DEMO-3

Models and Representations

(version 3.6c, March 2013)

Jan L.G. Dietz

Overview (1)

The ontological model of an organisation in DEMO-3 consists of the integrated whole of four sub models, each taking a specific view on the organisation. The modelled organisation can be (any part of) any of the three aspect organisations of an enterprise: the B-organisation, the I-organisation, or the D-organisation.

Construction Model

The Construction Model (CM) of an organisation is the ontological model of its construction: the composition, the environment, and the structure (i.e. the active and passive mutual influences among the elements in the composition and between these and the elements in the environment). Thus, the CM contains the identified actor roles in the composition and in the environment, the identified transaction kinds among the actor roles in the composition, and between these and the actor roles in the environment. In addition, the CM contains the information links from the actor roles in the composition to the internal transaction kinds and to the selected 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 by means of *Action Rule Specifications* (ARS).

Overview (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 Universal Transaction Pattern (UTP). Regarding the transition space, the PM contains the coordination events (creations of coordination facts) 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 UTP. Therefore, a PSD contains only the occurrence laws between transaction processes, expressed in links between process steps. There are three kinds of them: response links, causal links, and waiting links.

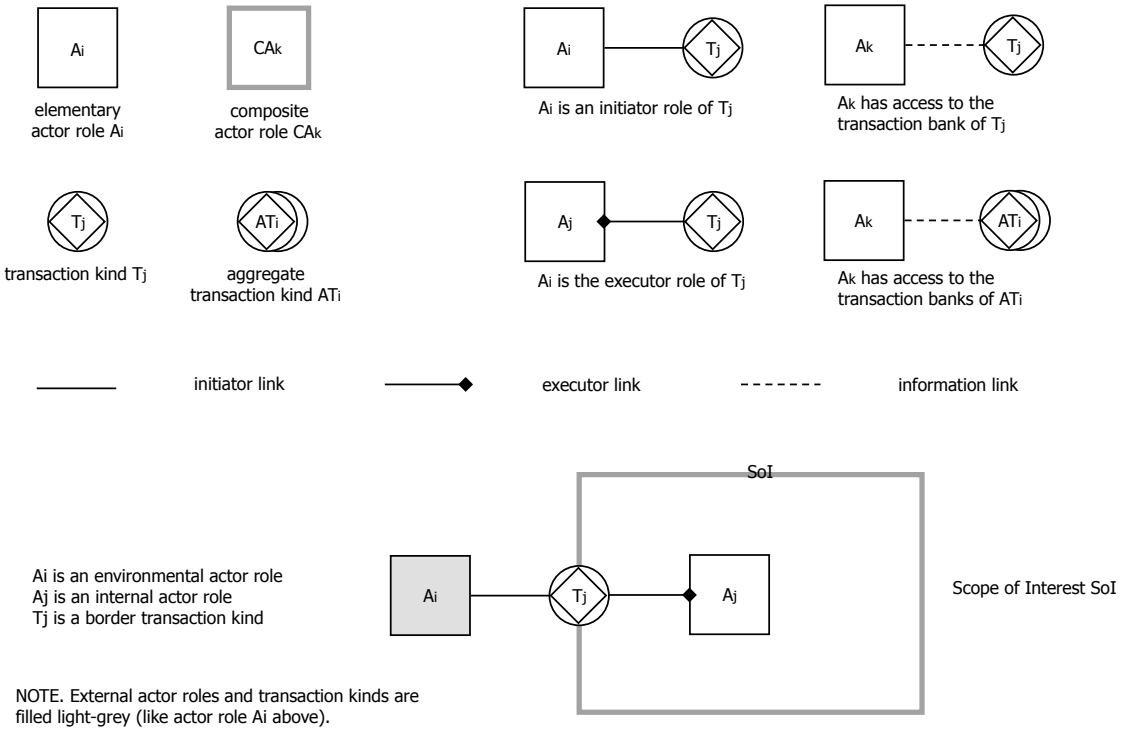
A PM is represented in a *Process Structure Diagram* (PSD), optionally complemented by a *Transaction Pattern Diagram* (TPD) for one or more transaction kinds.

Fact Model

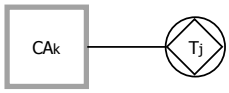
The Fact Model (FM) of an organisation is the ontological model of the state space and the transition space of its production world. Regarding the state space, the FM contains all identified fact kinds (both declared and derived), as well as the existence laws that apply. Three kinds of existence laws are distinguished: reference laws, unicity laws, and dependency laws. Regarding the transition space, the FM contains the production event kinds (results of transactions) as well as the applicable occurrence laws. Note: although the transition space of the production world of an organisation is completely determined by the transition space of its coordination world, it is often illustrative to show the occurrence laws between transaction product kinds.

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

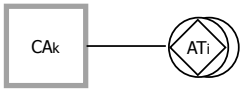
Legend of the OCD (1)



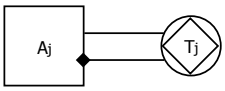
Legend of the OCD (2)



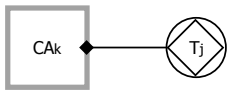
an actor role in CA_k is an initiator role of T_j



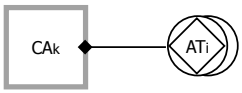
actor roles in CA_k are initiator roles in transactions in AT_i



self-activating actor role Aj (executor and initiator of transaction kind T_j)

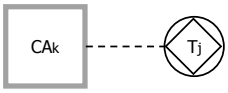


an actor role in CA_k is the executor role of T_j

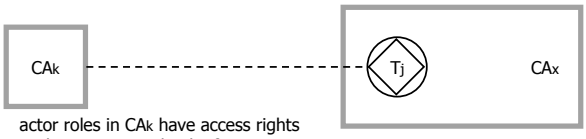


actor roles in CA_k are executor roles in transactions in AT_i

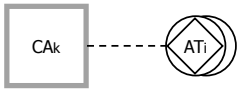
shorthand notation of self-activating actor role Aj (executor and initiator of transaction kind T_j)



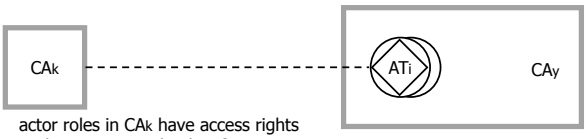
actor roles in CA_k have access rights to the transaction bank of T_j



actor roles in CA_k have access rights to the transaction bank of T_j



actor roles in CA_k have access rights to the transaction banks of AT_i

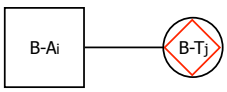


actor roles in CA_k have access rights to the transaction banks of AT_i

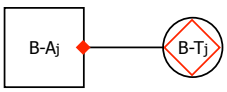
Legend of the OCD (3)

The transaction sort (original, informational or documental) of a transaction kind are 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 cannot be colored.

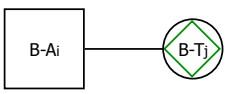
The aspect organisation to which a transaction kind and its executor role belong are indicated by adding the prefix B- or I- or D- to the respective id's. Examples: B-T17, I-T18, D-T19, B-A17, I-A18, D-A19.



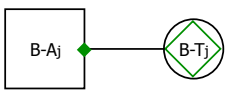
$B\text{-}A_i$ is an initiator role of original transaction kind $B\text{-}T_j$



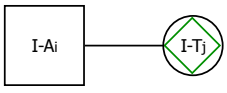
$B\text{-}A_i$ is the executor role of original transaction kind $B\text{-}T_j$



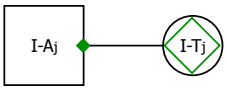
$B\text{-}A_i$ is an initiator role of informational transaction kind $B\text{-}T_j$



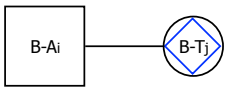
$B\text{-}A_i$ is the executor role of informational transaction kind $B\text{-}T_j$



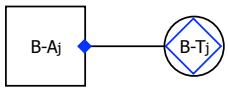
$I\text{-}A_i$ is an initiator role of informational transaction kind $I\text{-}T_j$



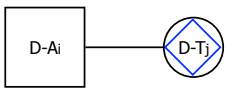
$I\text{-}A_i$ is the executor role of informational transaction kind $I\text{-}T_j$



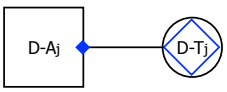
$B\text{-}A_i$ is an initiator role of documental transaction kind $B\text{-}T_j$



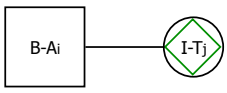
$B\text{-}A_i$ is the executor role of documental transaction kind $B\text{-}T_j$



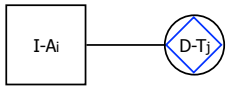
$D\text{-}A_i$ is an initiator role of documental transaction kind$D\text{-}T_j$



$D\text{-}A_i$ is the executor role of documental transaction kind $D\text{-}T_j$



$B\text{-}A_i$ is an initiator role of informational transaction kind $I\text{-}T_j$



$I\text{-}A_i$ is an initiator role of documental transaction kind $D\text{-}T_j$

Legend of the TPT

The **Transaction Product Table** (TPT) is a table of transaction kinds. The syntax of a TPT entry in EBNF is:

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

transaction kind id = "T", {digit}-; Example: T17
transaction kind name = {lower case letter}-; Example: rental start
product kind id = "P", {digit}-; Example: P17
product kind formulation= unary predicate formulation, property formulation;
unary predicate formulation = variable name, "**is**", perfect tense verbal expression;
Example: Rental **is** contracted
property formulation = "**the**", property kind name, "**of**", variable name, "**is**", perfect tense verbal expression;
Example: **the** first fee **of** Rental **is** paid
variable name = upper case letter, {lower case letter};

NOTE 1. "EBNF" stands for "Extended Backus-Naur Form". It is the international standard syntactic meta language defined in ISO/IEC 14977.

NOTE 2. The SBVR syntax is an allowed alternative for the presented syntax. Example: [rental] is contracted.

Legend of the BCT

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 in EBNF is:

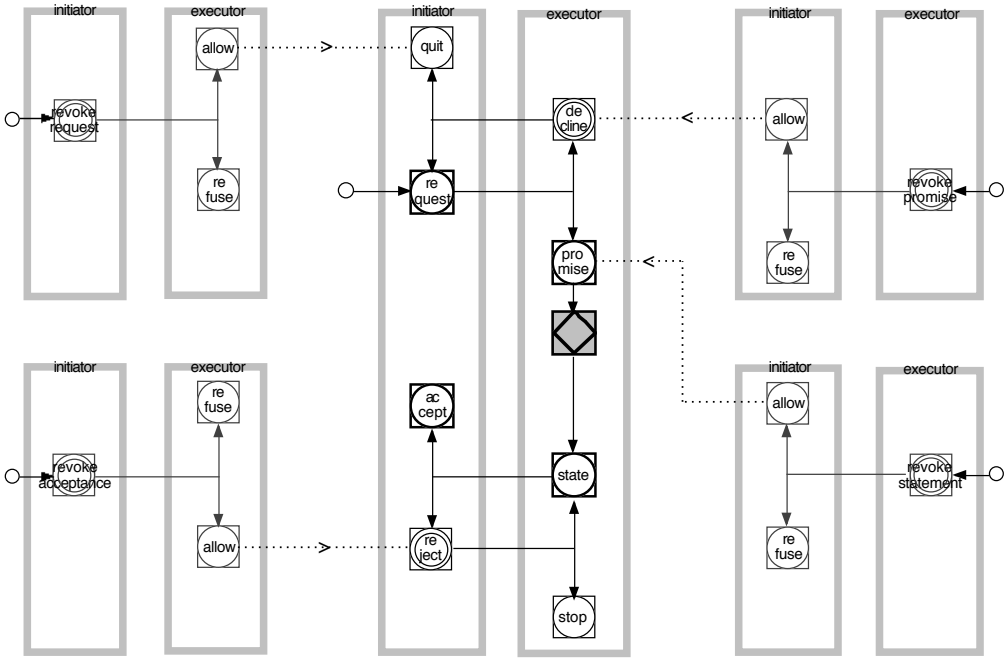
BCT entry = transaction kind id, (fact kind id, fact kind formulation) | (product kind id, product kind formulation);

transaction kind id = "T", {digit}-; Example: T17
fact kind id = "F", {digit}-; Example: F17
fact kind formulation = object class name | property kind formulation;
object class name = {upper case letter}-; Example: RENTAL
property kind formulation = "**the**", property kind name, "**of**" | "**in**", domain variable reference;
domain variable reference = variable name {"**of**" | "**in**", variable name};
Example: **the** renter **of** Rental
Example: **the** daily rental rate **of** Car Group **in** Year
variable name = upper case letter, {lower case letter};

product kind id = "P", {digit}-; Example: P17
product kind formulation= unary predicate formulation, binary predicate formulation;
unary predicate formulation = variable name, "**is**", perfect tense verbal expression;
Example: Rental **is** contracted
binary predicate formulation = "**the**", property kind name, "**of**", variable name, "**is**", perfect tense verbal expression;
Example: **the** first fee **of** Rental **is** paid

NOTE 1. "EBNF" stands for "Extended Backus-Naur Form". It is the international standard syntactic meta language defined in ISO/IEC 14977.
NOTE 2. The SBVR syntax is an allowed alternative for the presented syntax. Example: the renter of [rental]

Legend of the TPD (1)



Legend of the TPD (2)

A box represents an act (C-act or P-act), a disk represents a C-fact, and a diamond represents the P-fact of the transaction. The solid line between a C-act and its resulting C-fact is a *causal link*. Note: in the 'contracted' form of the TPD, these links are contained in the combined symbol of a process step.

An act with it resulting fact is called a process step. They are represented by the combination of a box and a disk or diamond.

The grey-lined rectangles represent the responsibility areas of the two actor roles.

A solid arrow from a process step S1 to a process step S2 represents a *response link*. It means that the occurrence of the coordination event (i.e. the creation of the C-fact) of S1 is an agendum for the actor role who is responsible for performing S2.

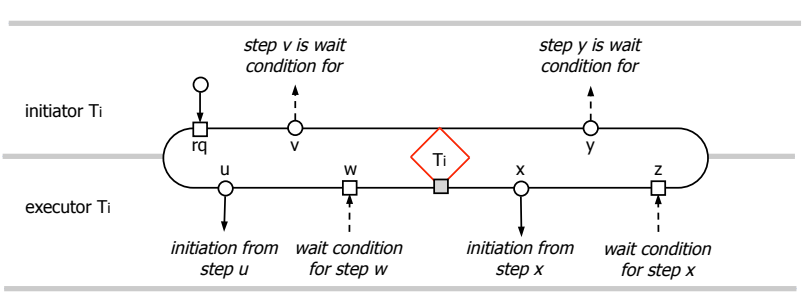
A small disk outside a responsibility area may represent any coordination event, of the same transaction or in another transaction (of the same or another transaction kind).

The split of a response link into two links represents an exclusive or.

A double disk represents a discussion step.

A dashed line with a ">" in the middle indicates the transaction step to which the transaction process returns after a successful revocation (allow step).

Legend of the PSD (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 process goes on in three phases: the proposition phase (left from the diamond), the execution phase (between the left and the right bracket of 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 PSD (2)

A process step consists of a coordination act (represented by a small box) and its resulting coordination fact (represented by a small disk).

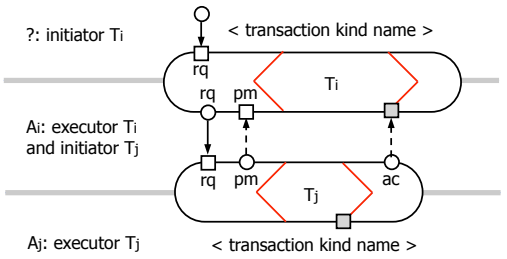
A solid arrow represents a *response link*. A response link starts from (the fact of) a process step and ends in the request act of a transaction kind. A process step may have several outgoing response links. This means that several transaction kinds are initiated from it.

A dashed arrow represents a *waiting link*. A waiting link starts from (the fact of) a process step and ends in (the act) of a process step, or in the execute act. A process step may have several outgoing waiting links. It means the occurrence of the coordination event is a wait condition for the performance of several steps.

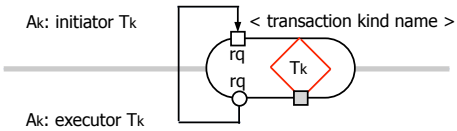
A process step may also have several incoming waiting links. It means that performing (the act of) the step has to wait for the occurrence of several coordination events. How exactly is specified in the Action Model.

Legend of the PSD (3)

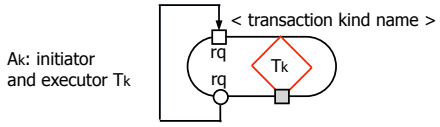
Example of transaction kind T_i with enclosed transaction kind T_j



Example of a self-initiating transaction kind T_k

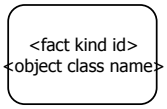


Alternative notation:



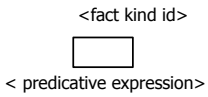
Legend of the OFD (1)

SHAPES and NOTATIONS

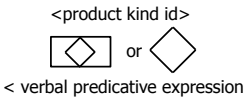


notation of the *extension* of a unary fact kind;
the extension is the *class* of *objects* to which the intension applies;
By convention, the object class name is written in capitals.

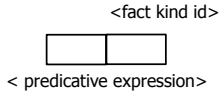
extension_of_fact_kind = {x|intension_of_fact_kind(x)}



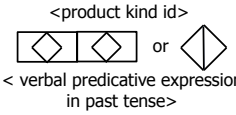
notation of the *intension*
of a unary fact kind;
a unary fact type has one role;



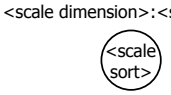
notation of the *intension*
of a unary product kind;



notation of the *intension*
of a binary fact type;
a binary fact type has two roles;



notation of the *intension*
of a binary product kind;



notation of the *extension* of a *scale*;
a scale is a unary fact kind that solely applies to abstract objects (like numbers);

NOTE 1. The intension of a fact type is a *predicate* over one or more objects; every object plays a specific *role*.

NOTE 2. A predicative sentence explains the intension of a fact type. It contains as many variables (placeholders for objects) as there are roles in the fact type. Variable names start with a capital. Examples: "Person is a student", "Person lives in Town".

NOTE 3. The shapes of external fact types are colored (filled) light-grey.

Legend of the OFD (2)

SCALES: dimensions, sorts, and measurement units

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

dimension	scale unit(s)	sort
TIME	day, hour, minute, second, millisecond, …..	I
	(week, month, year, etc. are dependent on the calendar used)	
MONEY	dollar ($), euro (€) etc.	R
MASS	… kg, g, mg, …	R
LENGTH	… m, cm, mm, …	R
AREA	… m^2, cm^2, mm^2, …	R
VOLUME	… m^3, cm^3, mm^3, …	R
VELOCITY	… m/s, …	R
TEMPERATURE	^oK, ^oC, ^oF	I
NUMBER	< just counting >	A

Note 1. Each dimension belongs to a particular scale sort: Ordinal (O), Interval (I), Ratio (R) or Absolute (A).

Note 2. Only the scale units in the SI (Système International) are considered.

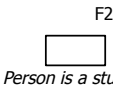
Note 3. In ontological models, one often does not care about the scale unit; then mentioning the scale dimension is sufficient.

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.

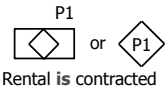
Legend of the OFD (3)

DECLARING FACT KINDS and PRODUCT KINDS

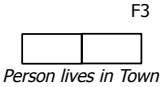
extensional declaration of the unary fact kind F1;
the extension of a declared unary fact kind is also called *category*.



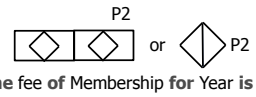
intensional declaration of the unary fact kind F2



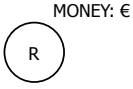
intensional declaration of the unary product kind P1



intensional declaration of the binary fact kind F3



intensional declaration of the binary product kind P2



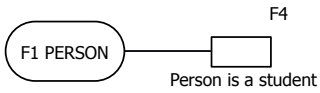
extensional declaration of the scale with dimension MONEY and unit € (euro);
because the dimension is MONEY, this scale is a rational scale.

NOTE 1. Declaration means that one introduces an original (i.e. not derived) fact type in a Fact Model. Derived fact kinds may be introduced in the same way, but then an asterisk must be added to the fact kind id. Example: F17*. In addition, the derivation rule must be provided. Sometimes, it is convenient to specify derived fact kinds graphically (see slide 19).

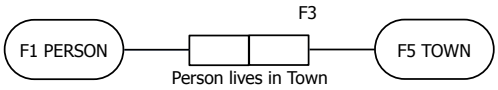
NOTE 2. Scales cannot be declared intensionally.

Legend of the OFD (4)

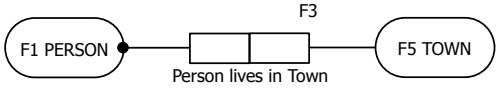
REFERENCE LAWS and DEPENDENCY LAWS



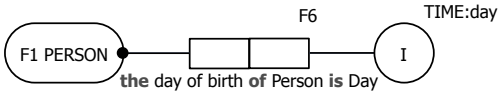
the line between the shapes of F1 and F4
represents a *reference law*:
for all Object **it holds that** Object **is a** student **implies** Object **is a** person



the lines between F1 and the left role of F3,
and between F5 and the right role of F3
represent *reference laws*:
for all Object1, Object2 **it holds that** Object1 lives in Object2
implies Object1 **is a** person **and** Object2 **is a** town



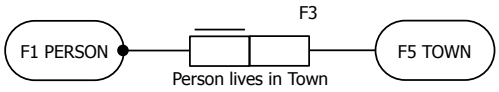
the black dot on the edge of the shape of F1
represents a *dependency law*:
for all Object1 **in** PERSON **there is a** Object2 **in** TOWN
such that Object1 lives in Object2



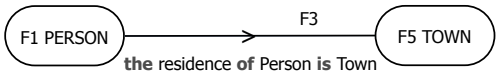
the black dot on the edge of the shape of PERSON
represents a *dependency law*:
for all Object1 **in** PERSON **there is a** Object2 **in** TIME:day
such that Object1 is born on Object2

Legend of the OFD (5)

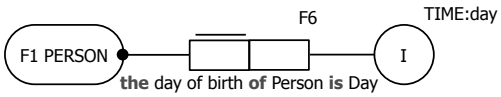
UNICITY LAWS and shorthand notations



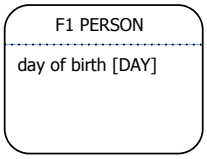
the line above the shape of the first role of F3
represents a *unicity law*:
for all Person **it holds that if**
Person lives in Town1 **and**
Person lives in Town2
then Town1 **is identical to** Town2



shorthand notation of the construct above;
F3 is called now a *property* of person;
the predicative expression "Person lives in Town" is replaced by the
semantically equivalent nominative expression
"**the** residence **of** Person **is** Town"



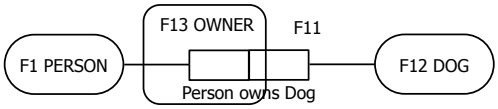
the line above the shape of the first role of F6
represents a *unicity law*:
for all Person **it holds that if**
the day of birth **of** Person **is** Day1 **and**
the day of birth **of** Person **is** Day2
then Day1 **is identical to** Day2



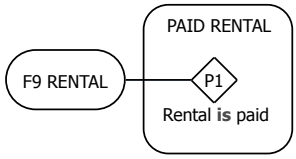
shorthand notation of the construct above;
F6 is called now an *attribute* of person;
the nominative expression "**the** day of birth **of** Person **is** Day"
is replaced by the shorthand "day of birth", and the scale reference
"TIME:day" is replaced by he shorthand "DAY"

Legend of the OFD (6)

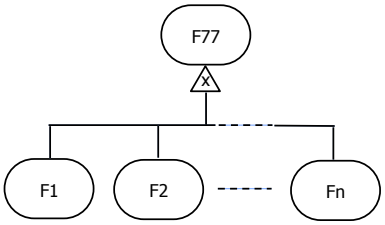
graphical specifications of DERIVED FACT KINDS



By drawing an object class shape around the shape of the first role of F11, the *derived* fact kind F13 is specified graphically;
the extension of F13 is the set of persons for which it holds that there is a Dog such that Person owns Dog.
F13 is called a *specialisation* of F1

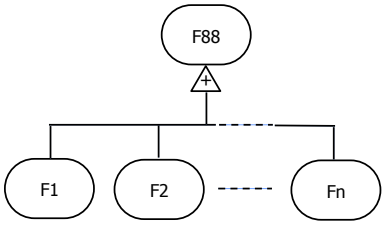


By drawing an object class shape around the shape of P1, the *extension* of P1 is shown graphically.
P1 may now serve as a reference domain in other fact kinds (and product kinds).



graphical specification of the *aggregation* F77 of fact kinds F1, F2, …. , Fn

The extension of F77 is the cartesian product of the extensions of F1 … Fn



graphical specification of the *generalisation* F88 of fact kinds F1, F2, …. , Fn

Every instance of F88 is an instance of F1 or … or Fn

Derived Fact Specification Language

<transaction kind id> <fact kind id> <fact kind reference>, where:

<transaction kind id>	::= T<numeral string>	Example: T17
<fact kind id>	::= F<numeral string>	Example: F17
<fact kind reference>	::= <general reference>|<special reference>	
<general reference>	::= <verbal predicative expression>	Example: Person lives in Town
<special reference>	::= <object class reference>|<property kind reference>	
<object class reference>	::= <object class name>	Example: RENTAL
<property kind reference>	::= <nominal predicative expression>	
<property kind reference>	::= **the** <property name> **of** <variable> **is**	Example: **the** renter **of** Rental **is**

transaction kind name = "T", {lower case letter}-;
variable name = upper case letter, {lower case letter};
property formulation = "the", property name, "of", variable name, "is", object reference;

Existence Law Specification Language

basic action rule syntax

when < transaction kind > **of** < object identifier(s) > **is** < transaction status >
if < boolean logical expression >
then < transaction kind > **of** < object identifier(s) > **must be** < perfective form of C-act >
else < transaction kind > **of** < object identifier(s) > **must be** < perfective form of C-act >

Note1: < transaction status > = requested, promised, executed, stated, accepted, etc.

Note2: < perfective form of C-act > = requested, promised, executed, stated, accepted, etc.

Action Rule Specification Language

basic action rule syntax

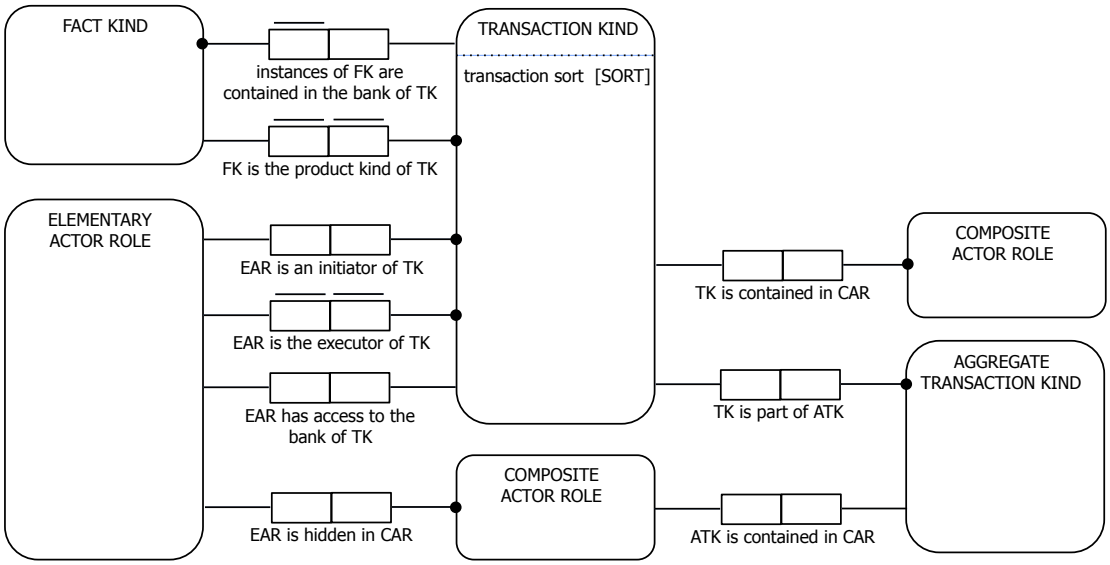
action rule specification = event part, assess part, response part;

event part= agendum clause, [with clause], [while clause];
agendum clause = "**when**", transaction kind name, "**for**", ["new"], variable name, "is", ("requested" | "promised" | "stated" | "accepted");
with clause = "**with**", {property formulation}-;
property formulation = "**the**", property name, "**of**", variable name, "**is**", object indicator;
(* 'variable name' in 'property formulation' must be the same as 'variable name' in 'agendum clause' *)
object indicator = object name | "**some**", variable name;
(* 'variable name' in 'object indicator' must be the 'variable name' of the range of the property kind *)
while clause = "**while**", {event clause}-;

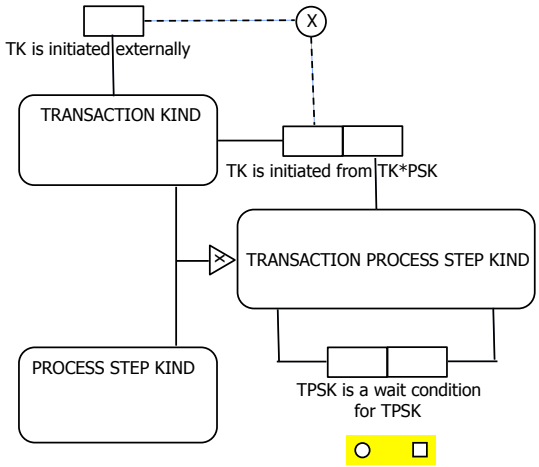
assess part = justice part, sincerity part, truth part;
justice part = "*justice:*", "<no specification>" | {fact formulation}-;
sincerity part = "*sincerity:*", "<no specification>" | {fact formulation}-;
truth part = "*truth:*", "<no specification>" | {fact formulation}-;
fact formulation =

response part = "**if**", "complying with", ("request" | "promise" | "state" | "accept"), "is considered justifiable", "**then**", action clause, ["**else**" action clause]

Meta Model of Construction Model



Meta Model of Process Model



Performer of process step

INITIATOR	EXECUTOR
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

NOTE. Initiating transaction kind TK means performing the request in a TK (by the initiator role of TK)

Yet to be delivered

Meta Model of Fact Model