MIML light – Referenz

1. MIML – Multimodal Integration Markup Language

MIML beschreibt den strukturellen Aufbau von multimodalen Äußerungen. Besondere Beachtung wird hierbei auf die Zeitlichkeit von einzelnen Bestandteilen der Äußerungen – auch zwischen verschiedenen Modalitäten – gelegt:durch Beschreibung einer Abfolge, zeitlicher Relationen oder Konstraints über Zeitpunkten.
MIML light stellt eine vereinfachte Untermenge von MIML dar: Es existieren nur sprachliche Äußerungen und es wird auf die Aufteilung in Unternetzwerke verzichtet. MIML light ist damit noch stärker an die Notation für ein Zustandsnetzwerk angelehnt.

2. Die MIML Umgebungs-TAGs

MIML beschreibt neben den Strukturen der Äußerungen auch die benutzten Lexika.
Unterhalb der obersten Ebene (dem `definition`-TAG) können sowohl Lexika (s. 3.), als auch Definitionen der Struktur der Netzwerke folgen (entsprechen der Grammatik).

3. Lexika

Allgemein kann MIML beliebige Lexika für beliebige Modalitäten zur Verfügung stellen (im Kontext der virtuellen Werkstatt arbeiten wir mit einem sprachlichen Lexikon und einer Sammlung von Gesten). Für MIML light definieren wir nur einfache sprachliche Lexikoneinträge:

```
<wordtype function="createAction">
    <word>besorg,besorge,erzeug,erzeuge,generiere,gib</word>
</wordtype>
```

Die terminalen Symbole – als atomare Bestandteile der Äußerung – werden abstrahiert zu einem nichtterminalen Symbol, dass aus unserer Grammatik dann referenziert werden kann: Es beschreibt üblicherweise abstrakt die Funktion, die die Terminale in einer Äußerung übernehmen. Die einzelnen Terminale werden dabei in einer `<word>`-Umgebung als Komma-separierte Liste aufgeführt.

In einer `wordtype`-Umgebung können auch mehrere `word`-Umgebungen auftauchen, die dann verschiedene Wertebelegungen als Rückgabewert zur Verfügung stellen:

```
<wordtype function="type">
    <word value="Objekt">Objekt,Ding,Teil</word>
    <word value="Schraube">Schraube</word>
    <word value="Block">Block,Klotz,Wuerfel</word>
</wordtype>
```

In der `wordtype`-Umgebung wird die Funktion innerhalb der strukturellen Beschreibung beschrieben (im Beispiel: Objekte sind von einem Typ – als da wären Schraube, Block, …). In den einzelnen `word`-Umgebungen können gleichzeitig auch noch standardisierte Wertzuweisungen geschrieben werden, wie sie benötigt werden zum Beispiel in einem ATN, um gesammeltes Wissen während der Abarbeitung des ATNs auch *semantisch* zu repräsentieren (im Beispiel: wenn geäußert wurde "Erzeuge einen Block" oder "Generiere einen Klotz", so soll beim Parsen des Typs als Repräsentation aufgebaut werden, dass der Typ `Block` ist.

4. Strukturelle Beschreibung

Die Grammatik wird als Struktur in einer `requirement`-Umgebung definiert: Hierbei wird sowohl festeglegt, wie Äußerungen aussehen dürfen, als auch semantische Information aufgebaut.
Ein `requirement` wird begonnen mit einer Definition von Variablen – die die semantischen Wertbelegungen enthalten sollen:

```
<frame>
    <slot name="attribute" type="singleslot"/>
</frame>
```

Ein `slot` kann während des Parse-Vorgangs gefüllt werden (ein Wert zugewiesen werden) und später von außen (oder intern durch eine eingebundene Funktion) wieder ausgelesen werden.
In der Beschreibung eines ATNs durch MIML werden hier dadurch gleichzeitig am Anfang schon alle notwendigen Variablen definiert.
Danach folgt die strukturelle Beschreibung (`description`), in der zuerst die zeitliche Beziehung der folgenden Teiläußerungen festgelegt werden: Im Fall von MIML light sind diese immer einfach sequentiell angeordnet.
Die Grammatik wird nun durch speech-TAGs angegeben: als atomare verlangte Wörter oder aus dem Lexikon benutzte Wörter einer bestimmten Wortgruppe (über function festgelegt):

```
<speech>
    <function name="createAction"/>
</speech>
<optional>
    <speech>
        <word> mir </word>
    </speech>
</optional>
```

Eine `optional`-Umgebung gibt an, dass darin enthaltene Bestandteile nur möglich, aber nicht erforderlich sind.
Zur Verzweigung können `select`-Konstrukte eingeführt werden, in denen dann verschiedene Pfade als `<choice>`s angegeben werden können.
Während des Parse-Vorgangs gesammelte Informationen können in den Variablen/Slots zwischengespeichert werden zur späteren Verwendung (als semantische Information):

```
<speech>
    <function name="type"/>
    <fill-slot name="type"/>
</speech>
```

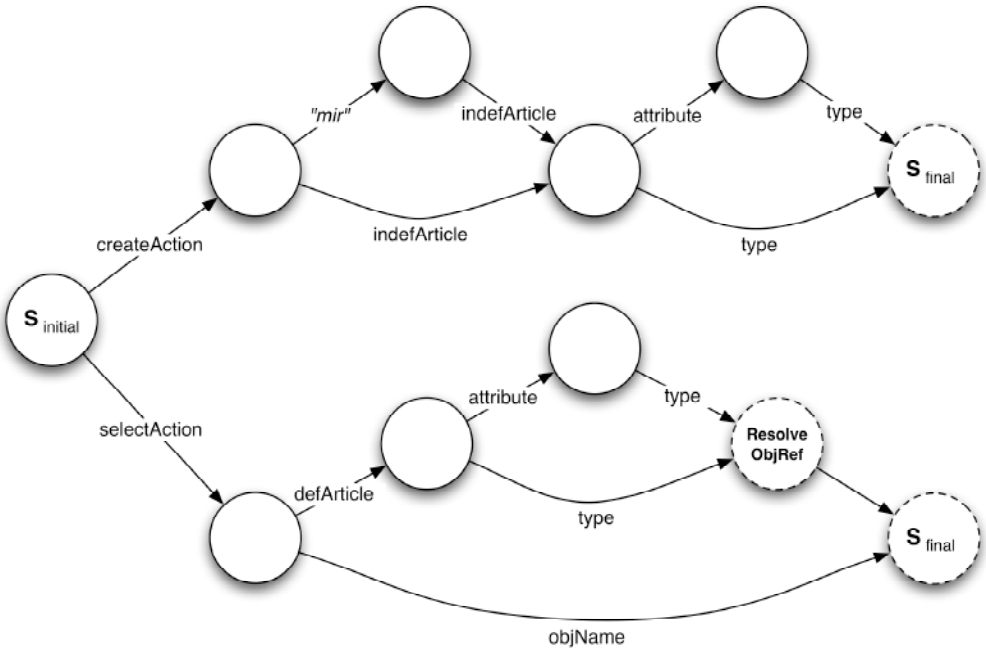
Der Slot wird nun gefüllt mit dem im Lexikon definierten `value` für das entsprechend geparste Wort.
Im Ablauf des Parsens (oder am Ende) können extern Funktionsaufrufe angestoßen werden. In der `execute`-Umgebung wird zuerst der Funktionsaufruf angegeben (als `apiCommand`) und können zusätzliche Parameter übergeben werden (gefüllt aus den Slots des aktuellen requirements):

```
<execute>
    <apiCommand name="selectAction">
        <set-param name="objName" sourceslot="identifier"/>
```

```
        </apiCommand>
    </execute>
```

5. Beispiel

Im folgenden ein vollständiges Beispiel für eine MIML light-Definition für das in der Graphik dargestellte ATN:




```
<?xml version="1.0" standalone="no"?>
<definition start="topLevel">
  <wordtype function="createAction">
    <word>besorg,besorge,erzeug,erzeuge,generiere,gib</word>
  </wordtype>
  <wordtype function="selectAction">
    <word>nimm,Nimm</word>
  </wordtype>
  <wordtype function="indefArticle">
    <word>ein,einem,eine,einer,einen,eines</word>
  </wordtype>
  <wordtype function="defArticle">
    <word>der,die,das,dem,den,des</word>
  </wordtype
  <wordtype function="attribute">
    <word value="blau">blau,blaue,blauen,blaues,blauem,blauen,blauer</word>
    <word value="gelb">gelb,gelbe,gelben,gelbes,gelber,gelbem</word>
    <word value="rot">rot,rote,roten,rotes,rotem,roter</word>
    <word value="gruen">gruen,gruene,gruenen,gruenes,gruener,gruenen</word>
  </wordtype>
  <wordtype function="type">
    <word value="Objekt">Objekt,Ding,Teil</word>
    <word value="Schraube">Schraube</word>
    <word value="Block">Block,Klotz,Wuerfel</word>
    <word value="Teil">Ding,Dingsbums,Etwas,Objekt,Teil</word>
    <word value="Dreierleiste">Dreier,Dreierleiste,Dreilochleiste</word>
    <word value="Fuenferleiste">Fuenferleiste,Fuenflochleiste</word>
    <word value="Leiste">Holzleiste,Leiste</word>
    <word value="Schlitzschraube">Schlitzschraube</word>
    <word value="Sechskantschraube">Sechskantschraube</word>
  </wordtype>
  <wordtype function="objName">
    <word value="block_1">block_1,block_eins</word>
```

```
  <word value="block_2">block_2,block_zwei</word>
  <word value="block_3">block_3,block_drei</word>
  <word value="block_4">block_4,block_vier</word>
  <word value="leiste_1">leiste_1,leiste_eins</word>
  <word value="leiste_2">leiste_2,leiste_zwei</word>
  <word value="leiste_3">leiste_3,leiste_drei</word>
  <word value="schraube_1">schraube_1,schraube_eins</word>
  <word value="schraube_2">schraube_2,schraube_zwei</word>
</wordtype>

<requirement name="topLevel" function="topLevel">
  <frame>
    <slot name="attribute" type="singleslot"/>
    <slot name="type" type="singleslot"/>
    <slot name="identifier" type="singleslot"/>
  </frame>
  <description>
    <temporalrelation type="sequential">
      <select>
        <choice>
          <speech>
            <function name="createAction"/>
          </speech>
          <optional>
            <speech>
              <word> mir </word>
            </speech>
          </optional>
          <speech>
                 <function name="indefArticle"/>
          </speech>
          <optional>
            <speech>
              <function name="attribute"/>
              <fill-slot name="attribute"/>
            </speech>
          </optional>
          <speech>
            <function name="type"/>
            <fill-slot name="type"/>
          </speech>
          <execute>
            <apiCommand name="createAction">
                        <set-param name="apiType" sourceslot="type"/>
                        <set-param name="apiAttributes" sourceslot="attribute"/>
            </apiCommand>
            <fill-slot source="apiResult" target="identifier"/>
          </execute>
        </choice>

        <choice>
          <speech>
            <function name="selectAction"/>
          </speech>
          <select>
            <choice>
              <speech>
                <function name="defArticle"/>
              </speech>
              <optional>
                <speech>
                  <function name="attribute"/>
                  <fill-slot name="attribute"/>
```

```
                </speech>
              </optional>
              <speech>
                <function name="type"/>
                <fill-slot name="type"/>
              </speech>
              <execute>
                 <apiCommand name="resolveObjectReference">
                     <set-param name="apiType" sourceslot="type"/>
                     <set-param name="apiAttributes" sourceslot="attribute"/>
                 </apiCommand>
                 <fill-slot source="apiResult" target="identifier"/>
              </execute>
            </choice>
            <choice>
              <speech>
                <function name="objName"/>
                <fill-slot name="identifier"/>
              </speech>
            </choice>
          </select>
          <execute>
            <apiCommand name="selectAction">
                 <set-param name="objName" sourceslot="identifier"/>
            </apiCommand>
          </execute>
        </choice>

      </select>

    </temporalrelation>
  </description>
</requirement>

</definition>
```