

## 0.1 Auswahl der verwendeten Technologien

Ein zentraler Teil einer Architektur ist die Auswahl der verwendeten Technologien. Diese Technologien sollen die Lösung der Aufgaben einer Software vereinfachen.

Im DMF müssen folgende Aufgaben gelöst werden:

1. Modelldatei Parsen und AST generieren
2. AST auslesen und verarbeiten
3. Kommunikation mit verschiedenen IDE's
4. Generieren von Codedateien in verschiedenen Sprachen
5. Integration mit verschiedenen Build Tools

### 0.1.1 Parser

Der Parser für das DMF muss große Dateien wiederholt mit kleinen Änderungen parsen. Diese Anforderung stammt aus der Notwendigkeit des AST's um Syntaktische und Semantische Fehler, sowie die verschiedenen Tokens(siehe Abschnitt LSP) nach jeder Eingabe an die IDE zu übermitteln. Hierbei ist Latenz die höchste Priorität, denn die Reaktionsfähigkeit der IDE beeinflusst die Geschwindigkeit mit der Entwickelt werden kann.

Zusätzlich muss der Parser auch von jeder anderen Komponente des DMFs verwendet werden. Deshalb ist hier die Einschränkung der anderen Technologien unerwünscht.

### XText

XText ist ein Framework der Eclipse Foundation.

Es bietet die Möglichkeit eine DSL mit verschiedenen Modellen zu modellieren und Regeln automatisch zu überprüfen. XText setzt auf Modellierung vieler Bestandteile und generiert andere Komponenten komplett. Dies ermöglicht eine schnelle Entwicklung wenn die Anforderungen perfekt zu XText passen. XText schränkt stark ein, wo Anpassungen vorgenommen werden können. So ist es nicht vorgesehen die LSP-Server Implementierung anzupassen, obwohl XText nicht alle Features des LSP-Protokolls unterstützt. Dateigeneration und die Verarbeitung des AST's müssen auch mit dem Java-Interfaces von XText vorgenommen werden. Dies setzt immer die Verwendung von JVM basierten Sprachen voraus. Jede JVM-Implementierung benötigt beachtliche Zeit zum Starten weshalb Code Generation immer auf den Start Warten muss.

Abschließend waren an XText die nicht funktionierenden Beispiel Projekte sowie zwingende Entwicklung in Eclipse sehr abweisend. Ein Framework welche eine einfache und flexible Entwicklung ermöglichen soll, sollte nicht schwer und nur in einer IDE zu entwickeln sein.

## Treesitter

Treesitter ist ein Open Source Framework zur Generierung von Parsern. Dabei wird die Grammatik mithilfe einer Javascript API definiert. Mithilfe der Treesitter CLI wird aus der Javascript Datei der Parser generiert. Der generierte Parser nutzt C. C eignet sich hier sehr gut, da es die höchste Performance und die Möglichkeit es in jeder anderen Sprache zu nutzen bietet. Das Nutzen von C ist für jede Sprache eine Voraussetzung, um mit dem Betriebssystem zu kommunizieren. C's größter Nachteil, die manuelle Speicher Verwaltung, wird durch die Generation des Parsers gelöst. Die bereitgestellten Schnittstellen übergeben Strukturen welche vom Aufrufer verwaltet werden.

**Iteratives Parsen** Ein großes Unterscheidungsmerkmal von Treesitter ist die Möglichkeit iterativ zu parsen.

*With intelligent [node] reuse, changes match the user's intuition; the size of the development record is decreased; and the performance of further analyses (such as semantics) improves.[?]*

Beim iterativen Parsen ist das Ziel den AST nicht bei jedem Parse Durchlauf neu zu erstellen, sondern möglichst viel des AST's wiederzuverwenden. Für das Iterative Parsen muss der AST sowie die bearbeiteten Textstellen an Treesitter übergeben werden. Die Durchlaufszeit des iterativen Parsedurchlaufs hängt nicht mehr der Länge der kompletten Modelldatei ab, sondern nur von den neuen Terminalsymbolen und Modifikationen im AST:

*Our incremental parsing algorithm runs in  $O(t + \text{slg}N)$  time for  $t$  new terminal symbols and  $s$  modification sites in a tree containing  $N$  nodes [?]*

## ANTLR

ANTLR ist sehr ähnlich zu Treesitter. Die größten Unterschiede sind die API's zum Schreiben der Grammatiken und die Möglichkeit iterativ zu Parsen. Zusätzlich unterstützt ANTRL nur Java, C# und C++. Dies zwingt einen in der Wahl der Implementierungssprache ein.

## Auswahl Parser

Für das DMF Framework wurde Treesitter verwendet. Die Exellente Performance sowie die Flexibilität bei der Implementierung der restlichen Komponenten hoben Treesitter von den restlichen Technologien ab.

### 0.1.2 AST Verarbeiten

Bei der Verarbeitung des AST's müssen verschiedene Regeln abgearbeitet werden und der Inhalt des AST's in einem Modell vorbereitet werden. Essentiell für die Verarbeitung ist die Zusammenarbeit mit den folgenden Komponenten.

Die Auswahl der Technologie für diesen Schritt basiert auf der Auswahl für die folgenden Schritte.

### 0.1.3 Kommunikation mit verschiedenen IDE's

Damit ein Framework die Entwicklung nicht einschränkt muss es in verschiedenen "Integrated Development Environments" (IDE's) genutzt werden können. Viele IDE's stellen Schnittstellen für Plugins bereit. Dazu zählen IntelliJ, Eclipse, NeoVim und VSCode. Jede Schnittstellen ist jedoch unterschiedlich, wodurch die Entwicklung von vielen Verschiedenen Plugins nötig wäre.

**Language Server Protokoll** Eine einfachere Möglichkeit bietet das "Language Server Protokoll" (LSP). Dieses Protokoll bietet die Möglichkeit, dass viele verschiedene IDE's eine Server Implementierung nutzen. Im Fall von Zed und Eclipse lassen sich LSP-Server sogar ohne jegliche Plugins einbinden. Wobei hier auf die schlechte Unterstützung des LSP-Protokolls in Eclipse hingewiesen werden muss. IntelliJ und NeoVim nutzen Plugins, um LSP-Server anzubinden. VSCode bietet eine API und einen einfachen LSP-Client in ein kleines Plugin zu implementieren. Im LSP-Server können gebündelt Logik und Protokoll implementiert werden.

LSP wird hauptsächlich über die Standard-Eingabe und -Ausgabe oder über einen Server Socket transportiert. Es wird ein JSON-RPC Format genutzt. Der LSP-Server muss somit JSON, Std-In und Std-Out, sowie Server Sockets unterstützen.

#### Typescript

Von der VSCode Dokumentation wird die Implementierung eines LSP-Servers in Typescript empfohlen. Dafür werden Bibliotheken bereit gestellt. Typescript eignet sich gut, für die JSON Parsing und für die Verwendung von Server Sockets. Probleme entstehen bei Typescript bei den Themen Performance, Anbindung an den Parser und bei der Fehlerbehandlung.

#### Golang

Golang ist eine Sprache welche für die Entwicklung von Backends ausgelegt wurde.

Es werden die Anforderungen für JSON-Parsing, Std-IO und Server Sockets erfüllt, durch die große Standard Bibliothek erfüllt. Es gibt keine Bibliothek welche das komplette Protokoll beinhaltet. Dieses kann jedoch durch die Unterstützung von LLMs schnell generiert werden.

Golang bietet zusätzlich eine simple Anbindung an den Parser und die Möglichkeit sehr einfach Parallelität einzubauen. Besonders erwähnenswert ist die Geschwindigkeit eines Golang Programmes und die Startgeschwindigkeit ohne auf Speichersicherheit zu verzichten.

## Java

Java bietet eine Bibliothek mit "lsp4j" eine Bibliothek zur einfachen Entwicklung. Bei der Einbindung des Parsers gestallten sich jedoch zusätzliche Herausforderungen da der Java Code Plattform unabhängig kompiliert wird und Plattform abhängigen Code aufrufen muss. Java benötigt für die Ausführung eine installierte Instanz der JRE. Die JRE muss nicht nur zusätzlich zum LSP-Server verwaltet werden, sondern benötigt zusätzlich Zeit zum Starten. So muss der Entwickler länger warten bis seine Entwicklungsumgebung bereit steht.

### 0.1.4 Generation von Code Dateien in verschiedenen Sprachen

Ziel des DMFs ist es große Mengen an Sourcecode zu generieren. Dabei soll den Entwicklern die Wahl zwischen mehreren Zielsprachen gegeben werden. Diese Generation wird beim Build und damit sehr häufig ausgeführt. Eine langsame Generation wird jeder Organisation viel Geld kosten.

Die Generation muss somit schnell und Zielsprachen unabhängig sein. Sie muss auch aus von den Build Tools gestartet werden.

## Golang Templates

Golang Standardbibliothek bietet die Möglichkeit Templates zu definieren. Diese Templates werden hauptsächlich für die Generierung von HTML genutzt. Da sie Golang die Templates nicht nur für HTML sondern auch für generelle Texte anbietet, können diese auch für jede Zielsprache genutzt werden.

Die Anforderungen an einen Webserver (Geschwindigkeit, Ressourcen schonend, simpel) komplementieren die Anforderungen an einen Codegenerator sehr gut. Golang Templates stechen besonders für ihre Integration in IDE's wie z.B. in IntelliJ heraus.

## Java

Es gibt mehrere Template Engines für Java. Einige Beispiele wäre FreeMaker oder Apache Velocity. Beide sind gut unterstützt und bieten alle nötigen Features für die Generierung von Code.

## Typescript

Für Typescript gibt es viele Template Engines. Zu den bekannten gehören Eta, liquidjs und squirrelly. Sie bieten alle die Möglichkeit verschiedene Zielsprachen zu generieren und können mit nodejs ausgeführt werden.

## Auswahl

Da Golang eine exzellente Unterstützung in IntelliJ hat und keine zusätzliche Installation wie NodeJs oder JRE verwalten muss, fiel meine Wahl auf Golang. Mit der Wahl für Golang für den Generator, ist auch die Wahl für die Verarbeitung des AST's und für den LSP-Server.

### 0.1.5 Integration mit verschiedenen Build Tools

Damit eine Generation während des Buildvorgangs ist essentiell, um sicherzustellen das der generierte Code aktuell ist. Damit der Neugeneration werden auch alle eventuelle Anpassungen in den Dateien überschrieben, wodurch Fehler vermieden werden.

#### **Maven**

Maven ist ein sehr verbreitetes Build Tool für Java. Maven unterstützt Plugins welche während des Builds ausgeführt werden und in der Maven Konfiguration konfiguriert werden können. Die API für Maven Plugins ist in Java geschrieben. Dieses Plugin muss den Generator aufrufen. Dies ist möglich indem die Datei des Generators ausgeführt wird.

#### **NPM**

NPM ist das führende Build Tool für Typescript Projekte. NPM unterstützt die Ausführung von Terminal Befehlen. Der Generator kann somit über das Terminal ausgeführt werden.