Erweiterungen und Änderungen an D°

Versionen

Datum	Notiz	Version
31.12.2020	Erste Veröffentlichung	1.0

Autoren

Name	Institution	Email
Fabian Bruckner	Fraunhofer ISST	fabian.bruckner@isst.fraunhofer.de

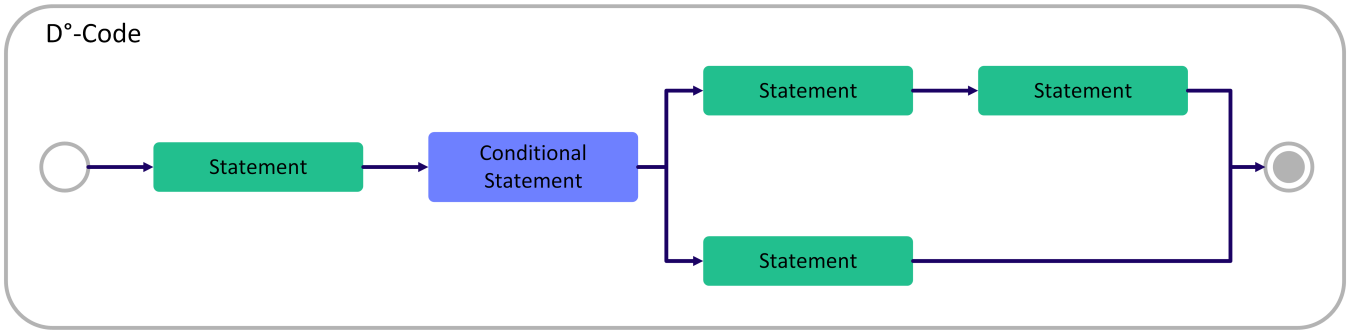
Im Rahmen der Entwicklung und Verwendung von D° sind diverse Aspekte identifiziert worden, welche:

- Erweiterungen am Funktionsumfang von D° erforderlich machen, um D° in gegebenen Szenarien einsetzen zu können.
- Modifikationen oder Erweiterungen an D° nahelegen, um die Komplexität zu reduzieren und somit Benutzbarkeit zu verbessern.

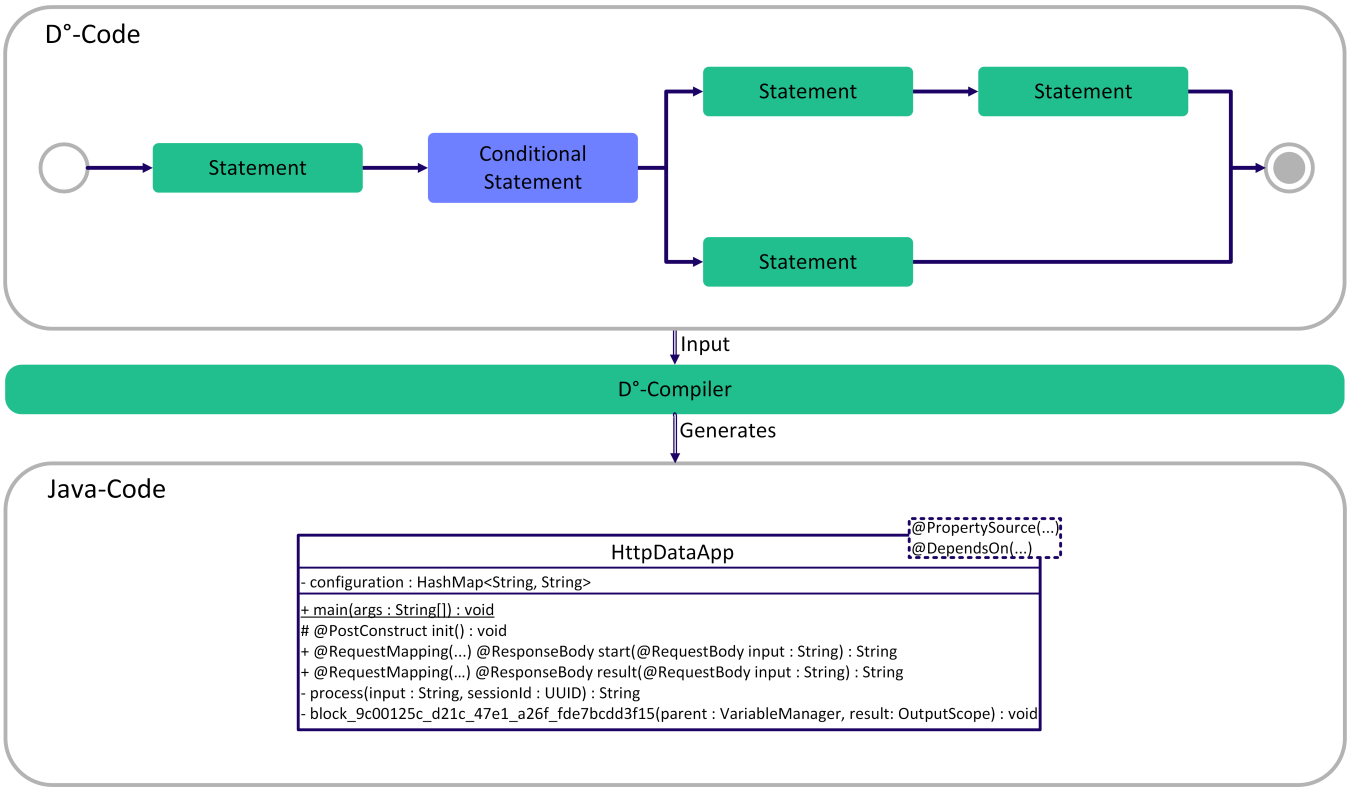
Einige dieser Aspekte wurden im Rahmen des Projekts T3.6 adressiert und sind in diesem Dokument entsprechend dokumentiert.

Metadaten für Spracherweiterungen

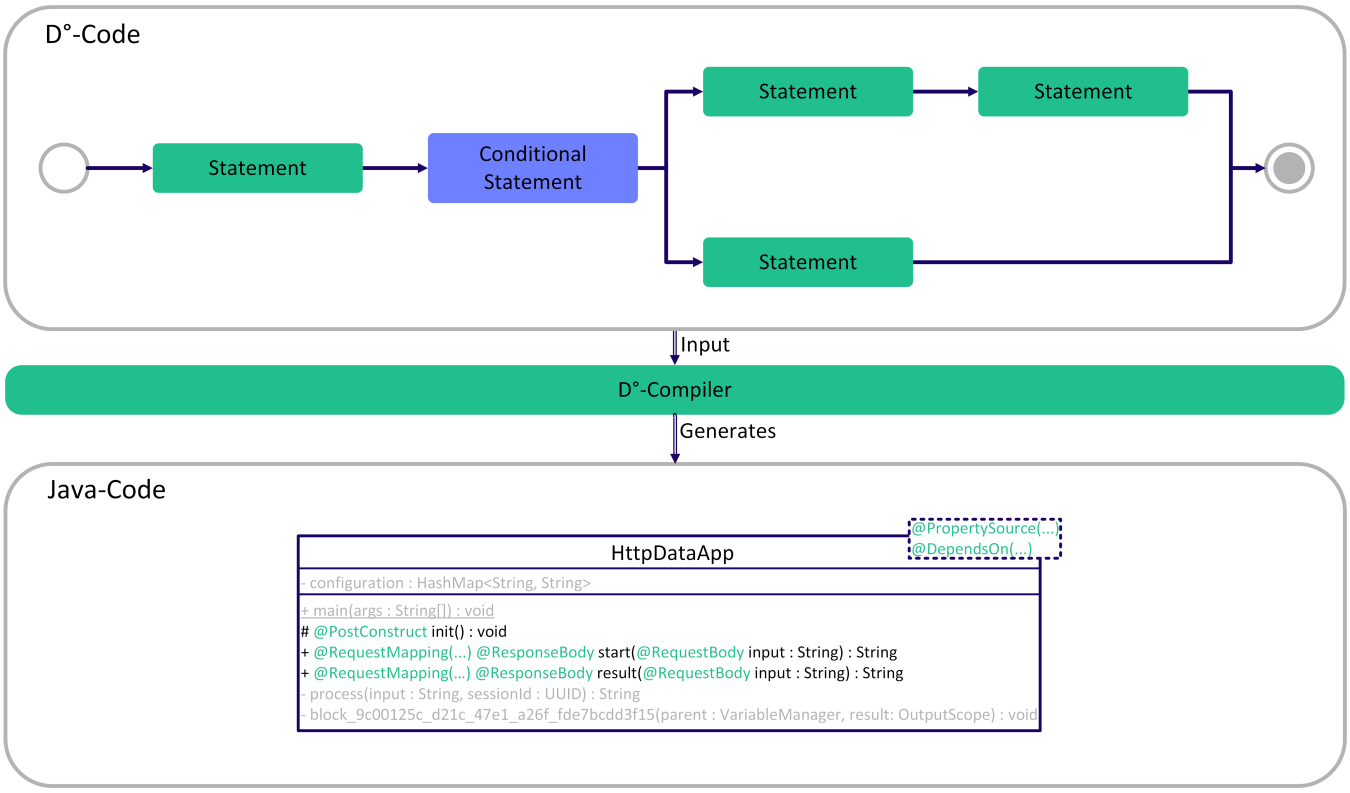
D° verfügt im Auslieferungszustand über keinerlei Datentypen, Policies oder Aktivitäten, welche verwendet werden können um Applikationen zu erstellen. Diese werden dem Compiler über Erweiterungen zur Verfügung gestellt und in die erzeugen Applikationen integriert. Die Elemente, welche in den Erweiterungen enthalten sind, können verwendet werden um bspw. die Applikation, welche in der nachfolgenden Abbildung dargestellt ist, zu erstellen.



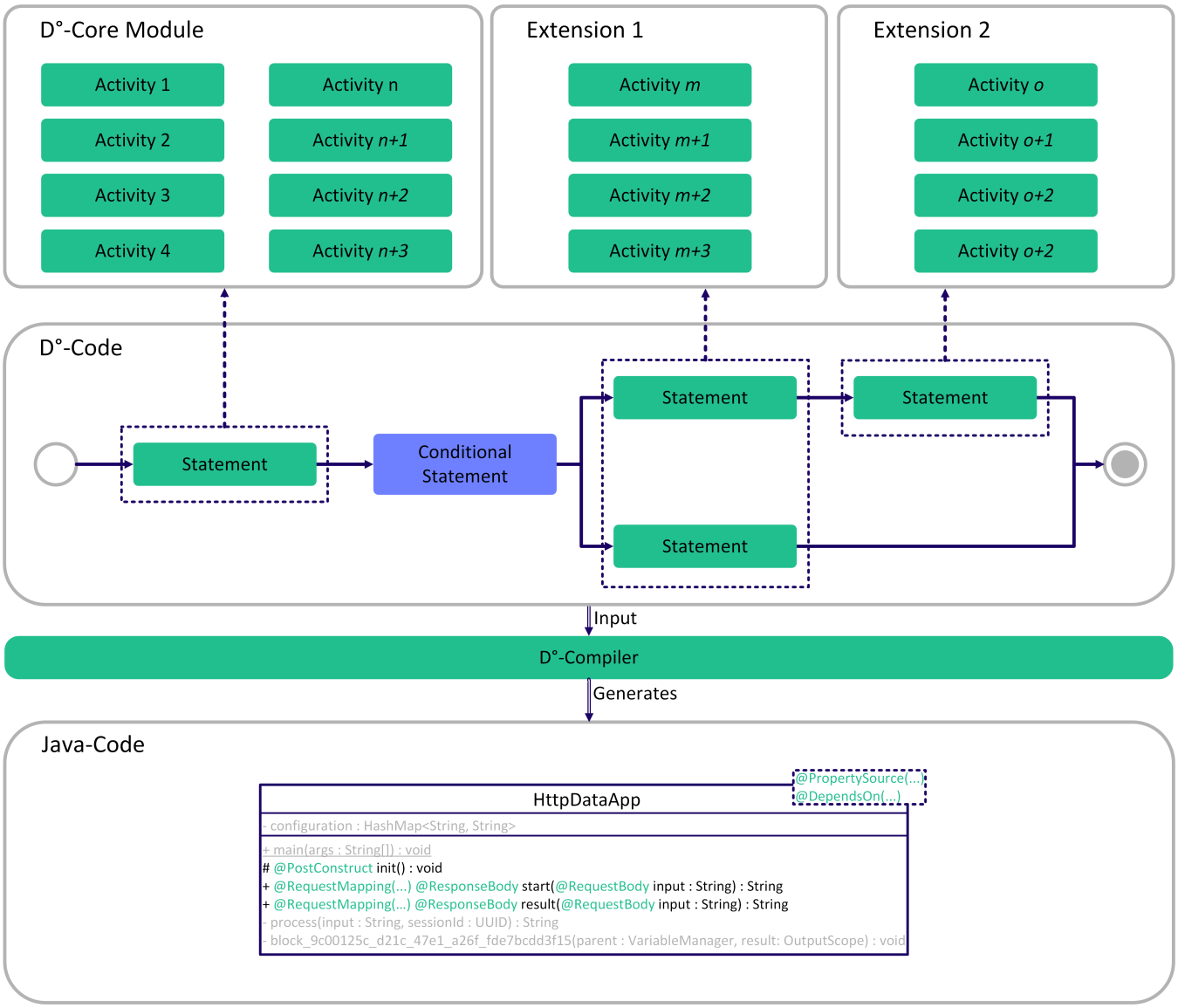
Diese D°-Applikation ist die Eingabe für den D°-Compiler, welcher im Anschluss daran eine Applikation in der verwendeten Hostsprache generiert. Für unsere Implementation ist die verwendete Hostsprache Java.



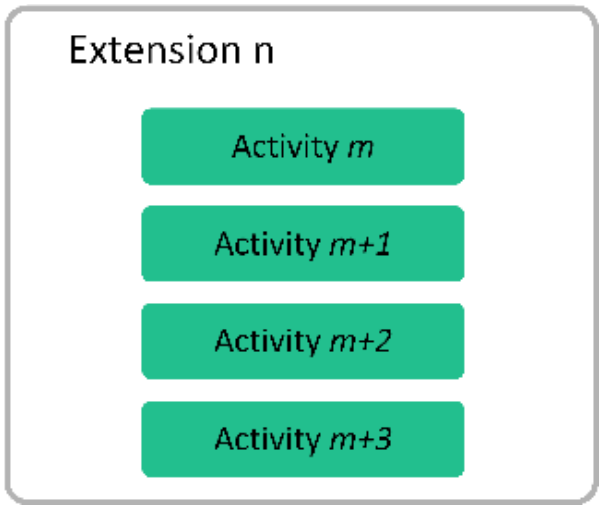
Dabei besitzt die generierte Applikation einen eigenen Technologie-Stack, was den generierten Code vereinfacht und gleichzeitig die gesamten Funktionalitäten der gewählten Technologie für D°-Applikationen verwendbar macht. Im Falle unserer Implementation verwenden alle generierten Applikationen das Spring Framework als Grundlage. Zur Veranschaulichung wurden in der nachfolgenden Abbildung die verwendeten Spring-Annotationen hervorgehoben.



Bezieht man die zuvor erwähnten Erweiterungen für D° mit ein, zeigt sich, dass die einzelnen Elemente, welche in der Applikation verwendet werden, von unterschiedlichen Erweiterungen bereitgestellt werden.

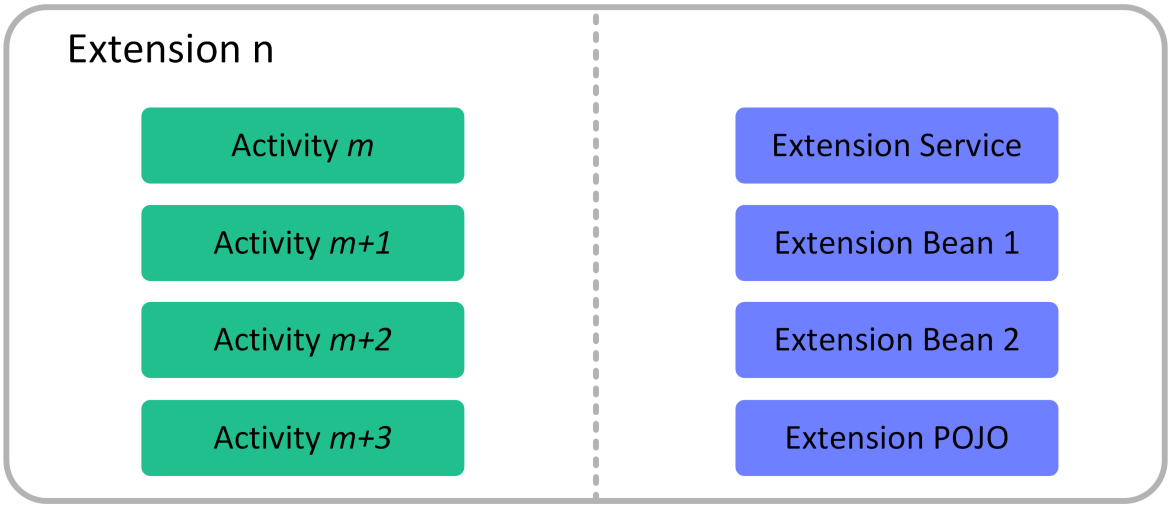


Das ist auch nicht problematisch, sondern direktes Ergebnis des Aufbaus von D°. Bei genauerer Betrachtung ergibt sich aber eine Einschränkung für die Erstellung von Erweiterungen. Zu diesem Zweck betrachten wir eine beliebige Erweiterung, welche eine beliebige Anzahl von verschiedenen Sprachelementen für D° enthält.

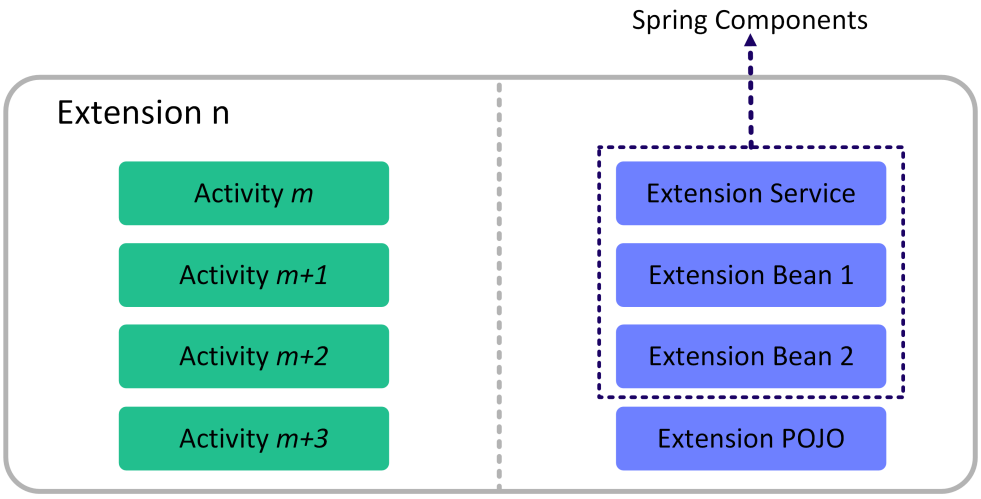


Für D° sind Aktivitäten atomare Elemente der Sprache. Aber in der verwendeten Hostsprache kann sich hinter jeder Aktivität eine beliebige Menge Programmcode befinden, welcher die eigentliche Logik bereitstellt. Dasselbe gilt für Policies.

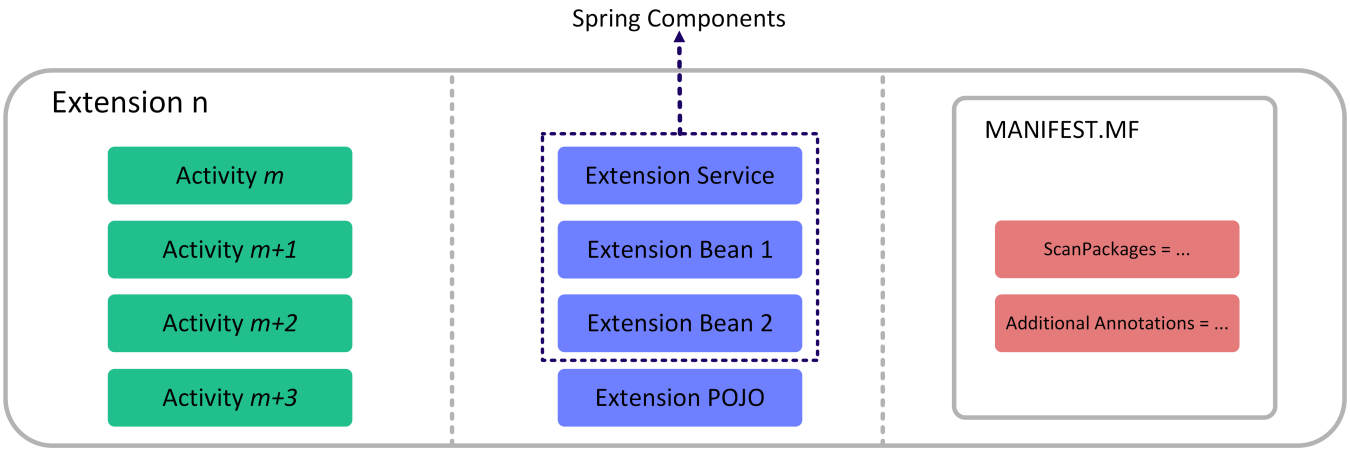
Daraus folgt, dass eine Erweiterung neben den D°-Sprachelementen, welche für den Anwender sichtbar und verwendbar sind, diverse zusätzliche Komponenten enthalten kann, welche zur Bereitstellung der Logik des jeweiligen Elements beitragen.



An dieser Stelle wird eine Limitation des Ansatzes sichtbar. Theoretisch könnten die zusätzlichen Komponenten, welche Teil der Erweiterung sind, auf Funktionalitäten des verwendeten Technologie-Stacks zugreifen. Am Beispiel unserer Implementation, welche auf das Spring Framework setzt, würde dies bedeuten, dass durch die Applikationen spezielle Konfigurationen/Annotationen benannt werden müssen, welche anschließend von dem Compiler in die generierte Applikation integriert werden. Dieser "Rückkanal", welcher es Erweiterungen erlaubt Einfluss auf den Codegenerator zu nehmen, war in D° bisher nicht vorgesehen.



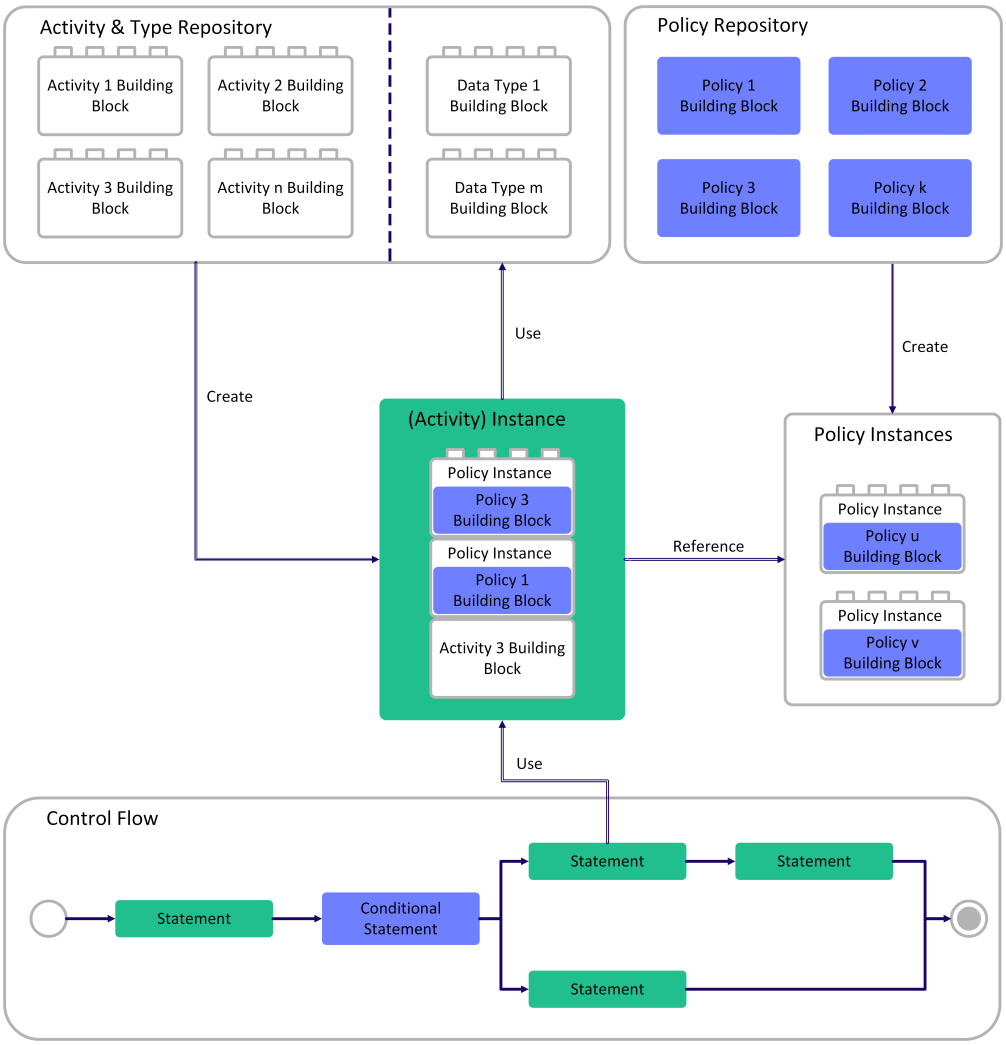
Aus diesem Grund war es notwendig eine flexible Möglichkeit bereitzustellen, welche es Erweiterungen erlaubt Informationen an den Compiler zu übergeben. Dabei sollen diese Metadaten der Erweiterungen nicht losgelöst von der eigentlichen Erweiterung gespeichert werden. Da dies nur solche Erweiterungen betrifft, welche auch eigenen Programmcode in der Hostsprache bereitstellen, lässt sich eine komfortable Lösung für diese Einschränkung finden. Die betroffenen Erweiterungen werden dem Compiler in Form eines JAR-Archives zur Verfügung gestellt. Dabei hat jedes JAR die Möglichkeit Metadaten über die Datei MANIFEST.MF, welche in das JAR integriert ist, bereitzustellen. Dieser Mechanismus wird verwendet, um für D° passende Metadaten in die Erweiterungen zu integrieren und diese anschließend im Compiler wieder auszulesen.



Schatteninstanzen

In der verfügbaren Dokumentation von D° finden sich Informationen darüber, wie zwischen den Definitionen und Instanzen von Sprachelementen in D° unterschieden wird und welche Implikationen dies hat.

An dieser Stelle wird das Thema kurz zusammengefasst: Sprachelemente, die über Programmcode in der verwendeten Hostsprache verwenden, werden in einem ersten Schritt als Definition angelegt. Diese Definitionen geben die Struktur und Schnittstellen vor, sind aber nicht direkt in D°-Applikationen nutzbar. Um zu einem Konstrukt zu gelangen ist es notwendig diese Definitionen zu instantiieren. Ein möglicher Teil dieser Instantiierung ist die Komposition von verschiedenen Sprachelementen. Das nachfolgende Schaubild stellt diesen Schritt dar.



Daraus ergibt sich keine direkte Problematik, aber ein Potential für erheblichen Mehraufwand, der vermieden werden kann. Häufig sollen Aktivitäten genutzt werden, ohne das von den zusätzlichen Möglichkeiten, welche die Instantiierung bereitstellen, Nutzen gemacht wird. Beispielsweise wird eine Aktivität, welche verwendet wird, um Log-Nachrichten auf die Konsole auszugeben häufig nicht mit zusätzlichen Policies versehen.

Der bisherige Entwicklungsstand von D° macht es in diesen Situationen erforderlich, dass eine "leere" Instanz der entsprechenden Aktivitätsdefinition erzeugt wird, welche wiederum in D°-Applikationen verwendet werden kann.

Dies führt potentiell zu einem hohen Maß von Redundanz und Mehraufwand bei der Erzeugung solcher Instanzen. Aus diesem Grund wurde der Compiler mit der Möglichkeit versehen für die Definitionen von Aktivitäten sogenannte "Schatteninstanzen" zu erzeugen, welche einer direkten Instanz einer Definition ohne Anpassung entspricht.

Dabei erzeugt der Compiler für jede Aktivitätsdefinition eine solche Schatteninstanz. Diese ist wie jede andere Aktivitätsinstanz nutzbar. Sofern der Compiler erkennt, dass eine solche Schatteninstanz in der aktuell übersetzten Applikation verwendet wird, wird die Schatteninstanz auch in das maßgeschneiderte Typ- und Aktivitätsrepository der Applikation exportiert.

Zugriff und Zuweisung von Attributen

Der bisherige Stand von D° erlaubt keinen Komfortablen Zugriff auf einzelne Attribute/Variablen sowie Änderungen an den gespeicherten werten. Der nachfolgende Codeblock zeigt, wie der Wert einer Variable im bisherigen Entwicklungsstand von D° verändert wird.

```
statusMsg = $Text(@write["Starting initialization."]);
statusMsg = $Text(@write["Changed content."]);
```

Dies ist wenig intuitiv und erzeugt Mehraufwand bei der Entwicklung mit D°. Aus diesem Grund wurde eine Möglichkeit geschaffen Variablen und Attribute zu verändern, welche sich an bekannten Programmiersprachen orientiert. Der nachfolgende Codeblock zeigt diverse Zuweisungsmöglichkeiten, welche dem Compiler jetzt hinzugefügt wurden.

```
statusMsg = $Text(@write["Starting initialization."]);
statusMsg = "eSun"

filament = $demonstrator_3d_printer.Filament();
filament.company = "dasFilament";
filament.company = statusMsg;
filament.diameter = "1.75";
```

Dabei ist zu beachten, dass jedes Literal in Form eines Strings angegeben werden muss. Dies gilt auch für Zahlen und bspw. boolsche Werte. Dies liegt daran, dass das von D° verwendete Typsystem Nukleus Stringbasiert arbeitet.

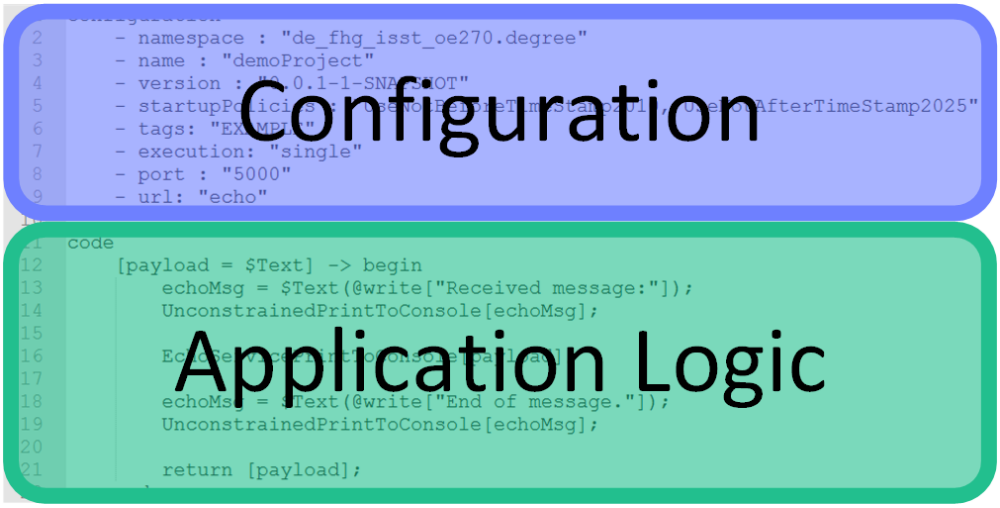
Hostsprachen Konfiguration

Wie aus der verfügbaren Dokumentation zu D° hervorgeht besteht jede D°-Applikation aus zwei Teilen:

1. Der eigentlichen Applikationslogik
2. Der Konfiguration der Applikation

Dabei enthält der Konfigurationsbereich alle Daten, die notwendig sind um die Applikation zu beschreiben. Gleichzeitig wird in diesem Konfigurationsbereich darauf verzichtet Einstellungen vorzunehmen, welche Abhängig von der gewählten Technologie und Hostsprache sind. Beispielsweise finden sich Informationen über verwendete Ports und Pfade welche verwendet werden, um die Applikation aufzurufen, im Konfigurationsbereich. Aber Einstellungen für spezielle Flags, welche in einer Applikation verwendet werden, finden sich nicht in diesem Konfigurationsbereich.

Die nachfolgende Abbildung zeigt eine beispielhafte D°-Applikation und hat die beiden Aspekte der Applikation entsprechend hervorgehoben.



Der Grund für diese Trennung ist, dass die Hostsprache von D° theoretisch ausgetauscht werden kann. Würden solche Daten im Konfigurationsbereich der Applikation abgelegt werden, würde eine Bindung zwischen dem D°-Code und der verwendeten Hostsprache entstehen, was zu vermeiden ist.

Aus diesem Grund wurde dem Compiler die Möglichkeit hinzugefügt beliebige `properties`-Dateien mit in den Übersetzungsvorgang einzubeziehen. Diese Dateien beinhalten beliebig viele `<key>=<value>`-Paare. Der Compiler liest diese Dateien aus, meldet eventuell vorhandene Konflikte und fügt die entsprechende Konfiguration in die erzeugte Applikation ein.