The DEMO Specification Language v4.6.1

Jan L.G. Dietz

jan.dietz@sapio.nl,

Abstract. Conceptual models must be expressed in a suitable language in order to communicate them. To avoid misunderstandings, this language should allow for formulating clear, unambiguous expressions. First order logic is a language that has all the properties one needs and wants, but it has the drawback that its common Peano-Russell notation puts off people who lack a background in logic and mathematics. The DEMO Specification Language (DEMOSL) offers a user-friendly look, although it is firmly based on first order logic. One produces precise formulations that resemble English sentences. The syntax of DEMOSL is defined in the so-called Extended Backus-Naur Form (EBNF). In addition to (formal) textual expressions, it allows for (formal) graphical representations. Except for the Action Model, one can represent the DEMO aspect models in diagrams. DEMOSL comprises specific diagrams for each of the other models: the Cooperation Model, the Process Model and the Fact Model. The diagrams are kept simple, which means that what cannot be expressed in a diagram, must be expressed in (formal) text. In addition to the explanation and illustration of DEMOSL, the meta model or schema of each of the four aspect models is presented and discussed, as well as of each of the kinds of diagrams and tables. They are also expressed in DEMOSL.

1 Introduction

In this document, the formal language is presented in which the ontological models of DEMO (Design and Engineering Methodology for Organisations)[1] are expressed. The language is called DEMO Specification Language, or DEMOSL for short. The distinction between models on the one side and the diagrams, tables and (formalised) texts in which they are expressed on the other side, is crucial (cf. MU theory in [1]): they constitute respectively the semantics and syntax of the language DEMOSL.

Expressions in DEMOSL are basically formal textual expressions in first order logic [2]. A large part of these textual expressions have also a graphical or tabular equivalent, because the appreciation of formal text in current practice is generally low, particularly by people without a logical/mathematical background. Consequently, the diagrams should be understood as graphical representations that are univocally transformable to (formal) texts and vice versa. The same holds for tabular expressions. They are only syntactic alternatives to represent the same semantics as formal texts. In spite of the current preference for diagrams to formal text, there are limits to the expressive power of diagrams: they can easily become impractical if one has to remember (too) many shapes, symbols and constructs. Therefore, the diagrams in DEMOSL are kept simple; they have a limited set of shapes, symbols and constructs. What cannot be expressed in a diagram, must be expressed in a formal text.

In Sec. 2 the basics of DEMOSL are presented: the definition of terms, the declaration and the derivation of types, and the ways in which time values are represented. The four aspect models in DEMO, thus the Cooperation Model (CM), the Action Model (AM), the Process Model (PM) and the Fact Model (FM) are discussed in [1]. Secs. 3 thru 6 contain their complete and formal definitions, including the meta model of each of them, as well as the various ways in which they can be expressed. Figures that cannot be easily inserted in the text, are put in Sec. 7, as an appendix.

2 The basics of DEMOSL

2.1 Definition of terms

In this section, the terms that are used in formal expressions of DEMOSL, are defined in the so-called Extended Backus-Naur Form (EBNF)[2], the international standard syntactic meta language, defined in ISO/IEC 14977[3]. As much as possible, English words are added to the syntactical elements of DEMOSL, which make the textual expressions pretty intuitive: they look like structured English sentences, quite unlike the common Peano-Russell notation[4]. To improve the readability, words, like articles and prepositions, are added. These added words are printed in bold. There are also a number of reserved terms, which connect DEMOSL to the EE theories in [1].

In EBNF, names are put between double quotation marks (“ and ”). The symbol “|” stands for “(exclusive) or”. The symbol “,” means “followed by”; the EBNF brackets “{“ and “}” enclose symbols that may be repeated an unlimited number of times; “}-“ as the closing bracket means that there is at least one occurrence. Where considered helpful, comments are inserted between “%” and “%”.

reserved term = coordination act name | coordination fact name | special term;

coordination act name = "request" | "promise" | "declare" | "accept" | "decline" | "reject" | "revoke" | "allow" | "refuse";

coordination fact name = "requested" | "promised" | "declared" | "accepted" | "declined" | "rejected" | "revoked" | "allowed" | "refused";

special term = "performer" | "addressee" | "ct" % creation time % | "et" % event time % | "ot" % operative time % | "now" % at any moment, the value of the variable now is the current time % | "set of" % set of %;

transaction kind id = "TK", {digit}-;	Example: TK17
multiple transaction kind id = "MTK", {digit}-;	Example: MTK2
product kind id = "PK", {digit}-;	Example: PK02
actor role id = "AR", {digit}-;	Example: AR8
transactor role id = "TAR", {digit}-;	Example: TAR17
composite transactor role id = "CTAR", {digit}-;	Example: CTAR01

transaction kind name = entity type name, ing-form of a verb (in lower case);
Examples: rental completing, deposit paying
actor role name = entity type name, nominal form of a verb (in lower case);
Examples: rental completer, deposit payer
entity type name = noun or nominal phrase (in lower case);
Examples: rental, deposit paid rental
object class name = noun or nominal phrase (in upper case);
Examples: RENTAL, DEPOSIT PAID RENTAL
value type name = noun or nominal phrase (in lower case);
Examples: year, car group
value class name = noun or nominal phrase (in upper case) between "{" and "}";
Examples: {YEAR}, {CAR GROUP}
property type name = noun or nominal phrase (in lower case);
Examples: renter, pick-up branch
attribute type name = noun or nominal phrase (in lower case);
Examples: age, daily rental rate
event type name = perfect tense verb (in lower case);
Examples: completed, paid
entity name = {letter | digit}-
Examples: John, 1089, Mary47
value name = {letter | digit}-
Examples: sedan, 2020, 2458270D

As discussed in the FI theory [1], the signifier of a conceptual thing can be a name, a noun or a sentence, depending on the kind of thing. Moreover, a name can be a

proper name, like ‘John’ or ‘John Smith’ for a person, or an identifier like ‘TK01’ for a transaction kind, ‘2272 BP’ for a postal code, and ‘069684996’ for a BSN (Dutch citizen identifier). Note: to avoid confusion, it is recommended to put signifiers (of any kind) between single quotation marks, as we did above. For example, one better write “value type ‘car group’” instead of “value type car group”, and “car group ‘sedan’” instead of “car group sedan”.

entity reference = definite entity reference | indefinite entity reference | indirect entity reference;

definite entity reference = entity type name, entity name;
Examples: rental ‘1089’, car ‘387462’

indefinite entity reference = “**some**”, entity type name;
Examples: **some** person, **some** car

indirect entity reference = “**the**”, property type name, {“**of**” | “**in**” | “**on**”, entity reference | value reference }-;
Examples: **the** renter **of** rental ‘1089’,
the mother **of the** member **of** membership ‘387’,
the car **of some** rental **in the** year ‘2019’

entity variable = “[“, entity type name , “]”;
Examples: [person], [car], [rental]

property variable = “**the**”, property type name, {“**of**” | “**in**” | “**on**”, entity variable | property variable | value variable | attribute variable }-;
Examples: **the** renter **of** [rental],
the father **of the** renter **of** [rental],
the renter **of some** rental **on the day** ‘2458270’,
the book **of some** loan **on the ending day of** rental ’12’

property condition = property variable, “**is**” | “**is not**”, property variable | entity reference;
Examples: **the** renter **of** [rental] **is the** driver **of** [rental],
the driver **of** [rental] **is not** person ‘92637’
the father **of the** driver **of** [rental] **is not** person ‘92637’

value reference = definite value reference | indefinite value reference | indirect value reference;

definite value reference = value type name, value name;
Examples: number ‘1089’, day ‘2458270’, car group ‘sedan’

indefinite value reference = "**some**", value type name;
Examples: **some** number, **some** day, **some** car group

indirect value reference = "**the**", attribute type name, {"**of**" | "**in**" | "**on**", entity reference | value reference}-;
Examples: **the** weight **of** car '387462',
the weight **of the** car **of** rental '1089',
the daily rental rate **of** car group 'sedan' **in the** year '2'

value variable = "[", value type name , "]";
Examples: [day], [car group]

attribute variable = "**the**", attribute type name, {"**of**" | "**in**" | "**on**", entity variable | property variable | value variable | attribute variable}-;
Examples: **the** age **of** [person],
the daily rental rate **of** [car group] **in** [year]
the deposit amount **of some** rental **on the day** '2458270',
the penalty **of some** loan **on the ending day of**
rental '1089'

attribute condition = attribute variable, "**is equal to**" | "**is unequal to**" | "**is greater than**" | "**is less than**" | "**is equal to or greater than**" | "**is equal to or less than**" , attribute variable | definite value reference;
Examples: **the** ending day **of** [rental] **is equal to or greater than**
the starting day **of** [rental],
the number **of** free cars **in the** car group **of** [rental]
on every day **between the** starting day **of** [rental]
and the ending day **of** [rental] **is greater than**
the number '0'

NOTE 1. The time indication in the last three lines of the second example (**on every** day …) is clarified in Sec. 2.4.
NOTE 2. The fact type "free cars" is a derived fact type. It has to be specified, as part of the Fact Model of an organisation.
NOTE 3. The mathematical comparisons above only apply to numerical instances of value types (or scale types) of the sorts Ordinal, Interval, Rational and Absolute (cf. Sec. 2.3).

2.2 Declaration and derivation of types

Types (or better: fact types) can be specified graphically and textually, both on the schema level and on the meta schema level. The graphical specification is discussed in Secs. 3 thru 6. Below, the textual specification is presented.

type declaration = entity type declaration | value type declaration | property type declaration | attribute type declaration | event type declaration;
entity type declaration = "entity type", entity type name, "**exists**";
Example: entity type 'rental' **exists**
value type declaration = "value type", value type name, "**exists**";
Example: value type 'car group' **exists**
property type declaration = "property type", property type name, "**exists**";
Example: property type 'renter' **exists**
attribute type declaration = "attribute type", attribute type name, "**exists**";
Example: attribute type 'starting day' **exists**
event type declaration = "event type", event type name, "**exists**";
Example: event type 'completed' **exists**

Derived types can be specified graphically or textually. Examples of graphically specified derived types are provided in Sec. 6. Examples of textually specified derived fact types are (the symbol "≡" means "is defined as"):

the rental charge **of** [rental] ≡ **the** duration **of** [rental] **times the** daily rental rate **of the** car group **of** [rental] **in the** year **of the** starting day **of** [rental],
the duration **of** [rental] ≡ **the** ending day **of** [rental] **minus the** starting day **of** [rental] +1,
the actual duration **of** [rental] ≡ **the** day **of the** et **of** (car returning **for** [rental] **is** accepted) **minus the** starting day **of** [rental] + 1,
the late return penalty **of** [rental] ≡ (**the** actual duration **of** [rental] **minus the** duration **of** [rental]) **times** (**the** late return penalty **in the** year **of the** starting day **of** [rental]);

2.3 Value types - dimensions, units and sorts

Values of variables, like the day of birth of a person, her/his length or weight, or the sort of a transaction kind (original, informational or documental), are basically measurements on a (measurement) scale. Commonly, six scale sorts are distinguished: Ordinal (O), Interval (I), Rational (R), Categorial (C), Absolute (A) and Boolean (B).

Ordinal scales have no zero point and no measuring unit, they are only orderings from less to more. A well-known example is the hardness of rocks: it is only possible to determine that the one rock is harder than an other rock.

Interval scales do have a measuring unit but no zero point. The measuring unit can be chosen freely. A well-known value type in this scale sort is temperature (Note, the temperature in degrees Kelvin does have a physical zero point but it is still considered an interval scale). Another example is time. Because of the very natural unit of a day, all calendars are based on this measuring unit.

Rational scales have also a freely choosable measuring unit but a fixed zero point. Well-known value types in this scale sort are length, surface, volume and velocity.

Categorial scales (also called nominal scales) are actually not real measurement scales, because there is no measurement unit and no zero point. They are (basically arbitrary) divisions. Examples are product categories in shops (like fruit, vegetables, dairy, etc.) and car groups in a car rental company (like sedan, mini, sports car, etc.).

The *Absolute* scale is actually not a real measurement scale, it is just counting: the number of apples in the basket and the number of persons in a car. Considered as a scale, it has a fixed zero point and a fixed measuring unit.

The *Boolean* scale consists of the two (logical) boolean values true and false.

In Table 1, the standard value types are presented. One may assume that they are always present. So, there is no need to define them explicitly in a DEMO model.

value type	**dimension**	**measuring unit**	**base type**	**scale sort**
time	TIME	Julian day, hour, minute…	integer	I
duration	TIME	number of days, ..	integer	A
amount	MONEY	dollar ($), euro (€), …	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}C, ^{o}F, K	real	I
number	NUMBER	< not applicable >	integer	A
truth value	BOOLEAN	< not applicable >	{true, false}	B
sort	SORTAL	< not applicable >	{O, I, D}	C

Table 1 Dimensions, units and sorts of value types

By the value type ‘sort’ is meant the distinction of the three sorts of organisation and production in the ALPHA theory [1]. Value classes (so the extensions of value types) are formulated as follows in BNF: “{“, dimension name, “ : ”, unit name, “}”. Examples of value classes are {MONEY : euro} or {MONEY : €}, and {TEMPERATURE : ^{o}C}. It is allowed to omit the unit, and thus to only mention the dimension. For example: amount {MONEY}. It is also allowed to abbreviate <dimension><unit> to <unit> if no confusion can arise. In that case, the unit is written in capital. For example: {JULIAN: day} may be abbreviated to {DAY}.

2.4 Representing time values

Points in time, like the day of birth of a person and the starting day of a car rental, are basically measured on the Julian time scale. Next to this scale, time units from the Gregorian Calendar, like year and month, may be used. The conversion between the Julian scale and the Gregorian Calendar is based on the converter of the US Navy[5].

So, in expressions like “the first day of the next month (from now)” and “the year of the starting day of [rental]”, the values of month and year are calculated by means of this converter. Note that the value of the universal variable ‘now’ is also expressed in the Julian scale.

Next to the common time durations (or intervals) like days, weeks, months and years, one may define arbitrary intervals by means of the expression “**on every** <time unit> **between** <point in time> **and** <point in time>”. An interval defined in this way is up to and including the second point in time. Here is an example from the case Rent-A-Car; it regards all days from, and including the starting day, up to, and including, the ending day:

> **the** number **of** free cars **in the** car group **of** [rental] **on every** day **between the** starting day **of** [rental] **and the** ending day **of** [rental] **is greater than**
> **the** safety minimum **in the** year **of the** starting day **of** [rental]

The preciseness with which points in time are measured and represented depends on the situation at hand. In normal enterprise situations, the smallest unit is probably the second. For example, the Julian value of ’30 April 2019 12:00 hours, 0 minutes and 0 seconds’ is ‘2458604.000000’, and the Julian value of ’30 April 2019 12:00 hours, 0 minutes and 1 second’ is ‘2458604.000012’.

2.5 The four aspect models

The ontological model of an enterprise in DEMO consists of the integrated whole of four aspect models or sub models, each taking a specific view on the enterprise’s organisation: the Cooperation Model, the Action Model, the Process Model and the Fact Model. The relationships between the four models is illustrated in Fig. 1. Recall that a model of a thing is its understanding within a theoretical framework (cf. MU theory in [1]). A DEMO model is the understanding of a Scope of Interest (SoI) within the framework of the EE theories (cf. Part B of [1]).

The *Cooperation Model* (CM) of an organisation is a model of the *cooperation* between its actors (cf. OMEGA theory in [1]), or, following the DELTA theory in [1], the *construction* of the organisation, i.e. of the transactor roles and the coordination structures among them.

The *Action Model* (AM) of an organisation is a model of its *operation*, i.e. the manifestation of the construction in the course of time (cf. PSI theory and DELTA theory in [1]). It comprises action rules and work instructions.

The *Process Model* (PM) of an organisation is a model of the (business) *processes* (consisting of transaction steps and process links between them) that take place as the effect of the activity of actors (cf. PSI theory in [1]). In systemic terms, it is a specification of the state space and the transition space of the coordination world (cf. DELTA theory in [1]).

The *Fact Model* (FM) of an organisation is a model of the *products* (independent fact types with their dependent fact types) of the organisation that actors bring about (cf. PSI theory in [1]). In systemic terms, it is a specification of the state space and the transition space of the production world (cf. DELTA theory in [1]).

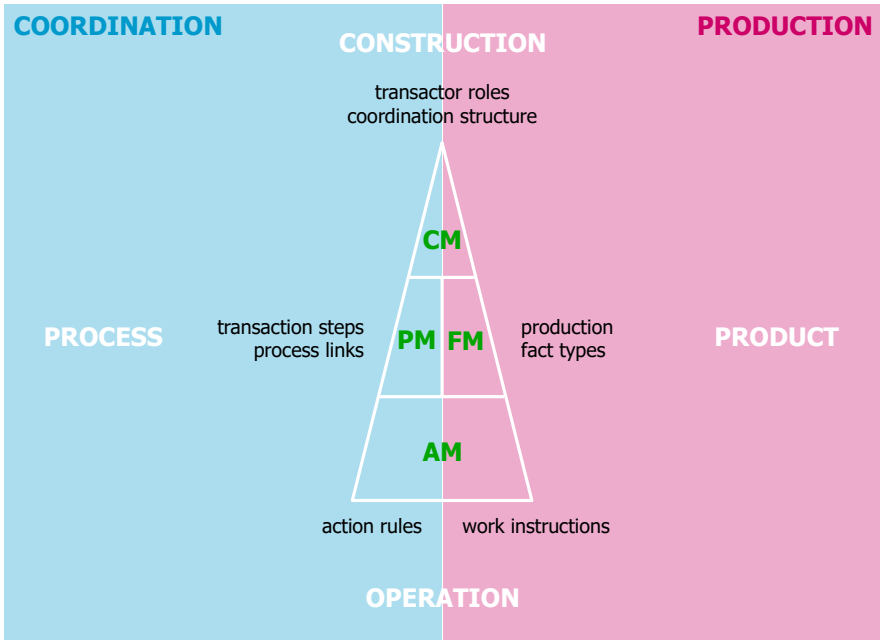


Fig. 1 The integrated DEMO aspect models

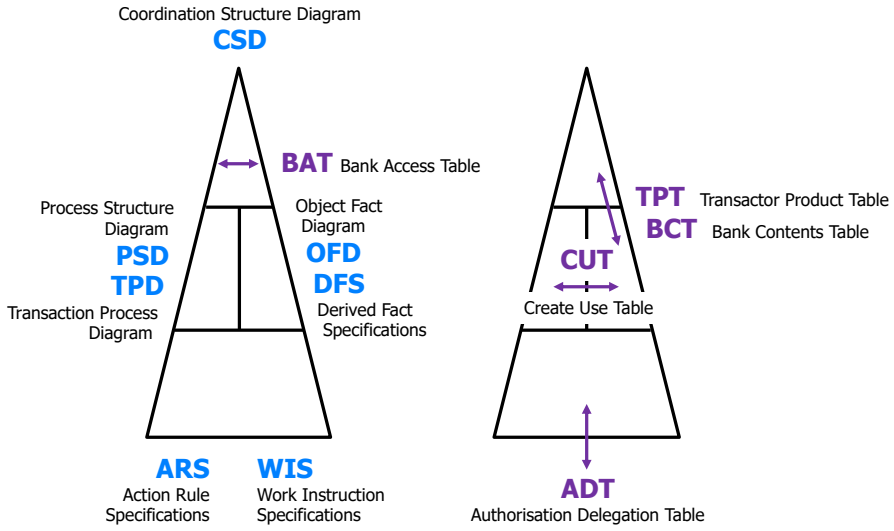


Fig. 2 Ways of expressing the four aspect models

As illustrated by the triangular shape in Fig. 1, and the division of this shape in the four aspect models, the CM and the AM cover both coordination and production, while the PM regards only coordination and the FM only production. The PM connects the CM and the AM, as far as the coordination between actors is concerned. In a similar way, the FM connects the CM and AM, as far as production is concerned. The

AM is the solid basis on which the other three models are standing. They are already 'contained' in the AM, so to speak, but they need yet to be 'extracted' from it. Lastly, there is nothing 'above' the CM. Fig. 15 contains the combined conceptual schemata of the CM, the AM and the PM, expressed in GOSL (cf. MU theory in [1]). Fig. 16, contains the meta schema of the FM. Every instance of the meta schema, so every FM, is itself a schema, namely of the production world of the modelled organisation.

Fig. 2 presents the ways in which the aspect models are expressed, in diagrams, tables and (formal) textual expressions. They are all formally defined in Figs. 17 through 25. The Bank Access Table (BAT) is an alternative way of expressing the interstriction structure (cf. Sec. 3). Therefore it is put on the left side of Fig. 2. The other tables, shown on the right side, are so-called cross model tables: each of them represents a specific relationship between two aspect models. The set of cross-model tables is by no means exhaustive: one may freely define new ones if there is a need. The presented tables are just the ones that are commonly used in practice. In addition, alternative diagrams may be developed to express the four models. As long as they are also formally defined, like the ones we discuss, there is no objection.

3 The Cooperation Model

The *Cooperation Model* (CM) of an organisation (or SoI) is a model of the *cooperation* between its actors (cf. OMEGA theory in [1]), or, following the DELTA theory in [1], the *construction* of the organisation, i.e. of the transactor roles and the coordination structures among them.

Within a SoI one may want to indicate a part on which one wants to focus, e.g. a legal unit like a company. This is done by colouring the actor role shapes outside the focus light-grey. The transactor roles inside the focus are called *internal*, and the transactor roles outside are called *environmental* (if there is interaction with an internal actor role) or *external* otherwise (cf. DELTA theory in [1]). A transaction kind of which both the initiator and the executor are internal, is called *internal*. If either of them is environmental, it is called a *border* transaction kind. Multiple transaction banks are by definition external and therefore also coloured light-grey. There are three coordination structures among transactor roles (cf. OMEGA theory in [1]): the interaction structure, the interimpediment structure, and the interstriction structure. These structures are graphically expressed in the *Coordination Structure Diagram* (CSD).

The legend of the CSD is exhibited in Figs. 3 and 4. The upper left part of Fig. 3 shows the *elementary* and *self-activating* transactor roles. The shapes and constructs in the lower part of Fig. 3 are considered to be sufficiently explained in the figure. The constructs hold for each of the three sorts of organisations as distinguished in the ALPHA theory in [1]: O-organisations, I-organisations, and D-organisations.

Fig. 4 shows on the right side how these sorts are distinguished. On the left side, it shows how external and environmental transactor roles are indicated, namely by colouring the shapes light-grey. The right side of the figure also shows how external (multiple) transaction kinds are indicated.

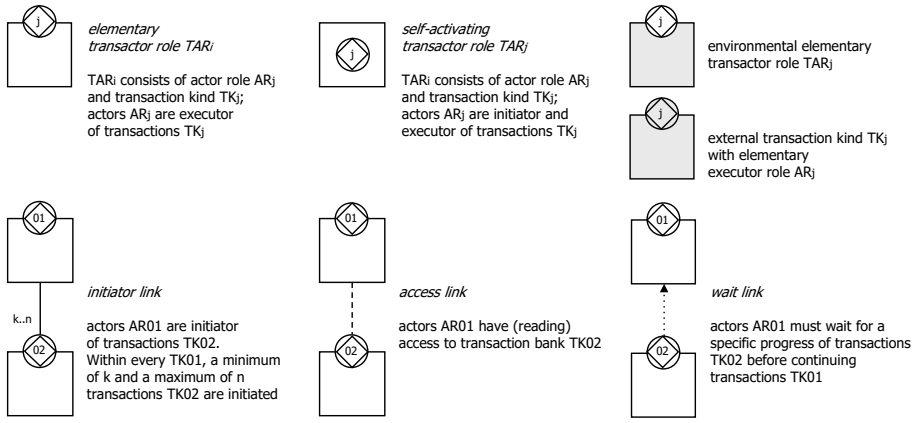


Fig. 3 Legend of the Coordination Structure Diagram (1)

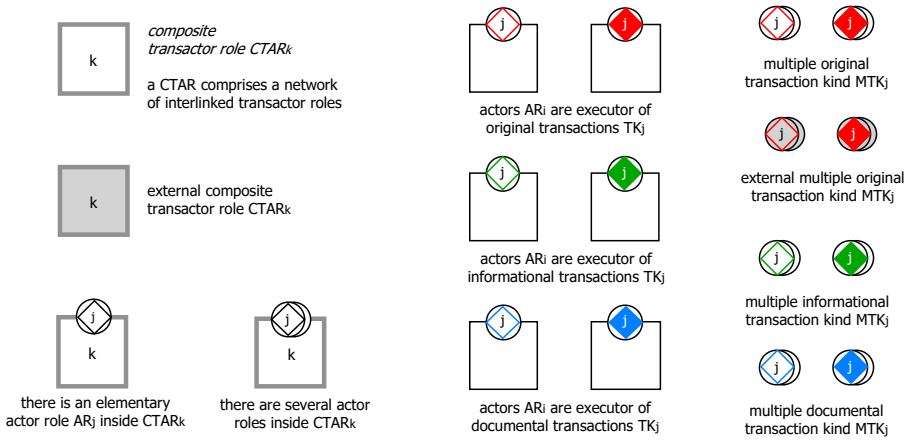


Fig. 4 Legend of the Coordination Structure Diagram (2)

The *interaction structure* consists of initiator links between transactor roles and transaction kinds. Through this structure, trees of transactor roles emerge, as illustrated by Fig. 5. It is the CSD of the case GloLog (cf. [1], Chap. 18). Initiator links are indicated by solid lines between transactor roles and transaction kinds. A cardinality range (k..n) may apply, as shown in Fig. 3.

By definition, the top of such an interaction tree is a self-activating transactor role (cf. OMEGA theory in [1]). The organisation of an enterprise, or any other SoI, often contains only subtrees of these trees. Then, the cut-off upper part is indicated by a composite transactor role. Most business processes in enterprises start in this way.

The *interimpediment* structure consists of *wait links* from transaction kinds to actor roles. A wait link expresses that actors in the connected actor role have to wait for a specific progress in transactions of the connected transaction kind before they can proceed their work (in their own transactions). In other words, the initiators or executors of these transactions impede actors in the connected actor role to carry on as

long as the wait condition holds. A wait link is indicated by a dotted arrow from a transaction kind to the impeded actor role. The interimpediment structure constitutes the *process dependencies* among the corresponding transaction processes: the acts of the connected actors depend on the progress of the connected transaction processes.

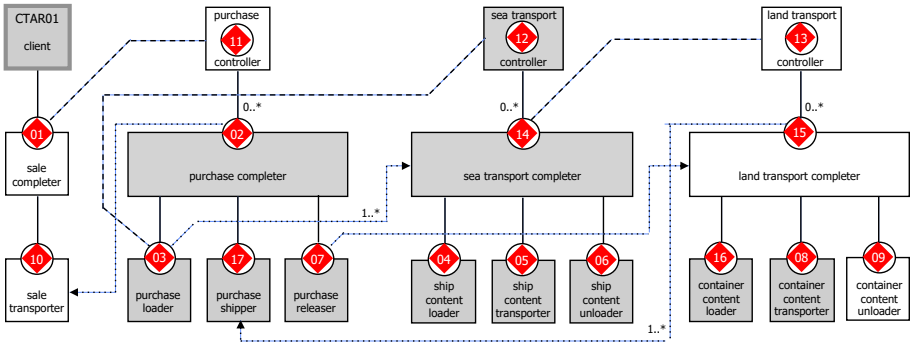


Fig. 5 The three coordination structures in the GloLog enterprise

If one abstracts from the realisation of the O-organisation of an SoI, and thus aims at producing its essential model (cf. ALPHA theory in [1]), a third coordination structure comes on the scene. This *interstriction*[6] *structure* consists of *access links* from actor roles to transaction kinds, which are now conceived as transaction banks. Access links are the ontological abstraction of the sharing transaction kinds between the O-organisation and the I-organisation of the SoI (cf. ALPHA theory in [1]). An access link expresses that actors in the connected actor role have reading access to the contents of the transaction bank (both to the C-facts and to the P-facts). Access links are indicated by dashed lines between actor roles and transaction kinds. The interstriction structure constitutes the *state dependencies* among the corresponding transaction processes: the acts of the connected actors depend on the current state of the connected transaction processes, represented by the facts in the transaction bank

The other transaction kinds between the O- and the I-organisation, so the remembering transaction kinds are abstracted from by considering the facts that are created by the initiator and the executor of a transaction to be stored in the transaction bank.

Fig. 17 shows the fact types in the schema of the CM of which instances are expressed in a CSD: the entity types ‘transaction kind’, ‘actor role’, their combination in the concept of ‘transactor role’, as well as the entity types ‘composite transactor role’ and ‘multiple transaction kind’, and the property types ‘executor link’, ‘initiator link’, ‘access link’, ‘wait link’ and ‘is part of’.

As an example, the CSD in Fig. 5 contains 18 transactor roles. Three of them are self-activating: TAR11, TAR12 and TAR13. There is one composite transactor role: CTAR01. The fourteen initiator links constitute the interaction structure. Next, there are four wait links: one from TK02 to AR10, one from TK03 to AR14, one from TK07 to AR15, and one from TK15 to AR17, together constituting the interimpediment structure. Lastly, there are three access links: one from AR11 to TK01, one from

AR12 to TK03, and one from AR13 to TK14, together constituting the interstriction structure. Note that the access links to external information sources are omitted. Otherwise, there would have been many more access links.

Following the OER method ([1], Chap. 12), the CSD of an organisation is obligatorily supplemented by a *Transactor Product Table* (TPT), for the sake of formulating product kinds formally and properly, right from the beginning. A TPT is a table of transaction kinds with their corresponding product kinds and executing actor roles (or executor roles). The syntax of a TPT entry is specified in EBNF as follows:

TPT entry = (transaction kind id, transaction kind name), (product kind id, product kind formulation), (actor role id, actor role name);
product kind formulation= entity variable | property variable | attribute variable, "**is**", perfect tense verb;

Examples: [rental] **is** contracted
the car **of** [rental] **is** returned
the fee **of** [membership] **in** [year] **is** paid

Fig. 21 shows the fact types in the schema of the CM of which instances are expressed in a TPT: the entity types 'transaction kind', 'product kind' and 'actor role', and the property types that determine the product kind of each transaction kind as well as its executing actor role. Because the product kinds are identical to the independent P-fact types in the FM, the TPT is called a cross-model table; it bridges the CM and the FM. Table 2 exhibits the TPT of the case Rent-A-Car (cf. [1], Chap. 15).

transaction kind	**product kind**	**executor role**
TK01 rental completing	PK01 [rental] **is** completed	AR01 rental completer
TK02 car taking	PK02 **the** car **of** [rental] **is** taken	AR02 car taker
TK03 car returning	PK03 **the** car **of** [rental] **is** returned	AR03 car returner
TK04 deposit paying	PK04 **the** deposit **of** [rental] **is** paid	AR04 deposit payer
TK05 invoice paying	PK05 **the** invoice **of** [rental] **is** paid	AR05 invoice payer

Table 2 TPT of the Rent-A -Car organisation

A CSD may be supplemented by a *Bank Contents Table* (BCT). This is a table that shows the fact types of which instances are created or used by the initiators or executors of the identified transaction kinds. These instances are considered to be contained in the corresponding transaction banks. The BCT of an organisation bridges its CM and FM. Table 3 exhibits the BCT of the case Volley (cf. [1], Chap. 12).

The syntax of a BCT entry is specified in EBNF as follows:

BCT entry = transaction kind id, (transaction kind name | multiple transaction bank id), multiple transaction bank name, (object class name | product kind formulation | property variable | attribute variable);

In Table 3, the fact types are grouped according to the transaction banks in which they are stored. P-fact types whose instances are used within the organisation but created outside it, are also listed in the BCT. Because one commonly doesn't know the specific transaction bank in which they reside, a multiple transaction bank is included that is considered to encompass this transaction bank. Fig. 22 shows the fact types in the schema of the CM of which instances are expressed in a BCT: the entity types 'transaction kind', 'multiple transaction kind' and 'P-fact type,' and the property type that determines which facts are contained in which transaction bank.

bank	**independent/dependent facts**
TK01 membership starting	MEMBERSHIP [membership] **is** started **the** starting day **of** [membership] **the** member **of** [membership] **the** amount to pay **of** [membership]
TK02 membership paying	**the** first fee **of** [membership] **is** paid **the** amount paid **of** [membership]
MTK01 persons facts	PERSON **the** day of birth **of** [person]
MTK02 Volley facts	YEAR **the** minimal age **in** [year] **the** annual fee **in** [year] **the** max members **in** [year]

Table 3 BCT of the Volley organisation

A CSD may also be supplemented by a *Bank Access Table* (BAT). It is an alternative representation of the interstriction structure of an organisation. A BAT is particularly suitable if there are many access links, whose expressions in a CSD would lead to a mess of crossing dashed lines. Table 4 shows the BAT of the Library organisation (cf. [1], Chap. 16). A "U" (from Uses) indicates that there is an access link from the actor role (in the row) to the transaction bank (in the column). The access links from the executor and initiator roles to transaction kinds are indicated respectively by "Ex" and "In". As follows from the PSI theory in [1], both the initiator and the executor of a transaction need access to the facts that are produced during the transaction process.

Fig. 23 shows the fact types in the schema of the CM of which instances are expressed in a BAT: the entity types 'transaction kind', 'multiple transaction kind', 'actor role' and 'composite actor role, and the property types that determine the existence of executor links, initiator links and access links. The executor links and initiator links are included because they imply the existence of an access link.

Bank Actor	TK01	TK02	TK03	TK04	TK05	TK06	TK07	TK08	TK09	MTK01	MTK02	MTK03
AR01	Ex	In								U	U	U
AR02		Ex									U	
AR03	U	In	In, Ex	U						U	U	
AR04	U			Ex				U	U	U	U	
AR05	U			In	In, Ex					U	U	
AR06	U			U		Ex	In	In	In	U	U	U
AR07							Ex				U	
AR08								Ex			U	
AR09									Ex		U	
CTAR01	In			In							U	
CTAR02						In					U	

Table 4 BAT of the Library organisations

4 The Action Model

The Action Model (AM) of an organisation is the ontological model of its operation (cf. PSI theory and DELTA theory in [1]). It comprises action rules and work instructions. In practice, the AM of an organisation is often incomplete or even absent. In such a case, the actors are supposed to base their decisions on their professional and general knowledge.

Action rules guide actors in responding to coordination events. As discussed in the PSI theory [1], actors are basically autonomous in deciding how to respond to coordination events, as well as how to perform production acts. In principle, there is an action rule for every coordination event kind whose occurrences have to be responded to. It specifies the facts in the production world and/or the coordination world whose presence or absence in the current state of the world must be assessed, as well as the (production and/or coordination) acts that must be performed, depending on the outcome of the assessment. In current practice, action rules are commonly called (imperative) *business rules* [3]. Below, the *Action Rule Specifications* are defined in EBNF. Multiple use is made of already defined terms, to keep the definition as short as possible. The bracket pair "[" and "]" indicates that the enclosed part is optional.

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

event part= agendum clause, [while clause], [with clause];

agendum clause = "**when**", transaction kind name, "**for**", entity variable, "**is**", perfect tense intention;

while clause = "**while**", {transaction kind name, "**is**", perfect tense intention}-;

with clause = "**with**", {(property variable, indefinite entity reference) | (attribute variable, indefinite value reference) }-;

assess part = rightness division, sincerity division, truth division;
rightness division = "*rightness*:" {property condition | attribute condition }- |
<informal specification>;
sincerity division = "*sincerity*:" {property condition | attribute condition }- |
<informal specification>;
truth division = "*truth*:" {property condition | attribute condition }- |
<informal specification>;
<informal specification> = a text between "*" and "*";

response part = "**if**", "performing the action after **then** is considered justifiable", "**then**", action clause, ["**else**" action clause];
action clause = {present tense intention, transaction reference, {with clause}}-;
transaction reference = transaction kind name, "**for**", entity type name;

As an example of an ARS, we present the specification of the action rule from the case Rent-A-Car (cf. [1], Chap. 15), in which the declaration of the deposit payment is responded to.

when deposit paying **for** [rental] **is** declared (TK04/da)

assess *rightness*: **the** performer **of the** declaration **is the** deposit payer **of** [rental]
the addressee **of the** declaration **is**
the rental completer **of** [rental]
sincerity: * the performer seems sincere in performing the declaration *
truth: **the** declared ot **of** deposit paying **for** [rental] **is within**
the promised ot **of** deposit paying **for** [rental];
the declared deposit amount **of** deposit paying **for** [rental] **is**
equal to the promised deposit amount **of** deposit paying
for [rental]
if *performing the action after **then** is considered justifiable*
then accept deposit paying **for** [rental] [TK04/ac]
to the performer **of the** declaration
else reject deposit paying **for** [rental] [TK04/rj]
to the performer **of the** declaration
with % < explanation of the reason for rejecting **> %**

Work instructions guide the executor of a transaction in performing the production act. Although they are sometimes expressed in work flows, these work flows do not represent business processes but processes in the production world. In DEMO only the final effect of production processes, i.e. the resulting product, is remembered.

Fig. 18 shows the position of action rules and work instructions in the combined conceptual model of the CM, AM and PM. Every action rule applies to one particular transaction kind step kind, but there may be several action rules that apply to the same

coordination event kind. As an example from the case Rent-A-Car (cf. [1], Chap. 15), the action rule specifications ARS-3, ARS-7 and ARS-9 all apply to the event kind TK01/pm, although with different additional while-conditions. Work instructions however are considered to apply to the execution of precisely one product kind, and vice versa, since they are basically product kind specific.

In order to show precisely the delegations of authority (cf. PSI theory in [1]), one may add an *Authorisation Delegation Table* (ADT). An ADT bridges the AM and the implementation of the organisation, in particular the assignment of tasks (T) to task performers (P). The definition of the ADT is presented in Fig. 25. A distinction is made between a global and a detailed ADT. The columns of an ADT represent tasks (T), ranging from single process steps (in a detailed ADT) to the responsibility ranges of actor roles (in a global ADT). The rows represent the task performers (P), ranging from functionaries to complete enterprises. An "A" at the crossing of a column and a row indicates that the performer is authorised to perform the task, a "D" that he/she has delegated authority. In a global ADT, only A's can occur since it is by definition not possible to delegate a complete actor role.

Table 5 exhibits the detailed ADT of the case Volley [1]. It shows that the functionary Secretary has delegated the authority to perform C-act kinds TK01/dc, TK01/da and TK02/rq to the functionary Administrator.

T/P	TK01/dc	TK01/da	TK02/rq
Secretary	A	A	A
Administrator	D	D	D

Table 5. Detailed ADT of the case Volley

5 The Process Model

The Process Model (PM) of an organisation is the ontological model of the state space and the transition space (cf. DELTA theory in [1]) of its coordination world. Regarding the state space, the PM contains, for all internal and all border transaction kinds, the process step kinds as well as the applicable existence laws, in full accordance with the Complete Transaction Pattern (CTP) (cf. Fig. 6). Regarding the transition space, the PM contains, for all internal and all border transaction kinds, the process step kinds as well as the applicable occurrence laws, including the cardinalities of the occurrences, in full accordance with the CTP. Because the intra-transaction occurrence laws are fully determined by the CTP, a PSD contains only the inter-transaction occurrence laws, expressed in process links between process steps in different transactions. There are two kinds of them: response links and wait links. By a *process step* is understood the performing of a C-act and the becoming existent of the corresponding C-fact. Consequently, a process step kind is the combination of a C-act kind and the corresponding C-fact kind.

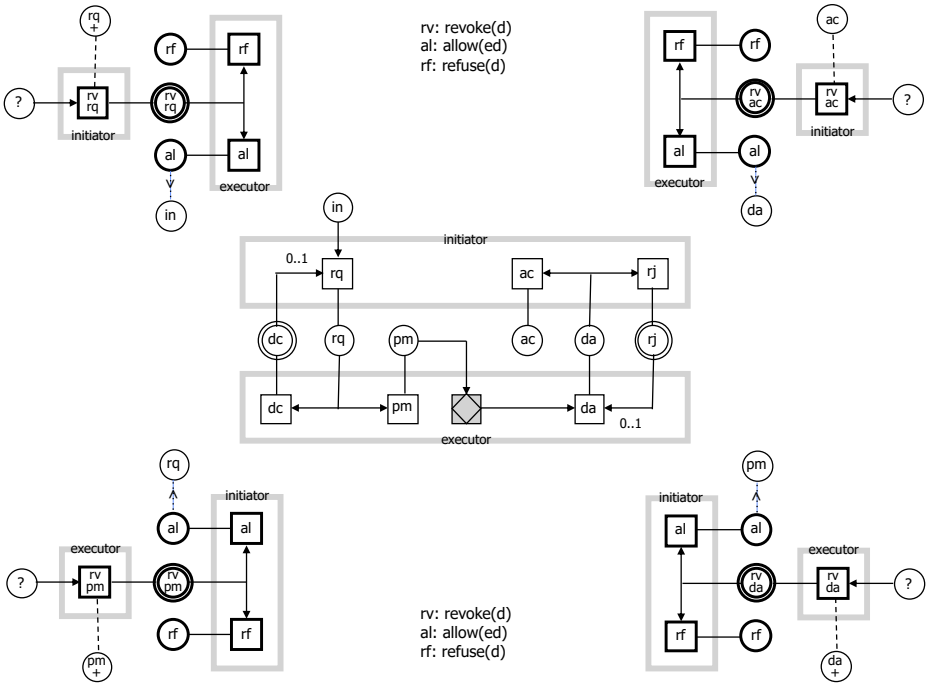
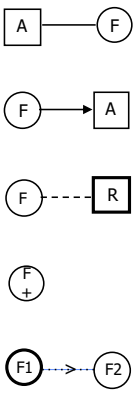


Fig. 6 TPD of the complete transaction pattern (CTP)

The PM of an organisation connects its CM and AM, as far as coordination is concerned (cf. Fig. 1). A PM is expressed in a Process Structure Diagram (PSD), optionally supplemented by Transaction Process Diagrams (TPD) and a Create Use Table (CUT). Because the PSD is based on the Complete Transaction Pattern (CTP), we present the Transaction Process Diagram (TPD) of the CTP in Fig. 6. The legend of the TPD is explained in Fig. 7.1.



causal link: performing C-act A causes the becoming existent of C-fact F; this is identical to saying that performing A causes the occurrence of the C-event F

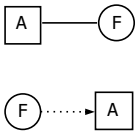
response link: C-act A is performed in response to the occurrence of C-event F

conditional link: in order to perform the *revocation* C-act R successfully, the C-fact F must exist (which is identical to saying that reached state F in the main process is reached)

F+ means: the state of the main process is F or further; e.g., if F is 'declared', then F+ means that the state is 'declared' or 'accepted' or 'rejected'

reversion link: reaching state F1 (in a revocation process) implies the instantaneous transition to state F2 in the main process

Fig. 7.1 Legend of the transaction process diagram (TPD)



causal link: performing C-act A causes the becoming existent of C-fact F; this is identical to saying that performing A causes the occurrence of the C-event F

wait link: performing C-act A has to wait for the occurrence of C-event F

Fig. 7.2 Legend of the Process Structure Diagram (PSD)

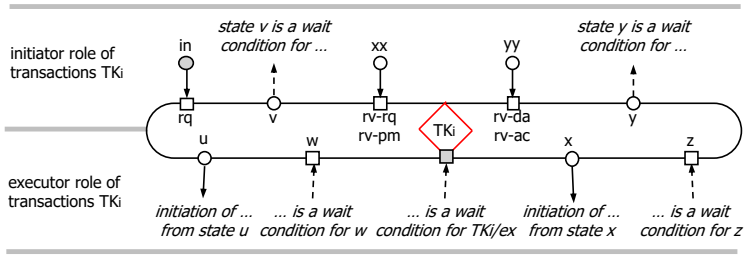


Fig. 8 General shape of the PSD

Figure 8 exhibits the general shape of transaction kinds in a *Process Structure Diagram* (PSD). The sausage-like shape arises from stretching the disk shape horizontally. One must imagine that there is a (non-proportional) linear time axis from left to right. The sort of the shown transaction kind is original, indicated by the red diamond (cf. Fig. 4). As discussed in the PSI theory in [1], a transaction proceeds in three phases: the order phase (to the left of the diamond), the execution phase (the diamond), and the result phase (to the right of the diamond). The state "in" is the initial state of the transaction process; it is some state in some transaction process. The indicators of the other states, as well as the indicators of the presented C-acts, are arbitrarily chosen letters. In order to show that response links and wait links can apply both to the order phase and to the result phase, they are drawn on both sides (u, v, w for the order phase, and x, y, z for the result phase). The P-act (named TKi/ex, "ex" from "execute") is indicated by a grey-coloured small box.

In a PSD, only those C-act kinds and C-fact kinds are shown that are connected to other transaction processes by means of *response links* or *wait links* (cf. Figs. 7.1 and 7.2). The shapes of C-act kinds (the small boxes) that are performed by the initiator are drawn on top of the 'sausage', so within the responsibility area of the initiator, whereas the shapes of C-act kinds that are performed by the executor are drawn at the bottom, so within the responsibility area of the executor, as is the shape of the P-fact kind (the small grey box). In principle, the same rule holds for the shapes of C-fact kinds (the small disks). So, the shapes of C-fact kinds created by the initiator are drawn on the top of the 'sausage', and the shapes of C-fact kinds created by the executor are drawn at the bottom. However, to avoid that the drawing of response links and wait links become a mess, it is allowed to put the shapes of C-fact kinds on the other side. For the same reason, one may duplicate the shapes of C-act and C-fact kinds. The states xx and yy represent states from which the revoke request or revoke promise, and the revoke declare or revoke accept are performed respectively.

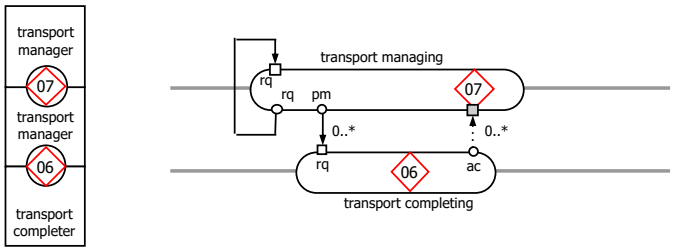


Fig. 9 PSD of the car transportation process of Rent-A-Car

The response links in Fig. 8 from C-fact kinds to (unspecified) C-act kinds are called *initiation links*. By definition, they start from the shape of a C-fact kind and end in the shape of the request act of some transaction kind. A C-fact shape may have several outgoing initiation links, meaning that transactions of several transaction kinds are initiated from it. Similarly, 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. Likewise, 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.

To response links as well as to wait links, cardinality ranges apply. A cardinality range *k..n* for a response link means that the C-act at the arrow side is performed a minimum number of times *k* and a maximum number of times *n*. The default value of *k* and *n* is 1; default values are commonly omitted in a PSD. Likewise, a cardinality range *k..n* for a wait link means that performing the C- or P-act at the arrow side is postponed until a minimum number of *k* and a maximum number of *n* C-events at the shaft side have occurred. The default value of *k* and *n* is again 1; default values are commonly omitted in a PSD. In order to illustrate the cardinality ranges in a PSD, Fig. 9 exhibits one of the PSDs from the case Rent-A-Car (cf. [1], Chap. 15). It shows in addition how self-activation is expressed in a PM, namely by a response link from the state promised (pm) to the act request [rq].

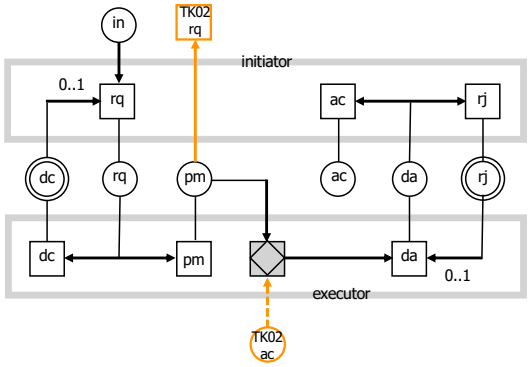


Fig. 10 The use of a TPD as a supplement to a PSD

Fig. 19 shows the fact types in the schema of the PM of which instances are expressed in a PSD: the entity types ‘transaction kind’, ‘actor role’, ‘transaction kind step kind’ and ‘general step kind’, as well as the property types ‘executor link’, ‘initiator link’, ‘response link’ and ‘wait link’. Note that every ‘sausage’ contains the complete transaction pattern. In order to show precisely the connections to and from other transaction kinds, the PSD of an organisation may be supplemented by a number of *Transaction Process Diagrams* (TPD).

An example of the use of a TPD in the case Volley, to show precisely the connections between the transaction kinds TK01 and TK02, is exhibited in Fig. 10. To save space, only the standard pattern of TK01 is shown. In the colour tangerine, the connections between the patterns of transaction kinds TK01 and TK02 are shown. In response to the event (TK01/pm), the act [TK02/rq] is performed. The performing of the P-act [TK01/ex] has to wait for the occurrence of (TK02/ac).

The PSD (and TPDs) of an SoI may be supplemented by a *Create Use Table* (CUT). A CUT is a cross-model table that connects the PM and the FM. It shows in which transaction steps instances of the fact types in the FM (cf. Sec. 6) are created (as an effect of performing the step) and in which steps they are used (in settling the agendum). The contents of a CUT is fully determined by the AM of the considered organisation. The syntax of a CUT entry is specified in EBNF as follows:
CUT entry = object class | P-fact type, C-act kind | “<derived>” | “<given externally>” | “<provided as parameter>” , C-fact kind;

P-fact type	**created in performing**	**used when settling**
MEMBERSHIP	TK01/rq	
PAID MEMBERSHIP	<derived>	
PERSON	<given externally>	
YEAR	<given externally>	
[membership] **is** started	TK01/ac	
the first fee **of** [membership] **is** paid	TK02/ac	TK01/pm
the member **of** [membership]	<provided as parameter>	TK01/rq, TK01/pm
the payer **of** [membership]	<provided as parameter>	TK01/rq, TK01/pm, TK01/da
the starting day **of** [membership]	<provided as parameter>	TK01/rq
the day of birth **of** [person]	<given externally>	TK01/rq
the minimal age **in** [year]	<given externally>	TK01/rq
the max members **in** [year]	<given externally>	TK01/rq
the annual fee **in** [year]	<given externally>	TK01/rq
the amount to pay **of** [membership]	TK02/rq	
the amount paid **of** [membership]	TK02/da	TK02/da
the first fee **of** [membership]	<derived>	TK01/rq, TK02/da
the number **of** members **on** [day]	<derived>	TK01/rq
the age **of** [person] **on** [day]	<derived>	TK01/rq

Table 5 CUT of the case Volley

As an example, Table 5 shows the CUT of the case Volley. All fact types, so entity types, value types, event types, property types and attribute types, that occur in the

FM are listed in the first column of the table. In the second column one indicates the acts by which facts of the type in the left column are *created*. For fact types whose instances are contained in external transaction banks, the indication is "<given externally>". For fact types whose instances are provided as parameter values in the with-clause of a when-clause of an action rule concerning the C-event type in the third column, the indication in the second column is "<provided as parameter>". Derived fact types are indicated by "<derived>". They have to be included in the Derived Fact Specifications, as part of the FM. In the third column one indicates the agenda during whose settling facts of the type in the left column are *used* (except for the entity and value types since they are already indirectly used in the property and attribute types).

6 The 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 (cf. DELTA theory in [1]). Regarding the *state space*, the FM contains the entity types, value types, property types and attribute types that are relevant for the modelled organisation as well as the existence laws that apply (cf. MU theory in [1]). The *transition space* of a production world is fully determined by the transition space of the corresponding coordination world (cf. PSI theory and DELTA theory in [1]), and thus by the PM of the considered SoI.

The FM of an organisation connects its CM and AM, as far as production is concerned (cf. Fig. 3). An FM is expressed in an *Object Fact Diagram* (OFD), supplemented by (textual) *Derived Fact Specifications* (DFS). If needed, (textual) existence laws may be added.

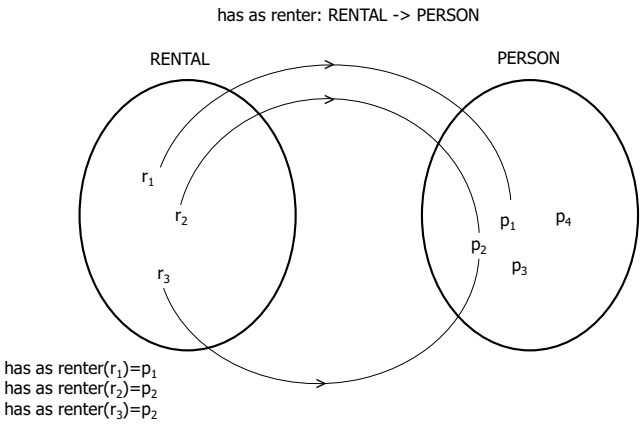


Fig. 11 Venn diagram notation of a (mathematical) function

Set theory and mathematical function theory help in understanding the relationships between the schema level and the instance level of the conceptual model of a world. Therefore, we base the explanation of the FM on it. The common way of representing sets in set theory is the Venn Diagram. In such a diagram, the shape of a set is an oval; symbols within the oval represent elements of the set (cf. Fig. 11). The

common way of representing functions (or binary relations in general) is to extend the Venn Diagram with connections between the elements of two (not necessarily different) sets. One set is called the domain of the function, the other one the range. A function maps the elements in the domain to the elements in the range. Fig. 11 exhibits an extended Venn Diagram, representing the function 'has as renter', having as domain the class RENTAL and as range the class PERSON.

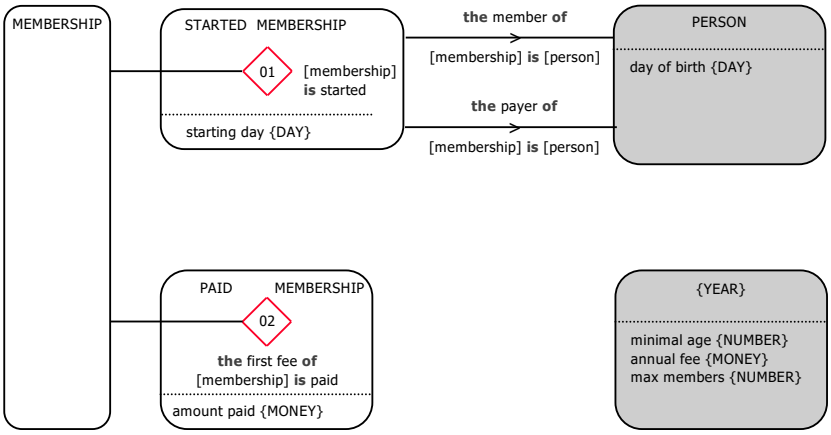


Fig. 12 OFD of the Volley organisation

The OFD is derived from this extended Venn Diagram: it consists of classes and of mappings between them. The classes are either *object classes* (i.e. sets of concrete objects of the same type) or *value classes* (i.e. sets of abstract objects of the same type, cf. FI theory [1]). The shape of a set or class in an OFD is a roundangle (the name is a contraction of "rounded rectangle"). The mappings between these classes represent property types or attribute types.

Property types are indicated by directed lines between classes. As an example in Fig. 12, the property type 'the member of [membership] is [person]' is a function that maps the class MEMBERSHIP to the class PERSON. One should imagine that the line between the roundangles represents the bunch of connections between elements in MEMBERSHIP and elements in PERSON. The ">" indicates that MEMBERSHIP is the domain of the function and PERSON the range.

Attribute types are indicated in a simpler way. This is possible because they are always pure (mathematical) functions, i.e. functions of which the cardinality range at the domain side is 0..*, and at the range side 1..1. The name of the attribute type is written in the roundangle of the class that is its domain. To the right of it, the name of the value class that is the range is written, between "{" and "}". The name of an object class and the list of attribute types that have this object class as domain, is separated by a dotted line.

Production *event types* are indicated by diamonds, the universal symbol of production (cf. PSI theory in [1]). They are expressed as (mostly) unary predicates concerning an entity type or class. For example, the event type 'the first fee of [membership]

is paid' concerns the entity type membership (or the object class MEMBERSHIP). An event type in the FM is identical to a product kind in the CM. Therefore, the numeral part of the product kind identifier (e.g. 02) is written in the diamond.

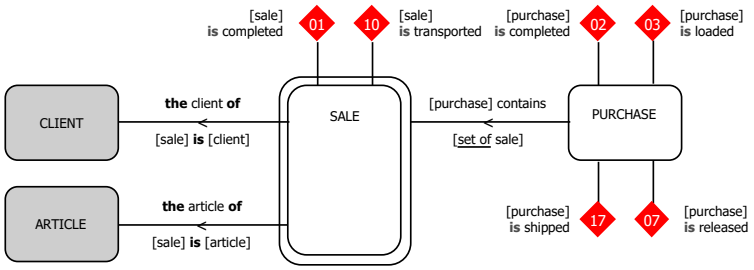


Fig. 13 part of the OFD of the GloLog organisation

In Fig. 13 a part of the OFD of the GloLog organisation ([1], Chap. 18) is exhibited in order to show how sets are expressed in an OFD. In the example, we need the class SET OF SALE (the extension of 'set of sale') as the range of the property 'contains', which has as domain PURCHASE. This class can graphically be defined by drawing a larger roundangle around the roundangle of SALE. The inverse of set of is member of. Thus, a sale is a member of a purchase. Another example of the use of this construct is the schema of the FM in Fig. 16.

Derived entity types can often be specified graphically, as is done in Fig. 12 for 'started membership' and 'paid membership'. They allow precise specifications of the attribute types 'starting day' and 'amount paid': both are functions with as domain the object classes STARTED MEMBERSHIP and PAID MEMBERSHIP respectively. Standard value classes like DAY and MONEY are assumed to be implicitly present in every OFD (cf. Sec. 2.3). The value class YEAR is explicitly included in the OFD in Fig. 12 because of the attribute types (that have to be specified) that have YEAR as their domain: 'minimal age', 'annual fee ' and 'max members'.

An OFD also exhibits the *existence laws* that can conveniently be specified graphically. For example, the OFD in Fig. 12 shows that the domain of the property type 'the member of [membership] is [person]' is the class MEMBERSHIP and that the range is PERSON. In addition it shows that every membership has exactly one person as its member, whereas a person can be member in 0, 1 or more memberships. This follows from the (default) cardinality range. Existence laws that cannot be specified graphically, must be specified textually.

External object classes, like PERSON, are coloured light-grey. It means that persons are created outside the focus of the SoI. But it must be possible to inspect their existence and to use their properties or attributes, like the day of birth. All standard value classes are external, as discussed before, and thus also coloured light-grey. For the complete legend of the OFD, the reader is referred to Sec. 6.3.3, since the graphical formalism of the language GOSL presented there, is identical to the graphical formalism of the OFD. Fig. 16 presents the meta schema of the FM. As one may expect, it is identical to the general meta schema (cf. MU theory in [1]).

As said in the introduction, derived fact types (of all kinds) that cannot be specified graphically, must be specified textually. As follows from the CUT in Table 4 and the OFD in Fig. 12, there are three attribute types in the case Volley that have to be specified textually. It is done in Fig. 14. Days are values in the Julian time dimension (cf. Table 1). So, the age of a person is expressed in the number of days that the person exists. If needed, it can be transformed to years in the Gregorian calendar or in any other calendar (cf. Sec. 2.4). Fig. 14 also contains the existence laws that apply to Volley. They are the *declarative* counterparts of the (imperative) business rules [1] or action rules that are discussed in Sec. 4. In other words, action rules, or imperative business rules in general, are the operationalisation of declarative business rules, which are basically first order logical formulas concerning the production world of an organisation.

Derived Fact Specifications

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]) +1) **times the** annual fee **in the** year **of the** starting day **of** [membership]

Existence Laws (Declarative Business Rules)

[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]
[membership] **is** started **on** [day] **implies that the** number of members **on** [day] **is less than or equal to the** max members **in the** year **of** [day]

Fig. 14 Derived Fact Specifications and Existence Laws of the Volley organisation

7 Appendix

Hereafter, Figs. 15 thru 25 are presented. They constitute an appendix to the document, because inserting them in the text would worsen the readability.

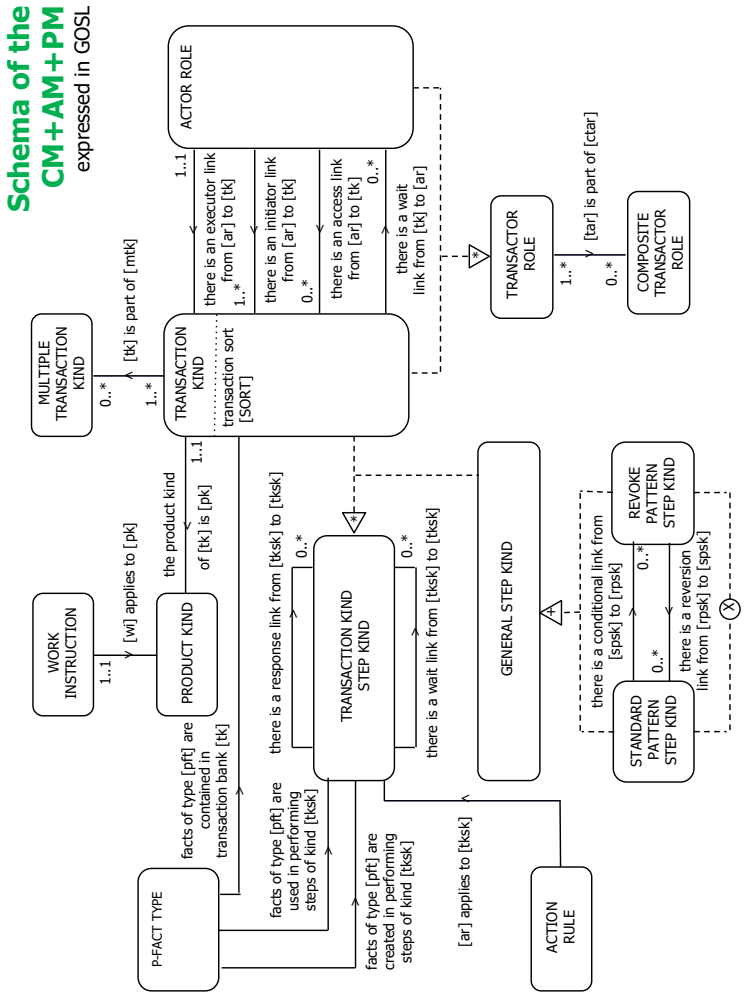


Fig. 15 Combined schema of the CM, AM, and PM

The OFD above is an expression of a schema in GOSL (cf. MU theory in [1]) that specifies the state space of the 'world' that is covered by the CM, the AM and the PM of an organisation.

Recall that the default cardinality range of a property type at the domain side is 0..* and at the range side 1..1. Default values are commonly not indicated in a schema.

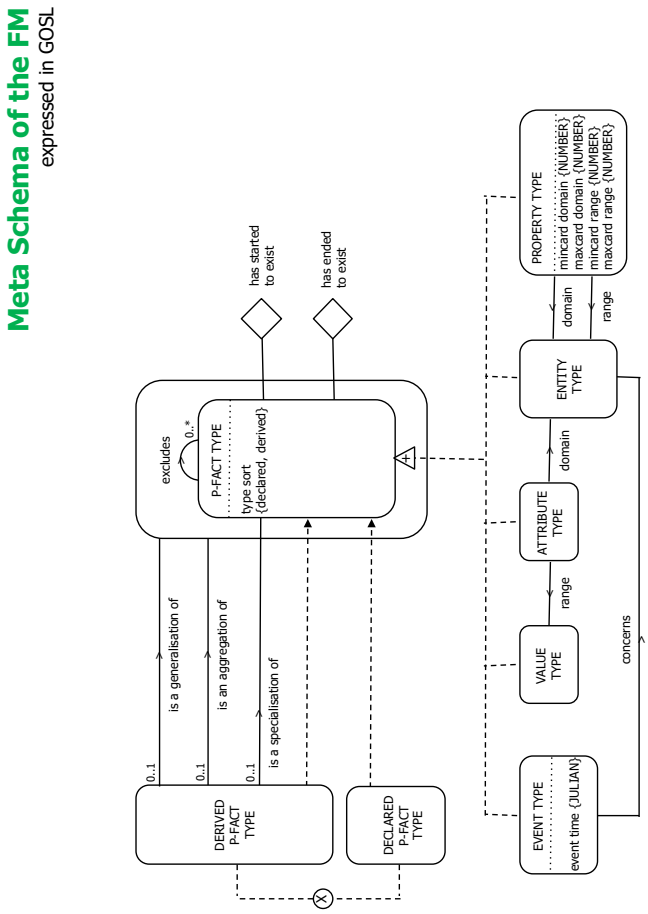


Fig. 16 Meta schema of the FM

The OFD above is an expression of the meta schema of the state space of the production world of an organisation, in GOSL. An example of a state space schema is the one that is exhibited in Fig. 12.

Note that the property types are formulated in a concise form: the references to the elements in the domain and the range are omitted.

Recall that the default cardinality range of a property type at the domain side is 0..* and at the range side 1..1. Default values are commonly not indicated in a schema.

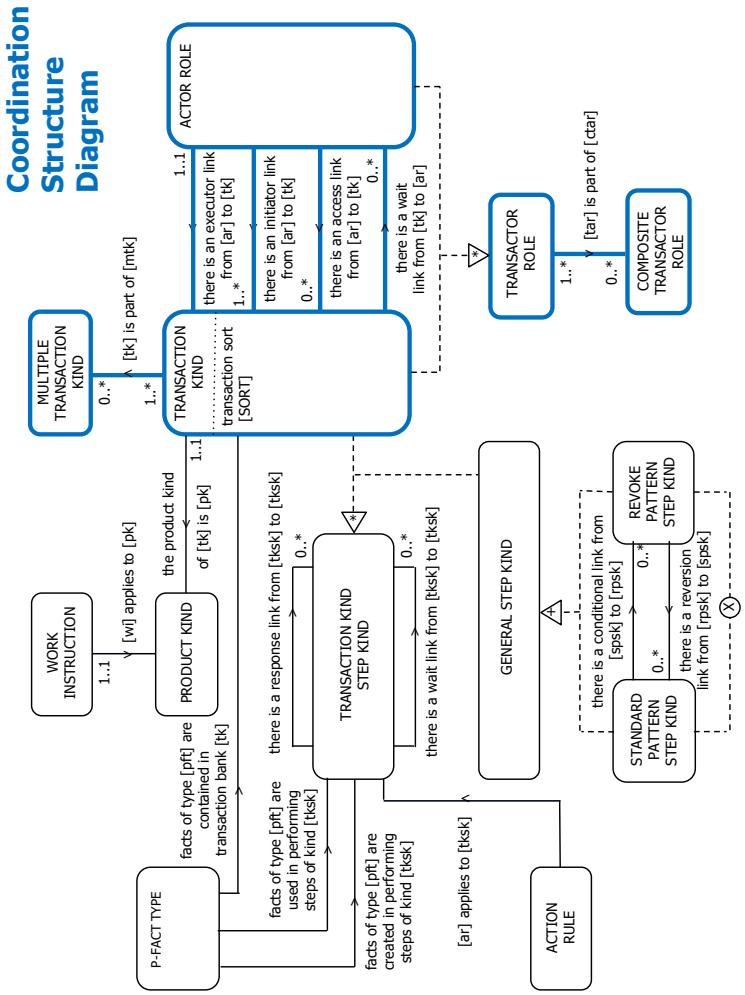


Fig. 17 Definition of the Coordination Structure Diagram (CSD)

The blue coloured and bold-lined parts above collectively define the (semantic) contents of a *Coordination Structure Diagram* (CSD). Thus, every CSD represents the existence, in the chosen SoI, of a number of transaction kinds, actor roles (and consequently transactor roles) as well as composite transactor roles and multiple transaction kinds. In addition, it represents the existence of a number of executor links, initiator links, access links and wait links. Note that a transactor role is the combination of a transaction kind and the actor role that has its executor role.

The instances of the property type 'is part of' (which exist between transaction kinds and multiple transaction kinds, as well as between transactor roles and composite transactor roles) may be implicitly given. It is important yet to know that a multiple transaction kind is a collection of transaction kinds, and that a composite transactor role is a collection of transactor roles. Fig. 5 exhibits an example of a CSD.

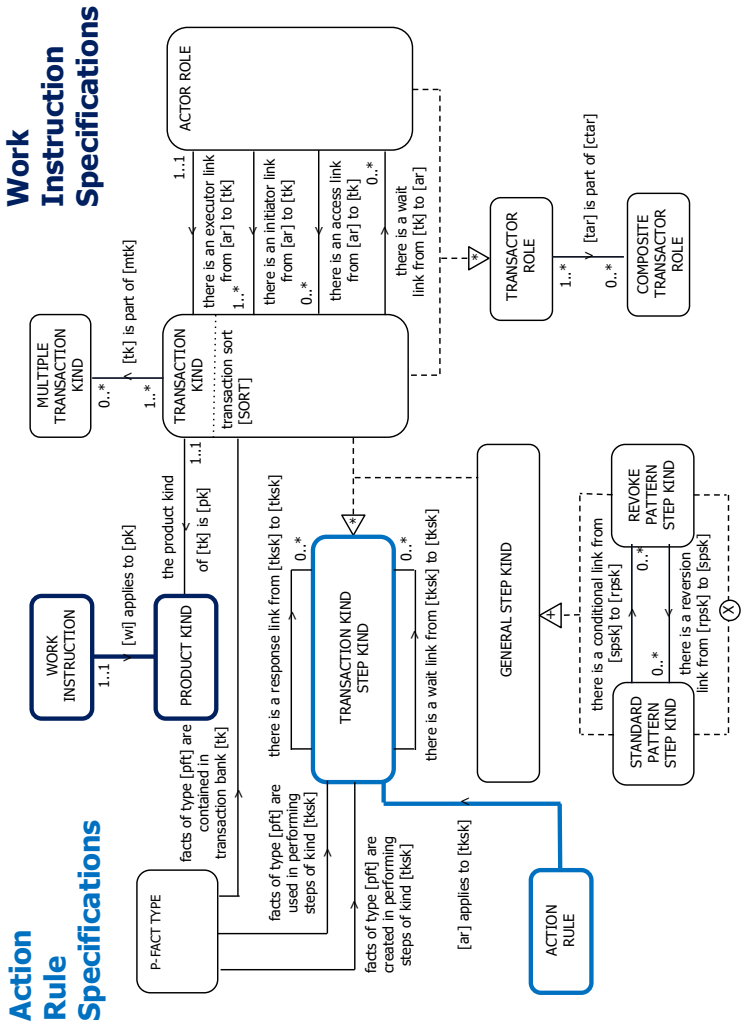


Fig. 18 Position of the action rules and the work instructions

The meaning of the blue coloured and bold-lined parts above is that every action rule (like the example rule in Sec. 4) applies to one transaction kind step kind, e.g. TK04/da in the essential model of the Rent-A-Car organisation (cf. [1], Chap. 15), but there may be several action rules that apply to the same transaction kind step kind (or co-ordination event kind). They differ however in the contained while clauses. Action rules are expressed in Action Rule Specifications (ARS).

The meaning of the dark blue coloured and bold-lined parts above is that every work instruction applies to one product kind, and vice versa. In other words, work instructions are product specific. Work instructions are expressed in Work Instruction Specifications (WIS).

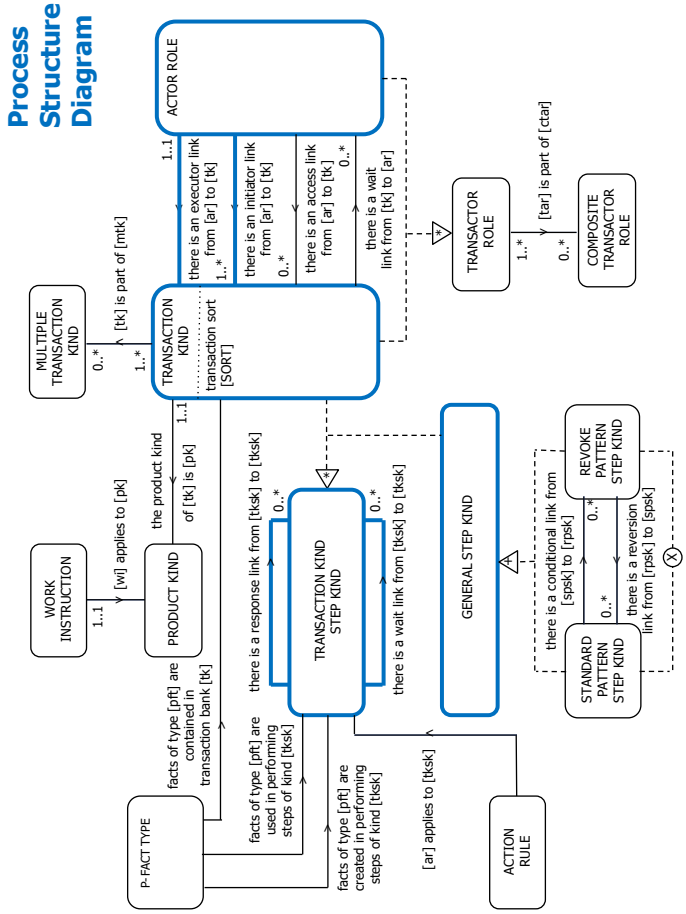


Fig. 19 Definition of the Process Structure Diagram (PSD)

The blue coloured and bold-lined parts above collectively define the (semantic) contents of a *Process Structure Diagram* (PSD). Thus, every PSD represents the existence, in the chosen SoI, of a number of transaction kinds and actor roles, as well as transaction kind step kinds, where every transaction kind step kind (e.g. TK04/da) is defined as the aggregation of a general step kind (e.g. 'da') and a transaction kind (e.g. TK01). In addition, it represents the existence of a number of executor links and initiator links, as well as a number of wait links between transaction kind step kinds. Fig. 9 exhibits an example of a PSD.

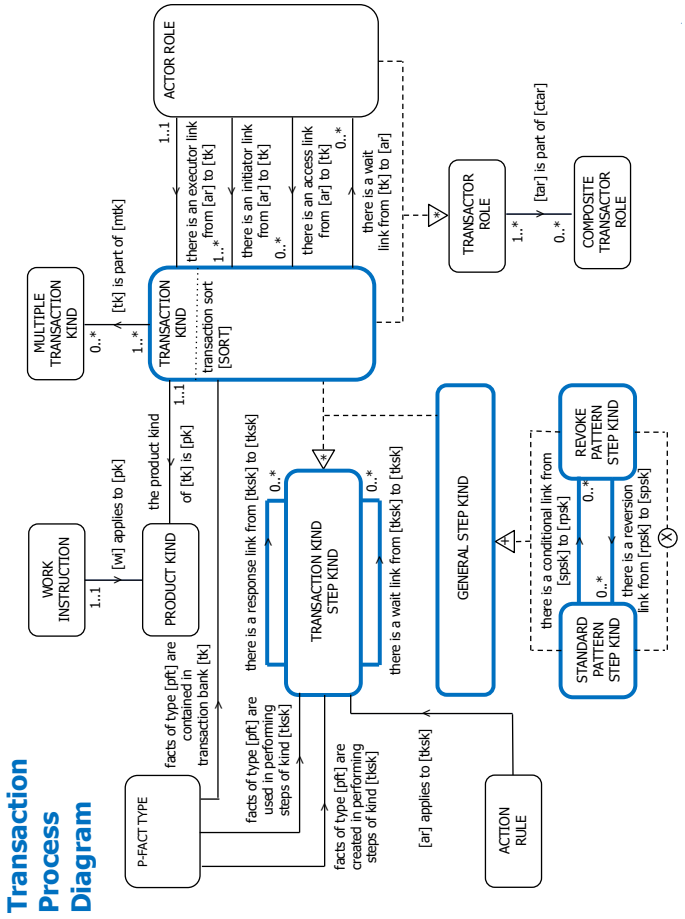


Fig. 20 Definition of the Transaction Process Diagram (TPD)

The blue coloured and bold-lined parts above collectively define the (semantic) contents of a *Transaction Process Diagram* (TPD). Fig. 6 exhibits the complete transaction pattern (CTP) expressed in a TPD. A typical use of this TPD is discussed in the case Fixit (cf. [1], Chap. 13). Another typical use is to show precisely the interrelationships of transactions. An example of this way of using the TPD is exhibited in Fig. 10.

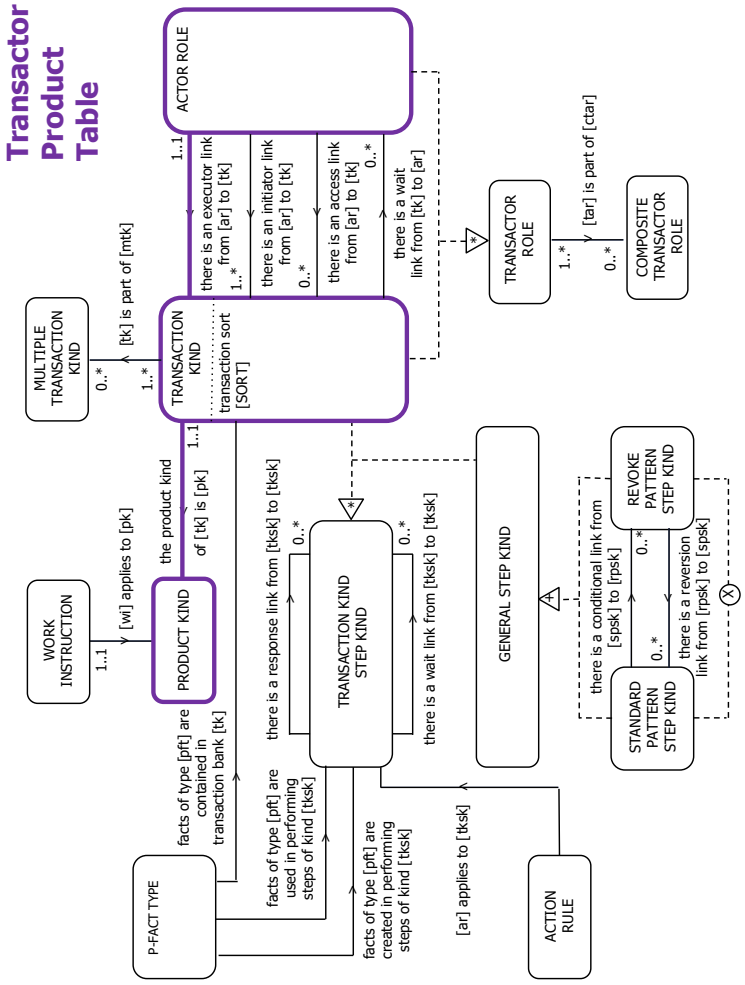


Fig. 21 Definition of the Transactor Product Table (TPT)

The purple coloured and bold-lined parts above collectively define the (semantic) contents of a *Transactor Product Table* (TPT). Thus, every TPT represents the existence, in the chosen SoI, of a number of transaction kinds, actor roles, and product kinds. In addition, it expresses which actor role is the executor role of a transaction kind, and which product kind is associated with the transaction kind. Table 2 exhibits an example of a TPT.

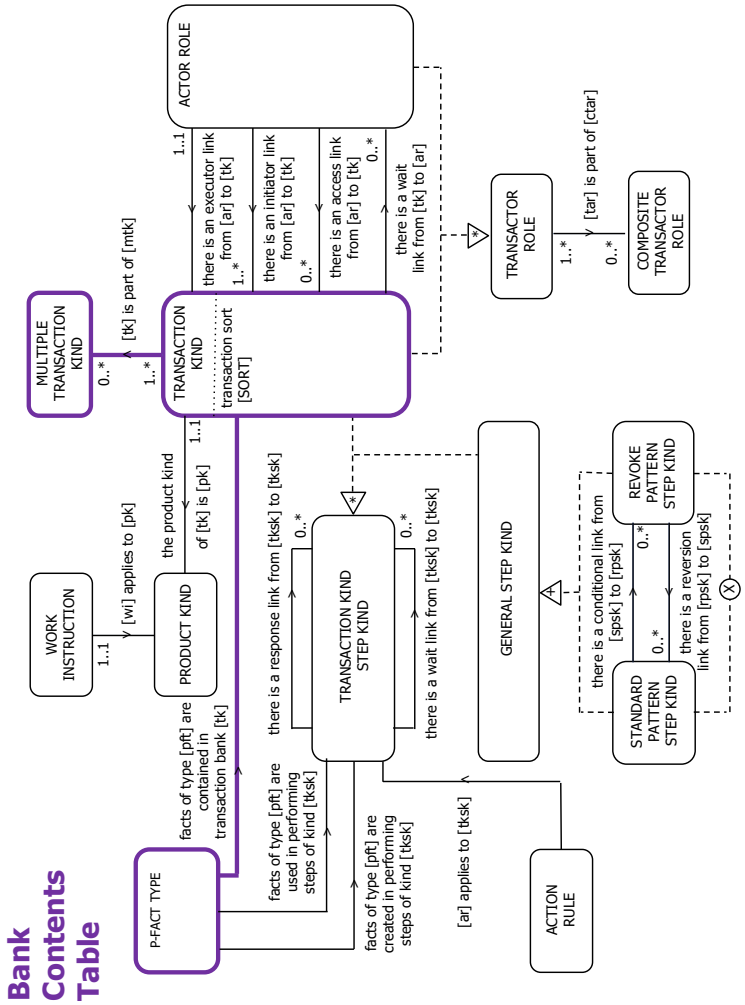


Fig. 22 Definition of the Bank Contents Table (BCT)

The purple coloured and bold-lined parts above collectively define the (semantic) contents of a *Bank Contents Table* (BCT). Thus, every BCT represents the existence, in the chosen SoI, of a number of transaction kinds, multiple transaction kinds, and P-fact types. In addition, it represents for every P-fact type in which transaction kind (now interpreted as a transaction bank) instances of it are contained.

The instances of the property type 'is part of' (between transaction kind and multiple transaction kind) may be implicitly given. It is important yet to understand that a multiple transaction kind is a collection of transaction kinds. Table 3 contains an example of a BCT.

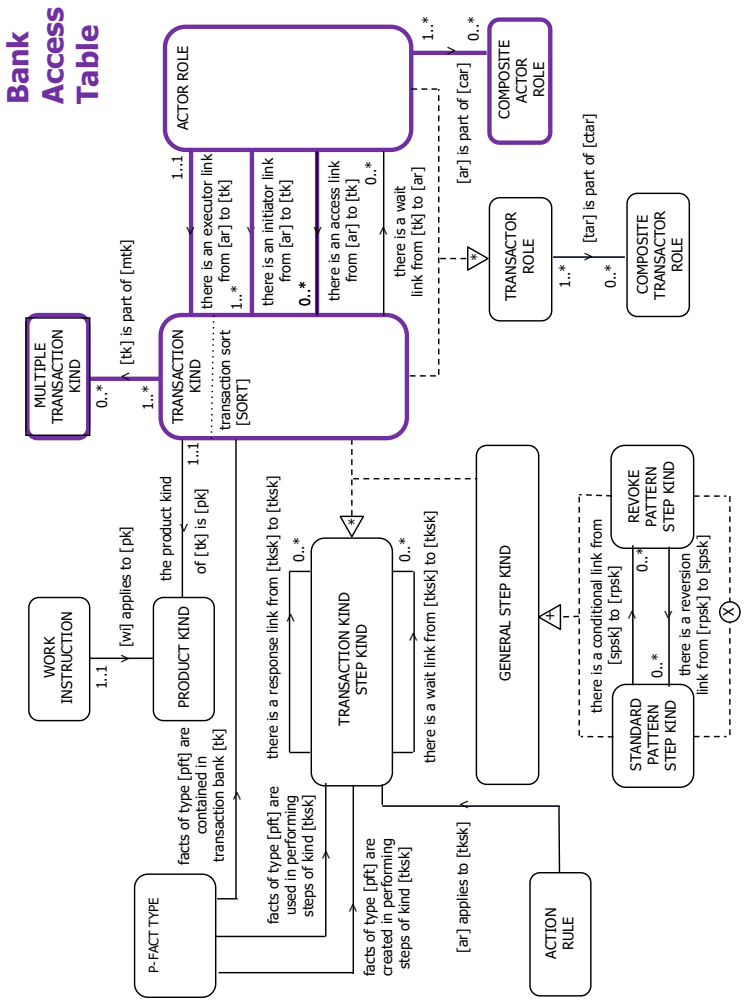


Fig. 23 Definition of the Bank Access Table (BAT

The purple coloured and bold-lined parts above collectively define the (semantic) contents of a *Bank Access Table* (BAT). Thus, a BAT represents the existence, in the chosen SoI, of transaction kinds, multiple transaction kinds, and actor roles, as well as of access links from actor roles to transaction banks, including the executor role and the initiator roles. To clarify this, the entity type 'composite actor role' is added to the schema (which can be fully deduced from the composite transactor role).

A BAT represents the interstriction structure of an SoI, as an alternative to drawing access links in the CSD. Access links may also exist between actor roles and multiple transaction kinds, and between composite actor roles and (multiple) transaction kinds. An example of a BAT is presented in Table 4.

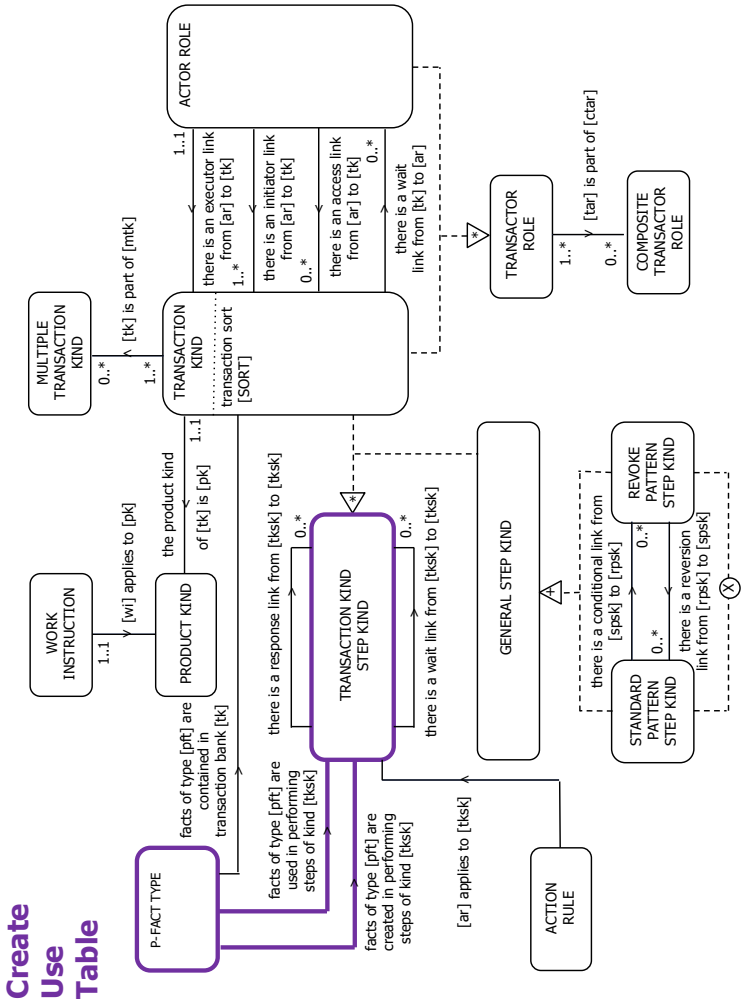


Fig. 24 Definition of the Create Use Table (CUT)

The purple coloured and bold-lined parts above collectively define the (semantic) contents of a *Create Use Table* (CUT). Thus, every CUT represents the existence, in the chosen SoI, of a number of transaction kind step kinds and P-fact types. In addition, it expresses for every P-fact type, in which transaction kind step kind its instances are created and in which transaction kind step kind its instances are used. Table 5 contains an example of a CUT.

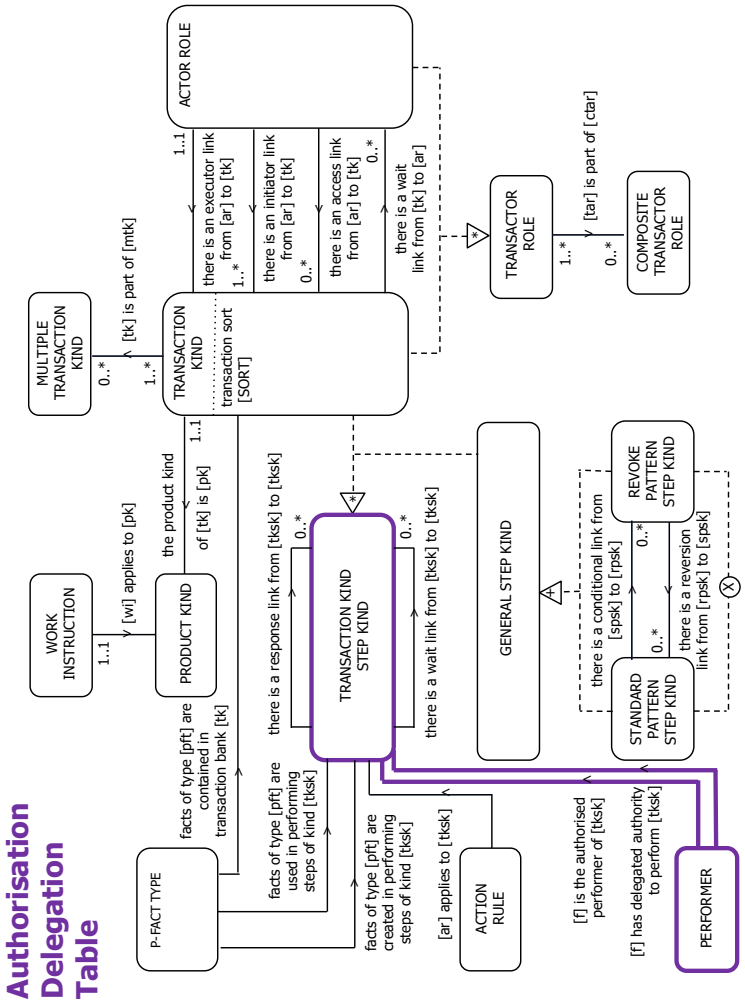


Fig. 25 Definition of the Authorisation Delegation Table (ADT)

The purple coloured and bold-lined parts above collectively define the (semantic) contents of the *Authorisation Delegation Table* (ADT). Note that the object class PERFORMER is added to the schema, to make the definition possible.

The columns of an ADT represent tasks (T), ranging from single process steps (in a detailed ADT) to the responsibility ranges of actor roles (in a global ADT). The rows represent the task performers (P), ranging from functionaries to complete enterprises. An "A" at the crossing of a column and a row indicates that the performer is authorised to perform the task, a "D" that he/she has delegated authority. In a global ADT, only A's can occur since it is by definition not possible to delegate a complete actor role (cf. PSI theory in [1]). Table 5 contains an example of a detailed ADT.

References

1. Dietz, J.L.G., H.B.F. Mulder, and SpringerLink (Online service), Enterprise Ontology A Human-Centric Approach to Understanding the Essence of Organisation. 2020, Springer International Publishing : Imprint: Springer: Cham.
2. Sowa, J.F., Knowledge representation : logical, philosophical, and computational foundations. 2000, Pacific Grove: Brooks/Cole. xiv, 594 p.
3. Dietz, J.L.G., *On the nature of business rules*, in *Advances in Enterprise Engineering I*. 2009, Springer: Berlin-Heidelberg.

End notes

[1] DEMO is extensively discussed in Dietz, J.L.G., Mulder, J.B.F.: *Enterprise Ontology - a human-centric approach to understanding the essence of organisation*, Springer, 2020, ISBN 978-3-030-38853-9

[2] https://en.wikipedia.org/wiki/Extended_Backus–Naur_form

[3] https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[4] https://en.wikipedia.org/wiki/Peano–Russell_notation

[5] https://aa.usno.navy.mil/data/docs/JulianDate.php

[6] “restriction” originates from the Latin verb “stringere”, meaning trimming, curtailing. The word “interstriction” expresses that actors restrict each others decision freedom or ‘play area’.