Model Interchange Framework (Stand 16.02.2014)

# Interchange Description Language

Die “Interchange Description Language” (IDL) dient dazu, die Rahmenbedingungen für eine Modelltransformation zu beschreiben. Zu diesen Rahmenbedingungen zählen u.a. Angaben zu anwendbare Diagrammtypen, Art und Ablauf der Transformation.

Das formale Grundgerüst einer Beschreibungsdatei sieht wie folgt aus:

[public] interchange ‘%Titel%‘ (%Zielformat%) [<> %Diagrammtyp%] {

[description: ‘%Text%‘]

@script: ‘%Pfad%‘ [, %Parameter%, …]

[…]

}

Ausdrücke in eckigen Klammern stehen dabei für optionale Angaben. Beginnt und endet ein Ausdruck mit „%“, so handelt es sich dabei um einen Parameter, der ausgefüllt werden muss. Es folgen nun zwei Beispiele für eine korrekte und vollständig ausgefüllte Interchange-Beschreibung:

public interchange 'Prolog Facts 2 - MIL' (pl) <> epc, oepc {

description: 'Exports a bflow\* diagram as Prolog facts. Based on export description language.'

@script: '/exportscripts/files/prolog.vt', insertAttributes=true

}

Listening 1

interchange 'UML' (umlx|uml|xml) {

description: 'Creates an UML diagram for the given model.'

@script: '/exportscripts/files/uml1.vt', factor=4, canGrow=true, defaultSize=40.5

@script: '/exportscripts/files/umlX.vt'

@script: '/exportscripts/files/uml2.vt', plain=true

}

Listening

Es folgt nun eine Auflistung und Beschreibung der optionalen Parameter:

|  |  |  |
| --- | --- | --- |
| Name | Wertebereich | Beschreibung |
| public |  | Wenn gesetzt, dann ist das Austauschformat nicht nur für Add-ons sondern auch für den gewöhnlichen Exportvorgang verfügbar. Das bedeutet, dass das Austauschformat über den Exportdialog auswählbar ist. |
| description | String | Die hier angegebene Beschreibung wird innerhalb des Exportdialogs verwendet. |
| Diagrammtyp | String | Siehe dazu weiter unten. |
| Parameter | Boolean, Float, Integer | Die hier angegebenen Parameter werden für das jeweilige Skript an die Instanz weitergeben, die das Skript verarbeitet. Dies kann zum Beispiel ein XSLT- oder Template-Processor sein. |

Abschließend noch ein Wort zu den auszufüllenden Parametern.

|  |  |
| --- | --- |
| Name | Beschreibung |
| Titel | Dieses Feld steht für den Titel bzw. Name des Austauschformats. Dieser wird in der Auswahlanzeige des Exportdialogs verwendet. |
| Zielformat | Dieses Feld steht für eine Auflistung von Dateiendungen, die für Ziel- bzw. Quelldateien eines Transformationsvorgangs verwendet werden können. Die Auflistung muss mindestens ein Kürzel enthalten. Mehrere Angaben können durch „|“ getrennt werden. Standardmäßig wird für einen Export immer das erste Kürzel für die Zieldatei verwendet. |
| Diagrammtyp | Legt eine Auflistung von Diagrammtypen fest, für die das jeweils definierte Austauschformat anwendbar ist. **Diese Angabe ist optional.** Wird keine Angabe gemacht, so ist das Austauschformat für alle Diagrammtypen verfügbar. Die Auflistung muss mindestens eine Angabe enthalten. Als Angabe gilt dabei jeweils die Dateiendung der Modelldatei des jeweiligen Modelleditors. Mehrere Angaben können mittels Komma getrennt werden. |
| Text | Siehe dazu die Beschreibung des optionalen Parameters „description“. |
| Pfad | Gibt den relativen Pfad zur Skriptdatei an. Dabei wird von der Lage der Beschreibungsdatei ausgegangen. |
| Parameter | Siehe dazu die Beschreibung des optionalen Parameters „Parameters“. |

Zur Installation von Skripten bitte in den entsprechenden Kapiteln der EMTB-Dokumentation nachschauen.

# Modelltransformationen

Das „Model Interchange Framework“ unterstützt eine breite Palette an Transformationsarten. Dazu zählen zum Beispiel XLST- oder Template-basierende Umwandlungen. Außerdem ist die Erzeugung von Graphiken oder UML-Diagrammen möglich. …

## XSLT-basierende Transformationen

Text …

## Template-basierende Transformationen

Bei dieser Form der Transformation kommt eine Template-Processing-Engine zum Einsatz. Einfach ausgedrückt bedeutet das, dass für das Zielformat bereits eine grobe Struktur vorliegt und nur noch gewisse Bereiche davon ausgefüllt werden müssen. Daher ist diese Form der Transformation für die meisten Anwendungszwecke ausreichend. Das folgende Beispiel zeigt ein einfaches Template.

Liste aller Knoten:

#foreach( $shape in $shapes )

Knotenname: $shape.Name, Typ: $shape.Type

#end

- Abschluss -

Listening

Das Ergebnis des Transformationsprozesses könnte dann beispielsweise wie folgt aussehen:

Liste aller Knoten:

Knotenname: Telefon klingelt, Typ: Event

Knotenname: Hörer abnehmen, Typ: Function

Knotenname: Stille, Typ: Event

Knotenname: Hörer auflegen, Typ: Function

- Abschluss –

Anhand des Beispiels ist erkennbar, dass jedes Template aus einem statischen und einem dynamischen Anteil bestehen kann. Während die statischen Teile für jeden Transformationsvorgang gleich sind, bedingen die dynamischen Teile dem zugrundeliegenden Modell. Dynamische Anweisungen beginnen typischer Weise mit dem Symbol „#“. Das Zeichen „$“ signalisiert einen Zugriff auf eine Variable.

Die Engine, die zur Umsetzung der Transformation verwendet wird, ist Apache Velocity[[1]](#footnote-1). Die komplette Syntax, die in einem Template zum Einsatz kommt, beruht auf diesem Framework. Zum Erstellen eines eigenen Templates, sollte man sich daher mit der entsprechenden Dokumentation[[2]](#footnote-2) vertraut machen.

In den folgenden Abschnitten erfolgt nur eine Dokumentation der Variablen und deren Eigenschaften, die seitens des Model-Interchange-Frameworks für jeden Transformationsvorgang zur Verfügung gestellt werden.

### Verfügbare Variable und deren Eigenschaften

Die folgenden Variablen stehen bei jedem Transformationsvorgang zur Verfügung.

|  |  |  |
| --- | --- | --- |
| Variable | Eigenschaft | Bedeutung |
| model | - | Stellt den Zugriff auf Eigenschaften bereit, die das Modell besitzt. |
|  | Id | Id des Modells |
|  | Type | Typ des Modells |
|  | Attributes | Key-Value-Map, die alle dem Modell zugeordneten Attribute besitzt |
| shapes | - | Array, das alle im Modell vorkommenden Knoten beinhaltet. Jeder Knoten innerhalb dieses Arrays besitzt die folgenden Eigenschaften. |
|  | Id | Id des Knotens |
|  | Type | Typ des Knotens |
|  | Attributes | Key-Value-Map, die alle dem Knoten zugeordneten Attribute besitzt |
|  | Name | Name des Knotens |
|  | Width | Breite der Figur, die den Knoten repräsentiert |
|  | Height | Höhe der Figur, die den Knoten repräsentiert |
|  | X | X-Koordinate der Figur, die den Knoten im Modell repräsentiert |
|  | Y | Y-Koordinate der Figur, die den Knoten im Modell repräsentiert |
|  | Image | Bild der Figur, die den Knoten repräsentiert |
| edges | - | Array, das alle im Modell vorkommenden Kanten beinhaltet. Jede Kante innerhalb des Arrays besitzt die folgenden Eigenschaften. |
|  | Id | Id der Kante |
|  | Type | Typ der Kante |
|  | Attributes | Key-Value-Map, die alle der Kante zugeordneten Attribute besitzt |
|  | Source | Gibt den Knoten an, den die Kante als Quelle hat. Das hinterlegte Objekt besitzt dabei die gleichen Eigenschaften wie jeder Knoten. |
|  | Target | Gibt den Knoten an, den die Kante als Ziel hat. Das hinterlegte Objekt besitzt dabei die gleichen Eigenschaften wie jeder Knoten. |
| params | - | Key-Value-Map, die die für die Transformation hinterlegten Parameter beinhaltet. Siehe dazu das Kapitel „Interchange Description Language“. |

Der nächste Abschnitt zeigt ein vollständiges Beispiel für ein Template.

### Beispiel für ein Template

Das folgende Beispiel erzeugt aus einem EPC-Modell eine Prolog-Faktenbasis, die im Rahmen der von bflow\*[[3]](#footnote-3) verwendeten sogenannten „Erweiterten Modellprüfung“ verwendet wird.

:- dynamic arc/2.

:- dynamic and/1.

:- dynamic or/1.

:- dynamic xor/1.

:- dynamic event/1.

:- dynamic function/1.

:- dynamic region/1.

#set( $items = {})

#foreach( $shape in $shapes)

#set( $type = $shape.Type.split("\.")[4].toLowerCase() )

#set( $j = $foreach.count)

#set( $id = "i\_$j")

#if($type == "event" || $type == "function" || $type == "region")

#set( $items[$shape] = $id )

$type($id).

elementname($id,'$shape.Name').

bflow\_id($id,'$shape.Id').

#foreach( $key in $shape.Attributes.keySet() )

attribute('$shape.Id', '$key', '$shape.Attributes.get($key)').

#end

#end

#if($type == "and" || $type == "or" || $type == "xor")

#set( $items[$shape] = $id )

$type($id).

bflow\_id($id,'$shape.Id').

#end

#end

#foreach( $edge in $edges )

#set( $type = $edge.Type.split("\.")[4].toLowerCase() )

#set( $sourceShape = $edge.Source )

#set( $targetShape = $edge.Target )

#set( $src = $items[$sourceShape] )

#set( $tgt = $items[$targetShape] )

#if($type == "arc")

arc($src,$tgt).

bflow\_id($src,$tgt,'$edge.Id').

#end

#end

Listening

### Layout der Templates

Steigt der Umfang und die Komplexität der Templates, so wird man dazu übergehen, diese verstärkt mittels Einrückungen und Zeilenumbrüchen zu strukturieren. Da aber jedes Zeichen außerhalb der Template-Direktiven als Ausgabezeichen interpretiert wird, führen derartige Einrückungen zu einem verzerrten Ausgabelayout. Sollte das nicht gewünscht sein, so kann man mit den folgenden Maßnahmen dem entgegenwirken.

Das MIF unterstützt die Möglichkeit, eine Vorverarbeitung der Templates durchzuführen. Um dieses Feature zu aktivieren, muss am Beginn des Templates folgender Zusatz notiert sein: „@preprocess“. Diese Kennzeichnung bewirkt, dass das MIF alle Einrückungen (Leerzeichen, Tabulatorsprünge) und Zeilenumbrüche aus dem Template entfernt. Somit kann das Template strukturiert werden, ohne das das eine Auswirkung auf das Ausgabelayout hat.

Soll das Ausgabelayout aber an gewissen Stellen derartige Einrückungen beinhalten, so kann dies wie folgt realisiert werden. Fügen Sie an den betroffenen Stellen folgende Direktiven in das Template ein:

|  |  |
| --- | --- |
| Direktive | Wirkung |
| ~t | Tabulatorsprung |
| ~nl | Zeilenumbruch |

**Wichtiger Hinweis:**Sollten in einem Template Kommentare (beginnen mit der Direktive „##“) vorhanden sein, so müssen sie bei aktiver Vorverarbeitung **immer** mit einem manuell gesetzten Zeilenumbruch abgeschlossen werden. Andernfalls wird das Template nicht das gewünschte Ausgabelayout erzeugen können.

Beispiel:

@preprocess

## Das ist ein Kommentar~nl

#foreach($shape in $shapes)

#set($id = $shape.Id)

~t$id~nl

#end

Listening

Das Ergebnis sieht bspw. wie folgt aus:

\_Ax99alsjf8t

\_o9oo83jnnma

\_iuaL-aufu2d

## Eigene Transformationen entwickeln

Text …

1. http://velocity.apache.org/ [↑](#footnote-ref-1)
2. http://velocity.apache.org/engine/releases/velocity-1.7/user-guide.html [↑](#footnote-ref-2)
3. http://www.bflow.org [↑](#footnote-ref-3)