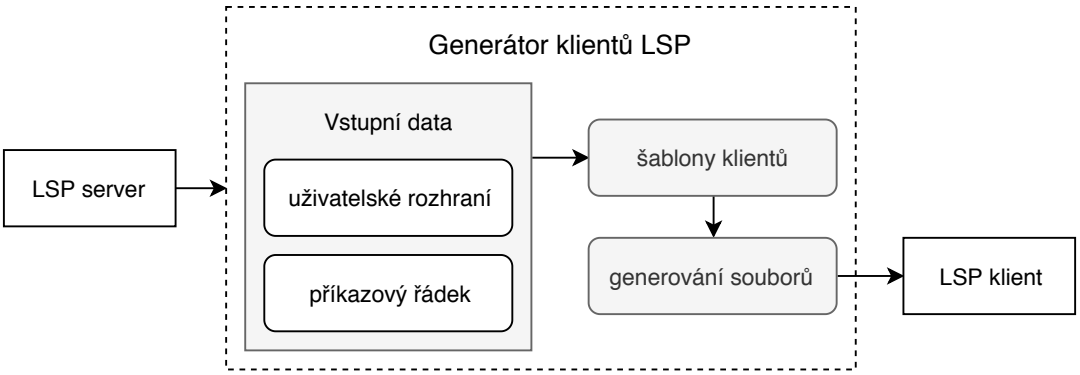
Šablony s předponou `java` jsou více komplexnější a jejich výstupem je plně funkční jazykový klient pro server implementovaný v jazyce *Java*, který je bez problémů instalovatelný do jednoho s podporovaných vývojových prostředí.

6.4 Interní logika generování klientů

Navržená interní logika generování klientů odpovídá schématu znázorněnému na obrázku 6.7. Princip spočívá v předpřipravení šablon klientů pro každé vývojové prostředí. Šablony jsou následně doplněny o informace získané od uživatele ze vstupu generátoru a nakonec vygenerování všech souborů do výsledného projektu nového klienta viz obr. 6.9.

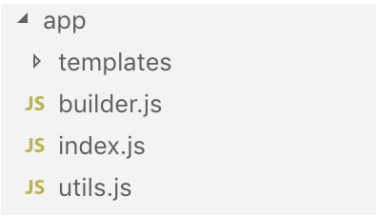
Parametry pro jednotlivé šablony jsou složeny z argumentů získaných z uživatelského rozhraní generátoru a parametrů, které se na základě těchto argumentů

vytvoří specificky pro každou šablonu zvlášť. Parametry jednotlivých šablon se liší z důvodů rozdílné implementace jazykových klientů pro různá vývojová prostředí. Liší se např. implementace zásuvných modulů, použité knihovny podporující *LSP*, programovací jazyk ve kterém je plug-in vytvořen atd.



Obr. 6.7: Blokové schéma vytvořeného generátoru klientů *LSP*.

Celá logika vytvořeného generátoru klientů je umístěna v adresáři **`app`**, který se dále dělí na adresář **`templates`** a tři soubory vytvořené v *Javascriptu* (obr. 6.8).



Obr. 6.8: Hlavní soubory vytvořeného generátoru klientů *LSP*.

Kořenovým souborem celého generátoru je **`index.js`**, obsahující kompletní řídící logiku. Jsou zde vytvářeny výzvy uživatelského rozhraní, získávány parametry šablon, generovány soubory atd.

Uspořádání parametrů do jednotlivých šablon je pro generátor řešeno prostřednictvím dílčích funkcí v souboru **`builder.js`**.

Soubor **`util.js`** obsahuje všechny zbylé pomocné funkce generátoru, pro vlastnosti, které nebylo možné pokrýt obecnými knihovnami jazyku *Javascript* (např. metody pro kontrolu vstupních parametrů výzev uživatelského rozhraní generátoru apod.).

Po úspěšném připravení všech parametrů jsou všechny soubory šablony doplněny o získané informace a je vygenerován výsledný projekt zásuvného modulu nového klienta *LSP* pro zvolené vývojové prostředí.

```
Generating client files...
Copying LSP server to client resources...
   create libs/lsp-server.jar
   create .classpath
   create .gitignore
   create .project
   create build.properties
   create libs/README.md
   create META-INF/MANIFEST.MF
   create OSGI-INF/l10n/bundle.properties
   create plugin.xml
   create pom.xml
   create README.md
   create src/org/vutbr/lsp/client/Activator.java
   create src/org/vutbr/lsp/client/LSPStreamConnectionProvider.java
   create src/org/vutbr/lsp/java/completion/JavaCompletionProposalComputer.java
   create src/org/vutbr/lsp/xml/completion/XMLCompletionProposalComputer.java
```

Obr. 6.9: Ukázka výstupu generátoru klientů *LSP* pro šablonu *Eclipse*.

Posledním krokem generování klienta je zkopírování souboru serveru *LSP* (archiv `.jar`), který byl uveden na vstupu generátoru, do dostupných zdrojů zásuvného modulu – adresář **`libs`**.

6.5 Ověření funkčnosti generátoru

Součástí vytvořeného generátoru klientů *LSP* jsou i jednotkové testy. Pro implementaci testů byl použit testovací framework jazyka *Javascript* – ***Mocha*** [44].

Testy slouží pro kontrolu struktury všech dostupných šablon generátoru. Ověřuje se správnost parametrů hlavních souborů dané šablony a zároveň dostupnost všech souborů, které má každá šablona obsahovat viz obr. 6.10.

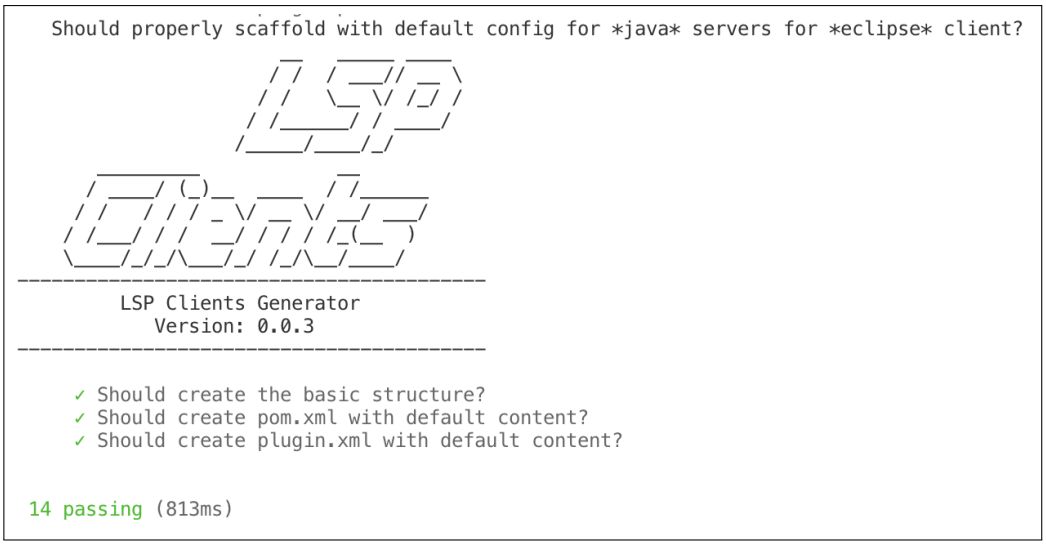
Pro spuštění jednotkových testů generátoru je nejprve zapotřebí nainstalovat modul testovacího frameworku *Mocha*. Testovací modul se nainstaluje příkazem **`npm install --global mocha`**. Testy se poté spustí příkazem **`npm test`** v kořenovém adresáři generátoru klientů *LSP*.

Kapitola dále popisuje ověření funkčnosti vygenerovaných klientů pro různé jazykové servery na vstupu generátoru a pro rozdílná vývojová prostředí na výstupu.

Prvním testovaným serverem je ***XML Server*** [45] od společnosti *IBM*, pro který byl prostřednictvím generátoru klientů *LSP* vygenerován nový jazykový klient do vývojového prostředí *VS Code*.

Další byl pro testování na vstup generátoru použit server ***Camel Language Server*** a výstupem byl zvolen jazykový klient pro vývojové prostředí *Eclipse*.

Oba servery použité pro ověření funkčnosti generátoru jsou implementované jazykem *Java* a proto je možné prostřednictvím generátoru vygenerovat plně funkční jazykové klienty a jednoduše ověřit zdali je ve vývojovém prostředí následně dostupná rozšířená jazyková podpora pro daný programovací jazyk.



Obr. 6.10: Ukázka jednoho výstupu z jednotkových testů generátoru klientů *LSP*.

Pro testování byli záměrně vybráni klienti pro vývojová prostředí *Eclipse* a *VS Code*, protože se jedná o známé implementace pro lokální instalaci, spuštění a demonstraci funkčnosti.

Ověření přítomnosti a funkčnosti jazykového klienta je demonstrováno na předpřipravených souborech a je využito vlastnosti kontroly a oznamování chyb jazykovým serverem pro daný programovací jazyk.

Generování a ověření klienta pro server *XML*

Prvním testovaným serverem *LSP* je *IBM XML Server*, který do vývojového prostředí přidává rozšířenou jazykovou podporu pro jazyk *XML*. Hlavní parametry potřebné pro vygenerování nového klienta pro server *XML* jsou (obr. 6.11):

- implementace serveru – **`java`**,
- balíček serveru *XML* – **`xml-lsp-server.jar`**,
- textový identifikátor serveru – **`xml`**,
- podporované soubory – **`xml`**,
- klient byl vybrán pro vývojové prostředí *VS Code* – **`vscode`**.

Jakmile je generování souborů dokončeno je zapotřebí nového klienta zkompilovat. Kompilace se provede příkazem **`npm link`** v kořenovém adresáři vytvořeného klienta.

Pro jednoduchou demonstraci funkčnosti nově získané rozšířené jazykové podpory pro jazyk *XML* byl připraven ukázkový soubor **`example.xml`**, který obsahuje syntaktické chyby. Pokud je soubor otevřen ve vývojovém prostředí *VS Code* bez přítomnosti vygenerovaného klienta, je soubor správně rozpoznán jako *XML*, ale chyby, které obsahuje, nejsou nijak detekovány viz obr. 6.12.

```
? LSP client name? xml-client-example
? LSP server written in? java
? Path to LSP server (.jar)? /java-lsp-servers/final/xml-lsp-server.jar
? LSP server ID? xml
? Supported file extensions? xml
? LSP client for? vscode
Generating client files...
Copying LSP server to client resources...
   create package.json
   create libs/lsp-server.jar
   create .project
   create .vscode/extensions.json
   create .vscode/launch.json
   create .vscode/settings.json
   create .vscode/tasks.json
   create gulpfile.js
   create language-configuration.json
   create libs/README.md
   create package-lock.json
   create README.md
   create src/extension.ts
   create tsconfig.json
   create tslint.json
```

Obr. 6.11: Generování jazykového klienta pro server *IBM XML*.

Snadný způsob ověření funkčnosti jazykového klienta je otevřít projekt ve vývojovém prostředí *VS Code* a následně spustit projekt jako nové rozšíření prostřednictvím dostupné konfigurace **`launch.js`**, kterou klient obsahuje (`Debug > Start Without Debugging`).

Bude otevřeno nové okno vývojového prostředí *VS Code*, které již bude mít nainstalováno rozšíření vygenerovaného klienta *LSP* pro jazyk *XML*.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <invoice xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="invoice.xsd">
4     <date>2017-11-30_INVALID</date>
5     <number>5235_INVALID</number>
6     <products>
7       <product description="laptop" price="700.00_INVALID"/>
8       <product price="40.00"/>
9       <product description="mouse" price="30.00" INVALID="" />
10    </products>
11    <payments>
12      <payment amount="770.00" method="credit_INVALID"/>
13    </payments>
14  </invoice>
```

Obr. 6.12: Ukázkový soubor *XML* bez kontroly chyb.

Nyní při otevření stejného ukázkového souboru bude prostřednictvím komunikace klienta a serveru provedena kontrola chyb a detekované problémy budou v editoru již správně zvýrazněny viz obr. 6.13.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<invoice xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="invoice.xsd">
  <date>2017-11-30_INVALID</date>
  <number>5235_INVALID</number>
  <products>
    <product description="laptop" price="700.00_INVALID"/>
    <product price="40.00"/>
    <product description="mouse" price="30.00" INVALID="" />
  </products>
  <payments>
    <payment amount="770.00" method="credit_INVALID"/>
  </payments>
</invoice>
```

Obr. 6.13: Ukázkový soubor *XML* s kontrolou chyb.

Generování a ověření klienta pro server *Camel Language*

Druhým testovaným serverem *LSP* je *Camel Language Server*, který do vývojového prostředí přidává rozšířenou jazykovou podporu pro jazyk *Apache Camel*. Hlavní parametry potřebné pro vygenerování nového klienta jsou (obr. 6.14):

- implementace serveru – **`java`**,
- balíček serveru *Camel Language Server* – **`camel-lsp-server.jar`**,
- textový identifikátor serveru – **`LANGUAGE_ID_APACHE_CAMEL`**,
- podporované soubory – **`xml, java`**,
- klient byl vybrán pro vývojové prostředí *Eclipse* – **`eclipse`**.

```
? LSP client name? camel-client-example
? LSP server written in? java
? Path to LSP server (.jar)? /java-lsp-servers/final/camel-lsp-server.jar
? LSP server ID? LANGUAGE_ID_APACHE_CAMEL
? Supported file extensions? xml, java
? LSP client for? eclipse
Generating client files...
Copying LSP server to client resources...
   create libs/lsp-server.jar
   create .classpath
   create .gitignore
   create .project
   create build.properties
   create libs/README.md
   create META-INF/MANIFEST.MF
   create OSGI-INF/l10n/bundle.properties
   create plugin.xml
   create pom.xml
   create README.md
   create src/org/vutbr/lsp/client/Activator.java
   create src/org/vutbr/lsp/client/LSPStreamConnectionProvider.java
   create src/org/vutbr/lsp/java/completion/JavaCompletionProposalComputer.java
   create src/org/vutbr/lsp/xml/completion/XMLCompletionProposalComputer.java
```

Obr. 6.14: Generování jazykového klienta pro server *Apache Camel*.

Jakmile je generování souborů dokončeno je zapotřebí nového klienta zkompilovat. Kompilace se provede příkazem **`mvn clean package`** v kořenovém adresáři vytvořeného klienta.

Pro jednoduchou demonstraci funkčnosti nově získané rozšířené jazykové podpory pro jazyk *Apache Camel* byl připraven ukázkový soubor **`camel-example.xml`**, který obsahuje syntaktické chyby.

Pokud je soubor otevřen ve vývojovém prostředí *Eclipse* bez přítomnosti vygenerovaného klienta, je soubor správně rozpoznán jako *XML*, ale chyby, které obsahuje, nejsou nijak detekovány viz obr. 6.15.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans
5
6      <camelContext id="cbr-example-context" xmlns="http://camel.apache.org/schema/spring">
7          <route id="cbr-route">
8              <from uri="timer:timerName?delay=1000rr" />
9              <to uri="direct:name" />
10         </route>
11     </camelContext>
12 </beans>
```

Obr. 6.15: Ukázkový soubor *Camel XML* bez kontroly chyb.

Snadný způsob ověření funkčnosti jazykového klienta je naimportovat projekt klienta do vývojového prostředí *Eclipse* (`File > Import... > General > Existing Projects into Workspace`) a následně projekt spustit jako novou aplikaci *Eclipse* (`Run > Run As > Eclipse Application`).

Bude otevřena nová instance vývojového prostředí *Eclipse*, které již bude obsahovat zásuvný modul vygenerovaného klienta *LSP* pro jazyk *Apache Camel*.

Nyní při otevření stejného ukázkového souboru bude prostřednictvím komunikace klienta a serveru provedena kontrola chyb a detekované problémy budou v editoru již správně zvýrazněny viz obr. 6.16.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans
5
6      <camelContext id="cbr-example-context" xmlns="http://camel.apache.org/schema/spring">
7          <route id="cbr-route">
8              <from uri="timer:timerName?delay=1000rr" />
9              <to uri="direct:name" />
10         </route>
11     </camelContext>
12 </beans>
```

Obr. 6.16: Ukázkový soubor *Camel XML* s kontrolou chyb.

7 Závěr

Cílem diplomové práce byl návrh nástroje pro ulehčení vytváření jazykových klientů *LSP* do různých vývojových prostředí pro rozdílné jazykové servery *LSP* a následná implementace navrženého nástroje postaveného nad moderními technologiemi a principy.

V rámci analýzy potřeb k řešení zadání diplomové práce bylo nejprve nutné se blíže seznámit s protokolem *Language Server Protocol* a jeho architekturou. Jedním z požadavků zadání práce byla možnost vytváření klientů pro různá vývojová prostředí, proto bylo dále zapotřebí nastudovat architektury a možnosti rozšiřování o nové zásuvné moduly různých vývojových prostředí. Blíže popsány byly v diplomové práci vývojová prostředí *Eclipse* a *VS Code*. Tato dvě vývojová prostředí byla vybrána, protože se jedná o často používané samostatné desktopové aplikace (*Standalone IDE's*). V návaznosti na požadavek generování různých klientů bylo nutné nastudovat implementaci klientů *LSP* pro každé vývojové prostředí, tzn. zejména dostupné knihovny pro podporu *LSP* v daném programovacím jazyce.

Základním předpokladem generátoru byla schopnost zpracovávat na vstupu libovolný server *LSP*, ale prvotním cílem nástroje byla zejména schopnost generovat klienty pro server *Apache Camel LSP*. Z toho důvodu je čtvrtá kapitola věnována projektu *Apache Camel* a popisu dostupné implementace jazykového serveru a jazykových klientů.

Pro splnění požadavku uživatelsky přívětivého generátoru, postaveného nad moderními technologiemi a principy, byl pro implementaci generátoru zvolen nástroj *Yeoman*, jehož popisu a vlastnostem byla věnována pátá kapitola.

Po úspěšné analýze potřeb k řešení zadání diplomové práce a získání všech potřebných znalostí byl navržen a vytvořen generátor klientů, pojmenovaný „*LSP Clients Generator*“, jehož hlavní vlastnosti a interní logika generování klientů byly detailně popsány v poslední kapitole práce. Součástí této kapitoly je i ověření funkčnosti generátoru. Prostřednictvím generátoru byli vygenerováni klienti pro dva rozdílné servery *LSP* pro dvě různá vývojová prostředí a byla ověřena komunikace klient-server, poskytující očekávanou rozšířenou jazykovou podporu.

Všechny zadané cíle této práce byly úspěšně splněny, ověřeny a výsledný projekt byl vytvořen tak, aby mohlo být nadále možné pracovat na jeho vývoji a rozšiřování o podporu dalších implementací jazykových serverů a většího množství dostupných vývojových prostředí.

Literatura

[1] Language Server Protocol. *Official page for Language Server Protocol* [online]. 2018 [cit. 2018-09-05]. Dostupné z: https://microsoft.github.io/language-server-protocol/

[2] JSON-RPC. *Wikipedia, The Free Encyclopedia* [online]. 6. Listopad 2018 [cit. 2018-12-10]. Dostupné z: https://en.wikipedia.org/wiki/JSON-RPC#Implementations

[3] *The application/json Media Type for JavaScript Object Notation (JSON)*. 2006, (RFC 4627). Dostupné také z: https://www.rfc-editor.org/rfc/rfc4627.txt

[4] JSON-RPC 2.0 Specification. *JSON-RPC* [online]. 4. 1. 2013 [cit. 2018-12-10]. Dostupné z: https://www.jsonrpc.org/specification

[5] Overview. *Language Server Protocol: Official page for Language Server Protocol* [online]. 2018 [cit. 2018-12-10]. Dostupné z: https://microsoft.github.io/language-server-protocol/overview

[6] LSP Specification. *Language Server Protocol: Official page for Language Server Protocol* [online]. [cit. 2018-12-10]. Dostupné z: https://microsoft.github.io/language-server-protocol/specification

[7] MOIR, Kim. Eclipse. *The Architecture of Open Source Applications* [online]. [cit. 2018-12-08]. Dostupné z: http://www.aosabook.org/en/eclipse.html

[8] Eclipse Platform: Platform architecture. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2018-12-05]. Dostupné z: http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Farch.htm&cp=2_0_1

[9] OSGi Architecture. *OSGi Alliance: The Dynamic Module System for Java* [online]. 1999 [cit. 2018-12-05]. Dostupné z: https://www.osgi.org/developer/architecture/

[10] Eclipse Platform: Plug-ins and bundles. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2018-12-05]. Dostupné z: http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fruntime_model_bundles.htm&cp=2_0_3_0_0

[11] Equinox. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2018-12-08]. Dostupné z: http://www.eclipse.org/equinox/

[12] *Eclipse, Android and Java training and support* [online]. 2016 [cit. 2018-12-08]. Dostupné z: http://www.vogella.com/

[13] JELÍNEK, Dominik. *Generátor kódu pro testovací rámec RedDeer: Instalace nových zásuvných modulů.* Brno, 2017, 55 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Petr Číka, Ph.D.

[14] Welcome to Eclipse Marketplace. *Eclipse Foundation* [online]. [cit. 2018-12-10]. Dostupné z: https://marketplace.eclipse.org/content/welcome-eclipse-marketplace

[15] Marketplace Client. *Eclipse Foundation* [online]. [cit. 2018-12-10]. Dostupné z: https://www.eclipse.org/mpc/

[16] Documentation for Visual Studio Code: Getting Started. *Visual Studio Code* [online]. [cit. 2019-03-08]. Dostupné z: https://code.visualstudio.com/docs

[17] Dokumentace Electronu: About Electron. *Electron* [online]. [cit. 2019-03-08]. Dostupné z: https://electronjs.org/docs/tutorial/about

[18] About Node.js. *Node.js* [online]. [cit. 2019-03-08]. Dostupné z: https://nodejs.org/en/about/

[19] Blink. *The Chromium Projects* [online]. [cit. 2019-03-08]. Dostupné z: http://www.chromium.org/blink

[20] *Monaco Editor* [online]. [cit. 2019-03-08]. Dostupné z: https://microsoft.github.io/monaco-editor/index.html

[21] Chromium. *The Chromium Projects* [online]. [cit. 2019-03-08]. Dostupné z: http://www.chromium.org/Home

[22] PATEL, Pavan. VS Code Architecture and Overview. *Community of Software and Data Developers* [online]. 16 Oct 2016 [cit. 2019-03-08]. Dostupné z: https://www.c-sharpcorner.com/article/vscode-architecture-and-overview/

[23] Language Server Extension Guide. *Visual Studio Code* [online]. [cit. 2019-03-08]. Dostupné z: https://code.visualstudio.com/api/language-extensions/language-server-extension-guide

[24] Yo Code - Extension and Customization Generator. *Npm* [online]. [cit. 2019-03-08]. Dostupné z: https://www.npmjs.com/package/generator-code

[25] Extension Marketplace. *Visual Studio Code* [online]. [cit. 2019-03-08]. Dostupné z: https://code.visualstudio.com/docs/editor/extension-gallery

[26] Eclipse LSP4J. *Eclipse Foundation* [online]. 2. Únor 2017 [cit. 2018-12-10]. Dostupné z: https://projects.eclipse.org/projects/technology.lsp4j

[27] Eclipse LSP4E. *Eclipse Foundation* [online]. 1. Únor 2017 [cit. 2018-12-10]. Dostupné z: https://projects.eclipse.org/projects/technology.lsp4e

[28] *VSCode Language Server - Node* [online]. [cit. 2019-03-08]. Dostupné z: https://github.com/Microsoft/vscode-languageserver-node/blob/master/README.md

[29] Apache Camel. *The Apache Software Foundation* [online]. [cit. 2018-12-10]. Dostupné z: http://camel.apache.org

[30] Enterprise Integration Patterns. *Apache Camel* [online]. [cit. 2018-12-10]. Dostupné z: http://camel.apache.org/enterprise-integration-patterns.html

[31] Spring Support. *Apache Camel* [online]. [cit. 2018-12-10]. Dostupné z: http://camel.apache.org/spring.html

[32] Using OSGi blueprint with Camel. *Apache Camel* [online]. [cit. 2018-12-10]. Dostupné z: http://camel.apache.org/using-osgi-blueprint-with-camel.html

[33] About the Scala DSL. *Apache Camel* [online]. [cit. 2018-12-10]. Dostupné z: http://camel.apache.org/scala-dsl.html

[34] Java DSL. *Apache Camel* [online]. [cit. 2018-12-13]. Dostupné z: http://camel.apache.org/java-dsl.html

[35] Apache ServiceMix. *The Apache Software Foundation* [online]. [cit. 2018-12-10]. Dostupné z: http://servicemix.apache.org

[36] Apache ActiveMQ. *The Apache Software Foundation* [online]. [cit. 2018-12-10]. Dostupné z: http://activemq.apache.org

[37] Apache CXF. *The Apache Software Foundation* [online]. [cit. 2018-12-10]. Dostupné z: http://cxf.apache.org

[38] Apache Karaf. *The Apache Software Foundation* [online]. [cit. 2018-12-10]. Dostupné z: http://karaf.apache.org

[39] Introduction to Apache Camel. *Baeldung* [online]. 2018 [cit. 2018-12-10]. Dostupné z: https://www.baeldung.com/apache-camel-intro

[40] Apache Camel LSP. *The Apache Camel LSP server implementation* [online]. 2018 [cit. 2018-09-05]. Dostupné z: https://github.com/camel-tooling/camel-language-server

[41] *Camel Tooling: Tools for Apache Camel* [online]. [cit. 2018-12-10]. Dostupné z: https://github.com/camel-tooling

[42] *Yeoman: The web's scaffolding tool for modern web apps* [online]. [cit. 2019-03-08]. Dostupné z: https://yeoman.io

[43] Inquirer.js. *Npm: Build amazing things* [online]. [cit. 2019-05-03]. Dostupné z: https://www.npmjs.com/package/inquirer

[44] *Mocha: Feature-rich JavaScript test framework* [online]. [cit. 2019-05-03]. Dostupné z: https://mochajs.org

[45] *XML Language Server* [online]. 4 Jun 2018 [cit. 2019-05-03]. Dostupné z: https://github.com/microclimate-devops/xml-language-server

Seznam symbolů, veličin a zkratek

API	Application Programming Interface
DSL	Domain-Specific Languages
FTP	File Transfer Protocol
IDE	Integrated Development Environment
JAR	Java ARchive
JDT	Java Development Tools
JSON	JavaScript Object Notation
LSP	Language Server Protocol
OS	Operating System
PDE	Plug-in Development Environment
PID	Process ID
RPC	Remote Procedure Call
SDK	Software Development Toolkit
URI	Uniform Resource Identifier
VS	Visual Studio
XML	eXtensible Markup Language

Seznam příloh

A Obsah přiloženého DVD

Přiložené DVD obsahuje elektronickou verzi diplomové práce, zdrojové soubory a soubory nezbytné k ověření funkčnosti generátoru. Hlavní dokument elektronické verze diplomové práce **dp-xjelin43.pdf** je umístěn v adresáři `dokumentace`. Zdrojové soubory jsou zabaleny do archivu a umístěny v adresáři `projekt`. Projekt lze po rozbalení naimportovat do vývojového prostředí *VS Code*. Poslední adresář `overeni-funkcnosti` obsahuje soubory sloužící pro ověření funkčnosti vygenerovaného klienta prostřednictvím generátoru klientů *LSP*. Soubor `README.txt` který je umístěn v kořenovém adresáři obsahuje dodatečné informace k projektu a prostředí ve kterém byl projekt vytvořen.

- `/` ..... kořenový adresář přiloženého DVD.
  - `dokumentace`
    - `dp-xjelin43.pdf` ..... elektronická verze diplomové práce.
  - `projekt`
    - `generator-lsp-clients.zip` ..... zdrojové soubory generátoru klientů.
    - `README.txt` ..... popis obsahu balíčku zdrojových souborů.
  - `overeni-funkcnosti`
    - `example.xml`. ..... ukázkový soubor pro server *XML LSP*.
    - `example.xsd`
    - `xml-lsp-server.jar`. ..... jazykový server *XML*.
    - `navod.pdf`. ..... kroky pro ověření funkčnosti generátoru klientů *LSP*.
  - `README.txt` ..... soubor s popisem obsahu DVD.