DEMOSL-3

DEMO Specification Language

(version 3.7, March 2017)

Jan L.G. Dietz
Mark A.T. Mulder

Introduction

This document presents the ontological models of DEMO-3, and the various ways in which they can be represented: diagrams, tables and (formal) textual expressions. The distinction between *models* and *representations* is important: they constitute respectively the semantics and the syntax of DEMOSL-3, the language for specifying graphically or textually DEMO-3 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 representations*, and vice versa.

Most people consider diagrams as the most convenient way of representing models. However, there are limits to their expressive power of diagrams; they can easily become (too) complicated. Therefore, the DEMO diagrams are kept simple. What cannot be said simply in a diagram, must be said in complementary textual expressions. The textual expressions in DEMOSL are quite intuitive: they look like structured English sentences.

The intended audience of this document is twofold. The first one consists of DEMO Professionals (Bachelors, Masters, Experts); they need to know DEMOSL profoundly for applying DEMO correctly. 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 aspect models

The ontological model of an organisation in DEMOSL-3 consists of the integrated whole of four aspect 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 (CM)

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 (AM)

The AM of an organisation is represented in *Action Rule Specifications* (ARS) and *Work Instruction Specifications* (WIS).

Process Model (PM)

The PM of an organisation 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 (FM)

The FM of an organisation is represented in an *Object Fact Diagram* (OFD), possibly complemented by the textual expressions of *Derived Fact Specifications* (DFS) and *Existence Law Specifications* (ELS).

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 concluding*
actor role name = {lower case letter}-, {" ", {lower case letter}};	*Example: rental concluder*
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*

entity variable = definite entity variable | indefinite entity variable | property variable;
definite entity variable = upper case letter, {lower case letter}, {" ", upper case letter, {lower case letter}} | "[", {lower case letter}-, {" ", {lower case letter}}, "]"
Examples: Person, [person], Car Group, [car group], Year, [year]
indefinite entity variable = "**some**", object class name;
*Examples: **some** PERSON, **some** YEAR*

property variable = "**the**", property kind name, "**of**" | "**in**" | "**on**", definite entity variable;
*Example: **the** member **of** Membership, **the** member **of** [membership]*
*Example: **the** daily rental rate **of** Car Group **in** Year, **the** daily rental rate **of** [car group] **in** [year]*
*Example: **the** number of cars **in** Car Group **on** Day, **the** number of cars **in** [car group] **on** [day]*

product variable = < definite entity variable regarding a product >;
Examples: Membership, [membership], Rental, [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= entity variable | property variable, "**is**", perfect tense sentence;

*Example: Rental **is** contracted*
*Example: **the** first fee **of** Membership **is** paid*
*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, et (event time), st (settlement time)

The Construction Model

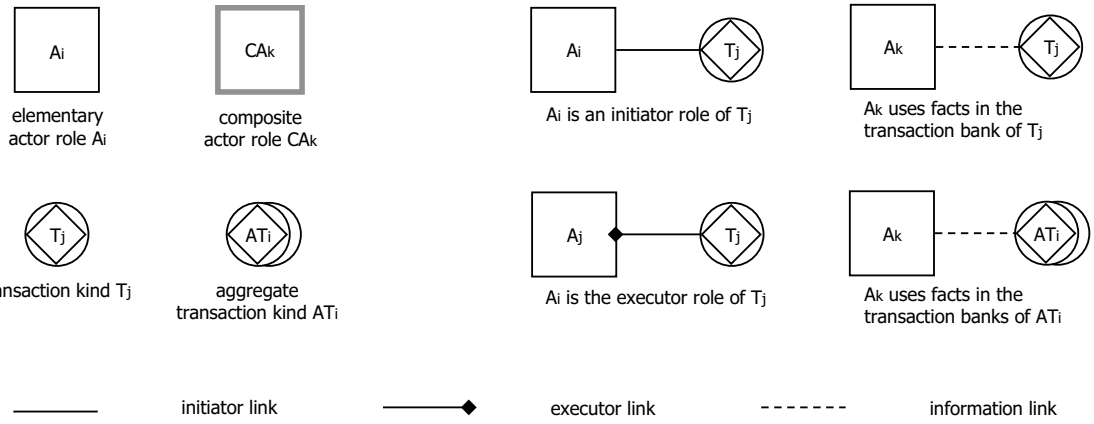
The Construction Model (CM) of an organisation is the ontological model of its construction: the *composition* (the internal actor roles, i.e. the actor roles within the border of the organisation), the *environment* (i.e. the actor roles outside the border of the organisation that have interaction with 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).

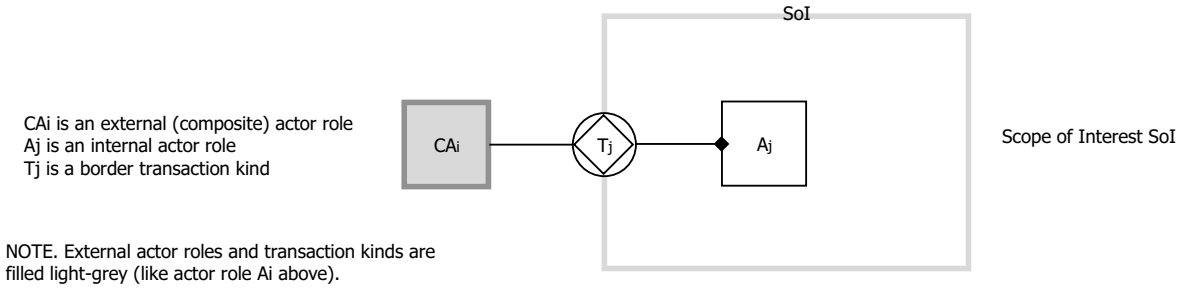
The CM of an organisation is represented in an *Organisation Construction Diagram* (OCD), a *Transaction Product Table* (TPT), and a *Bank Contents Table* (BCT).

The TPT is a cross-model table between the CM and the FM. More specifically, it relates the internal and border transaction kinds in the CM with the corresponding P-event types in the FM (which are called product kinds in the CM).

The BCT is a cross-model table between the CM and the FM. More specifically, it relates all transaction kinds (now interpreted as transaction banks) in the CM with the P-fact types in the FM, of which instances are contained in the transaction bank.

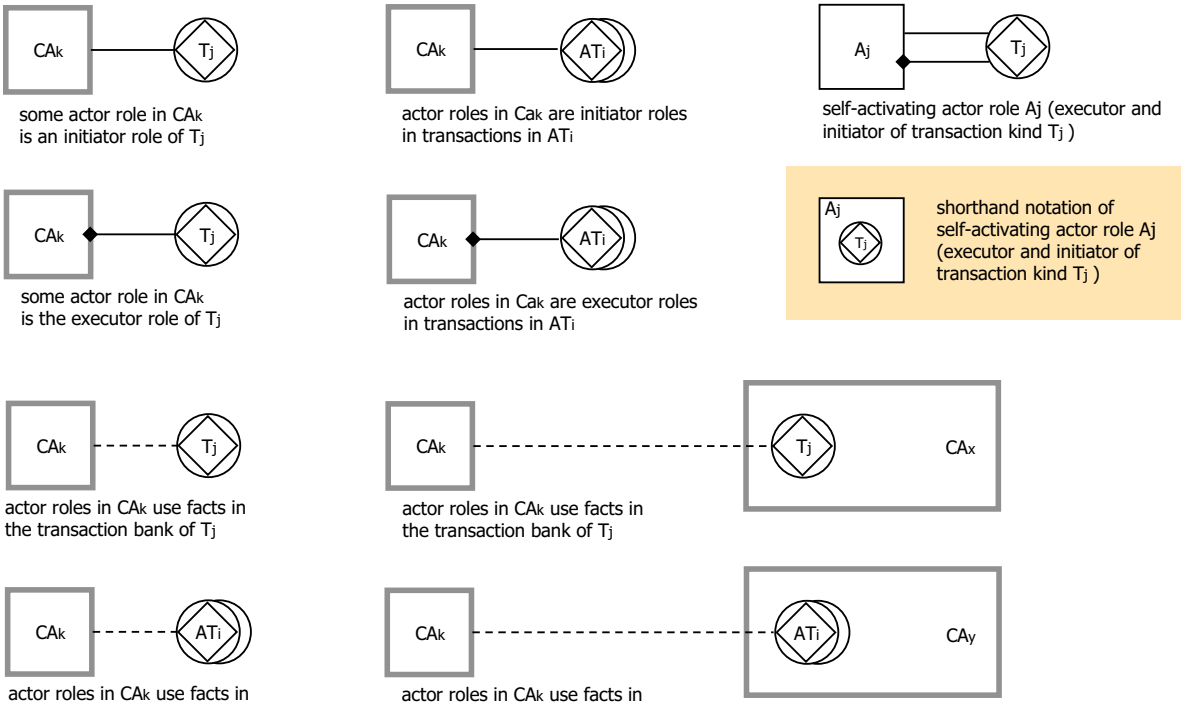
Legend of the Organisation Construction Diagram (1)





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

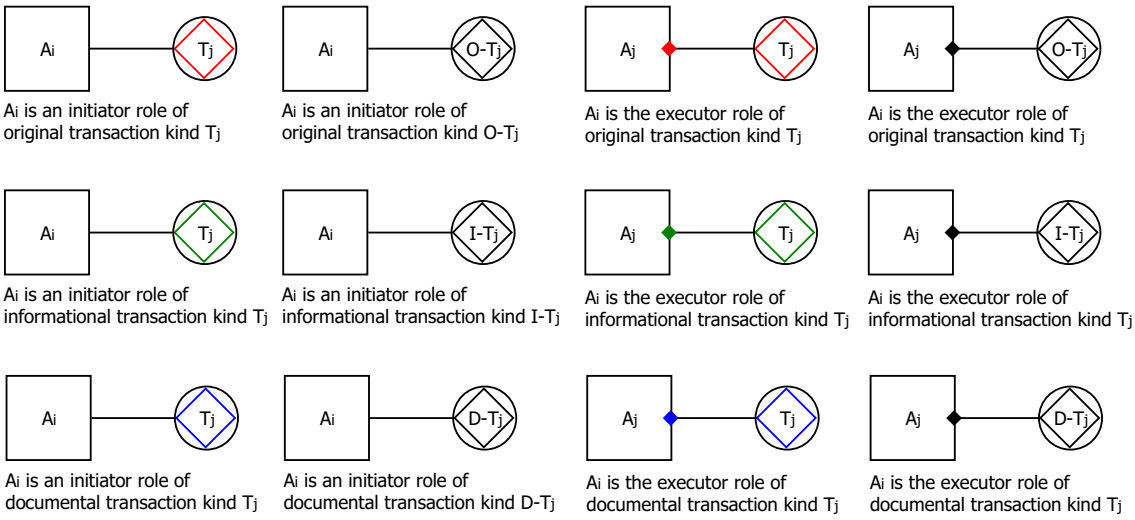
Legend of the Organisation Construction Diagram (2)



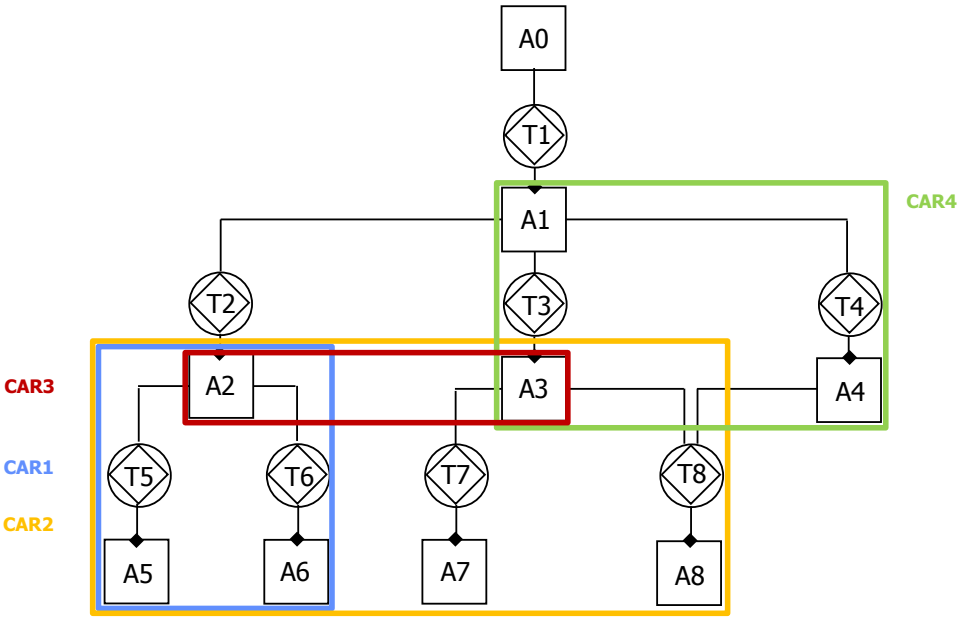
Legend of the Organisation Construction Diagram (3)

The *transaction sort* (original, informational or documental) of a transaction kind is indicated by colouring the diamond or its contour in the transaction symbol, as well as the 'executor diamond' on the edge of the actor role box red, green or blue respectively. The colours determine in addition the sort of the executor role.
Alternatively, the transaction sort can be indicated by adding the prefix O- or I- or D- to the respective transaction kind identifiers. The prefixes determine in addition the sort of the executor role. Examples: O-T17, I-T18, D-T19.

Because aggregate transaction kinds may contain transaction kinds of various sorts, they are not coloured.



Examples of composite actor roles



CAR1 has the executor role of T2.
CAR2 has the executor role of T2, T3 and T8.
CAR3 has the executor role of T2 and T3.
CAR4 has the executor role of T1.

CAR1 has no initiator roles in external transaction kinds.
CAR2 has no initiator roles in external transaction kinds.
CAR3 has the initiator role in T5, T6, T7 and T8.
CAR4 has the initiator role in T7 and T8.

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 concluding	P1 Rental **is** concluded
T2 rental payment	P2 **the** rent **of** Rental **is** paid
T3 car issuing	P3 **the** car **of** Rental **is** issued
T4 car returning	P4 **the** car **of** Rental **is** returned
T5 penalty payment	P5 **the** penalty **of** Rental **is** paid

transaction kind	product kind
T1 rental concluding	P1 [rental] **is** concluded
T2 rental payment	P2 **the** rent **of** [rental] **is** paid
T3 car issuing	P3 **the** car **of** [rental] **is** issued
T4 car returning	P4 **the** car **of** [rental] **is** returned
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

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]

The Action Model

The Action Model (AM) of an organisation is the ontological model of its operation. It consists of a set of *action rules* and a set of *work instructions*.

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.

Work instructions are optional. They guide the executor of a transaction in executing the production act.

An AM is represented in *Action Rule Specifications* (ARS) and *Work Instruction Specifications* (WIS).

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;

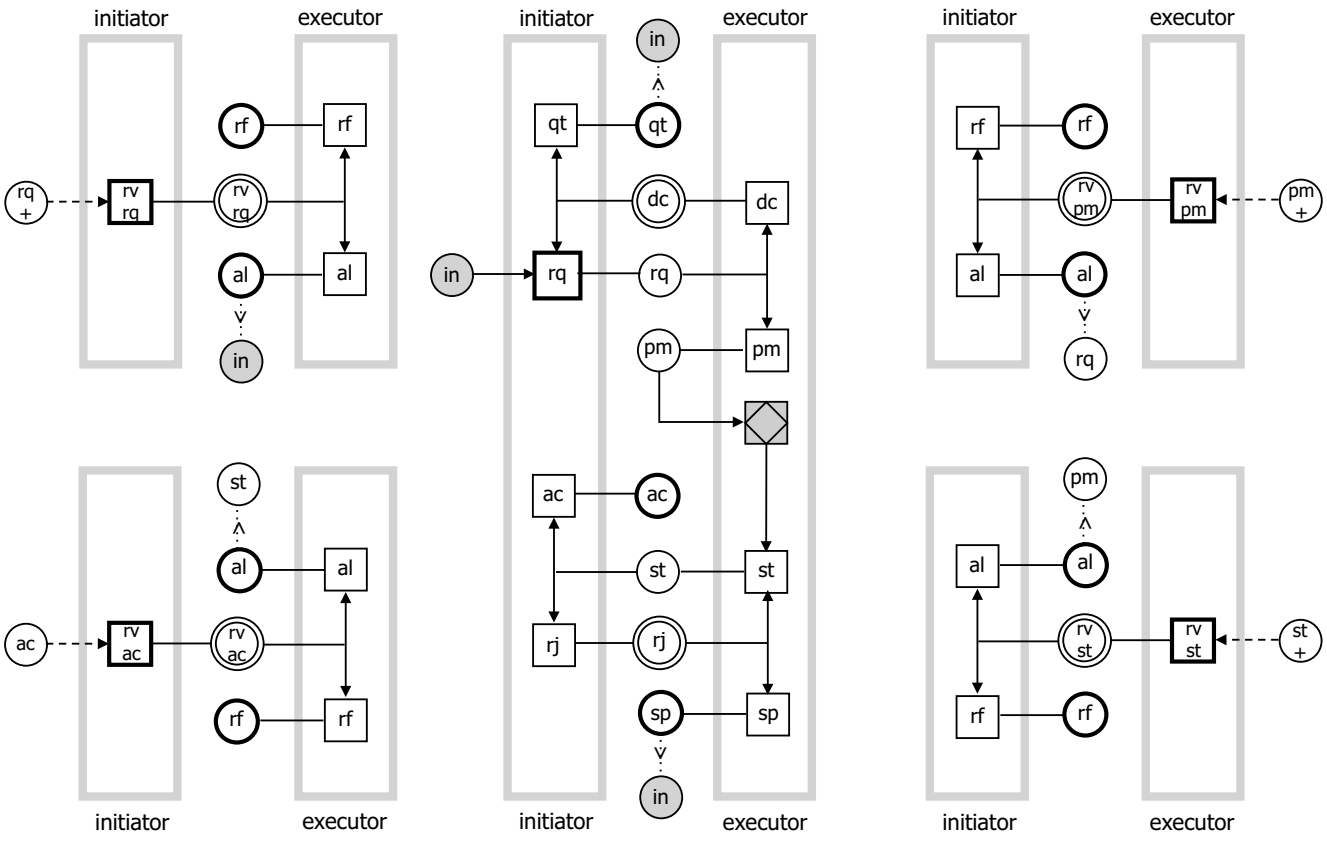
event part= agendum clause, [while clause], [with clause];
agendum clause = "**when**", new transaction reference, "is", perfect tense intention;
while clause = "**while**", {transaction reference, "is", perfect tense intention, {with clause};}-;
with clause = "**with**", {property kind formulation}-;

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 Complete Transaction Process Diagram



The 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 step kinds 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 intra-transaction occurrence laws are fully determined by the complete transaction pattern. Therefore, a PSD contains only the inter-transaction occurrence laws, expressed in links between process steps. There are two kinds of links: response links and wait links.

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.

Legend of the Transaction Process Diagram

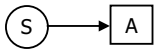
The pattern in the middle of the picture is the standard transaction pattern (STP). To save space, the symbols of the production act and the production fact are put together.

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).

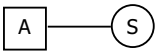
A white-filled box represents a *coordination act* (C-act), and a grey-filled box represents a *production act* (P-act). A white-filled disk represents a *coordination fact* (C-fact), and a grey-filled diamond represents the *independent production fact type* or *product kind* of the transaction. A double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

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

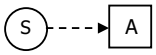
The combination of a C-act and its resulting C-fact is called a *process step*. Process steps are represented by a disk in a box. The corresponding compressed TPD is shown on the next slide,.



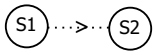
response link; A is performed in response to the occurrence of S



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



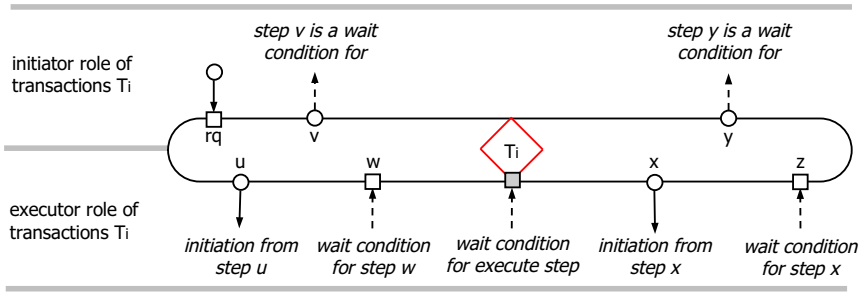
wait link; performing A has to wait for having reached status S



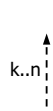
reversion link; reaching status S1 in some process entails the instantaneous reversion to status S2 in another process

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

Legend of the Process Structure Diagram (1)



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



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)

The sausage-like shape of a transaction kind in the PSD represents the complete transaction pattern. Every C-act kind and C-fact kind that is relevant for the relationships of transactions of a transaction kind with transactions of other transaction kinds is represented by its shape (small box or small disk) on the border of the sausage-like shape. The shapes must be drawn at the side of the corresponding phase (order phase or result phase).

A solid arrow represents an *initiation link*, which is a response link from an external C-fact kind. It starts from the shape of this C-fact kind and ends in the shape of the request act of a transaction kind. A C-fact shape may have several outgoing initiation links, meaning 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 kind shape and ends in a C-act kind shape or in the execute act kind shape (a grey-filled box). It means that the occurrence of an event of the C-fact kind is a wait condition for performing an act of the C-act kind.

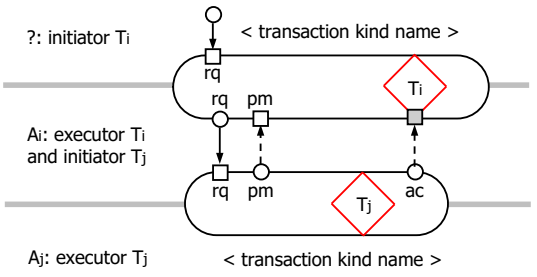
A C-fact kind shape may have several outgoing wait links. It means that the occurrence of an event of the C-fact kind is a wait condition for the performance of acts of several C-act kinds.

The C-fact kind shapes 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 P- or C-act kind shape may have several incoming wait links. It means that performing the act has to wait for the occurrence of a number of coordination events.

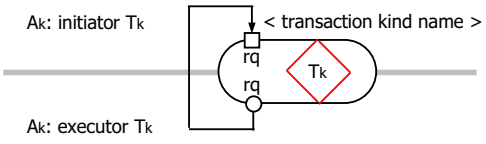
Legend of the Process Structure Diagram (3)

Example of transaction kind T_i with enclosed transaction kind T_j

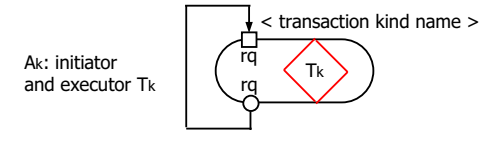


Note. The left and right bracket of the execution phase may be separated to show its duration.

Example of a self-initiating transaction kind T_k



Alternative notation:



The 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 P-fact types (both base and constructed), and the existence laws: reference laws, unicity laws, and dependency laws. Regarding the transition space, the FM contains the production event types, as well as the applicable occurrence laws. The production event types are identical to the transaction product kinds in the CM (TPT).

The transition space of the production world is completely determined by the transition space of its coordination world. Yet it may be convenient 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 the (textual) *Existence Law Specifications* that cannot be expressed in the OFD.

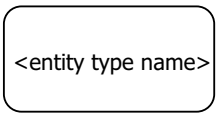
Legend of the Object Fact Diagram (1)

GRAPHICAL NOTATION of TYPES and CLASSES

GRAPHICAL DECLARATION of TYPES and CLASSES

PRN: PERSON = {x|person(x)}

PRN: person(x) ⇔ x ∈ PERSON



notation of the extension of an entity type

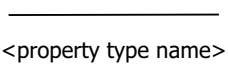
declaration of an entity type:
entity type 'rental' **exists**

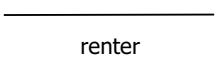
PRN
unary predicate 'rental'

notation of an event type

declaration of an event type:
event type 'concluded' **exists**

PRN
unary predicate 'concluded'





declaration of a property type:
property type 'has as renter' **exists**

PRN
binary predicate 'renter'

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

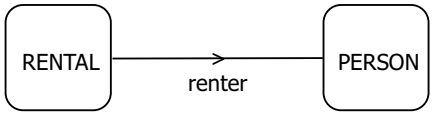
day of birth [JULIAN:day]

declaration of an attribute type:
attribute type 'day of birth' **exists**

PRN
binary predicate 'day_of_birth'

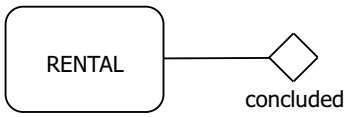
Legend of the Object Fact Diagram (2)

GRAPHICAL SPECIFICATION of REFERENCE LAWS



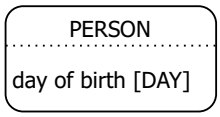
the domain of 'renter' is 'rental'
the range of 'renter' is 'person'

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



'concluded' concerns 'rental'

PRN
∀x: concluded(x) ⇒ rental(x)

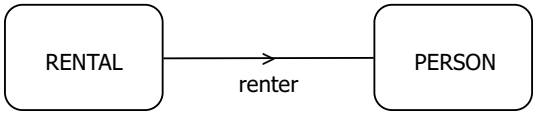


the domain of 'day of birth' is 'person'
the range of 'day of birth' is 'day'

PRN
∀x,y: day_of_birth(x,y) ⇒ person(x) ∧ day(y)
∀x,y,z: day_of_birth(x,y,) ∧ day_of_birth(x,z) ⇒ y=z
∀x ∈ PERSON: ∃y ∈ DAY: day_of_birth(x,y)

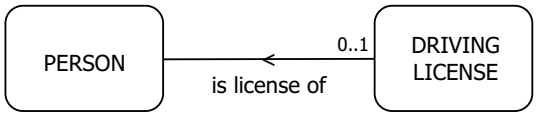
Legend of the Object Fact Diagram (3)

GRAPHICAL SPECIFICATION of CARDINALITY LAWS



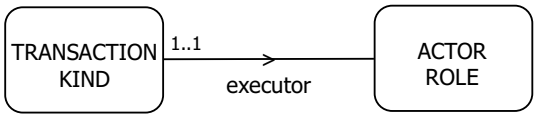
Every rental has exactly one person as the renter.
Every person is renter of zero, one or more rentals.

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



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

PRN
∀x,y: is_license_of(x,y) ⇒
driving_license(x) ∧ person(y);
∀x,y,z: is_license_of(x,y) ∧ is_license_of(x,z) ⇒ y=z
∀x ∈ DRIVING_LICENSE: ∃y ∈ PERSON: is_license_of(x,y)
∀x,y,z: is_license_of(x,y) ∧ is_license_of(z,y) ⇒ x=z

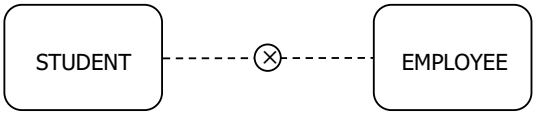


Every transaction kind has exactly one actor role as the executor role
For every every actor role, there is exactly one transaction kind in which it has the executor role

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

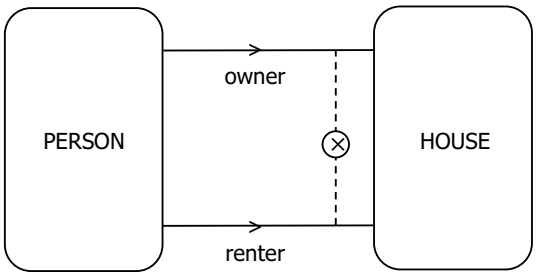
Legend of the Object Fact Diagram (4)

GRAPHICAL SPECIFICATION of EXCLUSION LAWS



PRN
STUDENT ∩ EMPLOYEE = Ø

a thing cannot at the same time be a student and an employee

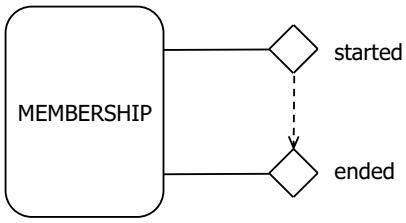


PRN
∀x,y: owner(x,y,) ⇒ ¬ renter(x,y)
∀x,y: renter(x,y,) ⇒ ¬ owner(x,y)

a person cannot at the same time be the owner of a house and the renter

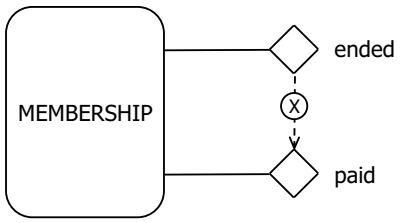
Legend of the Object Fact Diagram (5)

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 (future) 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 (future) occurrence of the event <membership is paid>

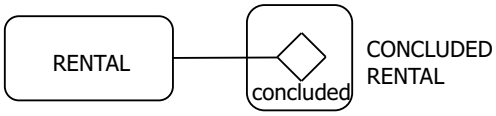
Legend of the Object Fact Diagram (6)

GRAPHICAL SPECIFICATION of SPECIALISATION

the entity type named "student"
is a specialisation of
the entity type named "person"

the rule that determines when a person
is (also) a student, has to be formulated
yet (probably in a textual way)

PRN
STUDENT ⊆ PERSON



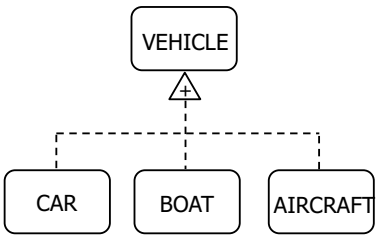
the entity type named "concluded rental"
is a specialisation of
the entity type named "rental"

the rule that determines when a rental
is (also) a concluded rental, is fully formulated,
namely in a graphical way

PRN
concluded_rental(x) ⇔ rental(x) ∧ concluded(x)
CONCLUDED_RENTAL⊆ RENTAL

Legend of the Object Fact Diagram (7)

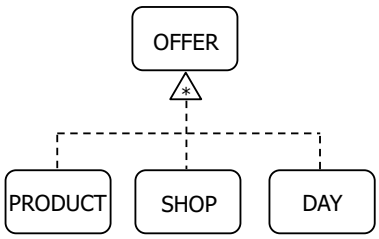
GRAPHICAL SPECIFICATION of GENERALISATION and AGGREGATION



VEHICLE *is a generalisation of*
CAR, BOAT and AIRCRAFT.
This is indicated by the "+" sign in the triangle.

Expressed in mathematical set theory, VEHICLE
is the union of CAR, BOAT and AIRCRAFT.
For every element x ∈ VEHICLE it holds that
x ∈ CAR or x ∈ BOAT or x ∈ AIRCRAFT.

PRN: B = A1 ∪ A2 ∪ A3



OFFER *is an aggregation of*
PRODUCT, SHOP and DAY.
This is indicated by the "*" sign in the triangle

Expressed in mathematical set theory, OFFER is
the cartesian product of PRODUCT, SHOP and DAY.
OFFER is a set of tuples (x, y, z) with
x ∈ PRODUCT, y ∈ SHOP, and z ∈ DAY.

PRN: B = A1*A2*A3

Legend of the Object Fact Diagram (8)

SCALES: dimensions, sorts, units and values

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. It is allowed to omit the scale unit, so to only mention the scale dimension.
Note 3. 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 4. 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.

Specification of the meta models in GOSL

The legends of the four aspect models of DEMO, as presented in the previous slides, can also be expressed in the General Ontology Specification Language GOSL, as proposed and discussed in the MU theory [TEE-05]. The resulting so-called *meta models* are presented in the next slides. The advantage of doing this, is that one can more clearly understand the relatedness of the four aspect models. In addition, it emphasises that the particular ways of expressing the models must be separated from their ontological meaning.

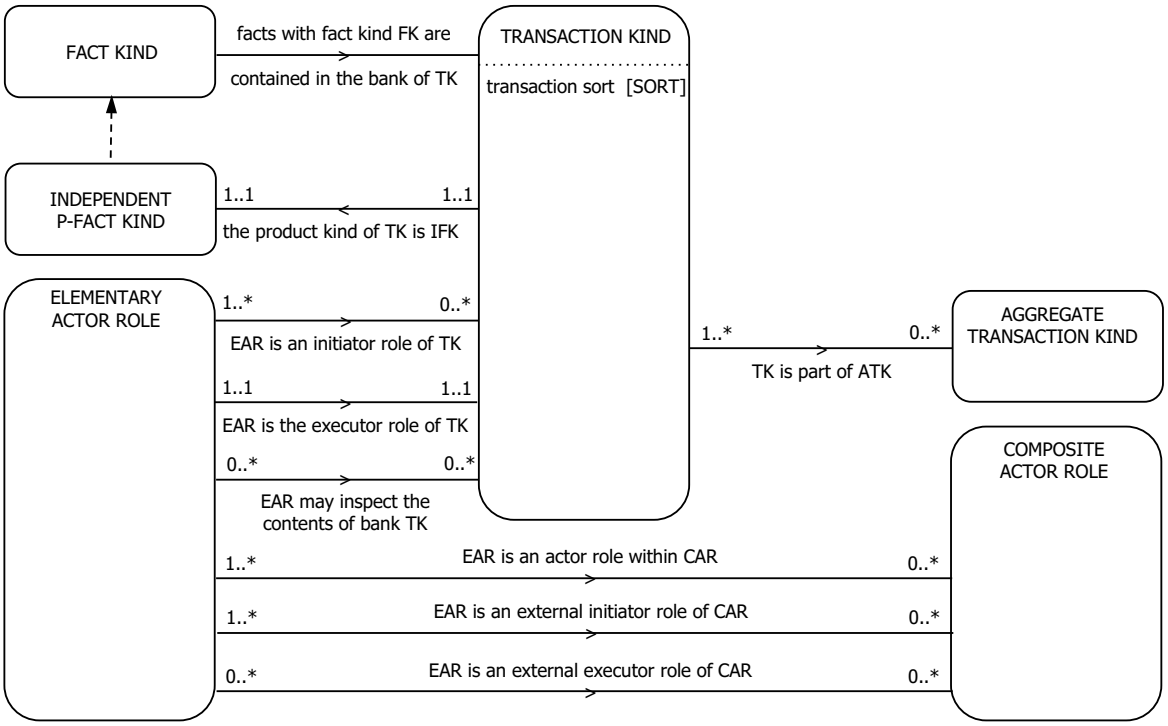
Of course, GOSL includes necessarily also a particular way of expressing meanings, notably a particular diagramming technique. Fortunately, it is the same one as is used above in specifying the Fact Model, notably the OFD. Defining the OFD in GOSL therefore yields the meta schema of all models (cf MU theory [TEE-05]).

As a help in understanding the meta models, one should apply the sapience "verification by instantiation" (from the book "The Essence of Organisation"). It means that one produces a few example instances of all the fact types in a meta model.

Note that every specific transaction is not an instance of a transaction kind but of the type 'transaction'. One of the properties of this transaction instance is its transaction kind, e.g. T1. A similar reasoning holds for actor role, fact kind, process kind, process step kind, etc.

The meta model of the FM is at the same time the meta model of all models (cf MU theory [TEE-05]) since it specifies the state space and transition space of any world. So, possible instances of the concept 'entity type' in this meta meta model, are (the types) transaction, transaction kind, actor, actor role, product, product kind, etc.

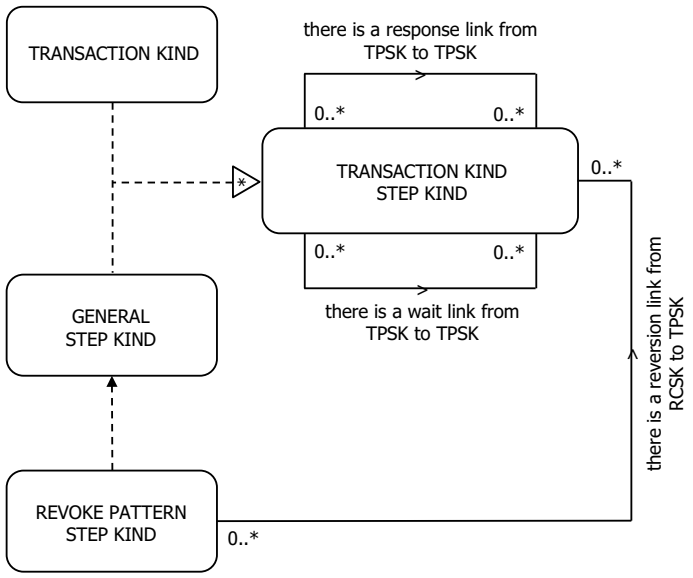
Meta Model of the Construction Model



NOTE1. By fact in "FACT KIND" is meant both coordination fact and production fact. The specialisation concerns only P-facts.
NOTE2. Every INDEPENDENT P-FACT KIND is identical to a P-EVENT TYPE in the meta model of the Fact Model.
NOTE3. An AGGREGATE TRANSACTION KIND comprises a set of transaction kinds.
NOTE4. A composite actor role comprises a set of actor roles, including the transaction kinds between them.

Meta Model of the Process Model

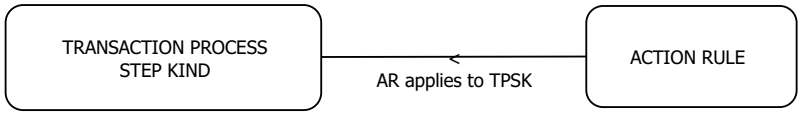
Examples of transaction process step kinds are: T01/rq, T01/ex, T01/st, T01/rv-rq, T01/rv-st



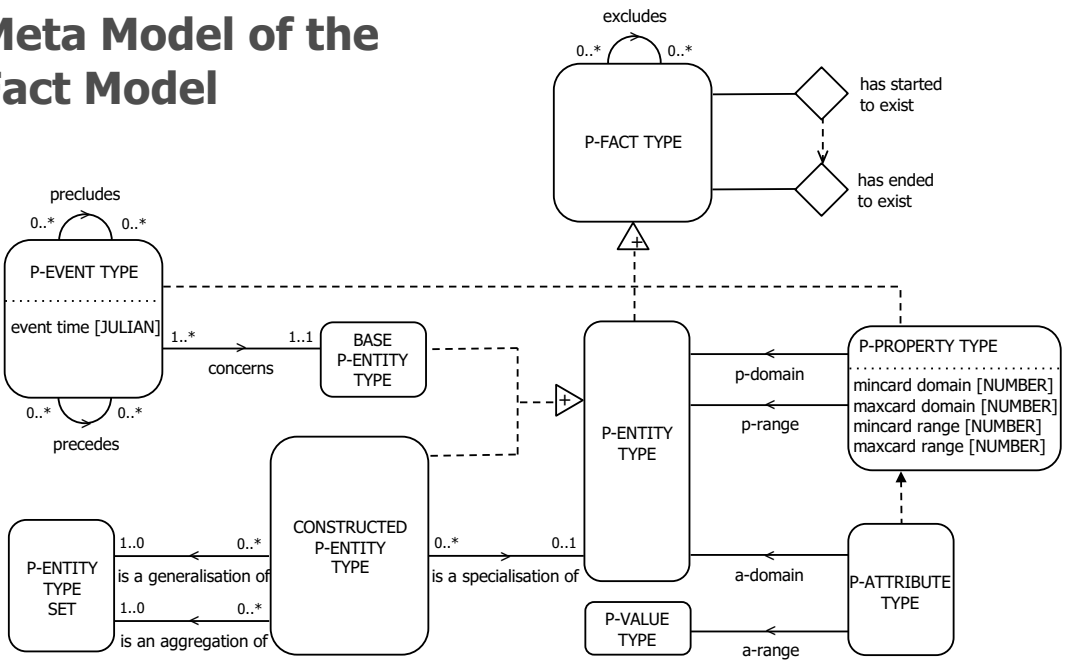
NOTE1. A process step is the combination of a C-act and the caused C-fact in the transaction pattern (slide 15).

NOTE2. The possible response links, the possible reversion links, and the possible intra-transaction wait links, are fully determined by the complete transaction pattern.

Meta Model of the Action Model



Meta Model of the Fact Model



NOTE1. The meta property types named "is a specialisation of", "is a generalisation of" and "is an aggregation of" exclude each other. Moreover, every constructed entity type is either a specialisation or a generalisation or an aggregation.
NOTE2. the meta entity type named "attribute type" is a specialisation of the meta entity type named "property type". Consequently, the meta property types named "a-domain" and "a-range" are respectively specialisations of the meta property types named "p-domain" and "p-range".
NOTE3. An entity type set is a set of (base or constructed) entity types.
NOTE4. Every P-EVENT TYPE is identical to an INDEPENDENT P-FACT KIND in the meta model of the Construction Model.

