Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Formalization in Method Engineering

Session 8

12 March 2019

Dr. Sietse Overbeek

Outline

- Semiotic levels
- Predicate Calculus
- Predicate Calculus for Meta-data modeling
  - Use cases
  - Petri-nets
- Motivation and discussion

Semiotic levels

In Information and Computer Sciences many special purpose languages (programming languages, scripting languages) or diagrammatic specification formalisms are used for describing data, process, architecture, software, system, etc.

In documentation and communication processes all kind of information is being transferred in terms of signs in a language. Signs themselves can be considered in terms of four inter-dependent levels of semiotics: empirics, syntax, semantics, and pragmatics. (Beynon-Davies, 2009).

1. **Empirics** is the study of the signals used to carry a message; the physical characteristics of the medium of communication, e.g., sound, light, electronic transmission, paper and ink, etc..

2. **Syntax** is concerned with the formalism used to represent a message.

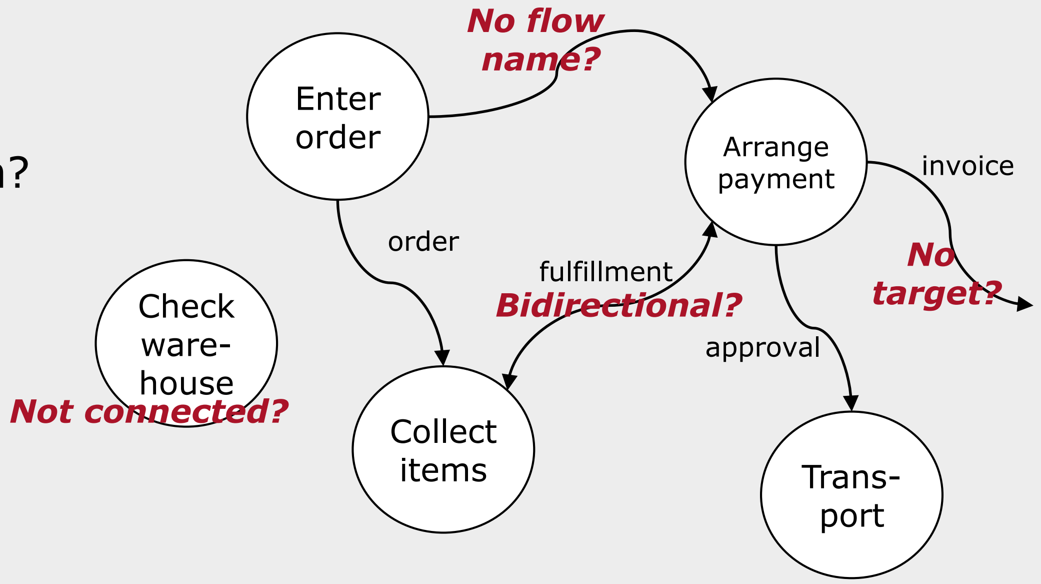
3. **Semantics** is concerned with the meaning of a message. Semantics considers the content of communication.

4. **Pragmatics** is concerned with the purpose of communication. Pragmatics links the issue of signs with the context within which signs are used.

See e.g. Beynon-Davies P. (2009). Business Information Systems. Palgrave, Basingstoke.

Communication: Example diagram

What is a correct diagram?



- Empirics: Are the circles properly displayed on the screen?
- Syntax: Is it according to the standards for this function modeling language?
- Semantics: Does it represent the reality of the IS?
- Pragmatics: Is it according to the purpose of the project?

Formalization of meta-models

Types of modeling techniques

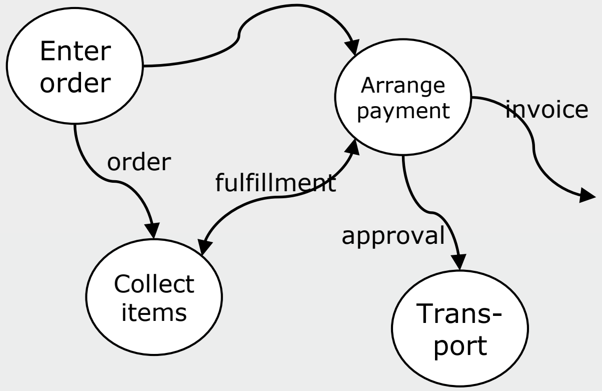
1. **Formal techniques**: techniques of which the syntax and the semantics are rigorously defined.
   Examples: predicate calculus and formal specification languages, like VDM [Bjorner 78], Z [Spivey 88], and OCL [UML 02].

2. **Structured techniques**: techniques of which the syntax is formally defined. Diagrammatic techniques for information systems modeling with precise rules that define which constructs are allowed and which are not.
   Examples are dataflow diagramming and data-modeling, use case diagramming, etc.

3. **Informal techniques**: techniques for which there is no complete set of rules to constrain the models created by the technique. Natural language and unstructured pictures are informal techniques.

Semantic correctness

- The semantic valuation function of structured techniques can in general not be made explicit, because the domain of values is the real world, of which we assume that it cannot be formalized. The meaning of the constructs in these techniques is therefore in general based on a common understanding communicated by means of examples.
- Semantics of a domain can be expressed in a so-called ontology, i.e. standardized vocabularies
- An ontology represents knowledge as a set of concepts within a domain, and the relationships between those concepts



Outline

- Semiotic levels
- Predicate Calculus
- Predicate Calculus for Meta-data modeling
  - Use cases
  - Petri-nets
- Motivation and discussion

Predicate Calculus

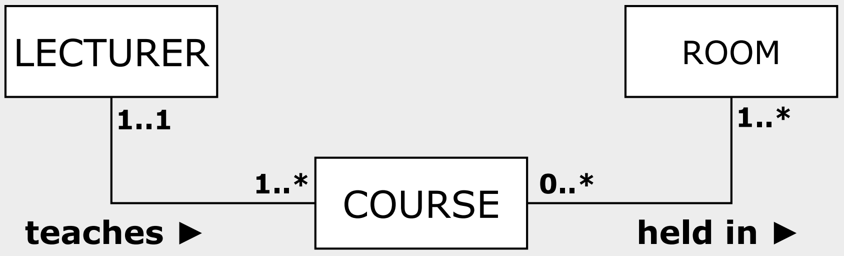
Predicate calculus is a mathematical formalism suitable for the syntactic structures of (meta-)data modeling

A set of formulas, called an algebraic structure, is defined consisting of:

1. Sets, that represent the concepts of the modeled technique
2. Predicates, that represent the associations between the concepts
3. Axioms formulate the rules of the technique.

Predicate calculus that we apply is also called *first order predicate calculus*

Example in M1-M0 level: Sets



The Sets, Predicates and Axioms need to be defined:

Three sets:

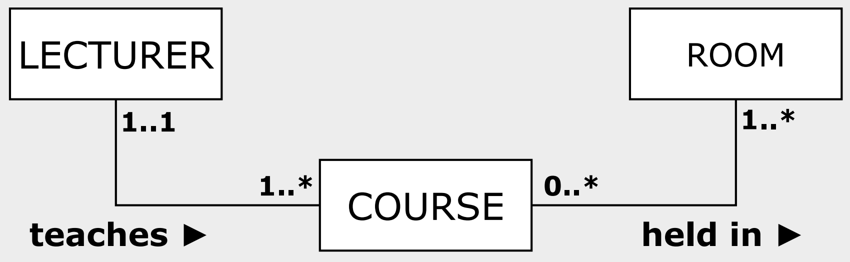
L : set of Lecturers
    L = {`Knuth, Dijkstra, Turing`}

C : set of Courses
    C = {`Programming, Architecture, Requirements, Logic`}

R : set of Rooms
    R = {`BBL079, BBL112, MIN211, MIN081, GEO001`}

Note: the *abstractions* L, C and R on the M1 level have *instances* at the M0 level listed as the set elements. (c.f. lecture 1 and 2)

Example in M1-M0 level: Predicates



Predicates are formulated over sets to designate associations.

Two predicates:
1. a lecturer teaches a course
2. a course is held in a room

predicate `teaches` **over** L x C where x denotes Cartesian product of the sets (next slide)

predicate `held` **over** C x R

In this notation `teaches(Dijkstra,programming)` means that the lecturer *Dijkstra* is teaching the course *programming, a*nd similarly for `held(programming,MIN211)`.

We use Courier font for the predicates for readability of these slides.

Example in M1-M0 level: Domain

Set L = {K,D,T}, C = {P,A,R,L}, R = {BBL079, BBL112, MIN211, MIN081, GEO001}

This M0 domain can be visualized in a table of the Cartesian product:

L x C

Course / Lecturer	P	A	R	L
K				
D				
T				

R x C

Course / Room	P	A	R	L
BBL079				
BBL112				
MIN211				
MIN081				
GEO01				

Assume the following predicates are valid in M0

```
teaches(K,A), teaches(D,P), teaches(D,R), teaches(T,L)

held(A,BBL079), held(P,BBL079), held(P,BBL112), held(P,MIN211),
held(R,MIN211), held(L,MIN211), held(L,GEO001)
```

teaches:

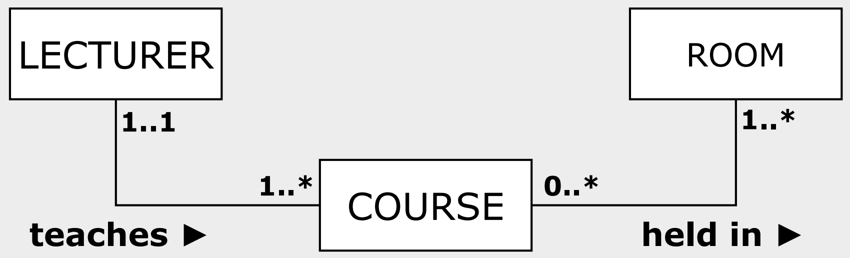
Course / Lecturer	P	A	R	L
K		√		
D	√		√	
T				√

held:

Course / Room	P	A	R	L
BBL079	√	√		
BBL112	√			
MIN211	√		√	√
MIN081				
GEO01				√

Note: check the multiplicities of the data-model!

Example in M1-M0 level: Axioms



Rules can now be formulated in terms of the predicates and the logical operators, called axioms.

An example is the axiom that each lecturer teaches at least one course:

U1: $\forall l \in L\ \exists c \in C$: teaches(l,c)

Similarly, a course is taught by at most one lecturer

U2: $\forall c \in C\ \forall l1, l2 \in L$: [teaches(L1,c) **and** teaches(L2,c) $\Rightarrow$ L1=L2]

Where
- $\forall$: for all
- $\exists$: there is
- $\in$: element of

So $\forall l \in L\ \exists c \in C$: For all lecturers there is a course

Predicate Calculus for meta-data models

Meta-data models capture static aspects of a method.

The meta-models consist of:

1. Concepts: mapped upon Sets
2. Associations between those concepts: mapped upon Predicates
3. Rules that must hold for these concepts and associations: mapped upon Axioms

This formalization is then performed at the M2-M1 level.

Outline

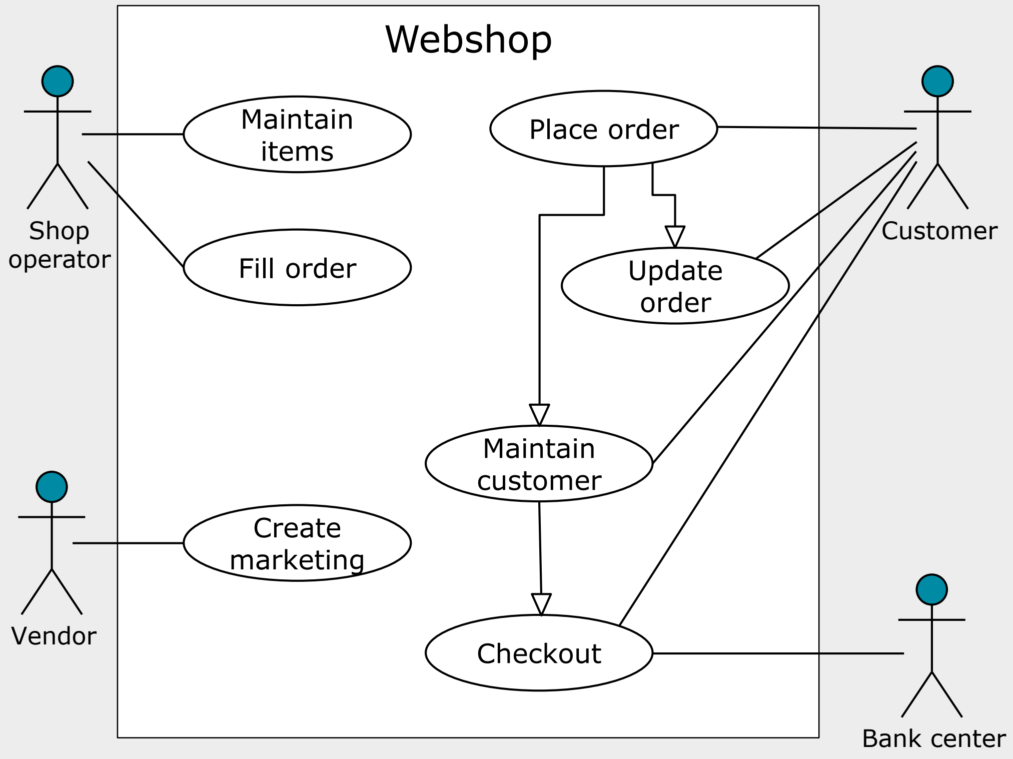
Example technique: Use Case

Use cases

- Actors (puppets)
- Use case (in ovals)
- Communications (lines)
- Relationship (arrows)
- System boundary (box)

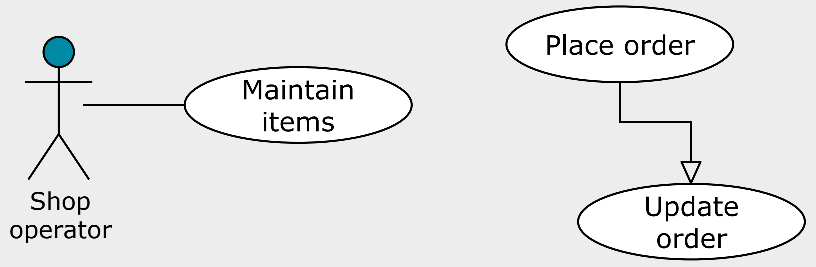
Note:

We simplify and ignore the system boundary and the <<include>> and <<extend>> relationships

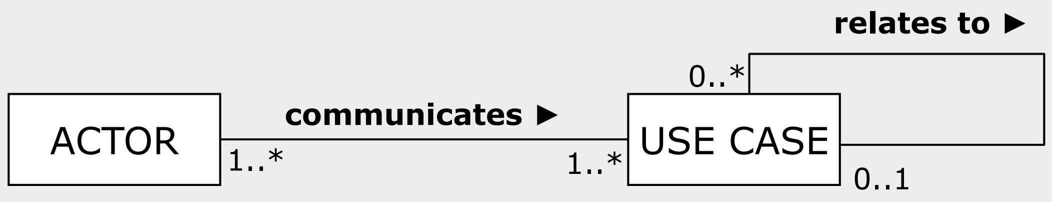


M2-M1 level: Use Case meta-model

Concepts of Use Cases
1. Actors
2. Use case



Associations of Use Cases
1. communicates
2. relates to



Sets

Two sets:

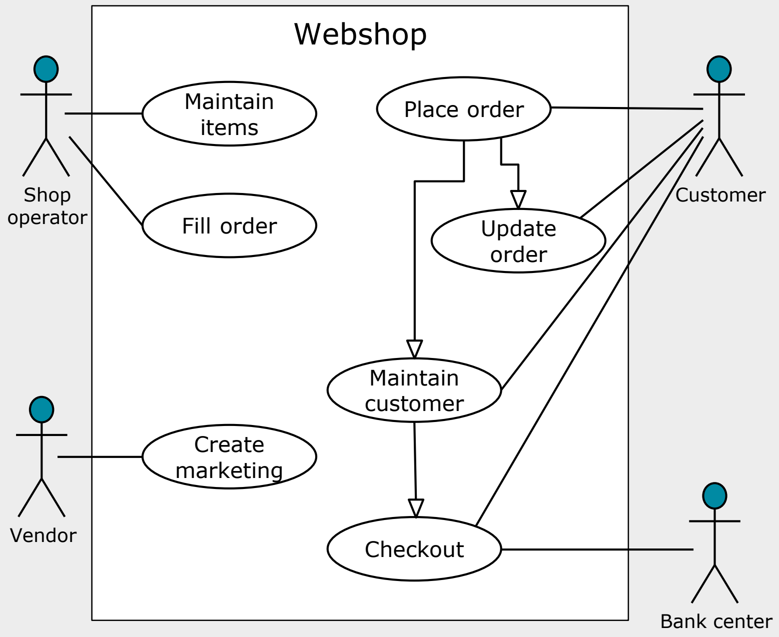
A : set of Actors

A = {`ShopOperator, Vendor, Customer, BankCenter`}

U : set of Use cases

U = {`PlaceOrder, UpdateOrder, FillOrder, MaintainItems, MaintainCustomer, CreateMarketing, Checkout`}

Note: the *abstractions* A and U on the M2 level have *instances* at the M1 level listed as the set elements.



Universiteit Utrecht

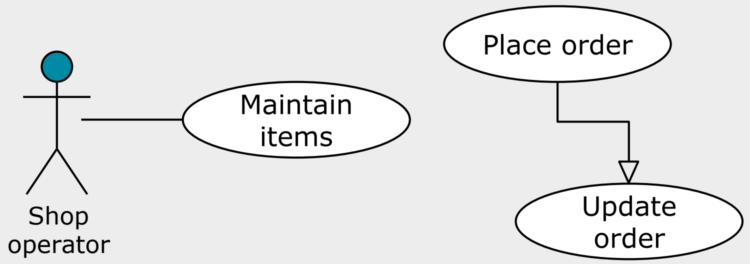
[Faculty of Science
Information and Computing Sciences]

Predicates

Predicates are formulated over sets to designate associations.

Two associations:

1. An Actor communicates in a Use case
2. A Use case relates to a Use Case



predicate `communicate` **over** A x U

predicate `relate` **over** U x U

In this notation `communicate`*(a,u)* means that the actor *a* communicates with the use case *u*, and similarly for `relate`*(u1,u2)* means that use case u1 is related to use case u2*.*

From the example:

`communicate`(ShopOperator,MaintainItems)

`relate`(PlaceOrder,UpdateOrder)

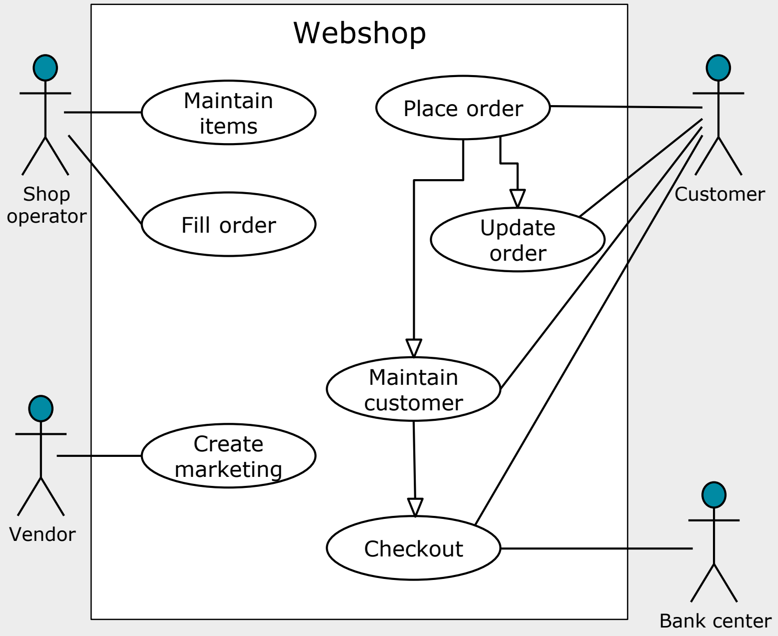
As the total number of communications is *8* and of relationships is *3* , so there are precisely *11* predicates valid. See next slide.

The basic predicates can not be defined in terms of other predicates, they are just assumed to be valid for certain values of the parameters.

Predicates listing

11 Predicates in M1:

```
communicate(ShopOperator,MaintainItems)
communicate(ShopOperator,FillOrder)
communicate(Vendor,CreateMarketing)
communicate(Customer,PlaceOrder)
communicate(Customer,UpdateOrder)
communicate(Customer,MaintainCustomer)
communicate(Customer,Checkout)
communicate(BankCenter,Checkout)
relate(PlaceOrder,UpdateOrder)
relate(PlaceOrder,MaintainCustomer)
relate(MaintainCustomer,Checkout)
```



Axioms for Use cases

Rules of the use case modeling technique can be formulated in axioms.

An example is the axiom that each Actor is associated to a Use case:

P1: $\forall a \in A\ \exists u \in U : \texttt{communicate}(a,u)$

Consequence: in a correct diagram there are *no* actors without a use case.

What is the formalization of P2 stating that all use cases are communicated by actors?

Auxiliary predicates

For simplicity of reasoning auxiliary predicates are introduced.

Formally we can do without them, but it makes the algebraic structure simpler.

For example:

predicate `basecase` **over** U as

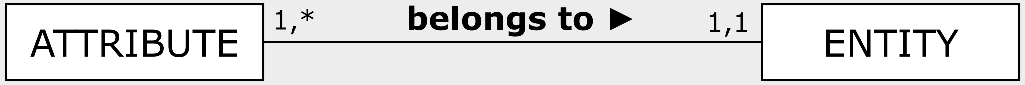
`basecase(u)` ≡ **not** ∃u′ ∈ U `relate`(u′,u)

where ≡ means: is defined as

I.e. a use case u is a base case if there are no related (extend, include) use cases.

So `MaintainItems` and `FillOrder` are base cases.
What are the others?

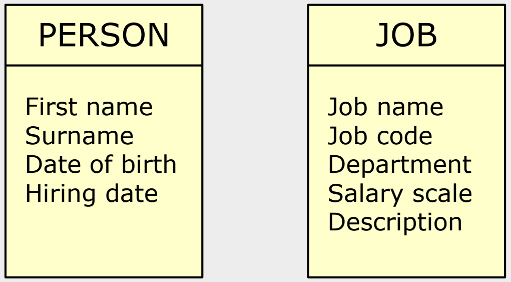
Example on Attributes and Entities



Algebraic structure

A: set of Attributes

E: set of Entities

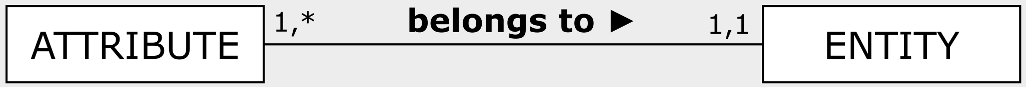


Predicate `belongs` **over** A x E

`belongs`(a,e) means attribute *a* belongs to entity *e*

```
belongs(Date_of_birth,PERSON)
belongs(Salary_scale,JOB)
```

Axioms for multiplicity



All entities have at least one attribute

D1: $\forall e \in E\ \exists a \in A : \texttt{belongs}(a,e)$

All attributes belong to an entity

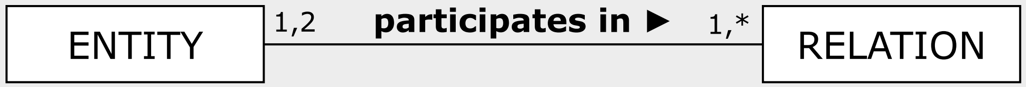
D2: $\forall a \in A\ \exists e \in E : \texttt{belongs}(a,e)$

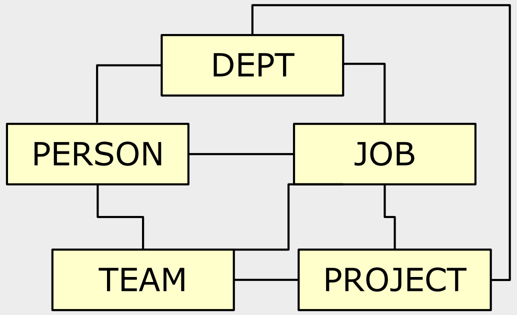
An attribute belongs just to one entity

D3: $\forall a \in A\ \forall e1,e2 \in E : [\texttt{belongs}(a,e1)\ \textbf{and}\ \texttt{belongs}(a,e2) \Rightarrow e1=e2]$

Note: identify D1, D2, and D3 in the above meta-model yourself

Axioms for multiplicity





E: set of Entities
R: set of Relations

Predicate `participate` **over** E x R

All entities participate in at least one relation

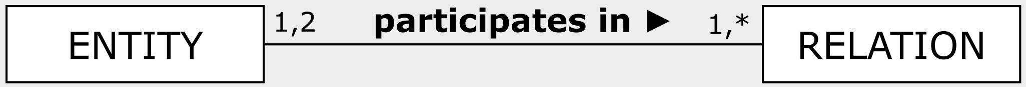
D4: $\forall e \in E\ \exists r \in R$: `participate`(e,r)

A relation has at most two entities

D5: $\forall r \in R\ \forall e1,e2,e3 \in E$: [`participate`(e1,r) **and** `participate`(e2,r) **and** `participate` (e3,r) $\Rightarrow$ e1=e2 **or** e2=e3 **or** e1=e3]

Formulate D6 yourself.

Auxiliary axiom for path



An Entity-Relationship diagram is always connected, i.e. it may not consist of two unconnected parts.

D7: ∀e1,e2∈E ∃r1,r2,…,rn∈R : [participate(e1,r1) **and** … **and** participate(e2,rn)]

However: formulations with … are not allowed in Predicate Calculus.
We solve this with an auxiliary predicate, called path.

predicate path **over** E x E **as**
path(e1,e2) ≡ ∃r∈R : [participate(e1,r) **and** participate(e2,r)] **or**
[∃ e3∈E : participate(e1,r) **and** participate(e3,r) **and** path(e2,e3)]

Now the rule D7 becomes:

D7: ∀e1,e2∈E : path(e1,e2)

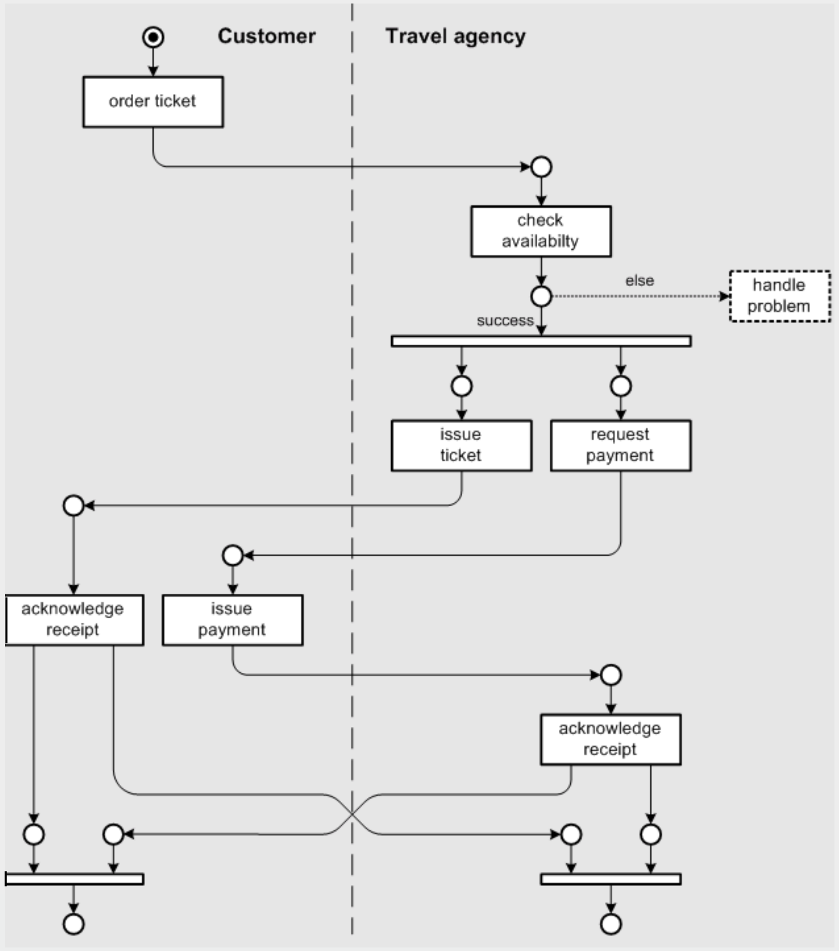
Outline

- Semiotic levels
- Predicate Calculus
- Predicate Calculus for Meta-data modeling
  - Use cases
  - Petri-nets
- Motivation and discussion

Example technique: Petri-nets

Petri-nets

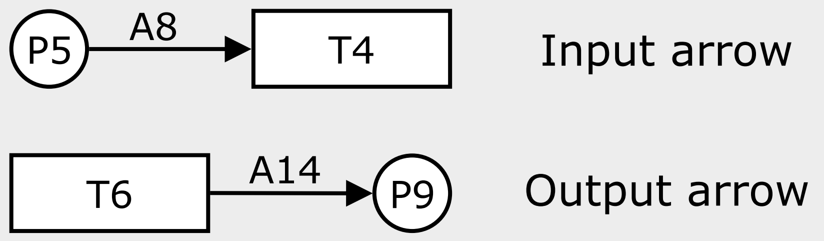
- Places (in circles)
- Transitions (in rectangles)
- Arrows

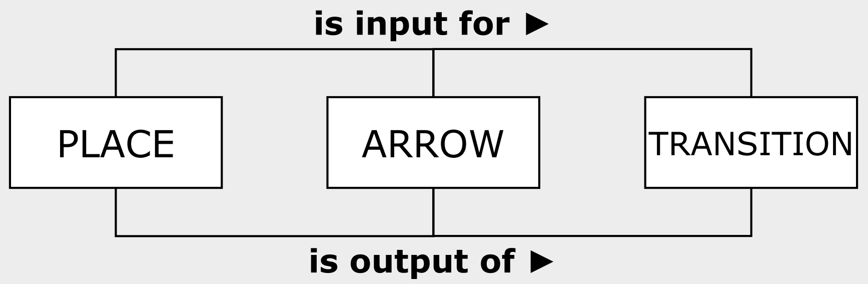


Meta-model

Petri-nets

- Place
- Transitions
- Arrows





Note: one of the rare occasions of a ternary association

Sets

Three sets:

S : set of places

S = {`start, P1, P2, P3,…, P11,end1, end2`}

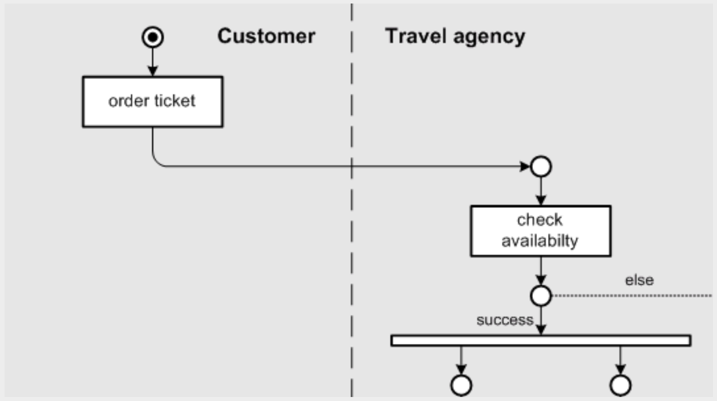
T : set of transitions

T = {`order ticket, check availability, handle problem, T1, issue ticket, request payment, acknowledge receipt1, issue payment, acknowledge receipt2, T2,T3`}

A : set of arrows

A = {`A1, A2, …, success, exit, …, A24`}

Input and Output Predicates



Predicates are formulated over sets to designate associations.

Two associations:
1. a place is input for a transition
2. a place is output for a transition

predicate `input` **over** S x A x T

predicate `output` **over** S x A x T

In this notation `input`*(s,a,t)* means that the place *s* is input to the transition *t* via the arrow *a*, and similarly for `output`*(s,a,t).*

From the example diagram:

```
input(P2,success,T4)
output(P1,A2,order ticket)
```

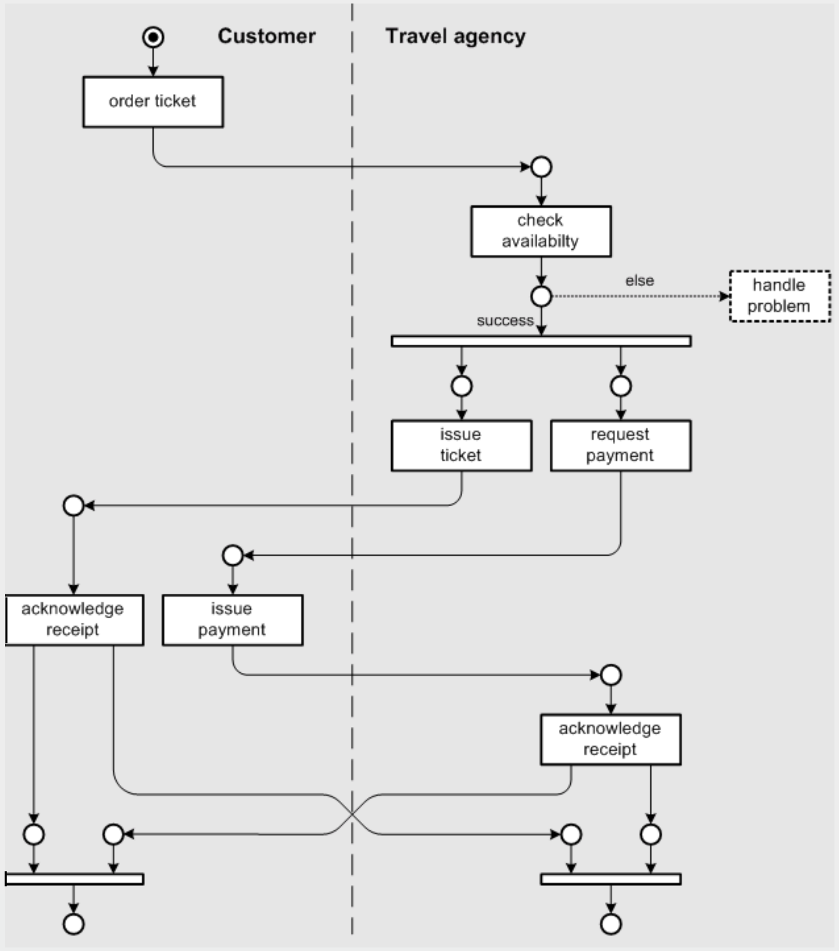
As the total number of arrows is *26*, there are precisely *26* predicates valid.

The basic predicates can not be defined in terms of other predicates, they are just assumed to be valid for certain values of the parameters.

Predicates listing

26 Predicates:

```
input(start,A1,order_ticket)
output(P1,A2, order_ticket)
input(P1,A3,check_availability)
output(P2,A4,check_availability)
.
.
.
output(end2, A24,T3)
```



Axioms

Rules for the Petri-net can be formulated in terms of the predicates and the logical operators.

An example is the axiom that each arrow connects a place to a transition as either input or output:

P1: $\forall a \in A\ \exists s \in S\ \exists t \in T\ [\texttt{input}(s,a,t)\ \textbf{or}\ \texttt{output}(s,a,t)]$

Consequence: in a correct diagram there are *no* dangling arrows.

Corollary for source

Define first a source as a place with arrows that are input for a transition

predicate `source` **over** S as

`source`(s) ≡ **not** $\exists a \in A\ \exists t \in T$ [`output`(s,a,t)]

From the definition of source and rule (P1) it follows that:

$\forall s \in S$ source(s) $\Rightarrow \exists a \in A\ \exists t \in T$: input(s,a,t)   (P2)

If a place is a source then an arrow points from that place to a transition

Outline

- Semiotic levels
- Predicate Calculus
- Predicate Calculus for Meta-data modeling
  - Use cases
  - Petri-nets
- Motivation and discussion

Motivation of formalization

- Formalization establishes a precise meaning of properties of techniques
- Rules in natural language always have room for misinterpretation
- Formalization only done for modeling techniques

- Rules determine a syntactically correct diagram
- Tools can check the correctness of a diagram
  - Checking during creation
  - Checking after creation

- Scientific work is greatly improved by formalization

Conclusions

- Formalization is required for proper scientific communication
- Formalization of rules can be pretty complex
- Before formalization starts a good scoping of the context and the collection of carefully described rules is to be performed