generation plugin from the Overture Tool and packaging it as a JAR executable. This JAR can then be run from VS Code using a command containing a desired output folder, a specific VDM dialect and other settings. Currently, this JAR file is located on the client side of VDM VSCode, but in the future, it will be moved to the VDMJ Language Server to fulfill the goal of decoupling the specification language functions from the VS Code extension. The UML translation JAR files will also be located on the client side for now.

2.6 Language Server Protocol

> *"The idea behind a Language Server is to provide the language-specific smarts inside a server that can communicate with development tooling over a protocol that enables inter-process communication.*
>
> *The idea behind the Language Server Protocol (LSP) is to standardize the protocol for how tools and servers communicate, so a single Language Server can be re-used in multiple development tools, and tools can support languages with minimal effort."* [12]

This is what Microsoft writes states the essence of the Language Server and LSP. Using traditional methods, developers would have to provide language support and functionality for each individual IDE resulting in a lot of repeated work. However, using LSP, all the language neutral information — such as data types, code complection etc. — can stay on the server and the client API can request this information through the LSP, thus getting rid of the need to do such implementations for every code editor.

Specification LSP (SLSP)

An extension of the LSP has to be made to support specification language specific features. An SLSP has been developed to support Proof Obligation Generation, Combinatorial Testing and translation features [15].

A Debug Adapter Protocol (DAP) is also an important counterpart to the LSP, however as we will not be needing debugging functionality for our implementation, we will provide no further description in this thesis.

For the core of this project, the Language Server will simply be for storing the executable JAR file that contain the UML transformation functions. However, it may also be possible to have the transformation be part of the Language Server. If the transformation were to happen through some third party extension, such as PlantUML (see chapter 3), the parsing of either VDM or UML to a language the third party extension can read could happen on the Language Server. Though it might be smarter to have it on the client side of VDM VSCode if the extension for visualisation is too dependent on the VS Code platform specifically.