Implementacja protokołu LSP dla wybranego środowiska zintegrowanego

(Implementation of a LSP protocol
for chosen IDE)

Wiktor Adamski

Praca inżynierska

Promotor: dr Wiktor Zychla

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

30 stycznia 2018 r.

Wiktor Adamski

.....

.....
(adres zameldowania)

.....

.....
(adres korespondencyjny)

PESEL: .....

e-mail: .....

Wydział Matematyki i Informatyki
stacjonarne studia I stopnia
kierunek: informatyka
nr albumu: 272220

Oświadczenie
o autorskim wykonaniu pracy dyplomowej

Niniejszym oświadczam, że złożoną do oceny pracę zatytułowaną *Implementacja protokołu LSP dla wybranego środowiska zintegrowanego* wykonałem/am samodzielnie pod kierunkiem promotora, dr Wiktora Zychli. Oświadczam, że powyższe dane są zgodne ze stanem faktycznym i znane mi są przepisy ustawy z dn. 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2006 r. nr 90, poz. 637, z późniejszymi zmianami) oraz że treść pracy dyplomowej przedstawionej do obrony, zawarta na przekazanym nośniku elektronicznym, jest identyczna z jej wersją drukowaną.

Wrocław, 30 stycznia 2018 r.

(czytelny podpis)

Streszczenie

Visual Studio Code to na pierwszy rzut oka prosty edytor kodu, jednakże system rozszerzeń pozwala rozbudować go do pełnoprawnego środowiska programistycznego. Ponieważ istnieje wiele edytorów, a każdy z nich posiadał swój interfejs programistyczny, powstał protokół LSP, który umożliwia napisanie logiki wspomagającej pisanie programu raz i użycie jej w wielu edytorach.

Visual Studio Code at first seems like a simple code editor, though its extension system allows to expand it into full-featured IDE. As there are many editors, and each of them has its own application interface, a LSP protocol was created, to allow writing helper logic once and use it in many editors.

Spis treści

Rozdział 1.

Preliminaria

Aby zrozumieć jak działa dostarczony serwer, należy wpierw zrozumieć architekturę rozszerzenia w systemie edytora Visual Studio Code, a także na czym polega omawiany protokół. W tym rozdziale poruszone zostaną:

- Struktura wtyczki rozszerzającej działanie edytora.
- Opis protokołu LSP.
- Visual Studio a Visual Studio Code.
- Parser Luaparse.

1.1. Struktura rozszerzenia Visual Studio Code

Sercem każdego rozszerzenia jest plik `package.json`, który przechowuje informacje na temat autora pakietu, warunki jego uruchomienia, a także wszystkie jego zależności:

```
{
    "name": "lua-lang",
    "description": "Lua language support",
    "author": "Wiktor Adamski",
    "license": "MIT",
    "version": "0.0.1",
    "engines": {
        "vscode": "^1.16.0"
    },
    "categories": [
        "Languages"
    ],
    "activationEvents": [
```

```
            "onLanguage:lua"
        ],
        "main": "./out/src/extension",
        "contributes": {
            "languages": [
                {
                    "id": "lua",
                    "aliases": [
                        "Lua",
                        "lua"
                    ],
                    "extensions": [
                        ".lua",
                        ".p8",
                        ".rockspec"
                    ],
                    "configuration":
                       "./language-configuration.json"
                }
            ],
        },
        "dependencies": {
            "vscode": "^1.1.5",
            "vscode-languageclient": "^3.4.2"
        }
    }
```

O ile serwer LSP może być napisany w dowolnym języku, część bezpośrednio łącząca się z edytorem (aktualna wtyczka) musi być w języku JavaScript dla środowiska uruchomieniowego node.js.

Duża część kodu który jest wspólny dla wszystkich rozszerzeń jest możliwa do automatycznego stworzenia przez generator kodu Yeoman. Proste polecenie `yo code` przeprowadzi nas przez kreator wtyczek i utworzy dodatkowe pliki konfiguracyjne, które pozwolą korzystać z edytora VS Code jako środowiska developerskiego.

1.2. Protokół Language Server Protocol

Protokół LSP [1] jest specjalizacją protokołu JSON-RPC, który przesyła dane między stronami komunikacji za pomocą obiektów JSON. Klient (edytor kodu) wysyła zapytania do serwera (program wspomagający) odpowiadające różnym akcjom podejmowanym przez programistę, np. zapytanie się o miejsce deklaracji danej zmiennej. Pierwszą wiadomością w trakcie połączenia jest wymiana możliwości za-

równo klienta (np. czy edytor wspiera przemianowanie zmiennej), jak i serwera (np. wskazanie definicji danego symbolu lub automatyczne uzupełnianie pisanego tekstu). Poniżej przedstawiam przykładowe zapytanie klienta:

```
{
    "jsonrpc": "2.0",
    "id": 1,
    "method": "textDocument/didOpen",
    "params": {
        ...
    }
}
```

1.3. Visual Studio (Code)

Wiele osób nie rozróżnia od siebie dwóch produktów Microsoftu. Visual Studio to zintegrowane środowisko programistyczne, nastawione głównie na pisanie programów w języku C#. Visual Studio Code jest natomiast otwartoźródłowym edytorem kodu opartym na silniku renderującym Electron od firmy Github, co sprawia, że jest on dosyć podobny do edytora Atom (również pod względem metodologii wtyczek).

1.4. Biblioteka Luaparse i drzewa rozbioru

Aby dostarczać jakiekolwiek sensowne informacje na temat kodu, potrzebne jest jego sparsowanie. Zajmuje się tym biblioteka Luaparse [2], która produkuje abstrakcyjne drzewa rozbioru programów napisanych w języku Lua. Drzewa reprezentowane za pomocą obiektów JavaScript są inspirowane na specyfikacji Mozilla Parser API. Przykładowo wyrażenie:

```
foo = "bar"
```

zostanie przełożone na drzewo:

```
{
    "type":"Chunk",
    "body":[{
        "type":"AssignmentStatement",
        "variables":[{
            "type":"Identifier",
            "name":"foo",
            "loc":{
                "start":{
                    "line":1,
```

```
                    "column":0
                },
                "end":{
                    "line":1,
                    "column":3
                }
            }
        }],
        "init":[{
            "type":"StringLiteral",
            "value":"bar",
            "raw":"\"bar\"",
            "loc":{
                "start":{
                    "line":1,
                    "column":6
                },
                "end":{
                    "line":1,
                    "column":11
                }
            }
        }],
        "loc":{
            "start":{
                "line":1,
                "column":0
            },
            "end":{
                "line":1,
                "column":11
            }
        }
    }],
    "loc":{
        "start":{
            "line":1,
            "column":0
        },
        "end":{
            "line":1,
            "column":11
        }
```

```
        },
        "comments":[]
    }
```

Rozdział 2.

Proces odpowiedzi na zapytania

Dysponując parserem kodu Lua, wystarczy odpowiednio przechodzić generowane przez niego drzewa, w celu odpowiedzi na poszczególne zapytania klienta. Punktem wejścia dla projektu będącego częścią niniejszej pracy jest artykuł [3] opisujący utworzenie prostego serwera LSP.

2.1. Zapytanie `Initialize`

Bibliografia

[1] Microsoft Language Server Protocol documentation, 2018.

[2] Oskar Schöldström Luaparse, 2013.

[3] Microsoft Creating Language Servers for Visual Studio Code 2018.

Dodatek A

Instrukcja uruchomienia rozszerzenia

Niniejszy dodatek opisuje instrukcję uruchomienia rozszerzenia w programie Visual Studio Code. Wymagana jest dodatkowo instalacja środowiska Node.js, które musi być dostępne z wiersza poleceń. Kroki do wykonania:

1. Nacisnąć kombinację klawiszy `Ctrl+‘`, otwierając tym samym wbudowane okno terminala.
2. Wykonać polecenie `npm install`, które zainstaluje brakujące pakiety zależne.
3. W sekcji **Debug** wybrać konfigurację **Launch Client** i nacisnąć przycisk zielonej strzałki (ewentualnie nacisnąć klawisz `F5`).
4. Zostanie otwarta druga instancja edytora, w której rzoszerzenie jest aktywne.