DEMOSL-3

DEMO Specification Language

(version 3.4, January 20156)

Jan L.G. Dietz

Introduction

This document presents the enterprise ontological models of DEMO-3, and the various ways of representing them in diagrams, tables and (formal) texts. The distinction between models and representations is important: they are the semantics and the syntax of the DEMOSL-3, the language for specifying graphically or textually DEMO models.

Expressions in DEMOSL are primarily represented in formal text. Therefore, the syntax of DEMOSL is defined in the Extended Backus-Naur Form (EBNF), the international standard syntactic meta language, defined in ISO/IEC 14977. However, many expressions also have a graphical equivalent.

Consequently, the DEMO diagrams should be understood as graphical representations, which are univocally transformable to textual DEMOSL expressions, and vice versa. Most people consider diagrams as very convenient model representations. But there are limits to their expressive power; they can easily become (too) complicated. Therefore, the DEMO diagrams are kept simple. What cannot simply be expressed in a diagram, must be expressed in a textual form. For that matter, the textual form of DEMOSL is quite intuitive: it looks like structured English.

The intended audience of this document is twofold. The first one consists of DEMO Professionals (Bachelors, Masters, Experts); they need to know DEMOSL profoundly. The second one are the developers of tools: modelling tools, simulation tools, etc. They need to have formal specifications of the syntax in order to verify the correctness of their software tools.

The sub models (1)

The ontological model of an organisation in DEMOSL-3 consists of the integrated whole of four sub models, each taking a specific view on the organisation: Construction Model, Action Model, Process Model and Fact Model. There are two ways of representing these models: *graphically* and *textually*.

Construction Model

The Construction Model (CM) of an organisation is the ontological model of its construction: the *composition* (the internal actor roles, i.e. the actor roles within the border of the organisation), the *environment* (i.e. the actor roles outside the border of the organisation that have interaction with internal actor roles), the *interaction structure* (i.e. the transaction kinds between the actor roles in the composition, and between these and the actor roles in the environment), and the *interstriction structure* (i.e. the information links from actor roles in the composition to internal transaction kinds and to external transaction kinds).

The CM of an organisation is represented in an *Organisation Construction Diagram* (OCD), a *Transaction Product Table* (TPT), and a *Bank Contents Table* (BCT).

Action Model

The Action Model (AM) of an organisation consists of a set of action rules. There is an action rule for every agendum kind for every internal actor role. An action rule specifies the (production and/or coordination) acts that must be performed, as well as the facts in the production world and/or the coordination world whose presence or absence in the state of the world must be assessed.

An AM is represented in *Action Rule Specifications* (ARS).

The aspect models (2)

Process Model

The Process Model (PM) of an organisation is the ontological model of the state space and the transition space of its coordination world. Regarding the state space, the PM contains, for all internal and border transaction kinds, the process steps and the existence laws that apply, according to the complete transaction pattern. Regarding the transition space, the PM contains the coordination event kinds as well as the applicable occurrence laws, including the cardinalities of the occurrences. The occurrence laws within a transaction process are fully determined by the complete transaction pattern. Therefore, a PSD contains only the occurrence laws between transaction processes, expressed in links between process steps. There are two kinds: response links and wait links (which represent interventions).

A PM is represented in a *Process Structure Diagram* (PSD), optionally complemented by a *Transaction Pattern Diagram* (TPD) for one or more of the transaction kinds.

Fact Model

The Fact Model (FM) of of an organisation is the ontological model of the state space and the transition space of its production world. Regarding the state space, the FM contains all identified fact kinds (both declared and derived), and the existence laws. Three kinds of existence laws are specified graphically: reference laws, unicity laws, and dependency laws; the other ones are specified textually. Regarding the transition space, the FM contains the production event kinds (results of transactions) as well as the applicable occurrence laws.

The transition space of the production world is completely determined by the transition space of its coordination world. Yet it may be illustrative to show the implied occurrence laws in an OFD.

The FM is represented in an *Object Fact Diagram* (OFD), possibly complemented by *Derived Fact Specifications* and *Existence Law Specifications*.

Basic terms and expressions in DEMOSL (1)

transaction kind id= "T", {digit}-; *Example: T17*
aggregate transaction kind id= "AT", {digit}-; *Example: AT2*
actor role id= "A", {digit}-; *Example: A17*
composite actor role id= "CA", {digit}-; *Example: CA1*
fact kind id= "F", {digit}-; *Example: F17*
product kind id= "P", {digit}-; *Example: P17*

transaction kind name = {lower case letter}-, {" ", {lower case letter}}; *Example: rental contracting*
actor role name = {lower case letter}-, {" ", {lower case letter}}; *Example: rental contracter*
object class name = {upper case letter}-, {" ", {upper case letter}}; *Examples: RENTAL, CAR GROUP*
fact kind name = {lower case letter}-, {" ", {lower case letter}}; *Examples: member, daily rental rate*

object variable = definite object variable | indefinite object variable | property variable;
definite object variable = upper case letter, {lower case letter}, {" ", upper case letter, {lower case letter}};
Examples: Person, Car Group, Year
indefinite object variable = "**some**", object class name;
*Examples: **some** PERSON, **some** YEAR*
property variable = "**the**", property kind name, "**of**" | "**in**" | "**on**", object variable;
*Example: **the** member **of** Membership*
*Example: **the** daily rental rate **of** Car Group **in** Year*
*Example: **the** number of cars **in** Car Group **on** Day*
*Example: **the** age **of** **some** PERSON*
product variable = < definite object variable regarding a product >;
Examples: Membership, Rental

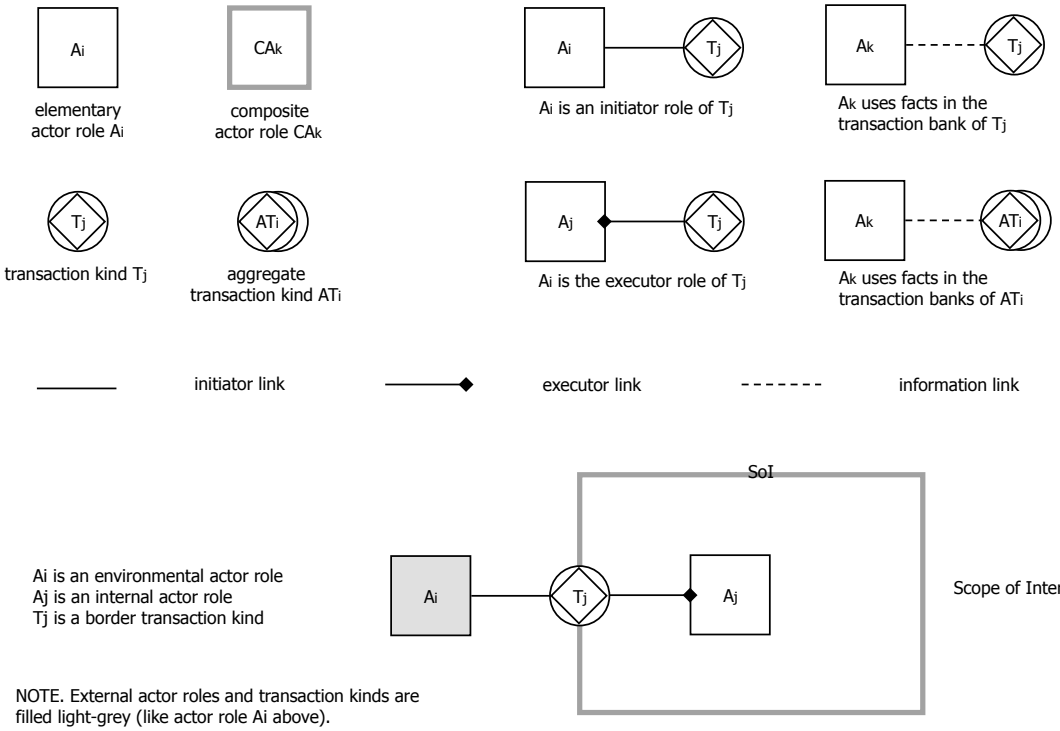
Basic terms and expressions in DEMOSL (2)

fact kind formulation = property kind formulation | attribute kind formulation | product kind formulation;
property kind formulation = property variable, "**is**", object variable;
*Example: **the** member **of** Membership **is** Person*
*Example: **the** daily rental rate **of** Car Group **in** Year **is** Money*
*Example: **the** son **of** Person **is** **some** PERSON*
attribute kind formulation = attribute variable, "**is equal to**" | "**is greater than**" | "**is less than**", value variable;
attribute variable = < property variable regarding an abstract object (= value) >;
value variable = < object variable regarding an abstract object (= value) > | value;
value = [dimension, ":"], (real | integer, unit) | boolean; < Cf. slide 15 >
Example: LENGTH: 2.4 m
Example: 2.4 m
product kind formulation= unary product kind formulation | binary product kind formulation;
unary product kind formulation = object variable | property variable, "**is**", perfect tense sentence;
*Example: Rental **is** contracted*
*Example: **the** first fee **of** Membership **is** paid*
binary product kind formulation = object variable | property kind variable, "**in**" | "**on**", object variable, "**is**", perfect tense sentence;
*Example: **the** fee **of** Membership **in** Year **is** paid*

present tense intention = "request" | "promise" | "state" | "accept" | "decline" | "quit" | "reject" | "stop" | "revoke" | "allow" | "refuse";
perfect tense intention = "requested" | "promised" | "stated" | "accepted" | "declined" | "quitted" | "rejected" | "stopped" | "revoked" | "allowed" | "refused";

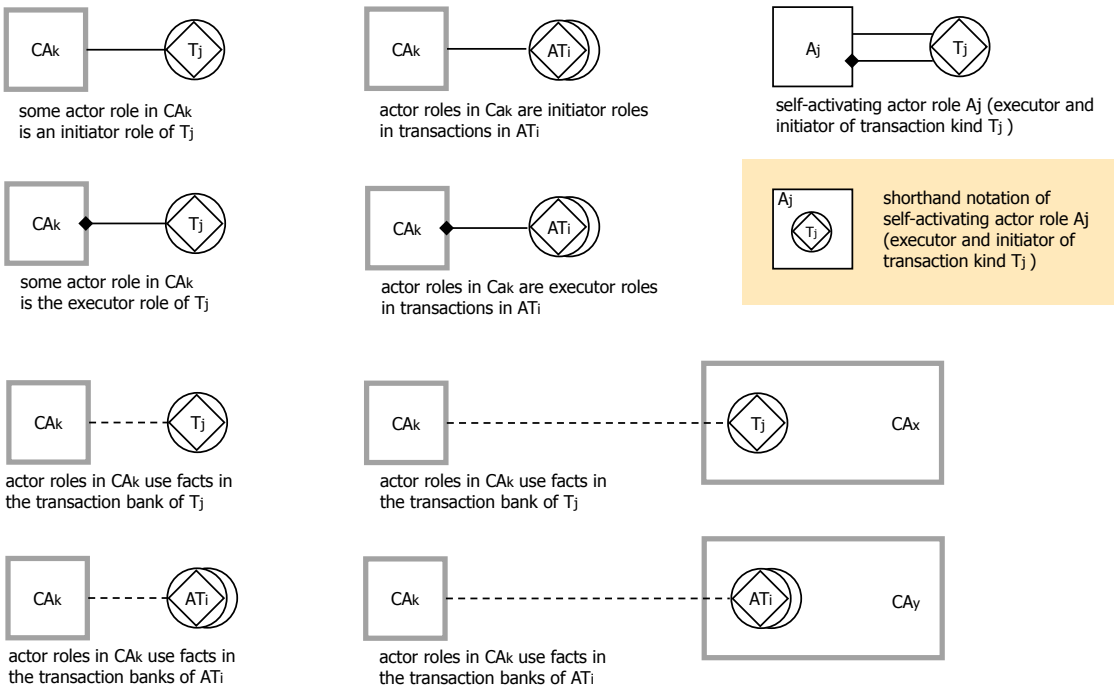
RESERVED WORDS
Reserved words are terms that have a definite meaning in DEMO. They are always underlined. Next to the 'present tense intention terms' and the 'perfect tense intention terms' (specified above), the next reserved words exist:
new, performer, addressee, production time

Legend of the Organisation Construction Diagram (1)



NOTE. External actor roles and transaction kinds are filled light-grey (like actor role Ai above).

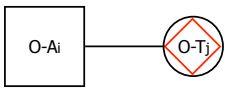
Legend of the Organisation Construction Diagram (2)



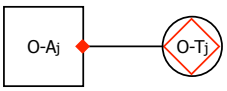
Legend of the Organisation Construction Diagram (3)

The *transaction sort* (original, informational or documental) of a transaction kind is indicated by coloring the diamond contour in the transaction symbol, as well as the 'executor diamond' on the edge of the actor role box red, green or blue respectively. Because aggregate transaction kinds may contain transaction kinds of various sorts, they are never colored.

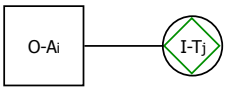
The *aspect organisation* (O-, I-, or D-organisation) to which a transaction kind and its executor role belong is indicated by adding the prefix O- or I- or D- to the respective id's. Examples: O-T17, I-T18, D-T19, O-A17, I-A18, D-A19.



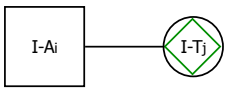
O-Ai is an initiator role of original transaction kind O-Tj



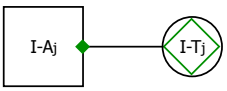
O-Ai is the executor role of original transaction kind O-Tj



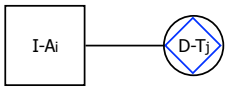
O-Ai is an initiator role of informational transaction kind I-Tj



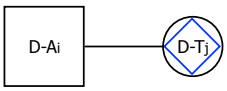
I-Ai is an initiator role of informational transaction kind I-Tj



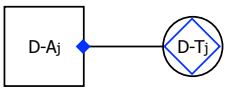
I-Ai is the executor role of informational transaction kind I-Tj



I-Ai is an initiator role of documental transaction kind D-Tj



D-Ai is an initiator role of documental transaction kindD-Tj



D-Ai is the executor role of documental transaction kind D-Tj

Action Rule Specifications

Below, the action rule specifications are defined in DEMOSL. Multiple use is made of terms that were already defined in slides 5 and 6, only to keep the definition as short as possible.

action rule specification = event part, assess part, response part;

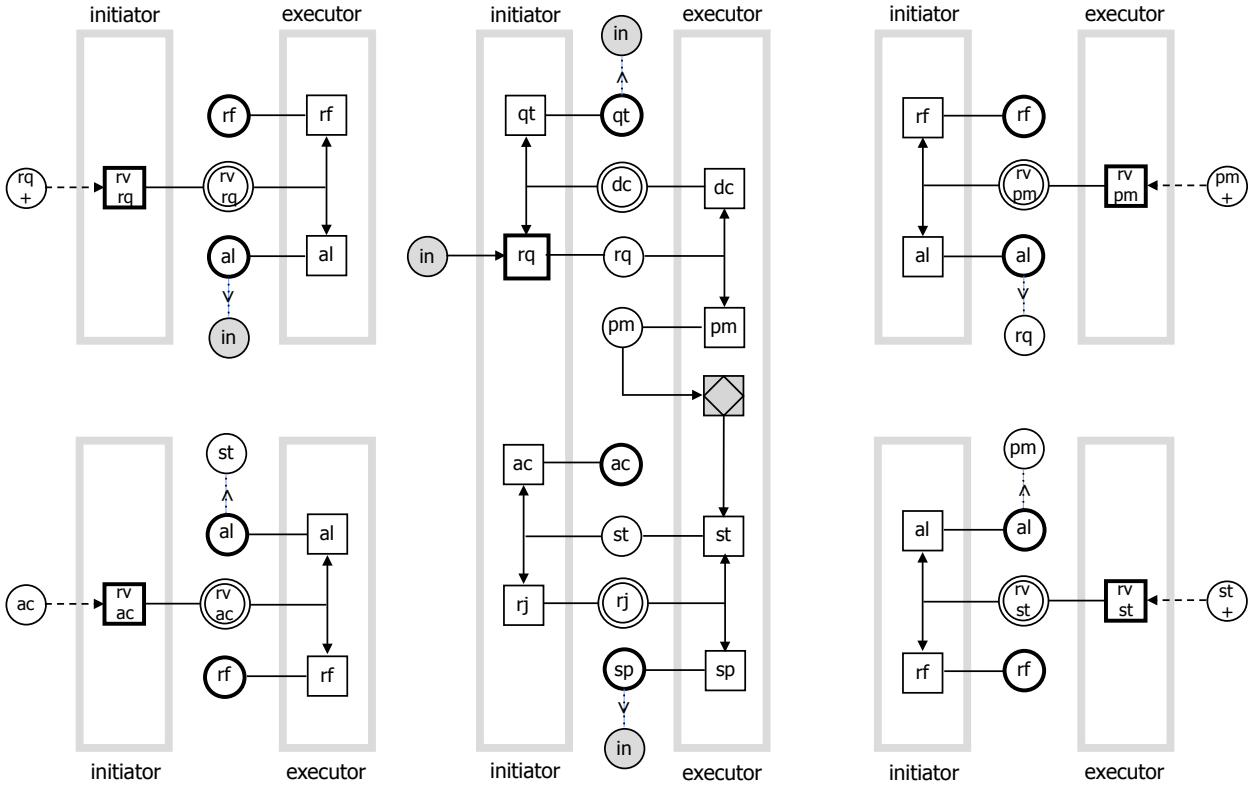
event part= agendum clause, [while clause], [with clause];
agendum clause = "**when**", new transaction reference, "is", perfect tense intention;
while clause = "**while**", {transaction reference, "is", perfect tense intention, {with clause};}-;
with clause = "**with**", {property kind formulation}-;

assess part = justice sub part, sincerity sub part, truth sub part;
justice sub part = "*justice:*", "<no specific condition>" | {fact kind formulation}-;
sincerity sub part = "*sincerity:*", "<no specific condition>" | {fact kind formulation}-;
truth sub part = "*truth:*", "<no specific condition>" | {fact kind formulation}-;

response part = "**if**", "complying with", present tense intention, "is considered justifiable", "**then**",
action clause, ["**else**" action clause];
action clause = {present tense intention, transaction reference, {with clause}};

transaction reference = transaction kind name, "**for**", product object reference;
new transaction reference = transaction kind name, "**for**", ["new"], product object reference;

The (extended) Transaction Process Diagram



Legend of the Transaction Process Diagram

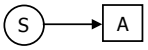
The pattern in the middle of the picture is the standard transaction pattern (STP). To save space, the symbols of the production act and the production fact are put together.

To the left and the right of this pattern are the four revocation patterns (rv-rq, rv-pm, rv-st, and rv-ac). They may be initiated from any status in the STP, except qt and sp, because these are transitory statuses (see below).

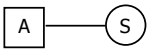
A white-filled box represents a *coordination act* (C-act), and a grey-filled box represents a *production act* (P-act). A white-filled disk represents a *coordination fact* (C-fact), and a grey-filled diamond represents the *independent production fact* or *product* of the transaction (P-fact). A double disk represents a discussion step.

Bold lined boxes represent the starting acts of a process. Bold lined disks represent terminal statuses.

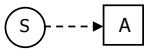
The combination of a C-act and its resulting C-fact is called a *process step*. Process steps are represented by a disk in a box. The corresponding compressed TPD is shown on the next slide,.



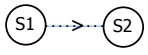
response link; A is performed in response to the occurrence of S



causal link; performing A causes status (C-fact) S



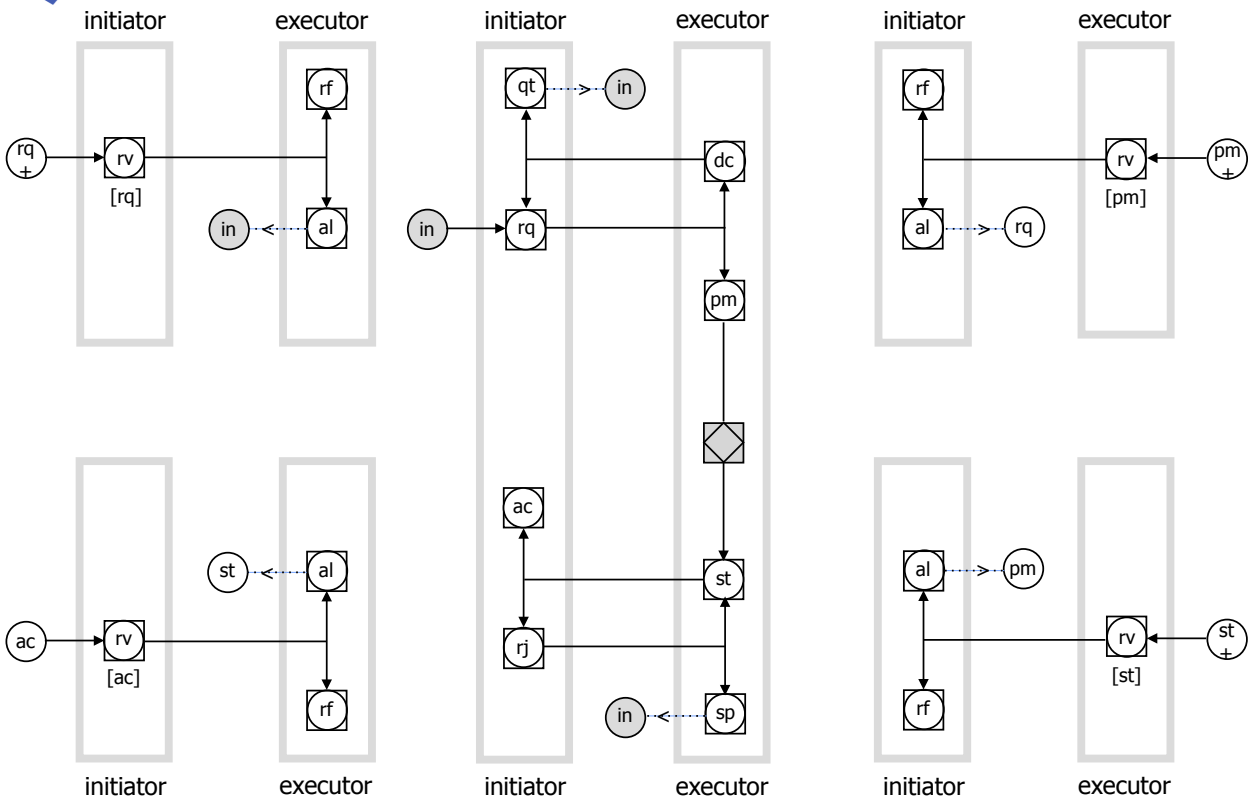
wait link; performing A has to wait for having reached status S



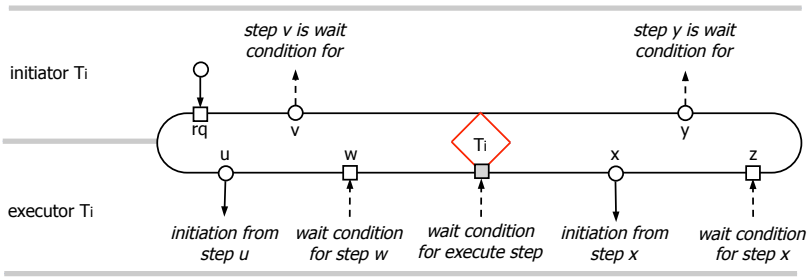
turn back link; reaching status S1 entails the instantaneous turning back to status S2; consequently, S1 is called a *transitory status*

S+ means: the (standard) process must be in status S or further

The compressed Transaction Process Diagram



Legend of the Process Structure Diagram (1)



k..n — the number of initiated transactions is minimally k and maximally n; the default values are 1..1

k..n — the number of waited for coordination events is minimally k and maximally n; the default values are 1..1

Note 1. There is a (non-proportional) linear time axis from left to right.

Note 2. A transaction proceeds in three phases: the *order phase* (left from the diamond), the *execution phase* (the diamond), and the *result phase* (right from the the diamond).

Note 3. The exhibited transaction kinds can also be of the sorts informational (green diamond) en documental (blue diamond)

Legend of the Process Structure Diagram (2)

A solid arrow represents an *initiation link*. It starts from a C-fact symbol and ends in the request act of some transaction kind. A C-fact symbol may have several outgoing initiation links. This means that transactions of several transaction kinds are initiated from it.

The request act must always be drawn within the swim lane of the initiator role.

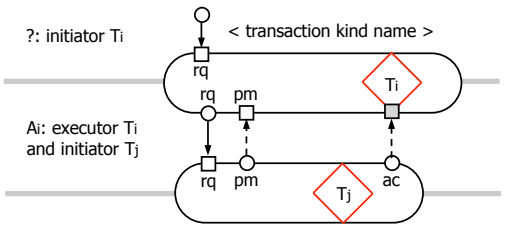
A dashed arrow represents a *wait link*. A wait link starts from a C-fact symbol and ends in a C-act symbol or in the execute act (grey-filled box). A C-fact symbol may have several outgoing wait links. It means the occurrence of the coordination event is a wait condition for the performance of several other process steps.

The C-fact symbols from which a wait link starts may be drawn on either the initiator or the executor side of the 'sausage'. However, they must be drawn in the proper phase (order phase or result phase).

A C-act symbol may have several incoming wait links. It means that performing the act has to wait for the occurrence of several coordination events. How exactly, is specified in the Action Model.

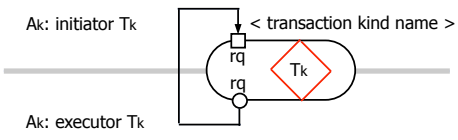
Legend of the Process Structure Diagram (3)

Example of transaction kind T_i with enclosed transaction kind T_j

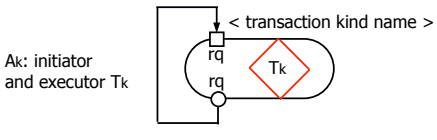


Note. The left and right bracket of the execution phase may be separated to show its duration.

Example of a self-initiating transaction kind T_k



Alternative notation:

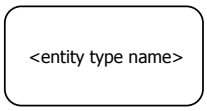


Legend of the Object Fact Diagram (1)

GRAPHICAL NOTATION of TYPES (CLASSES)

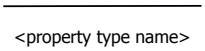
GRAPHICAL DECLARATION of TYPES (CLASSES)

NOTE: a type is the intension of a class, and a class is the extension of a type.
PRN PERSON = {x|person(x)} **PRN** person(x) ⇔ x ∈ PERSON



notation of the extension of an *entity type*

notation of an *event type*



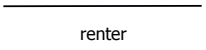
<attribute type name> [<dimension> : <unit>]

declaration of the entity type rental:
entity type rental **exists**

PRN *unary predicate* rental

declaration of the event type contracted:
event type contracted **exists**

PRN *unary predicate* contracted



declaration of the property type renter:
property type renter **exists**

PRN *binary predicate* renter

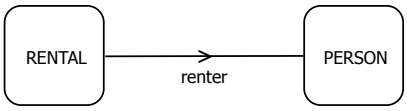
day of birth [JULIAN:day]

declaration of the attribute type day of birth:
attribute type day of birth **exists**

PRN *binary predicate* day_of_birth

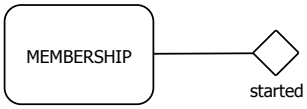
Legend of the Object Fact Diagram (2)

GRAPHICAL SPECIFICATION of REFERENCE LAWS



the *domain* **of** renter **is** RENTAL
the *range* **of** renter **is** PERSON

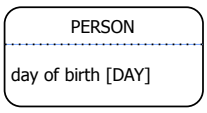
PRN ∀x,y: renter(x,y) ⇒ rental(x) ∧ person(y)



started *concerns* membership

PRN ∀x: started(x) ⇒ membership(x)

NOTE. Every event has the property <event time [JULIAN]>

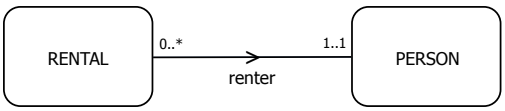


the *domain* **of** day of birth **is** PERSON
the *range* **of** day of birth **is** DAY

PRN ∀x,y: day_of_birth(x,y) ⇒ person(x) ∧ day(y)

Legend of the Object Fact Diagram (3)

GRAPHICAL SPECIFICATION of CARDINALITY LAWS

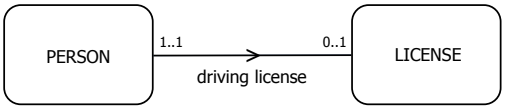


Every rental has exactly one person as renter
Every person is renter of zero, one or more rentals

NOTE. These are the default cardinalities; they may be omitted

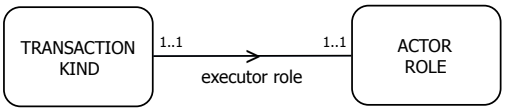
PRN ∀x,y: renter(x,y) ⇒ rental(x) ∧ person(y)
∀x,y,z: renter(x,y,) ∧ renter(x,z) ⇒ y=z
∀x ∈ RENTAL: ∃y ∈ PERSON: renter(x,y)

NOTE. Cardinalities are represented by "mincard .. maxcard"



Every person has zero or one driving licenses
Every license is the driving license of exactly one person

PRN ∀x,y: driving_license(x,y) ⇒ person(x) ∧ license(y)
∀x,y,z: driving_license(x,y,) ∧ driving_license(z,y) ⇒ x=z
∀y ∈ LICENSE: ∃x ∈ PERSON: driving_license(x,y)
∀x,y,z: driving_license(x,y,) ∧ driving_license(x,z) ⇒ y=z

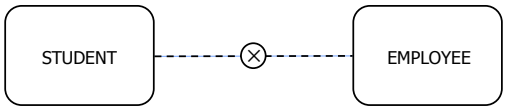


Every transaction kind has exactly one actor role as executor role
For every every actor role, there is exactly one transaction kind of which it has the executor role

PRN ∀x,y: executor_role(x,y) ⇒ x ∈ TK ∧ y ∈ AR
∀x,y,z: executor_role(x,y,) ∧ executor_role(x,z) ⇒ y=z
∀x ∈ TK: ∃y ∈ AR: executor_role(x,y)
∀x,y,z: executor_role(x,y,) ∧ executor_role(z,y) ⇒ x=z
∀y ∈ AR: ∃x ∈ TK: executor_role(x,y)

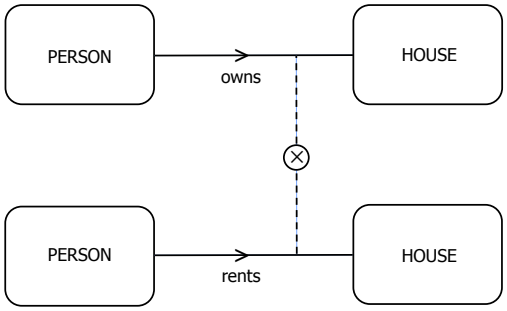
Legend of the Object Fact Diagram (4)

GRAPHICAL SPECIFICATION of EXCLUSION LAWS



if something is a student then it cannot be an employee,
and vice versa

PRN STUDENT ∩ EMPLOYEE = Ø



if a person owns a house, it cannot (also) rent a house,
and vice versa

PRN
∀x,y: owns(x,y,) ⇒ ¬ rents(x,y)
∀x,y: rents(x,y,) ⇒ ¬ owns(x,y)

Legend of the Object Fact Diagram (5)

TEXTUAL SPECIFICATION of EXISTENCE LAWS

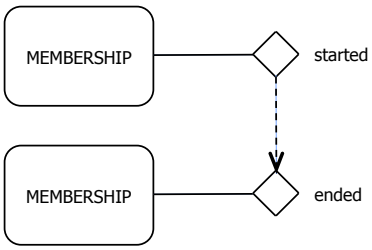
Existence laws that are not specified graphically, must be specified textually. Below, some examples are presented:

Membership **is** started **on** Day **implies that** Day **is the** first day **of some** Month **and** Month **is equal to or greater than** Current Month

Membership **is** started **on** Day **implies that the** age **of the** member **of** Membership **is equal to or greater than the** minimal age **in the** year **of** Day

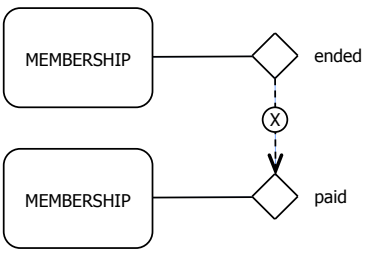
Legend of the Object Fact Diagram (6)

GRAPHICAL SPECIFICATION of PRECEDENCE LAWS and PRECLUSION LAWS



PRECEDENCE LAW

for all membership it holds that the occurrence of the event <membership is started> *precedes* the occurrence of the event <membership is ended>



PRECLUSION LAW

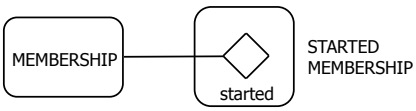
for all membership it holds that the occurrence of the event <membership is ended> *precludes* the occurrence of the event <membership is paid>

Legend of the Object Fact Diagram (7)

GRAPHICAL SPECIFICATION of SPECIALISATION, GENERALISATION, and AGGREGATION

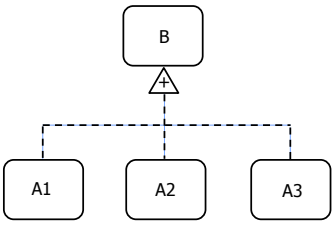
student **is a** *specialisation* **of** person

PRN STUDENT ⊆ PERSON



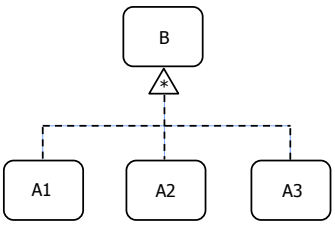
started membership **is a** *specialisation* **of** membership

PRN started_membership(x) ⇔ membership (x) ∧ started (x)
PRN STARTED_MEMBERSHIP⊆ MEMBERSHIP



graphical specification of the *generalisation* B
of the classes A1, A2, and A3.
This is indicated by the "+" sign in the triangle.

PRN B = A1 ∪ A2 ∪ A3



graphical specification of the *aggregation* B
of the classes A1, A2, and A3.
This is indicated by the "*" sign in the triangle

PRN B = A1*A2*A3

Legend of the Object Fact Diagram (8)

TEXTUAL SPECIFICATION of DERIVED TYPES

Derived types that are not specified graphically, must specified textually. Below, some examples are presented:

Person **is a** student ⇔ **there is a** Admission **and the** applicant **of** Admission **is** Person

the age **of** Person **on** Day **=** Day **minus the** day of birth **of** Person

the number of members **on** Day **= the cardinality of** STARTED MEMBERSHIP **on** Day

the first fee **of** Membership **=** (((12 **minus the** month **of the** starting day **of** Membership) **plus** 1) **divided by** 12) **times the** annual fee **in the** year **of the** starting day **of** Membership;

Legend of the Object Fact Diagram (9)

SCALES: dimensions, sorts, units and values

The next general scales are considered to exist always and everywhere; so, there is no need to declare them explicitly:

value type	dimension	unit(s)	base type	sort
duration	TIME	day, hour, minute, second, …..	integer	I
time	JULIAN	Julian day, hour, minute, second, …..	integer	I
amount	MONEY	dollar ($), euro (€) etc.	real	R
mass	MASS	… kg, g, mg, …	real	R
length	LENGTH	… m, cm, mm, …	real	R
area	LENGTH^2	… m^2, cm^2, mm^2, …	real	R
volume	LENGTH^3	… m^3, cm^3, mm^3, …	real	R
velocity	LENGTH/TIME	… m/s, …	real	R
temperature	TEMPERATURE	^{o}K, ^{o}C, ^{o}F	real	I
number	NUMBER	< just counting >	integer	A
truth value	BOOLEAN	logical value	{true, false}	B

Note 1. Each dimension belongs to a particular scale sort: Ordinal (O), Interval (I), Ratio (R), Absolute (A) or Boolean (B).
Note 2. Only the scale units in SI (Système International) are considered.
Note 3. It is allowed to omit the scale unit, so to only mention the scale dimension.
Note 4. It is allowed to abbreviate <dimension><unit> by only <unit> if no confusion can arise. The unit is then written in capital. Example: TIME:day may be abbreviated to DAY.
Note 5. The definitions of the time units larger than day (week, month, year, etc.) are dependent on the applied calendar. Therefore, they have to be defined additionally when needed.

Legend of the Transaction Product Table

The **Transaction Product Table** (TPT) is a table of transaction kinds and corresponding product kinds. The syntax of a TPT entry is defined in DEMOSL as follows:

TPT entry = transaction kind id, transaction kind name, product kind id, product kind formulation;

Examples:

transaction kind	**product kind**
T1 rental contracting	P1 Rental **is** contracted
T2 rental payment	P2 **the** rent **of** Rental **is** paid
T3 car pick up	P3 **the** car **of** Rental **is** picked up
T4 car drop off	P4 **the** car **of** Rental **is** dropped off
T5 penalty payment	P5 **the** penalty **of** Rental **is** paid

Legend of the Bank Contents Table

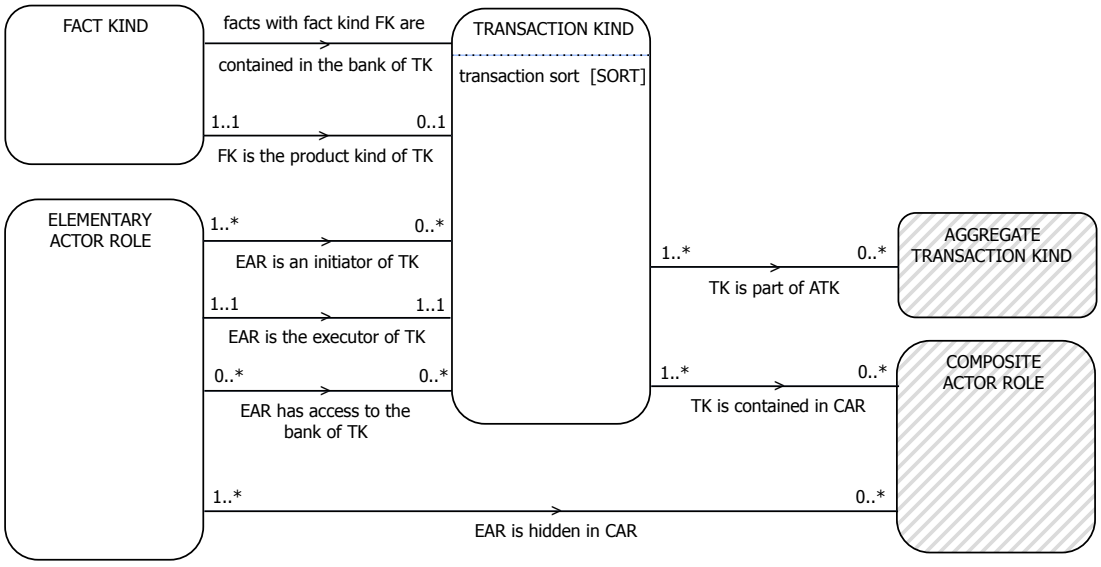
The **Bank Contents Table** (BCT) is a table that shows the fact kinds of which instances are contained in the transaction banks of the listed transaction kinds. The syntax of a BCT entry is specified in EBNF as follows:

BCT entry = (transaction kind id | aggregate transaction kind id),
(object class name | product kind formulation | property kind formulation);

Examples:

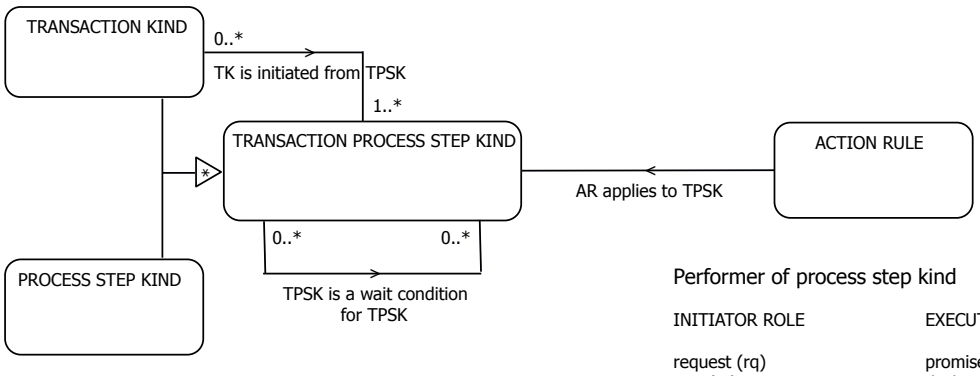
bank	independent/dependent facts
T1	MEMBERSHIP Membership **is** started **the** starting day **of** Membership **is** Day **the** member **of** Membership **is** Person
T2	**the** first fee **of** Membership **is** paid **the** amount paid **of** Membership **is** Money
AT1	**the** minimal age **in** Year **is** Integer **the** annual fee **in** Year **is** Money **the** max members **in** Year **is** Integer
AT2	PERSON **the** day of birth **of** Person **is** Day

Meta Model of the Construction Model



NOTE. The shapes of AGGREGATE TRANSACTION KIND and of COMPOSITE ACTOR ROLE are shaded grey to indicate that they are merely graphical notions. An aggregate transaction kind replaces a set of transaction kinds, and a composite actor role replaces a network of actor roles and transaction kinds.

Meta Model of the Process Model and Action Model



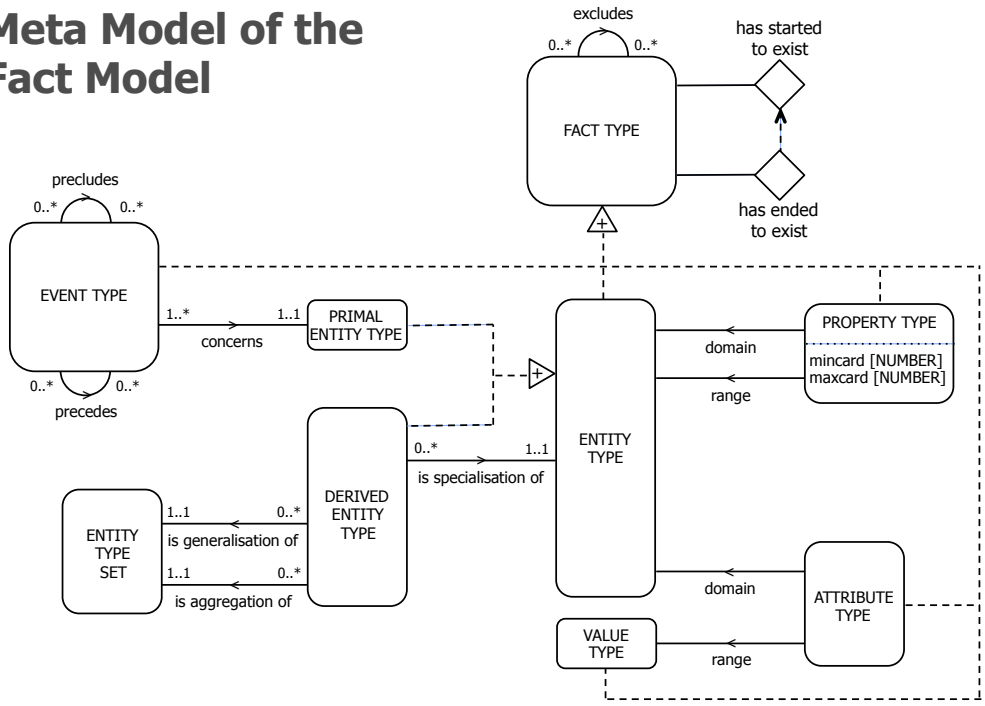
Performer of process step kind

INITIATOR ROLE	EXECUTOR ROLE
request (rq)	promise (pm)
quit (qt)	decline (dc)
accept (ac)	execute (□)
reject (rj)	state (st)
	stop (sp)
revoke-rq	revoke-pm
revoke-ac	revoke-st
allow revoke-pm	allow revoke-rq
refuse revoke-pm	refuse revoke-rq
allow revoke-st	allow revoke-ac
refuse revoke-st	refuse revoke-ac

NOTE1. Initiating transaction kind TK means performing the request in a TK (by the initiator role of TK) from some TPSK.

NOTE2. If the TPSK from which a TK is initiated is unknown, we say that TK is initiated externally. In the PSD, this is indicated by an empty disk.

Meta Model of the Fact Model



NOTE1. The property types <is specialisation of>, <is generalisation of> and <is aggregation of> exclude each other

NOTE2. Every event has the attribute <event time [JULIAN]>

NOTE3. A fact type is said to exist if the event <has started to exist> has occurred and the event <has ended to exist> has not yet occurred

NOTE4. An entity type set is a set of entity types (primal and/or derived)

