Towards a DEMO Description in Simplified Notation Script

Mark A. T. Mulder(✉), Rick Mulder, and Fiodor Bodnar

TEEC2, Hoevelaken, The Netherlands
markmulder@teec2.nl

Abstract. The core methodology of Enterprise Engineering (EE) is Design and Engineering Methodology for Organisations (DEMO) and has been the subject of modelling tools. This methodology can be split into a method or process part and a notation part, describing the meta-model and its visualisation. The way the notation of the methodology has been described for these tools has been of different detail levels. This paper describes the DEMO notation using the grammar of the Simplified platform as an exercise towards a complete notation grammar that can describe all existing and possibly future notations and also to complete the DEMO notation specification. The grammar is part of the Simplified platform, and the notation is the published definition of the notation part of the DEMO methodology. We have chosen a practical approach to developing the notation script and thinking out-of-the-box by not creating a theoretical box a priori.

Keywords: enterprise engineering · DEMO · modelling tools

1 Introduction

The DEMO [1] method is a core method (based on a theoretically founded method*ology*) within the discipline of EE [2]. The DEMO method focuses on the creation of so-called *essential* models of organisations. The latter models capture the organisational essence of an organisation primarily in terms of the actor roles involved, as well as the business transactions [9] (and ultimately in terms of speech acts [5]) between these actor roles. More specifically, an essential model comprises the integrated whole of four aspect models: the Construction Model (CM), the Action Model (AM), the Process Model (PM) and the Fact Model (FM). Each of these models is expressed in one or more diagrams and one or more cross-model tables. DEMO has strong methodological, and theoretical, roots [1,2,9].

After we built the Plena tool for modelling DEMO in the PhD project of Mulder [7], we continued our research to expand the modelling capability. This research found that the modelling capability could not be expanded within the existing Sparx Enterprise Architect (SEA) tool. We found many problems that we could not solve within this environment, e.g. functionality support, visualisation support, collaborative work support.

The problem we address in this paper is the complexity of the notations and the visualisation of that notation that we want to use. The example that we describe in this paper is about DEMO. DEMO is a method with complex diagrams that are linked together by the main elements, transactions. The transaction has multiple visualisations, dependent on the context. This makes the visualisation of the element non-trivial and demands variables for the visualisation. Furthermore, the description of the diagrams in the DEMO Specification Language (DEMOSL) has more requirements than the SEA tool can handle, e.g. the transactor and transaction kind that reference the same model element, attributes that are always a part of an entity and cannot be represented separately like ORM representation, and multiple visualisations of the same element in the same diagram. Other notations have similar issues (e.g. BPMN swim-lanes and visual attributes, IDEF0 layout restrictions, ArchiMate double representations). We needed a modelling environment to address all these visualisation issues.

Therefore, we have introduced Simplified modelling platform [6] that can create models according to specified notations (metamodels) in a web-based environment and present the models to relevant stakeholders. In summary, the Simplified modelling platform can contain any model that can be described as element-connection model with attributes (e.g. data models, process models (blocks and arrows), architecture models). The models can be visualised in table or diagram style where the representation is free to choose. Although this paper describes the notationscript from a DEMO method perspective, we have made notationscripts for more notations (e.g. ArchiMate, BPMN, Flowchart).

In this paper, we report on a study about the configuration effort within the Simplified modelling platform to support the use of DEMO 3 and 4 in practice answering the question on the minimal expressive power needed to describe these notations on this platform. This description should satisfy the model expression and the visualisation of the models. The flexibility in defining notations can help to easily define new domain specific notations with new semantics for specific problems.

Configuring the Simplified modelling platform support for DEMO requires an elaborate formalisation of the DEMO metamodel, as described in DEMOSL [4], and further specified to enable the automatic verification of models [7]. In order to translate the formalised DEMO metamodels, rules and visual notations to the notationscript we need the notation grammar that will support the definition of all required concepts and all the related visualisations.

Note that we have described the notation of the methodology in this paper while not describing any model. The visual and textual representations are part of the notation description and are represented in the metamodel structure. Also, we have chosen for a practical approach, thus not studying literature a priori. This approach might result in a grammar that partially exists. In hindsight, if we were able to use an available part of an existing notation grammar, then we would have to combine different parts from these notation grammars, which makes this option less feasible from a maintenance point of view. Also, the different points of view from different attempts make it very challenging to combine these efforts. Summarising, we have chosen to use our own notation grammar to define notations that will support the definition of required concepts and their related visualisations as will be explained in Sect. 3 and Sect. 6, respectively.

For the visualisation part of the grammar we have looked at implementations of SEA, ADOXX and draw.io. We used the convenient user interface concepts of these implementations and made our own implementation without copying anything.

Because DEMO version 3 and version 4 differ in the number of concepts, metamodel and visualisations we have decided to cover both versions. We see the new concepts of DEMO 4 as an extension on the concepts of DEMO 3 although the literature [3] leaves out options of the previous version [8], e.g. the Organisation Construction Diagram (OCD).

The remainder of this paper is structured as follows. Section 2 describes the research method utilised. Section 3 provides a overview of concepts that are currently supported by the proposed grammar for describing notations. In Sect. 4, we show how how DEMOSL of DEMO version 3.7 has been defined in the notationscript for the CM, PM and the FM. Section 5 addresses the definition of DEMOSL to the notationscript based on DEMO version 4.7.1. In Sect. 6 we continue on the specifics for defining visualisations in a notationscript. Lastly, before concluding, in Sect. 8 we discuss several challenges that future research will have to address.

2 Research Question and Method

The research question was to find a notation to flexibly describe at run-time the meta model of a model notation with the corresponding visualisations. The creation of this notationscript is an iterative process to find a suitable grammar that covers all meta model descriptions. This paper describes the creation of this notationscript which can be seen as an artefact in the sense of design science. We used Design Science Research (DSR) [10] to create the notation grammar and script. This artefact is an object that solves a problem by interaction with the context of that artefact. Thereafter the artefact must be implemented, validated, and evaluated.

3 Notations

Within the Simplified context, a notation can be self-contained, extending or replacing concepts from other notations. By extending other notations, one can add own elements without redefining previous notations. With the replacement functionality, one can restrict the use of a notation to a smaller subset of attributes of the concepts.

We have defined the notation as the highest user meta-level in the Simplified platform. At this level, we can define how models can be modelled and what rules the models must adhere to. In order to verbalise this metamodel, we have created a notation script language and corresponding grammar that can be used to specify notations. The notation script language is described in a notation grammar where the levels of abstraction are reflecting the functional components in the platform, and the transformation between the levels is either interpretation on run-time or compilation when parsing the notation script.

The current version of the notation script has a grammar for the following list of notation concepts which will be explained using the DEMO notations in Sects. 4 to 6.

```
element [extends <refname>] [replaces <refname>]
[comment "<comment>"]
( <name>
<typeRefName> | TEXT | URL | GUID | BOOL | INT | DATETIME
[<default>] [comment "<comment"] [,...] )
```

Listing 3.1. Element Grammar

Element is the base concept often referred to as 'block' (from 'blocks and arrows' in process modelling). In the notation script, the element concept supports the hereafter mentioned parameters. The *elementName* is the display name of the element. The *elementReferenceName* is the unique identifier within the notation. It can *extend* another element, taking the parameters of the extended element in addition to its own. It is also possible to *replace* an element to remove parameters that you do not want to use in your own notation. As in all descriptive concepts there is an option of adding comments. Lastly, an element can have any number of attributes defined, which we will explain later.

Connection is the base concept often referred to as 'arrow' (from 'blocks and arrows'). A connection has two extra parameters on top of the element parameters. The first parameter is *sourceElementReferenceName*, which defines the element of the source element, where the parameter matches the *elementReferenceName* of the referenced element. The second parameter is *targetElementReferenceName*, which defines the element of the target element just like the first parameter.

Typedef is the concept to represent a data type of an attribute (that in turn can be connected to an element or connection). The default supported types are *text*, *enumeration*, *boolean*, *date time*, *integer*, *URL*, and *UUID*. Within these types, restrictions on the types and the definition of the values of the enumeration can be added.

Toolbox focuses on the available elements and connections for a specific perspective. Similarly to element, Toolbox has *toolboxName*, *toolboxReferenceName*, and comments. The *toolboxContentNames* parameter allows the user to define which element and connections should appear. In addition, the *folder* parameter is used to create a categorisation in the toolbox.

```
virtualElement: VirtualKeyword ElementKeyword elementName
    elementReferenceName
   ElementKeyword elementReferenceName
   (ConnectionKeyword connectionReferenceName
    ElementKeyword elementReferenceName)+
   comment?
   ;
```

Listing 3.2. Virtual Element

Virtual Element is the concept of an element that starts to exist when a specific combination of elements and connections starts to exist and are connected. It has the *name* and *referenceName* parameters. In order to define which element and connection compose the virtual element, it has an *elementReferenceName* parameter, followed by one or more *connectionReferenceName* and *elementReferenceName* pairs. It contains the comments parameter like the previous concepts.

Rule is a way to check the restrictions of the model. By defining a logic query on the model, one can generate messages showing rules that have been violated. A rule is defined with a name, and an expression. This expression is a equivalent of a first-order logic expression. It also contains two optional parameters, namely a message can be added in case of rule violation, and comments.

Table is a visualisation of model information in a textual column-shaped format. Tables can be defined to list information in the scope of a repository, model, or a single diagram. Tables have name and Reference parameters. Their content is determined by a selectExpression parameters, which is like first-order logic, similar to rules. The display is handled by tableColumn definitions, which describe the name, span, and reference of the column.

Visual defines the shape of a model element or a model connection that can be displayed on a diagram. The parameters name and ReferenceName are present here. Additionally it has a parameter where a list of diagrams this element can appear on can be specified. The main focus of the visual definition is the visualisationScript, which is further explained in Sect. 6. Optionally there can be comments added.

Diagram is a special element that can hold other elements to visualise a specific perspective of the model. Diagram contains the name and ReferenceName parameters. It has a parameter for elementNames, which specifies the element that are meant to be on the diagram. This enables us to enforce a methodology, or let the user know when they have extra non-standard elements.

Cube is the multi-dimensional textual representation of a part of a model. Currently, a two-dimensional cube, matrix, is the only supported representation.

Behaviour defines the actions in the UI needed to realise the modelling process described in the methodology accompanying the notation. Behaviour is defined by specifying an action, such as adding an element to a diagram, or double clicking something, and a reaction, something that needs to happen in response to the action. It has contains the name parameter, and the behaviour rules parameter. The behaviour rules parameter is build up with an action and reference name, and a reaction and reference name. Optionally there is a final action such as alignment

Attributes can be added to elements, connections, folders, diagrams and virtual elements. The first parameter is the attributeName, which defines the attribute name. This is followed by the attributeType which can be a base type such as defined in typedef at the beginning of this list. In addition, it has the following three optional parameters. AttributeRequired, which defines if this is a mandatory attribute to fill, attributeDefault, which specifies the default value of the attribute, and comments.

Together, we expect these concepts to cover all of DEMO notation visualisations. When we are specifying DEMO 3 and 4 we will report on the current level of success on the specification.

4 DEMO 3 Notation

The notation of DEMO version 3.7 is formalised in DEMOSL [4] and some improvements were proposed in a PhD [7]. The formalisation of the DEMOSL enabled the automated verification and exchange of DEMO models and it has been implemented in the SEA add-on Plena. Whether the discussion on the relevance or usefulness of these concepts of DEMO is a discussion that can be done in another paper. We will focus on using the concepts and visualisation of these concepts as described in the above mentioned literature. The notation script of Simplified for DEMO 3 currently seems to contain all concepts of the aspect models CM, PM, and FM and we will explain the translation from DEMOSL to the notation script in this section. The DEMO 3 notation, as shown in listing 4.1, has been given the version number 3.7. This corresponds to the version of DEMOSL but does include the work of the mentioned PhD.

```
ScriptVersion01
Notation for DEMO version 3.7
comment "This version of DEMO is described in EO, TEOO and DEMOSL 3.7"
```

Listing 4.1. Script

Within DEMO 3, three enumerated data types exist as shown in listing 4.2. These data types describe all states of some attributes of the transaction kind, attribute type, and step kind.

```
typedef TRANSACTIONSORT ENUM (None, Original, Informational,
    Documental)
comment "The transaction sort type describes all possible Transaction
    sorts"
typedef ATTRIBUTEKIND ENUM (Original, Derived)
typedef STEPKIND ENUM (
  Initial, Request, Requested, Promise, Promised, Execute, Executed,
      State, Declare, Stated, Declared, Accept, Accepted,
  Decline, Declined, Quit, Quited, Reject, Rejected, Stop, Stopped,
  RevokeRequest, RevokedRequest, ...)
```

Listing 4.2. Type definitions

To prevent listing the whole notation in this paper we will summarise the element section of the DEMO 3 notation. The notation contains all DEMO 3 the elements, e.g. Actor, Entity Type, Attribute Type, Elementary Actor Role, Composite Actor Role, Transaction Kind, Aggregate Transaction Kind, and Transaction Process Step Kind. Special attention is needed for the Actor element as the DEMOSL does not specify this element. In the DEMO methodology the Actor-Function-Matrix [8, p.94] lists Actors that are linked to the Transaction Kinds. This Actor, as shown in listing 4.3, is not defined in the methodology but needs to be present to complete the connection between those concepts.

```
element Actor Actor37
    ( Name TEXT )
element "Transaction Kind" TransactionKind37
        ( Name TEXT
    , "Product Kind Name" TEXT
        , "Product Kind Formulation" TEXT
        , "Transaction Sort" TRANSACTIONSORT)
```

Listing 4.3. Elements

The connections within the model have been listed [7] and a summary of those findings are shown in Sect. 4. All connections between elements can be specified with the *connection* keyword, as shown in listing 4.4, and these have been specified in the notation script file (Table 1).

To → From ↓	ETK	ATK	EAR	CAR	ET	AT	CET	TPSK	AR
ETK	–	c	e	ce	–	–	–	–	–
ATK	–	–	–	–	–	–	–	–	–
EAR	ia	a	–	c	–	–	–	–	–
CAR	ia	a	–	c	–	–	–	–	–
ET	o	–	–	–	xsrg	–	–	–	–
AT	o	–	–	–	p	–	–	-	–
CET	o	–	–	–	–	–	–	–	–
TPSK	c	–	–	–	–	–	-	iy	tlwh
AR	–	–	–	–	W	–	–	–	–

Table 1. Element Property Types

	Property Types	
[c] contained in	[o] concerns	[i] initiator
[e] executor	[a] access to bank	[s] specialisation
[r] aggregation	[g] generalisation	[t] then
[l] else	[w] while	[h] when
[W] with	[x] excludes	[y] wait
[f] role of	[p] attribute of	

```
connection Initiator Initiator37e from ElementaryActorRole37 to
    TransactionKind37
connection Initiator Initiator37c from CompositeActorRole37 to
    TransactionKind37
connection "Attribute of Entity" AttributeOfEntity37 from
    AttributeType37 to EntityType37
```

Listing 4.4. Connections

The representation of models can be done in several ways. The most common notations involve diagrams and tables. To be able to verify the model we need to know what elements can exist on a diagram. This restriction does not limit the elements present on the diagram necessarily, but does give information about the verification of a 'pure' diagram. We have named the diagram according to the methodology [1] and match the elements present on those diagrams as shown in listing 4.5. We have chosen to have the connections allowed on the diagram to be derived from the elements that are allowed on the diagram. We could add them to the diagram as connection restrictions in a later stage but no specific requirement was found that did not allow for more connection types on a diagram.

```
diagram "Organisation Construction Diagram"
    OrganisationConstructionDiagram37
comment "OCD"
toolbox ToolboxOCD37
contains ( ElementaryActorRole37, CompositeActorRole37,
    TransactionKind37, AggregateTransactionKind37 )
```

Listing 4.5. OCD diagram

Before we show the definition of tables, we first introduce rules of a model. We have defined a grammar that mimics SQL and first order logic. Instead of adopting a complex specification language for rules we have started in a simplistic way to allow for the most common restrictions in a language. We have adopted the mathematical expressions 'for all', 'not exist' and 'exists' together with the logical 'and', 'or' and 'not'. The notation uses the mathematical representations of these terms (e.g. $\forall$, $\exists$, $\wedge$, $\vee$, !) which will not show up in the presented listings. When the logical rule collides with the given model, the message is presented to the user as shown in listing 4.6.

```
rule "No Reverse Association" (A connection Association(x, y) => !E
    connection Association(y, x))
Message "There can be no reverse association between two elements"
```

Listing 4.6. Rules

The same logical listing for rules is used in tables. This makes it possible to describe the table representation of a repository, model or diagram in a tabular format. This version has a one-dimensional representation where future versions are likely to have multi-dimensional representations allowing for more complex representation of model aspects like the Actor-Function-Table of DEMO.

```
table "Transaction Product Table" TransactionProductTable37
select (x."Identification",x."Name",x."Product Kind",x."Product Kind
    Formulation")
  (A element(x): x.Identification == "TransactionKind37")
column "transaction kind" span 2 data S0
column "tName" span 0 data S1
column "product kind" span 2 data S2
column "pName" span 0 data S3
```

Listing 4.7. TPT table

5 DEMO 4 Notation

The definition of the DEMO 4 notation has been derived from the notation of DEMO 3 in Sect. 4. DEMO 4 has an extra diagram notation compared to DEMO 3. DEMO4 also has some new connections that were not present in DEMO 3.

We will skip the parts of the notation specification that are the same as the DEMO 3 specification and show the DEMO 4 specific parts.

The elements of DEMO 4 are the same, but the names represent the DEMO 4 version, as shown in listing 5.1. The naming of some elements have changed like Aggregate Transaction Kind becomes Multiple Transaction Kind.

```
element EntityType EntityType40
       ( Identification TEXT
       , "Composite Entity" BOOL default "false" )
```

Listing 5.1. Element

The most challenging element is the Transactor. The transactor is the contraction of the Transaction Kind (TK) and the Elementary Actor Role (EAR), when these two elements are connected by an executor connection. In Fig. 1 we show a part of the metamodel of DEMO4 from DEMOSL 4.6.1. Furthermore, in the book [3], the diagram OCD is no longer discussed but the author has made it clear that DEMO 3 is still a valid notation. Additionally, we can read in the text on page 28 [3]: “Note that a transactor role is the combination of a transaction kind and the actor role that has its executor role.”.

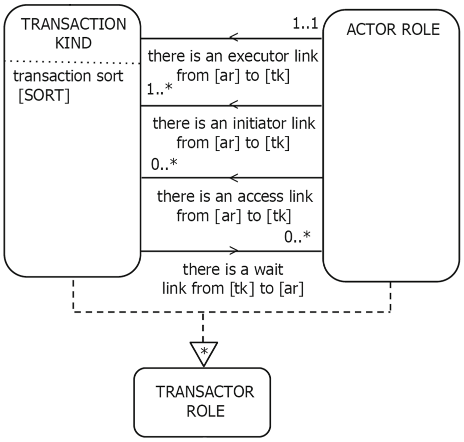
TRANSACTION KIND
transaction sort [SORT]
ACTOR ROLE
1..1
there is an executor link from [ar] to [tk]
1..*
there is an initiator link from [ar] to [tk]
0..*
there is an access link from [ar] to [tk]
0..*
there is a wait link from [tk] to [ar]
*
TRANSACTOR ROLE

Fig. 1. DEMO 4 metamodel on transactor [3]

The Coordination Structure Diagram (CSD) has been introduced to emphasise the tree structures in the construction and to show the transactors that form business processes together. That being said, we could be visualising a demo model in a OCD and a CSD simultaneously. In the OCD, the TK - executor - EAR are represented by an element - connection - element. In the CSD this construction is represented by a single element that has multiple parts. The upper part of the visual notation is the TK and the lower part is the EAR. But the element would be a single element on the diagram. To accomplish this we have created a virtual element. A virtual element is the combination of n+1 elements that are connected by n connections in a single line. On the one hand, when this combination of elements and connections start to exist the virtual element starts to exist. In addition, a virtual element that is created will create the underlying components simultaneously. On the other hand, when one of the components of the virtual element is removed from the model, the virtual element will cease to exist. Similarly, removing a virtual element will remove all of its components. In listing 5.2 the Transactor is defined for DEMO 4.

```
virtual element Transactor Transactor40
  element TransactionKind40
  connection Executor4e
  element ElementaryActorRole40
```

Listing 5.2. Virtual element

We could have solved this notation behaviour of a CSD with the behaviour statement of Sect. 6, but this virtual element concept can have some benefits for other notations in the future.

The TPT in DEMO4, as shown in listing 5.3, has a different meaning because the first letter is taken from the Transactor. Therefore, this table has a new definition in the notation.

```
table "Transactor Product Table" TransactionProductTable40
select (x."Name",x."Identification",x."Product Kind",x."Product Kind
     Formulation",z."Identification",z."Name")
 ( A element(x): x.elementname == "TransactionKind40"
 => E connection(y): y.Source == x.Id AND y.connectionName == "
      Executor"
 => E element(z): y.Target == z.Id AND z.refname == "
      ElementaryActorRole40"
 )
column "transaction kind" span 2 data S0
column "tName" span 0 data S1
column "product kind" span 2 data S2
column "pName" span 0 data S3
column "executor role" span 2 data S4
column "eName" span 0 data S5
```

Listing 5.3. TPT table

6 Notation Visualisation

Visualisation of models is a broader problem. In this paper we will keep the scope to the representation of model elements within a diagram representation of the model concepts. After defining the model concepts, we can define visualisations for these concepts. This visualisation is done by using a visual script which can contain a number of statements as described below. A visual script starts with an initialSizeStatement, specifying how large the objects should appear on the screen. Next, there are two possibilities. The first possibility is specifying penWidth, penColor, fillColor, and lineStyle, and subsequently defining a shape. The second possibility is using a groupStatement, which bundles together shapes defined within it. There are two types of groups, scaling, and non scaling. Scaling groups scale when the object would be scaled. Non-scaling groups do not scale and can be made by adding the noscale keyword, which is useful in certain cases such as text. Groups can contain other groups, but only if they are the same type, e.g. scale or noscale. Next, it is possible to define the shape that should be visualised. There are several basic shapes, and a free shape. We have a limited set of basic shapes, e.g. line, arc, polygon, rectangle and ellipse. The free shape can receive basic shapes, and will connect those to form one single closed shape. For these shapes, it is possible to define a minimum size using a minimumSizeStatement. Additionally, it is possible to display text, which is achieved using a printStatement.

Furthermore, there is the possibility of conditional visualisation using switch. The if statement can be accomplished by using a switch with a single case, since the default case is not required.

A certain concept can have multiple visualisations for multiple diagrams. This enables the modeller to define separate visuals for concepts such as a transaction kind, which has a different visualisation on an OCD compared to a Process Structure Diagram (PSD). Is is possible for an element to be visualised the same on all diagrams by substituting the diagramName by a star (*). In the model layer the visualisation of an element can occur multiple times on the same diagram.

In Simplified, the canvas has the positive x-axis to the right, and the positive y-axis down. Each grid square is 50×50 by default.

To accommodate the combination of the diagram and the elements and connections that can be on that diagram, the notation script can define the related elements and connections. The toolbox then can be presented to the user at the right moment while modelling. Though the toolbox shows the most likely options, other elements can be used in a notation when the diagram does not have to be pure for the methodology.

```
toolbox "OCD" ToolboxOCD37
comment "Toolbox with all elements for the OCD diagram"
(
  element TransactionKind37, element AggregateTransactionKind37,
  element ElementaryActorRole37, element CompositeActorRole37,
  connection Initiator37e, connection Initiator37c,
  connection Executor37e, connection Executor37c,
  connection Information37e, connection Information37c
)
```

Listing 6.1. Toolbox

In earlier versions of DEMO [1] the fact model has been modelled in a different notation. This former notation was closer to Object Role Modelling (ORM) and allowed for the attributes of entities to be modelled as elements in the fact model. The advantage of modelling attributes of entities as separate elements is the ability to reason about them. Connections to separated attributes can be made not only to the entity by the connection type 'attribute of entity' but also to other concepts. In order to model both earlier and current versions of DEMO, attributes can be visually modelled both as attributes of entities and as elements themselves while still using the same notation for the non-visual model.

The Action Rules Specification (ARS) is a concept that needs further research in itself. The complexity of action rules is bigger than a simple representation can visualise. Therefore, within this scope we just stick to the structure and a simple representation. For the ARS the 'with' specification [7] refers to attributes of entities. The connection from the 'when', 'then' and 'else' proposition to all relevant attributes can now be visually modelled and translated to a verbalisation and vise-versa. The visual challenge is to create a behaviour of the attribute element when it 'belongs' graphically to the entity. We have created the behaviour syntax to be able to define just that behaviour. This behaviour is of the same kind as the placing of activities on a swim-lane in a Business Process Model and Notation (BPMN) diagram.

```
behaviour "Attribute on Entity" on drop AttributeType37 on
    EntityType37
do link AttributeType37 to EntityType37
option align left,centre
```

Listing 6.2. Behaviour

Drawing all shapes of DEMO requires just a few mathematical basic shapes. The behaviour of these shapes, thought not all explicitly defined in DEMOSL 4.6.1, is quite challenging. We have decided to start with the basic set of figures and text that allows for defining the required shapes. The resizing of the transaction kind shape on a PSD is not yet optimised but will do the job in this first version. The conditional statements allow for changing the colour based on properties of the diagram or the element.

```
visualscript: initialSizeStatement vscrStatement* ;
visualScriptSettingStatements: vscrSStatement* ;
vscrSStatement:
      penWidthStatement |  penColorStatement
    | fillColorStatement | lineStyleStatement
    ;
vscrStatement:
      minimumSizeStatement | penWidthStatement
    | penColorStatement |  fillColorStatement
    | lineStatement |      arcStatement
    | anchorStatement |    polygonStatement
    | rectangleStatement | ellipseStatement
    | printStatement |     groupStatement
    | switchStatement |    lineStyleStatement
    | letStatement |       shapeStatement
    ;
```

Listing 6.3. Visual Grammar

The listing listing 6.4 is an example of the response link on a PSD. It will start with a circle and follows a solid line to the end of the connection with a rectangle at the end. All connection visualisations can have a line, begin, centre and end part as shown in Fig. 2 and Fig. 3.

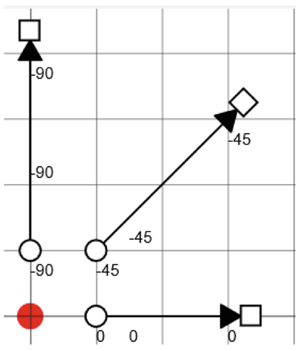

Fig. 2. Response link Simplified

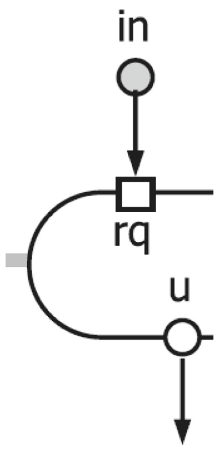

Fig. 3. Response link [3]

```
visual ResponseLinkPsd37 of ResponseLink37
on (ProcessStructureDiagram37)
    line {
          penwidth(2) / linestyle(solid)
          pencolor(0,0,0)
        }
        begin{
          initialsize(5,5) / fillcolor(255, 255, 255)
          ellipse(0, 0, 8, 8, 0)
        }
        end {
          initialsize(5,5) / fillcolor(0,0,0)
          polygon(0,0,3,10,90)
          fillcolor(255, 255, 255) / linestyle(solid)
          rectangle(10, -8, 15, 15, 0, 0)
        }
```

Listing 6.4. Visual Response link

For the preclusion connection, as shown in listing 6.5, a centre and end figure are used as shown in Fig. 4 and Fig. 5.

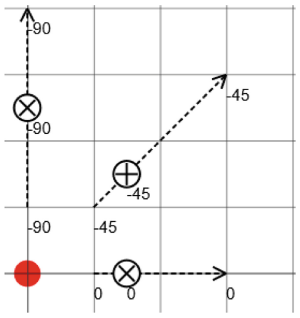

Fig. 4. Preclusion link Simplified

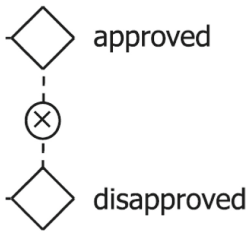

Fig. 5. Exclusion link [3]

```
visual PreclusionOfd37 of Preclusion37 on (ObjectFactDiagram37)
    line { penwidth(2)
      pencolor(0,0,0) / linestyle(dash) }
    centre { initialsize(5,5)    / linestyle(solid)
      ellipse(0,0,10,10,0)
      line(-6,-6,6,6,0) / line(-6,6,6,-6,0) }
    end { initialsize(5,5)    / penwidth(2)
      pencolor(0,0,0) / linestyle(solid)
      line(0, 0, -10, 5, 0) / line(0, 0, -10, -5, 0) }
```

Listing 6.5. Visual Preclusion

Elements also need a visualisation and the simplest visualisation is that of an EAR shown in listing 6.6. This is a square with the text in the square and some text beneath the square. As can be seen the graphical shapes will scale and the text will not scale along.

```
visual ElementaryActorRoleAll37 of ElementaryActorRole37 on (*)
{
  initialSize(50,50)
  group (0,0) scale {
    penWidth(2) / penColor(0,0,0)
    fillcolor(255,255,255) / rectangle(0,0,50,50,0,0)
  }
  group (0,0) noscale {
    print(10, 20, 40, 25, "{element.identification}", 0)
    print(-25, 60,100, 25, "{element.name}", 0)
  }
}
```

Listing 6.6. Visual EAR

All these visualisation concepts cover most of the CM, PM, FM but do not cover the AM fully yet.

7 Conclusion and Discussion

Creating a complete DEMO description in the Simplified Notation Script is on its way. Several aspects of the DEMO notation are not yet defined in the grammar of the notation script, but the main graphical representations are implementable in the notation. The notation script, as developed now, has been proved to be successful in the test model examples that we have created. Issues beyond the notationscript are withholding the release of the platform as a whole and do interfere with the public evaluation of the notationscript. Notwithstanding, this evaluation of the notationscript and its application shows that the first aspect model of DEMO can be successfully be created and used for practical modelling.

The notation will be enhanced in the coming period to accommodate all aspects of the DEMO notation. It is worth bearing in mind that the DEMO methodology comprises more than only the notation, e.g. process information. However, at this point in time we did not find a way to describe all this information in the notation script grammar. Therefore, some parts of the methodology support are still hard-coded until we have found a suitable way to describe the translation and generation of certain notation aspects.

The notation scripts will be published on a Git[1] to be used and improved by the community. This direct communication loop with the community will also benefit the specification of additional notations in Simplified in the future.

[1] https://gitlab.com/teec2/simplified/notations.

8 Future Research

During this attempt to describe DEMO in Simplified Notation Script we have come across several shortcomings that prevented us to be fully capable of modelling DEMO. First of all, the ARS grammar that was defined [7] is not included in the implementation yet and the visualisation of the ARS in an ARD is also not implemented yet either. Visualisation of the ARS is a subject that we will research further, parallel to the creation of the notation. Secondly, both the DEMO methodology and the aspect models with their specific visualisations make it quite challenging to make a solid definition to capture all concepts. Meaning that it will most likely be improved upon in the coming time. Lastly, compared to the hard-coded version of the DEMO modelling tool as an extension to SEA we still have to implement the generation options for the following diagrams: CSD, Transaction Pattern Diagram (TPD), Action Rules Process (ARP) and PSD.

Opportunities for improvement include the addition of a delete option to the notation script concepts as it currently supports only the definition, extension and replacement of the concepts. This would allow for a better accommodation of the language enhancements. Challenges of expanding the notation, and even connecting the notation to other notations are subjects that need more research.

Furthermore, the connection concept can be expanded upon. Currently, the grammar does not support multiple connections but the metamodel does support already n-ary connection. Although DEMO does not require n-ary connections in the current diagrams, we will have to extend the grammar to support this kind of connection for the addition of other diagrams and notations in the future. One of the concepts that needs the n-ary connections is the exclusion law in fact diagrams where two relations can be mutual exclusive.

We have only listed the research subjects of our R&D that are directly adjacent to the topic notation script.

References

1. Dietz, J.L.G.: Enterprise Ontology - Theory and Methodology. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-33149-2
2. Dietz, J.L.G., et al.: The discipline of enterprise engineering. Int. J. Organ. Design Eng. **3**(1), 86–114 (2013)
3. Dietz, J.L.G., Mulder, J.B.F.: Enterprise Ontology - A Human-Centric Approach to Understanding the Essence of Organisation. The Enterprise Engineering Series. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-38854-6
4. Dietz, J.L.G., Mulder, M.A.T.: Demo specification language 3.7 (2017)
5. Habermas, J.: The Theory for Communicative Action: Reason and Rationalization of Society, vol. 1. Boston Beacon Press, Boston (1984)
6. Mulder, M.A., Mulder, R., Bodnar, F., van Kessel, M., Gomez Vicente, J., et al.: The simplified platform, an overview. In: Modellierung 2022 Satellite Events (2022)
7. Mulder, M.: Enabling the automatic verification and exchange DEMO models. Ph.D. thesis, Radboud University Netherlands (2022)

8. Perinforma, A.P.C.: The Essence of Organisation, 3rd edn. Sapio, The Netherlands (2013)
9. van Reijswoud, V.E., Dietz, J.L.G.: DEMO Modelling Handbook, vol. 1, 2nd edn. Delft University of Technology (1999)
10. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43839-8