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 verschieden 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 vorraus. 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 seht 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 in tuition; 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 + slgN) 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 Grammtiken und die Möglichkeit iterativ zu Parsen.
Die Möglichkeit iterativ zu Parsen kann wichtige Zeit sparen, indem nicht die komplette Datei geparsed werden muss. Zusätzlich unterstützt ANTRL nur Java, C# und C++. Dies zwingt einen in der Wahl der Implementierungssprache ein. Da meine bevorzugte Wahl Golang ist, war dies ein zusätzlicher Grund ANTLR nicht zu benutzen.