Constraints between modeling perspectives

Manfred Jeusfeld
University of Skövde, Sweden
http://www.his.se

Resources

http://conceptbase.cc
Home page of the ConceptBase metamodeling system used in this tutorial

http://conceptbase.sourceforge.net/import-cb-appliance.html
virtual appliance including an executable ConceptBase installation

http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/2616290
Slides of a method engineering course (**highly recommended as preparation**)

http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/3810007
Supplementary material for this tutorial; use permitted under a CC license (see source files or README files in the subfolders)

http://link.springer.com/chapter/10.1007%2F978-3-319-39417-6_2

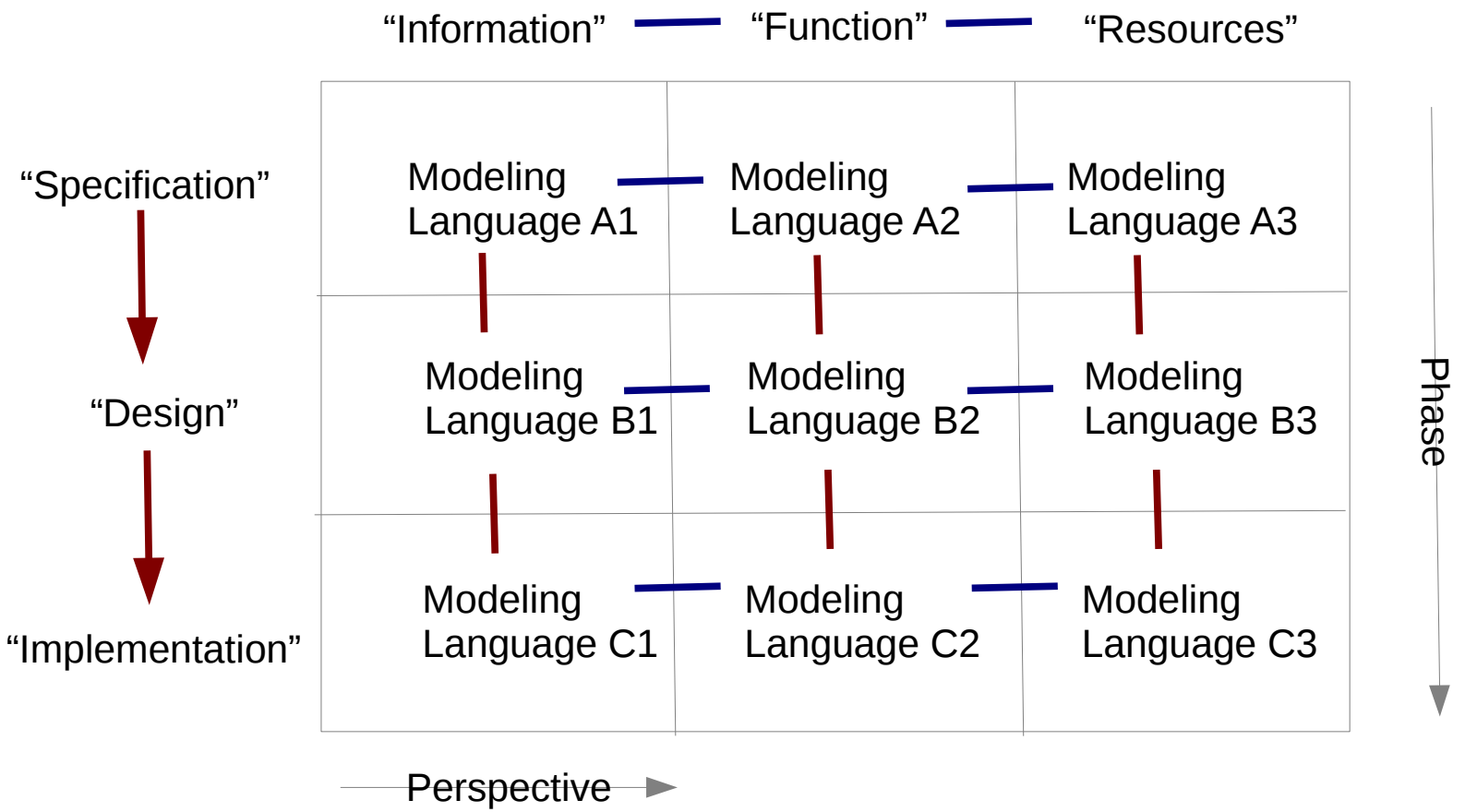
Manfred A. Jeusfeld: SemCheck - Checking Constraints for Multi-perspective Modeling Languages. In Dimitris Karagiannis, Heinrich C. Mayr, John Mylopoulos (eds.): Domain-Specific Conceptual Modeling - Concepts, Methods and Tools. Springer, 2016, pp. 31-53

http://book.omilab.org/psm/content/semcheck/info
SemCheck page at OMiLAB

Please check out the supplementary material provided above.
You are also advised to download and install ConceptBase from the homepage above. It installs on Windows, Mac OS, and Linux. You need Java 7 or higher.

Enterprise modeling frameworks

- Organize multiple modeling languages into a two-dimensional grid
- There are cross-notational links between perspectives (balance!) and development phases (map!)
- more than two dimensions are also possible, e.g. include time/versions



The Zachman framework

	Why	How	What	Who	Where	When
Contextual	Goal List	Process List	Material List	Organizational Unit & Role List	Geographical Locations List	Event List
Conceptual	Goal Relationship	Process Model	Entity Relationship Model	Organizational Unit & Role Rel. Model	Locations Model	Event Model
Logical	Rules Diagram	Process Diagram	Data Model Diagram	Role relationship Diagram	Locations Diagram	Event Diagram
Physical	Rules Specification	Process Function Specification	Data Entity Specification	Role Specification	Location Specification	Event Specification
Detailed	Rules Details	Process Details	Data Details	Role Details	Location details	Event Details

*) image freely licensed from Wikipedia:Zachman Framework

Enterprises are complex. As a consequence, enterprise models focus on certain levels (roles) and perspectives.

The partial models need to be synchronized since they make statements about the same reality (here: the enterprise).

Goals of this tutorial

... to introduce into **logic-based conceptual meta modeling**

... to show how different modeling languages can be integrated by **balancing rules**

... to demonstrate that variants of modeling languages can **share the same semantic core** (Petrinets, BPMN, STD)

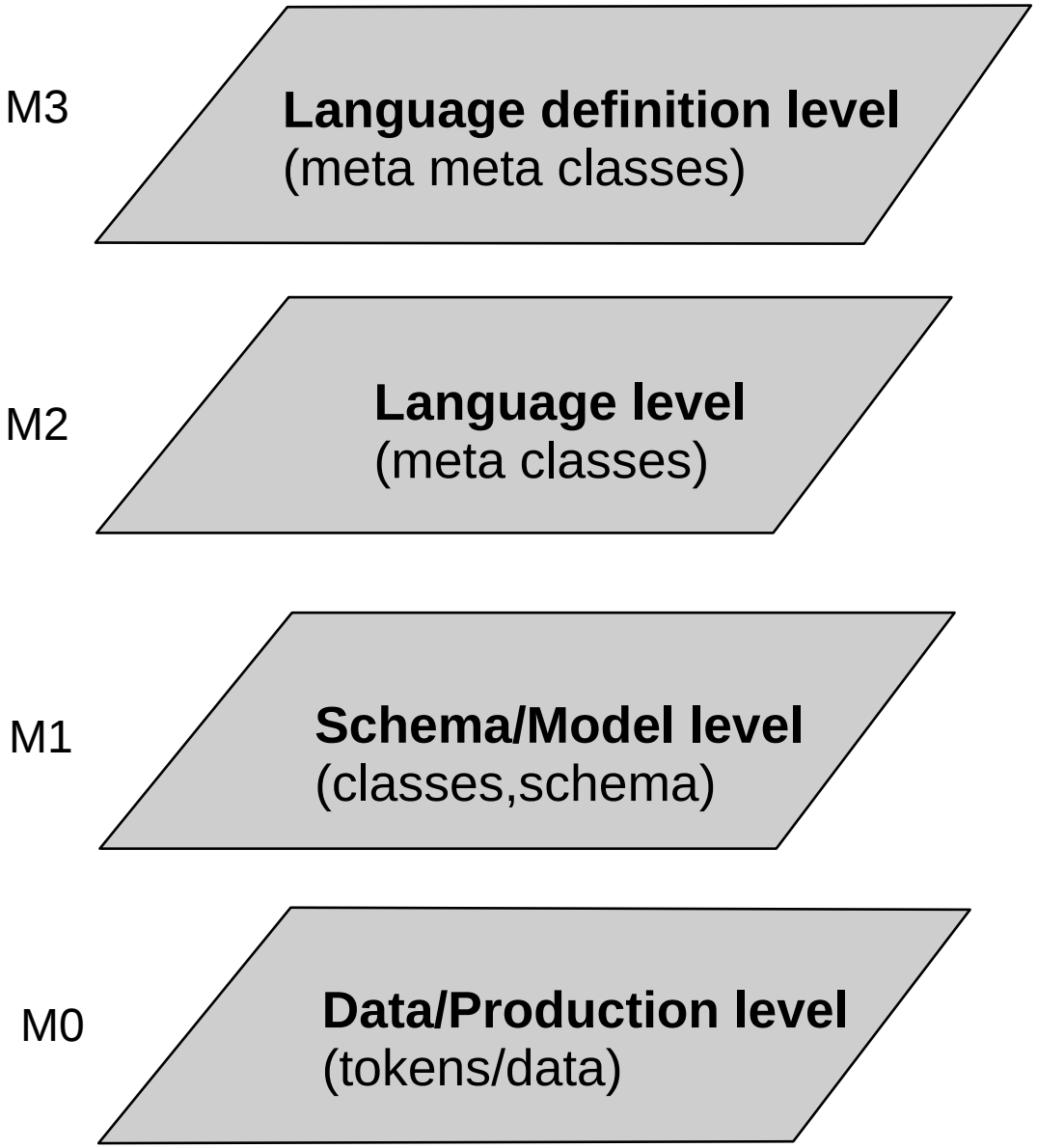
... and to listen to you!

Concrete and abstract statements

M0	Token (data) level: Bill earns 10000 Dollars.
M1	Class level: Employees have salaries.
M2	Meta class level: Entities are described by attributes.
M3	Meta meta class level: Information can be modeled by graphs.

The statement at the token level is an instance of the statement at the class level which is an instance of the statement at the meta class level and so on.
Goal: Map this to logic.

The Information Resource Dictionary System (ISO IRDS 1990)



Contains the facilities to define a notation (formal language for modeling/design/...)

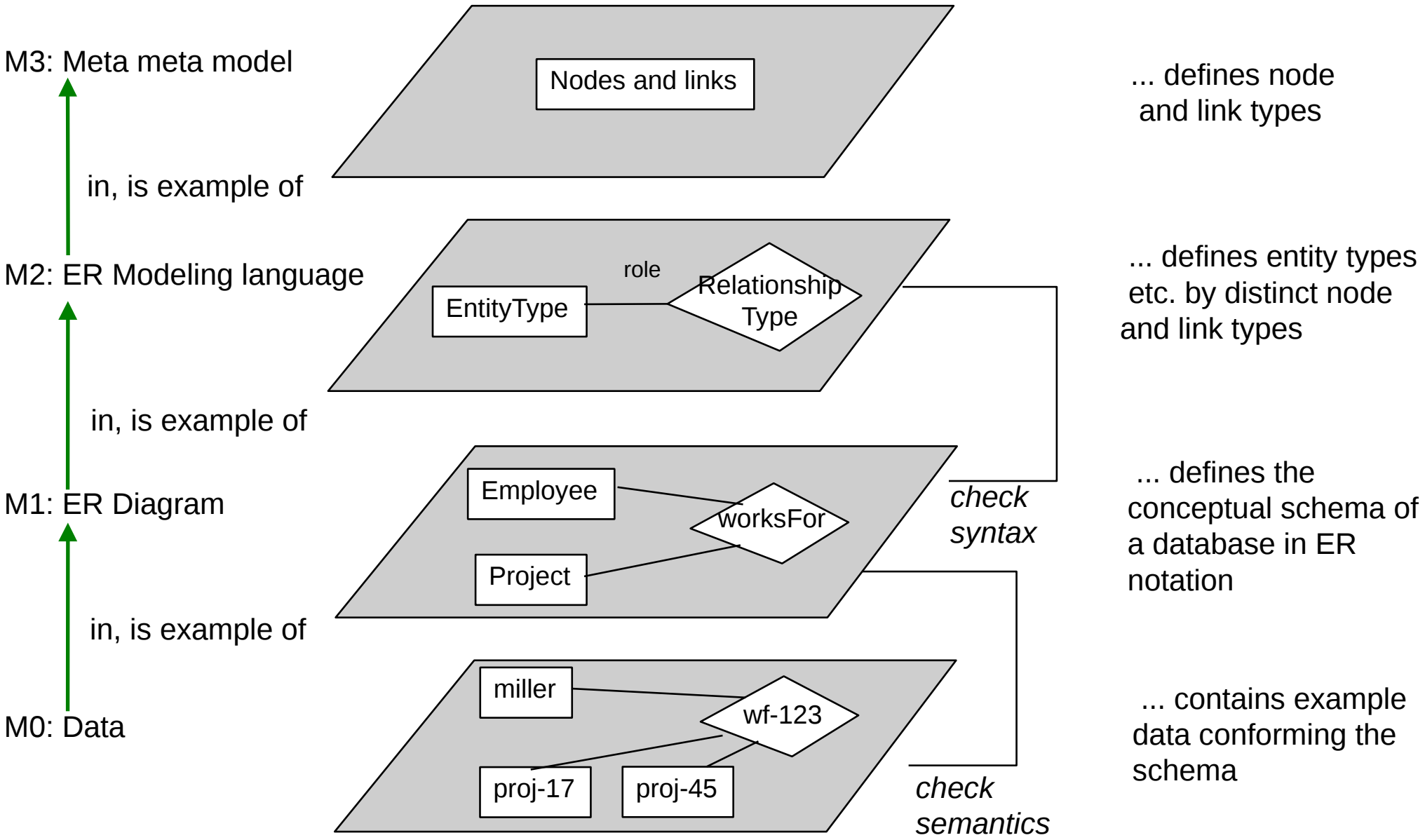
Contains the definition of notations using the facilities of the above level

Contains models/schemata/ programs written in a certain notation

Contains example data/process traces conforming the schemata/ models of the above level

N.B.: We sometimes use the term 'notation' as a synonym to 'language'.

The ERD Notation in IRDS levels



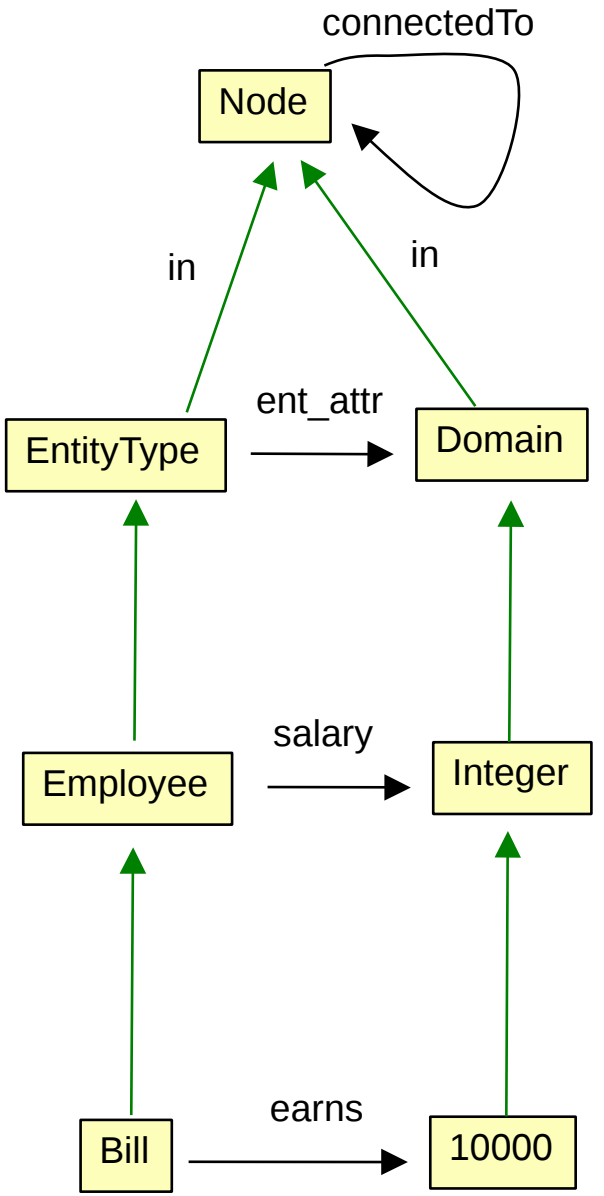
O-Telos: Graphical representation

```
(EntityType in Node)
(Domain in Node)
(EntityType connectedTo/ent_attr Domain)
```

```
(Employee in EntityType)
(Integer in Domain)
(Employee ent_attr/salary Integer)
```

```
(Bill in Employee)
(10000 in Integer)
(Bill salary/earns 10000)
```

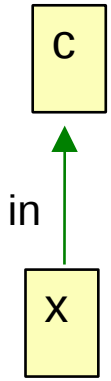
O-Telos facts



same information as graph

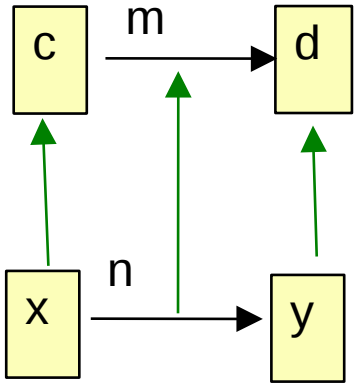
Instantiation and attribution in O-Telos

We need to represent data, schema, notation etc. uniformly as *objects*!




```
(x in c)
```

"the object x is an instance of the object c"

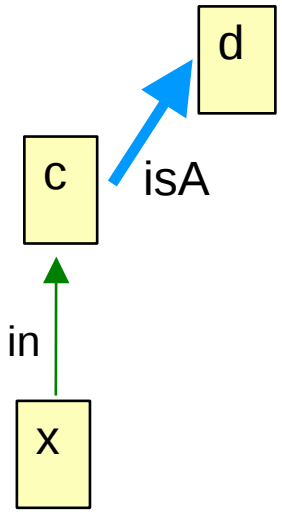



```
(x m/n y)

(x!n in c!m)
```

"the object x has a relation labeled n to y ;
this relation is an instance of the class-level relation m"

this denotes the link n of object x

Specialization in O-Telos

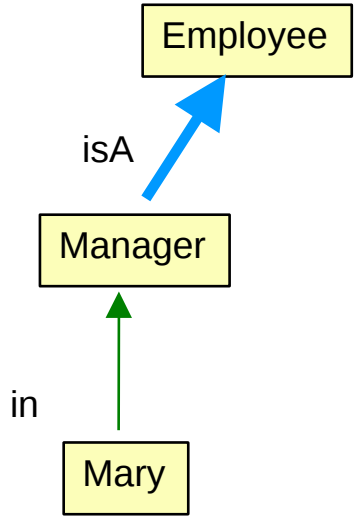



```
(c isA d)
```

"c is a subclass of d"

we want to achieve
that any instance
of c is automatically also an
instance of d *

Example:

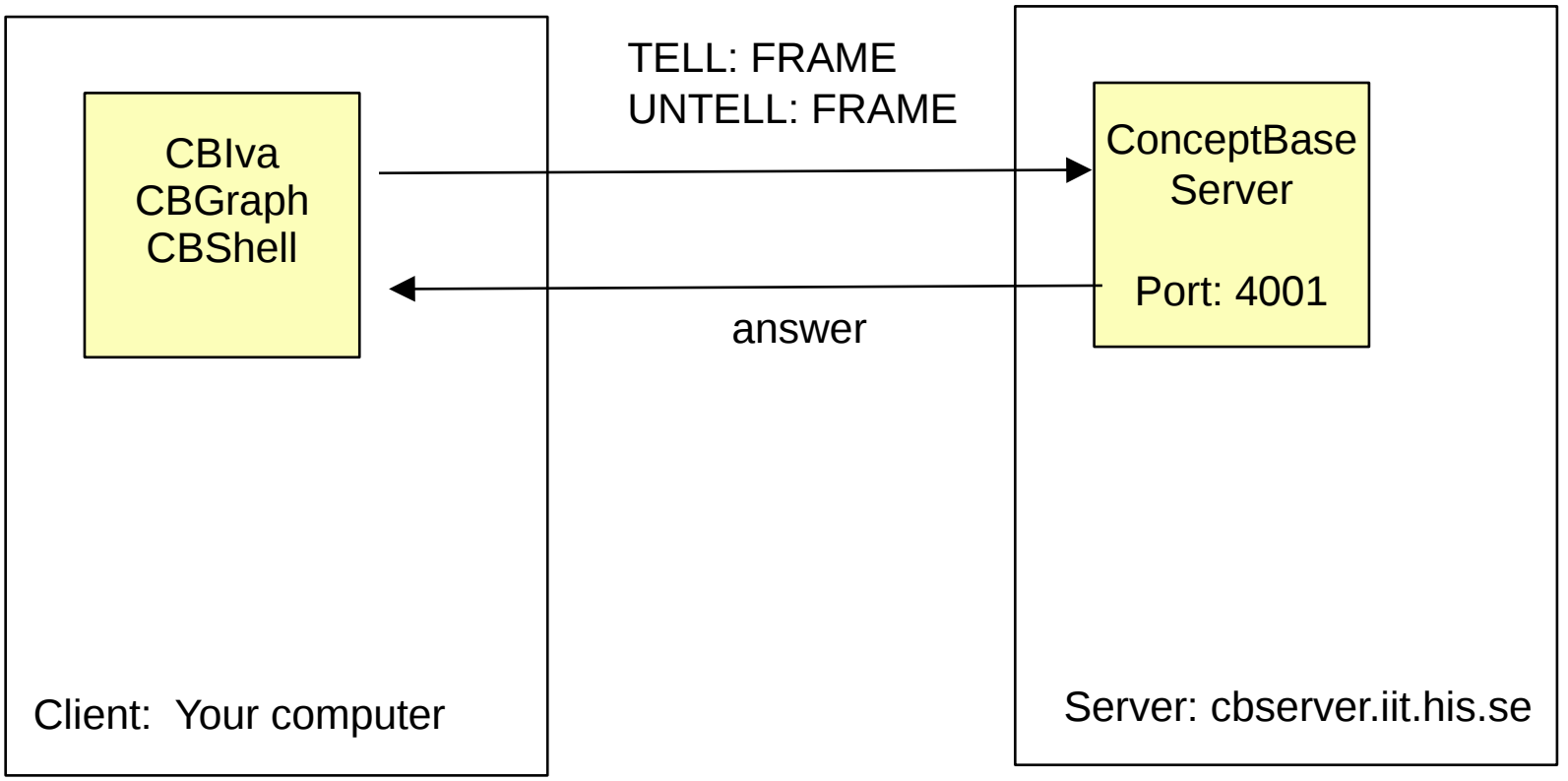



```
(Manager isA Employee)
(Mary in Manager)
```

*) This will be done by the pre-defined deductive rule

```
forall x,c,d (x in c) and (c isA d) ==> (x in d)
```

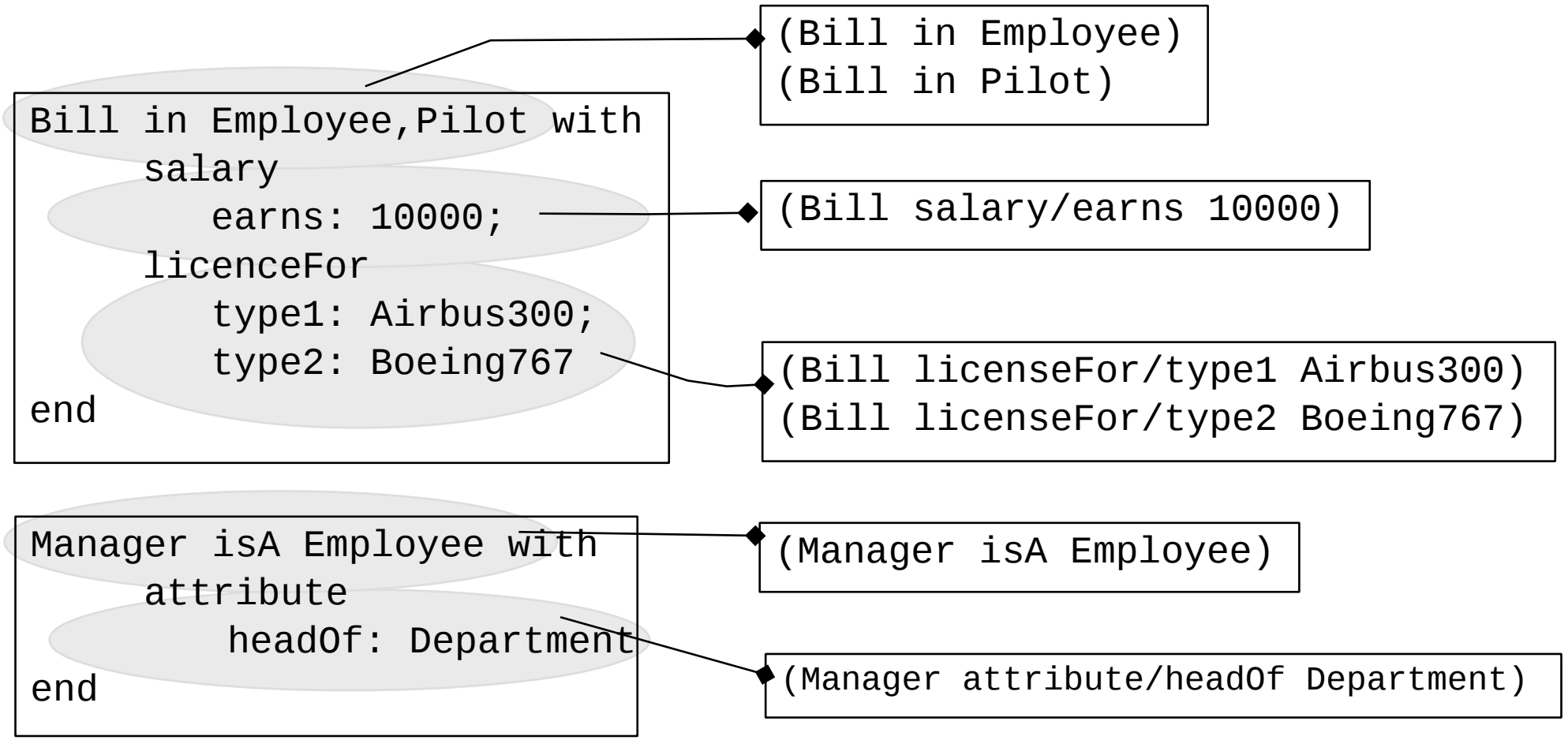
Principal interaction



You can also start the CBserver server on your local computer ("localhost") if the CBserver is supported for your platform

See also ConceptBase.cc User Manual

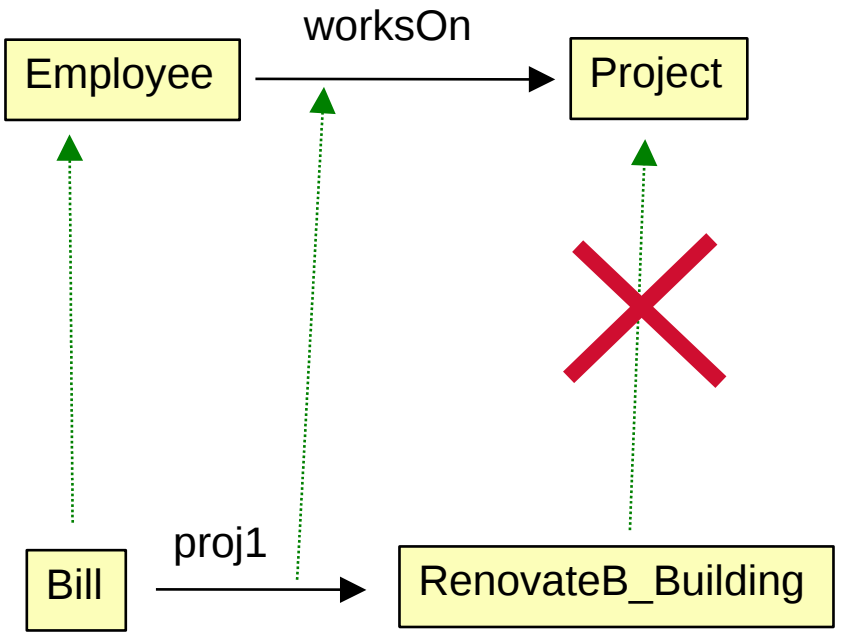
From frames to predicates



- The translation from frames to ground predicates and vice versa is straightforward

Violating the pre-defined constraint

```
Employee in Individual with
   attribute
      worksOn: Project
end

Bill in Employee with
   worksOn
       proj1: RenovateB_Building
end

RenovateB_Building in Individual
end
```



- Experience shows that ‘InstanceOf constraint 1’ is the most frequently violated constraint besides syntax errors.

Defining constraints

The predefined object **`Class`** offers attributes for rules and constraints

```
Class in Individual with
   attribute
      constraint: MSFOLconstraint;
      rule: MSFOLrule
end
```

· The only remarkable property of Class is that is makes ‘rule’ and ‘constraint’ available for its instances!

```
Employee in Class with
   attribute
      colleague: Employee;
      salary: Integer
   constraint
      con1: $ forall e/Employee s/Integer
            (e salary s) ==> (s < 200000) $
end
```

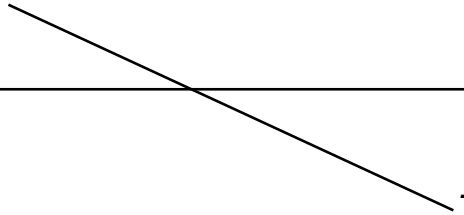
binding variables to classes

Indicates a formula

the label of the constraint carries no special meaning

Defining constraints (3)

```
EntityType in Class with
  attribute
     feature: Domain
  constraint
     atleastonefeature: $ forall et/EntityType exists d/Domain
                              (et feature d) $
end

Project in EntityType end
```



This instance of EntityType violates the constraint!

· Constraints at the notation level specify which models are ‘syntactically’ allowed. Compare to balancing rules!

NB: In this course, we use the term 'notation' as a synonym to '(modeling) language'.

Defining deductive rules

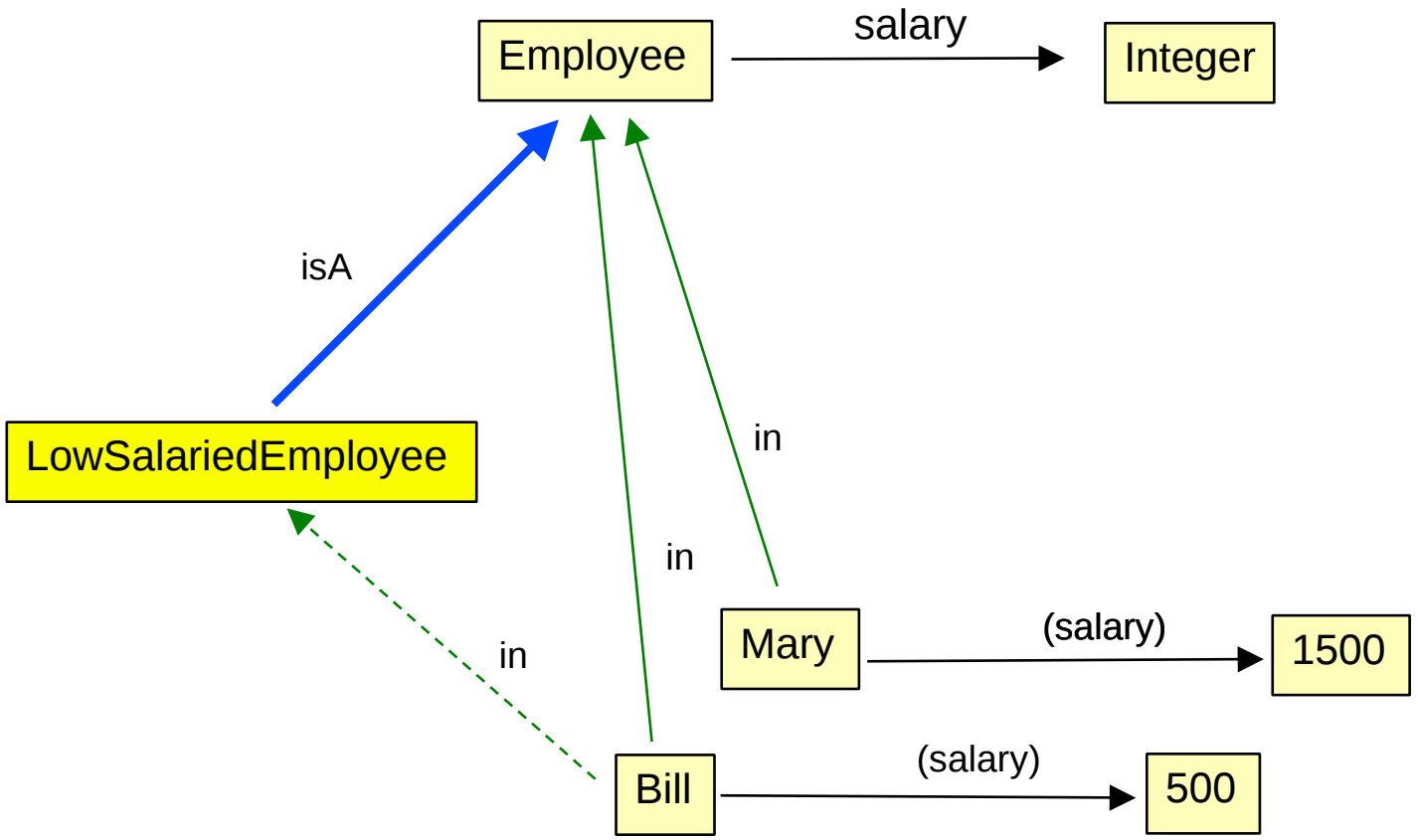
```
Employee in Class with
    attribute
        colleague: Employee;
       hasColleague: Employee;
        salary: Integer
    rule
        ru1: $ forall e1,e2/Employee
                        (e1 hasColleague e2) ==> (e1 colleague e2)$;
        ru2: $ forall e1,e2/Employee
                        (e2 hasColleague e1) ==> (e1 colleague e2)$;
        ru3: $ forall e1,e2,ez/Employee
                         (e1 colleague ez) and (ez colleague e2)
                                          ==> (e1 colleague e2)$

end
```

- Variables in conclusion predicate are forall-quantified. Further variables on the condition body.
- Allowed conclusion predicates: (x m y), (x in c)
- Rules compute new facts out of existing ground facts (P-objects)

Defining Query Classes (1)

· Query classes look like ordinary classes with constraints. However, their instances are ‘computed’.

· Example: LowSalariedEmployees are those employees who earns less than 1000.



Defining Query Classes (2)

```
LowSalariedEmployee in QueryClass isA Employee with
   constraint
      lowsal: $ exists s/Integer
                    (~this salary s) and (s < 1000) $
end
```

· The variable '~this' stands for any instance of Employee; it is recommended to use the '~' in conjunction with 'this', though 'this' is equivalent to '~this'

· The query class stands for a deductive rule:

```
forall ~this/Employee
   (exists s/Integer (~this salary s) and (s < 1000))
==> (~this in LowSalariedEmployee)
```

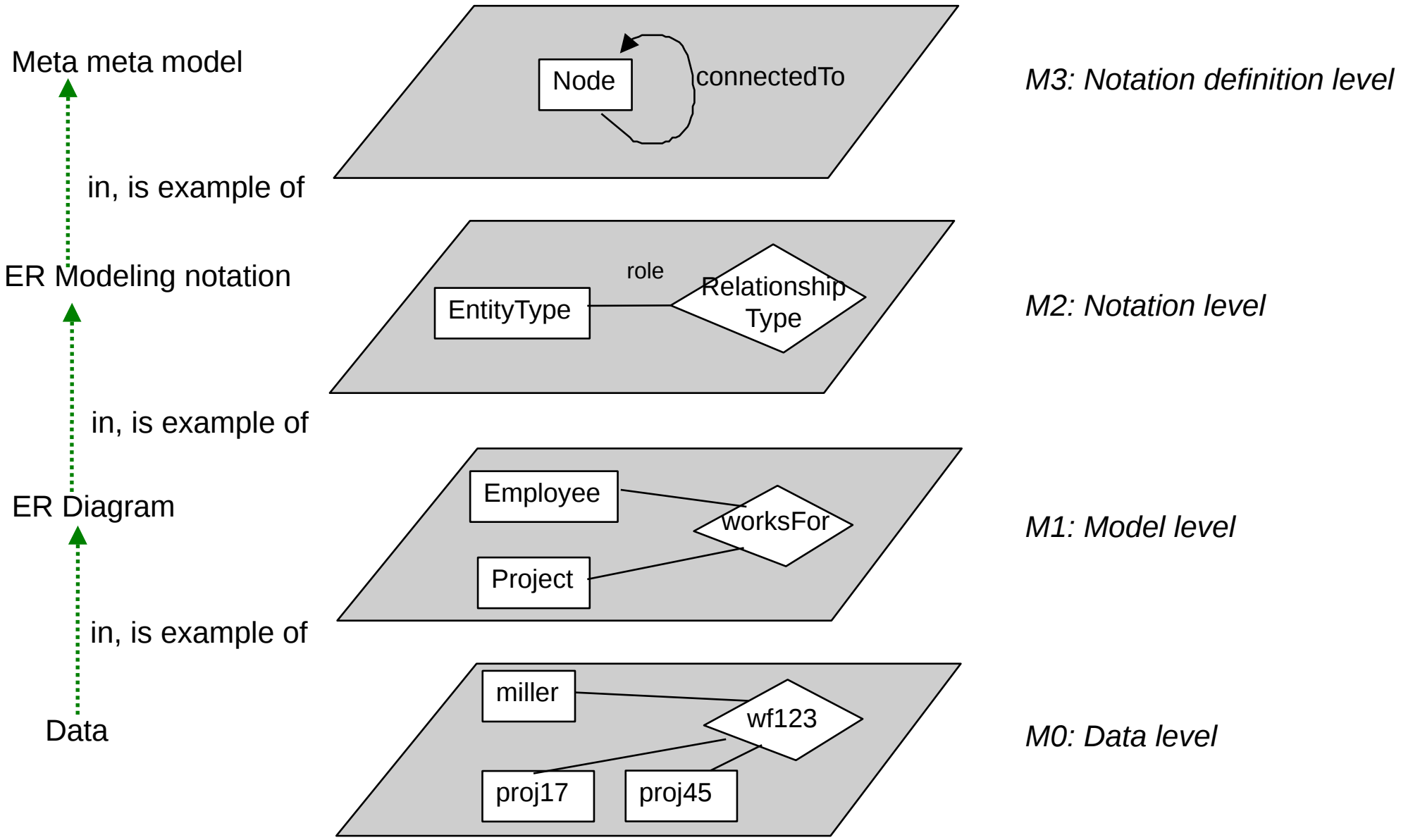
· Query classes are 'told' to the system like ordinary classes.

Equivalent formulation without query class

```
LowSalariedEmployee in Class with
    rule
        lowsal: $ forall thisE/Employee
                  (exists s/Integer (thisE salary s) and (s < 1000))
                   ==> (thisE in LowSalariedEmployee) $
end
```

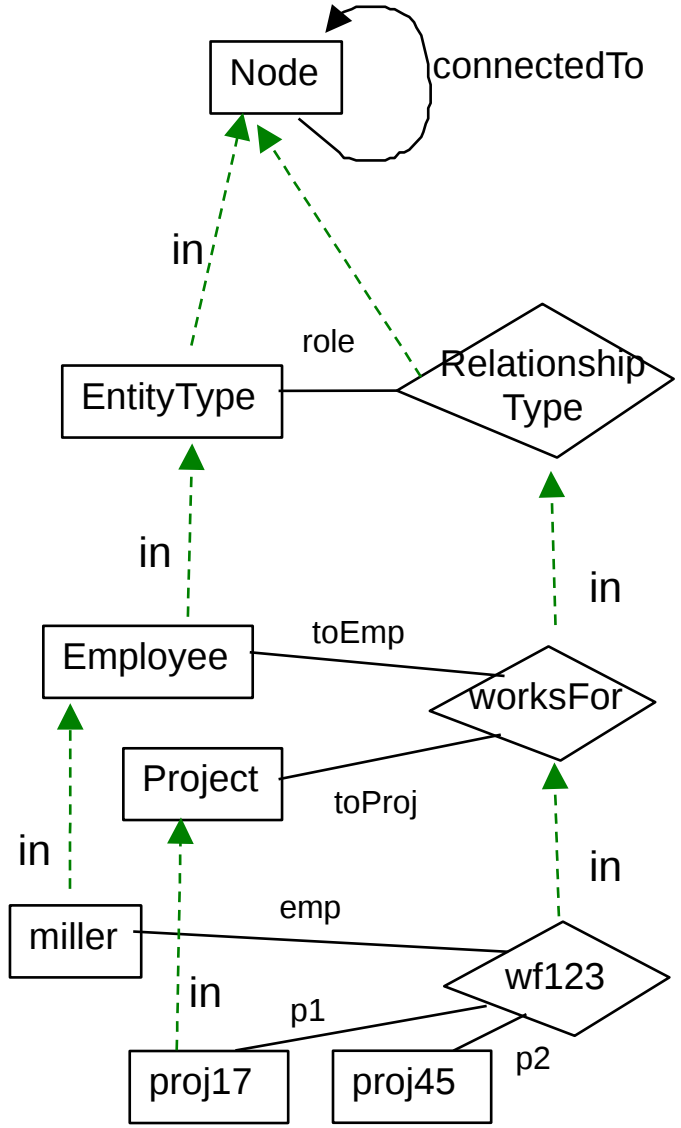
- This definition will work exactly like the query class definition on the previous slide!
- Query classes are “shortcuts” for deductive rules.

First example: Engineer the ERD-simple notation



A three-line definition of the ERD-simple notation

Graphical



Logical

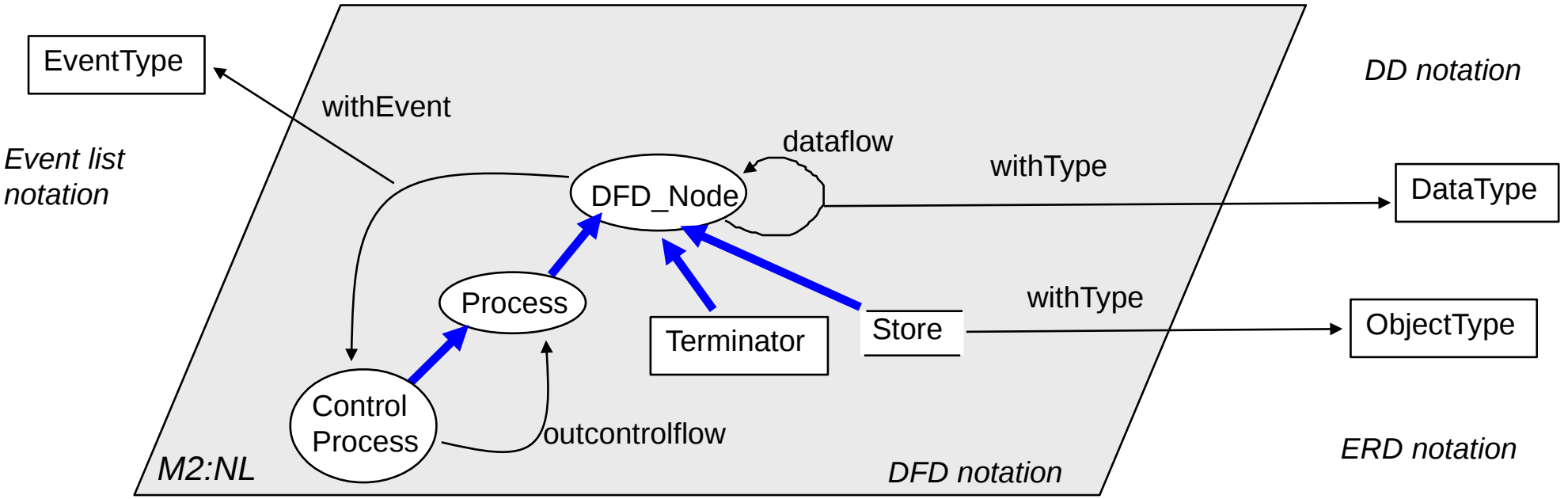
(Node attribute/connectedTo Node)

(EntityType in Node)
(RelationshipType in Node)
(RelationshipType connectedTo/role EntityType)

(Employee in EntityType)
(Project in EntityType)
(worksFor in RelationshipType)
(worksFor role/toEmp Employee)
(worksFor role/toProj Project)

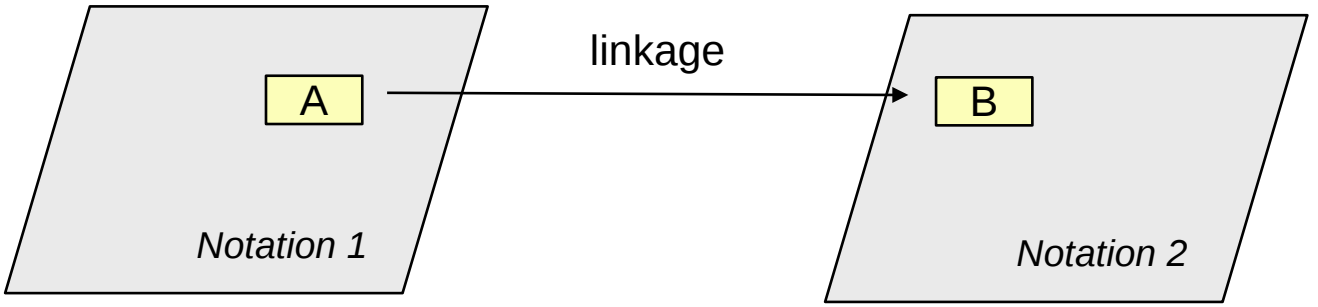
(miller in Employee)
(proj17 in Project), (proj45 in Project)
(wf123 in worksFor)
(wf123 toEmp/emp miller)
(wf123 toProj/p1 proj17)
(wf123 toProj/p2 proj45)

Inter-relationships between notations



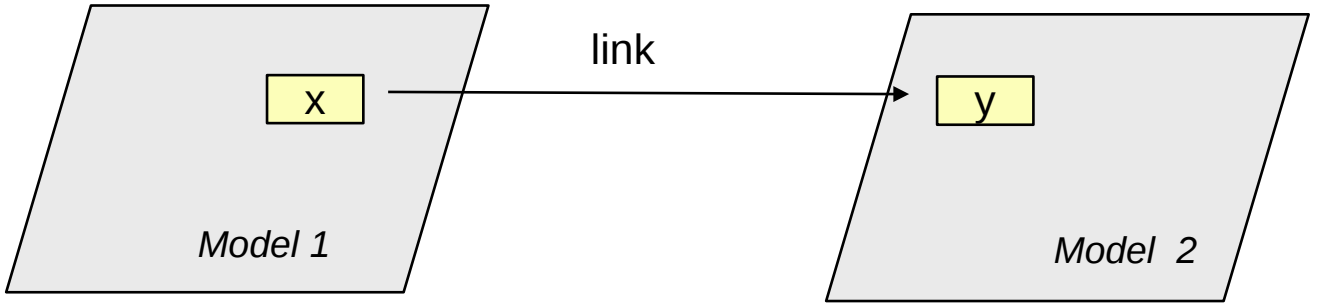
- The DFD symbols are connected to symbols of other notations
- In a comprehensive method, the notations are always inter-related. They can only be fully understood (see also stores in snapshot) when all inter-relationship are represented
- Inter-notational balancing rules are formalizing which combinations of models are allowed

Balancing between notations



“cond(x/A) and linkage(x,y) ==> cond(y/B)”

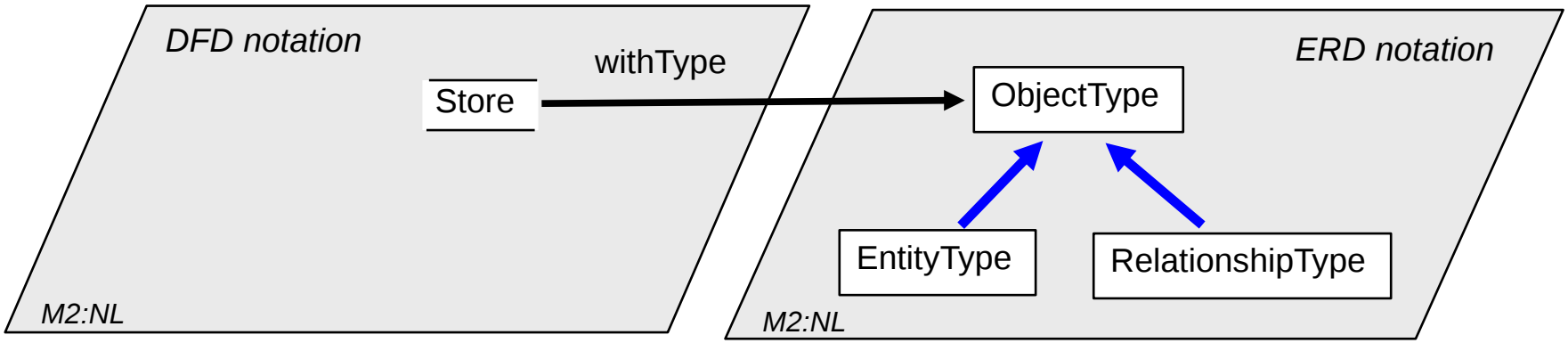
“cond(x/A) and linkage(x,y) and not cond(y/B) ==> inconsistent”



- If there is an object x of kind A fulfilling some condition and x is linked to some object y of kind B then some condition on y must hold

Balancing DFD vs. ERD

"(BR1) Each data store must be associated to an entity type or relationship type and vice versa."




```
Store in Class with
  constraint
      br1a: $ forall s/Store exists o/ObjectType (s withType o) $;
      br1b: $ forall o/ObjectType exists s/Store (s withType o) $
end
```

- Balancing rule as constraint would disallow any data store without an object type and any object type without a store

Balancing rules as query classes

```
QueryClass StoreWithoutType isA Store with
  constraint
      br1a: $ not exists o/ObjectType (~this withType o) $
end


QueryClass ObjectTypeWithoutStore isA ObjectType with
  constraint
      br1b: $ not exists s/Store (s withType ~this) $
end
```

- The solution with the query classes is preferable because it allows to represent incomplete models, e.g. a DFD with data stores that have not yet been associated to data types

- Models not violating the balancing rules generate empty answers (‘nil’) to both queries displayed above

Balancing DFD

"(BR2) Any process must have at least one input and one output."

```
QueryClass ProcessLacksInput isA Process with
  constraint
      br2a: $ not exists n/DFD_Node (n dataflow ~this) $
end


QueryClass ProcessLacksOutput isA Process with
  constraint
      br2b: $ not exists n/DFD_Node (~this dataflow n) $
end
```

- The corresponding balancing rule on data stores can be realized analogously

Balancing rule: Each relationship type must have at least one role

```
RelationshipType in Node with
   connectedTo
     role: EntityType
   constraint
     br1: $ forall r/RelationshipType exists e/EntityType
                 (r role e) $
end
```

· ‘worksFor’ fulfills the balancing rule but the subsequent relationship type does not:

```
hasDept in RelationshipType end
```

· Note that **we** (the method engineers) decide on the balancing rules that we want to enforce!
· We will see in the section on software process models that we can also specify **when** we want to check the rules.

Balancing rules as queries

· If a balancing rule is formulated as a constraint, then we are not able to enter a model to the repository which violated the balancing rule.

· This is sometimes inconvenient because modeling is a process which ends with a ‘consistent model’ but we would like to work with incomplete, inconsistent models in between.

· Solution: Formulate the balancing rule as query classes

```
BR1_violator in QueryClass isA RelationshipType with
  constraint
     br1: $ not exists e/EntityType (~this role e) $
end
```

negate the ‘positive’ constraint in order to find the violators of balancing rule BR1

· This trick would allow violations of BR1 while we can ask at any time which parts of the ERD-advanced model are inconsistent!

Negated form returns **violators** of the positive form

```
BR1_violator in QueryClass isA RelationshipType with
  constraint
     br1: $ not exists e/EntityType (~this role e) $
end
```

Deductive rule corresponding to the query class

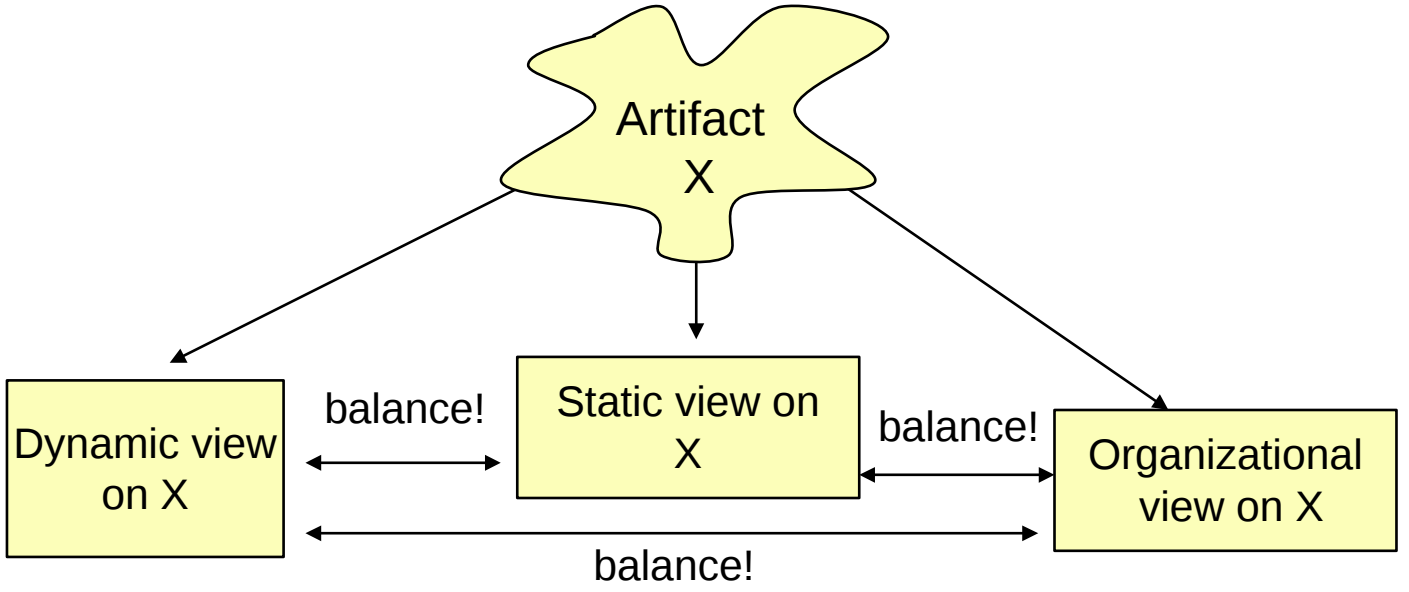
```
forall ~this/RelationshipType
   not exists e/EntityType (~this role e)
==> (~this in BR1_violator)
```

“return those relationship types that violate the 'positive' constraint”

is negation of the 'positive' constraint

```
forall r/RelationshipType
   exists e/EntityType (r role e)
```

Multiple models means partial viewpoints



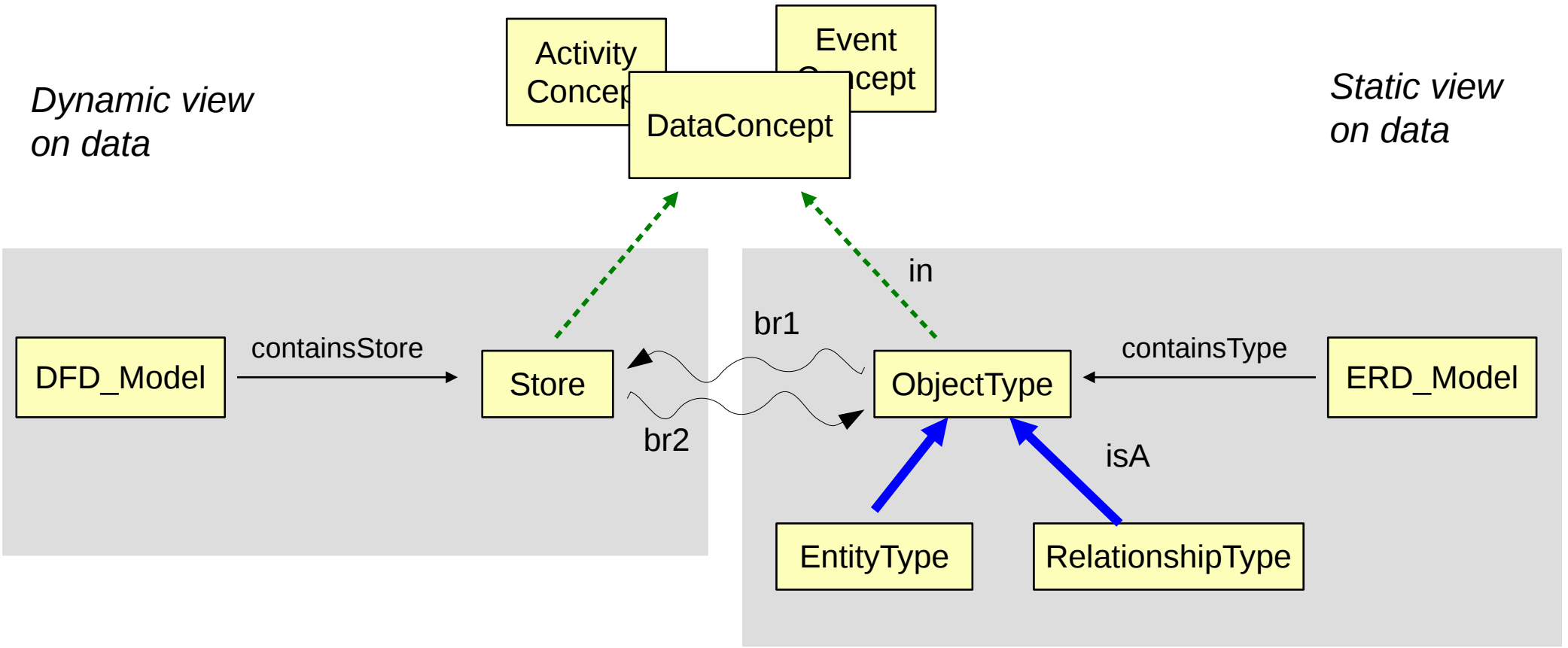
Each partial model contains only statements about X that are of interest in the partial model

We somehow “tear” the artifact X into parts that are interrelated, i.e. that are connected by means of balancing rules

So, a major ingredient of a ‘method’ are the balancing rules that control how the partial models are interrelated

Questions: How can we plan suitable notations for the perspectives and their balancing rules???

Example: the DataConcept in the Yourdan method

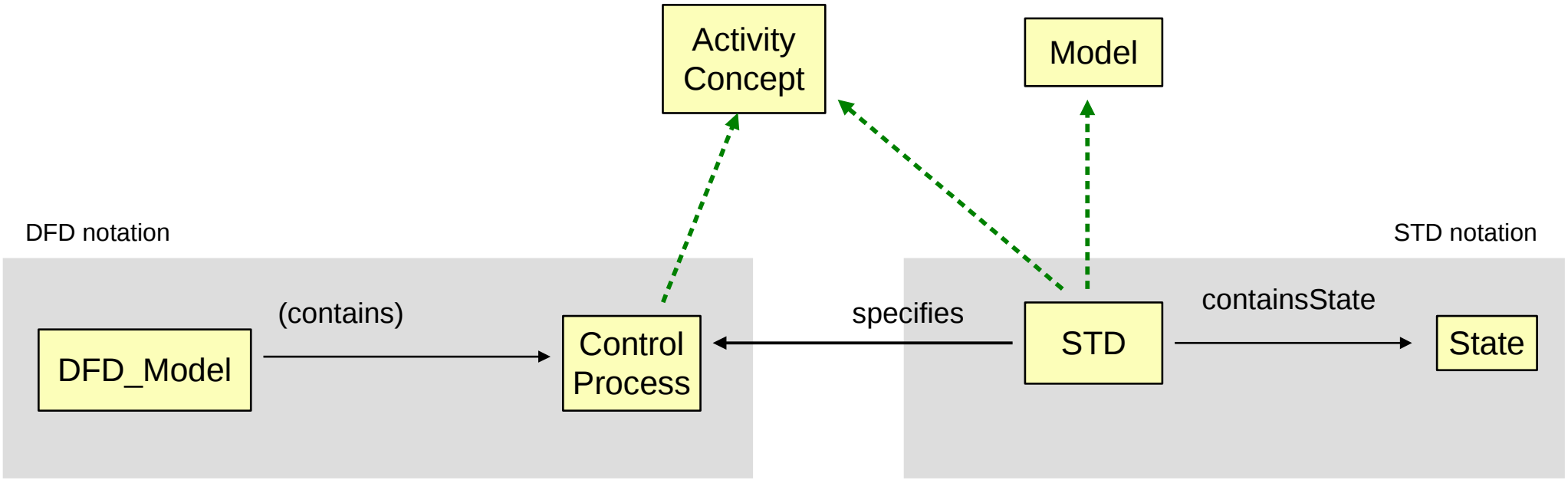


br1: any data store in a DFD model must be connected to an object type in the ERD model

br2: any object type in the ERD model must be connected to a data store in a DFD model

the classes ‘DataConcept’, ‘ActivityConcept’ etc. replace the more generic concept Node, i.e. we construct a richer Notation Definition Level!

Activities in DFD,STD models



This is similar to the previous situation. We will demand that any control process has an STD as specification and vice versa.

Note that the component symbol ‘State’ of STD are not directly linked to a DFD.

Creating multiple perspective methods

Goal: develop a new method with specialized modeling languages together with their balancing rules

Steps:

1. Develop a suitable “rich” notation definition level (not just “nodes plus connections”) which defines what kind of notations we are looking for

2. Derive perspectives by selecting parts of the concepts of the notation definition level

3. Instantiate these perspectives to obtain notations for them (one per perspective).

4. Balance the notations for the different perspectives.

CASE study: Modeling notations for analyzing business processe

Setting:

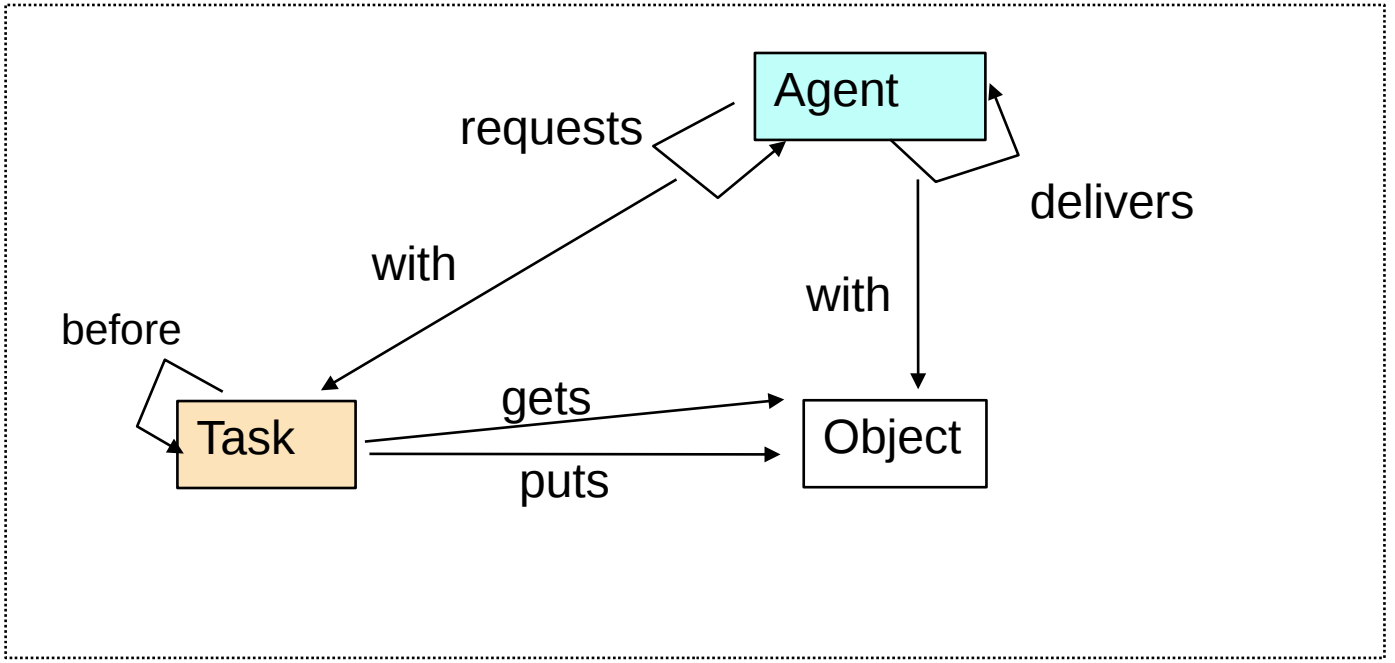
Organizations (like companies) do their business by assigning tasks among their employees. Those employees (‘agents’) send documents (‘objects’) to each other that are needed to fulfill the tasks. Tasks have a precedence order.

We are especially interested in the analysis of certain “undesired” business processes, e.g. an employee sends a document to another employee, however this employee never has received a task which requires that document

Our mission:

Find a suitable collection of modeling notations to represent and analyze such situations.

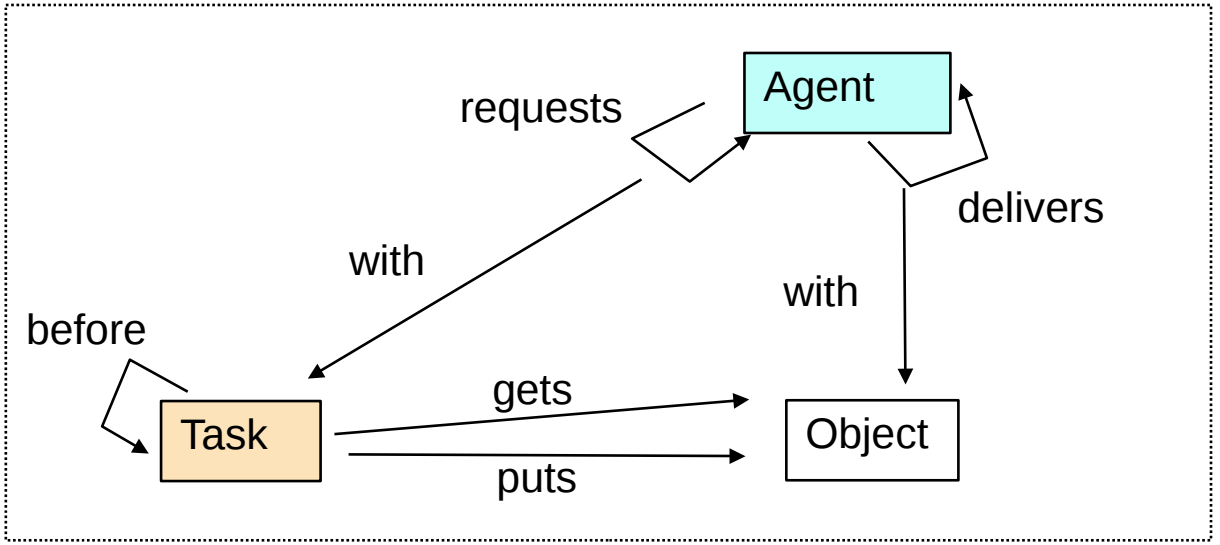
Step 1: Notation definition level



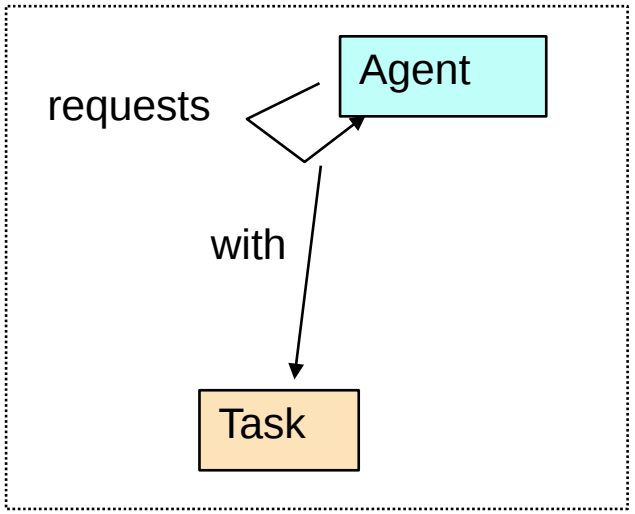
This notation definition level introduces the three key concepts mentioned in the case study. It encodes that we will be interested in modeling languages for the following kind of statements:

- Agents requests each other to do a certain task.
- Tasks have precedence order (“before”). They fetch (“get”) objects and they create (“put”) objects.
- Agents send objects to each other (“delivers”).

Step 2: Derive Perspectives

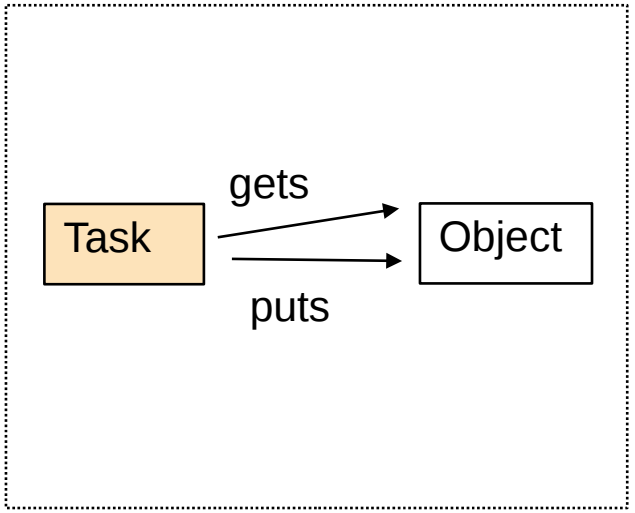


Notation definition level



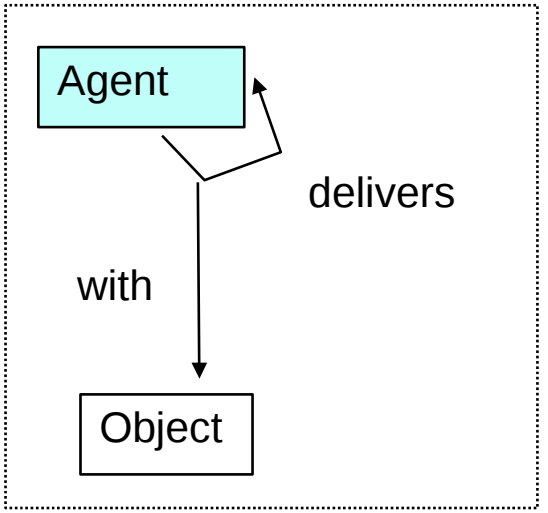
Task request perspective

+



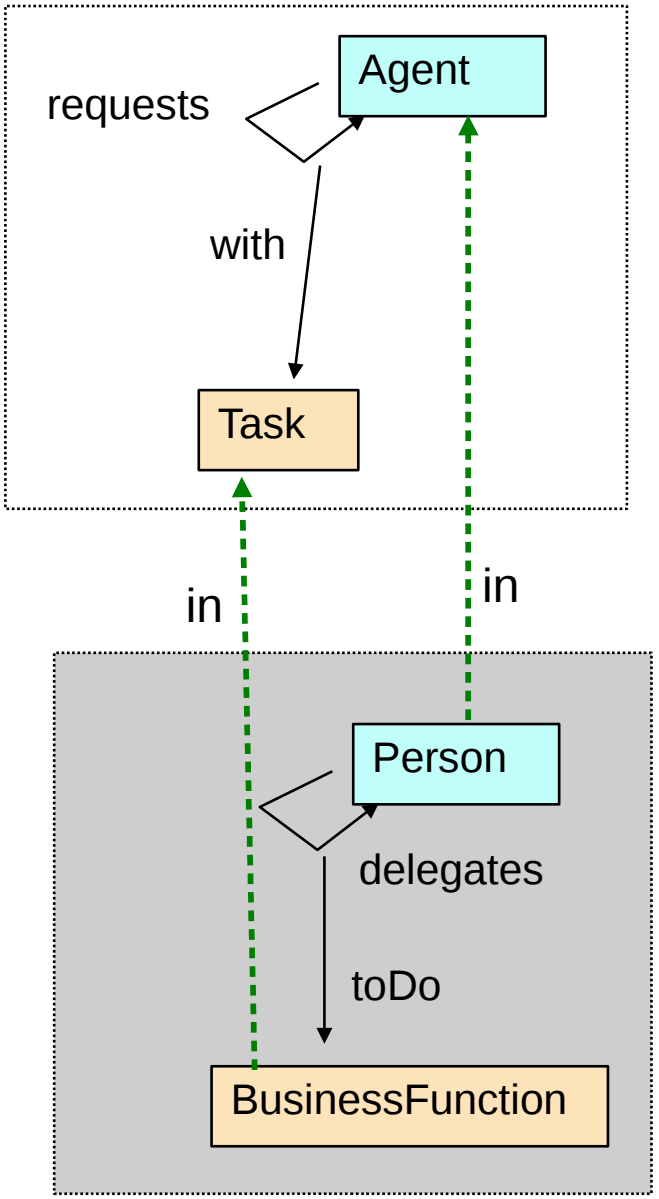
Task behavior perspective

+



Object flow perspective

Step 3.1: Notation definition for tasks requests



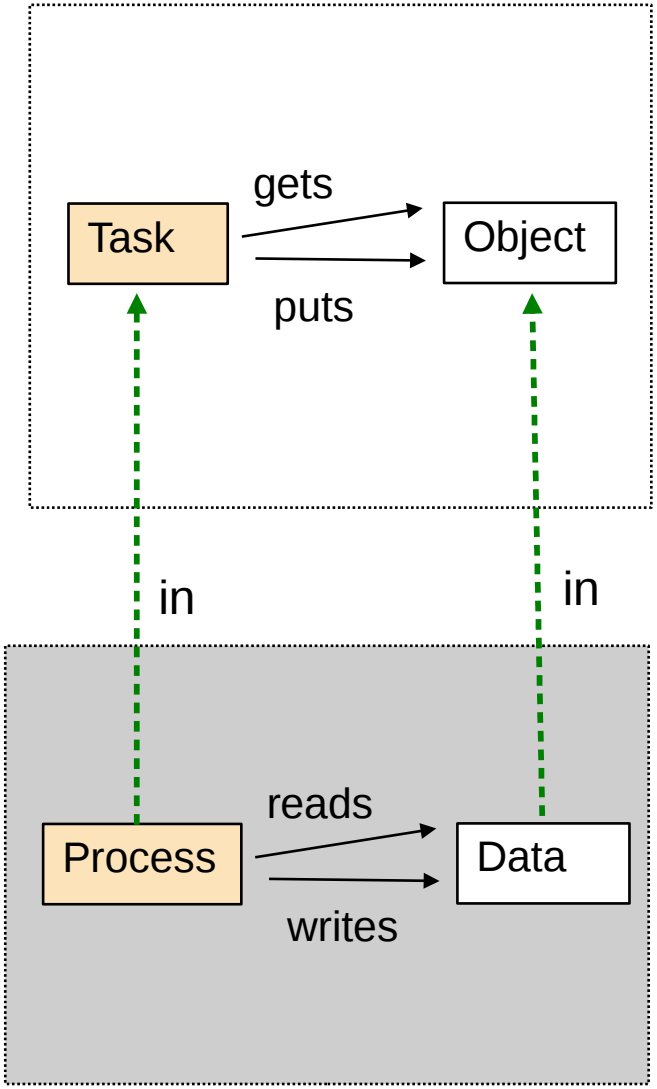
In the notation for task requests, we use a special node type “Person” for the agents. It is identified by the little person icon.

Tasks will be called “Business function” in our notation.

The notation can be adapted to the desired natural language (here: English). Note that the perspective is independent from the notation: it has generic terms for the kind of concepts.

In the example, we instantiate just one notation class per perspectice class. In general, we can have multiple instances. Moreover additional link types are possible that are not represented in the NDL.
Yourdan: dataflow is not in the richer NDL.

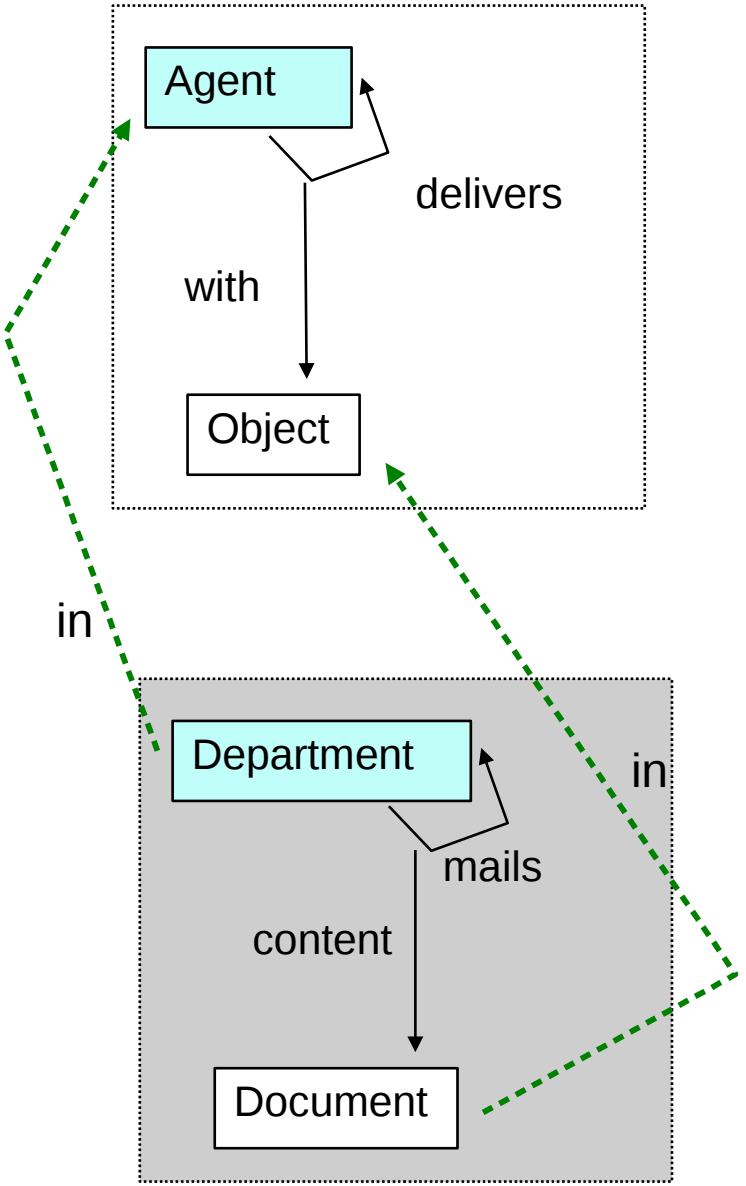
Step 3.2: Notation definition for task behavior



In the task behavior perspective, we decide to label tasks “process” and object “data”. This resembles the DFD notation (though it is not the same).

Note that the people who write models for task/process behavior may be different from those who define the models for the task requests!

Step 3.3: Notation definition for object flow



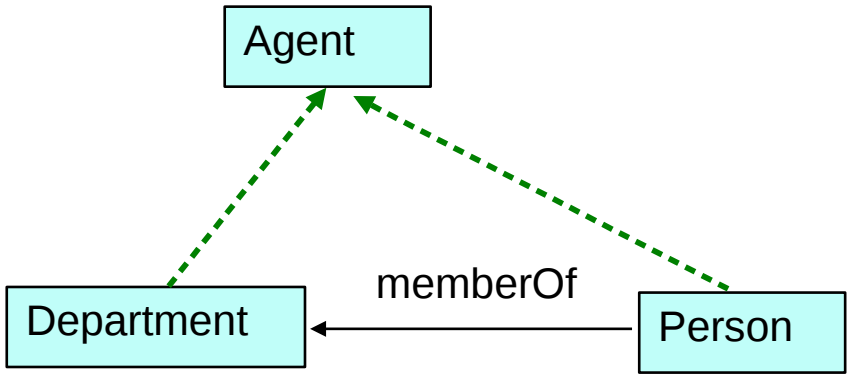
The notation for the object flow completes the instantiation of the perspectives. Here, the agents are departments.

Obviously, we want to express here that only departments send “official” documents to each other.

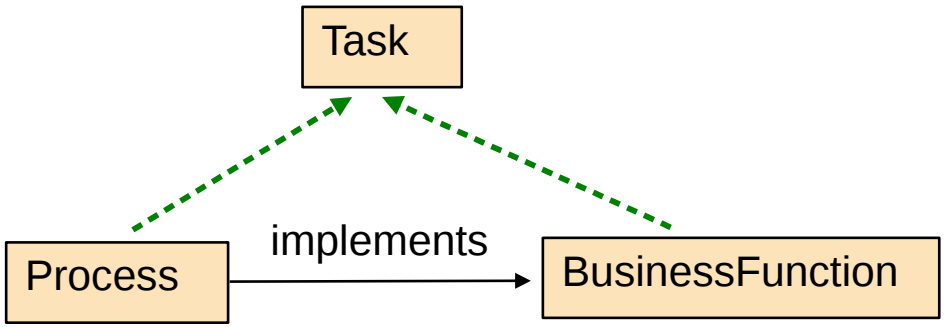
Note that the concepts Agent, Task, and Object appear in different flavor in the notations. This can also be observed in Yourdan’s collection of notations:

Data stores are not the same as entity/relationship types but the both model the same data concept!

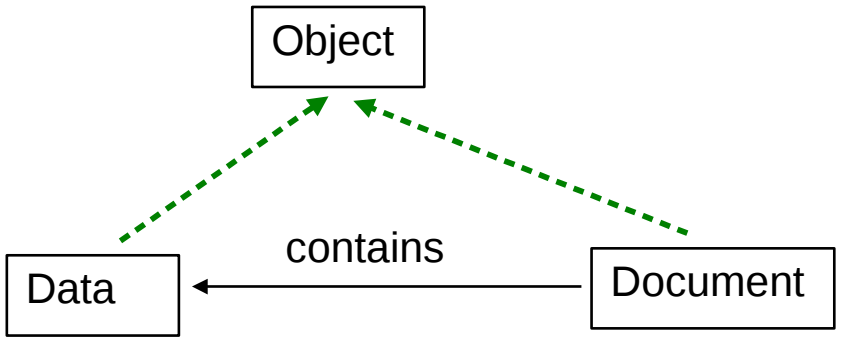
Step 4: Balance multiple viewpoints on notation definition concepts



Agents are regarded as “departments” in the one perspective and as “persons” in the other. Here, we can connect the two by a “memberOf” link. This is a kind of “partOf” relationship



As we know that business functions are passed between persons, we can interpret the link to “process” as being an implementation relationship: a piece of software implements a business function or supports it.

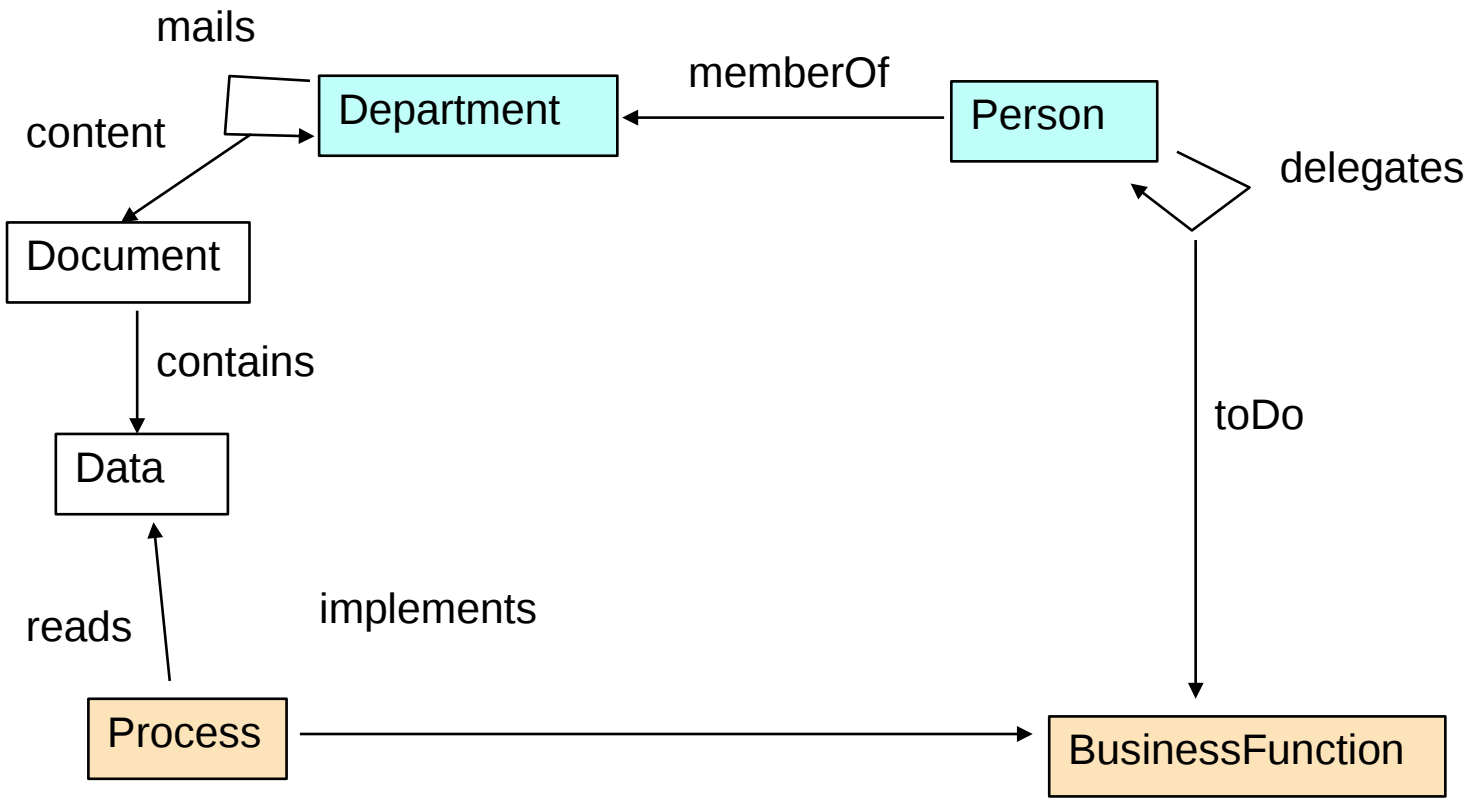


We can interpret “documents” as physical objects (files, paper) which contain data objects. When mailing a document between departments, the content is not essential. It becomes essential when a process manipulates its data.

Balancing rules for the new method

We limit ourselves to balancing rules that range over multiple notations

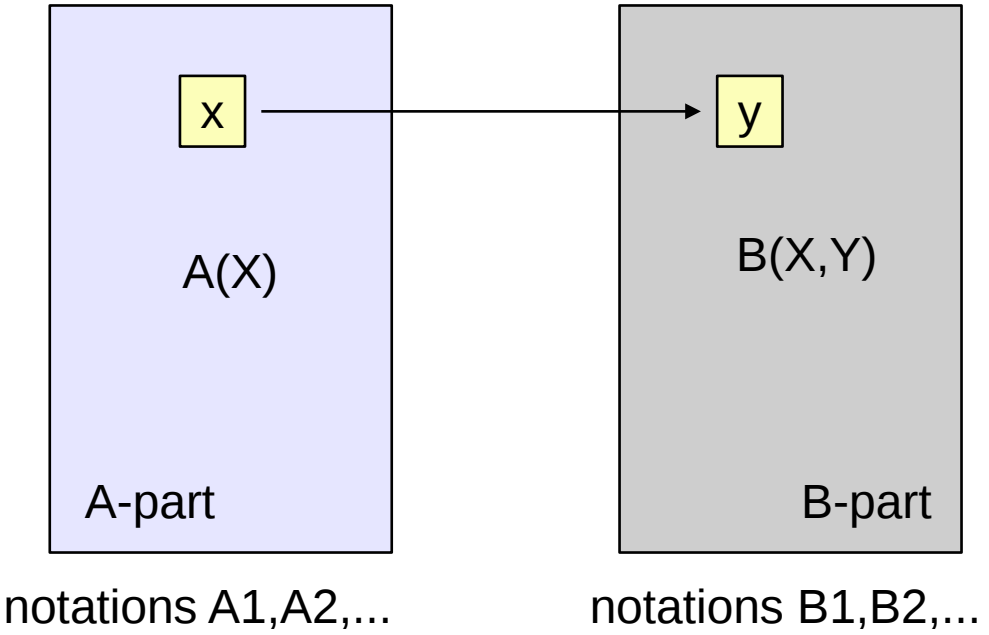
BR1: Whenever a person receives a request to perform a business function, her department must have received the documents containing all the data that are read by all the process that implement the business function!



http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/4047690

Strategy for designing the balancing rule

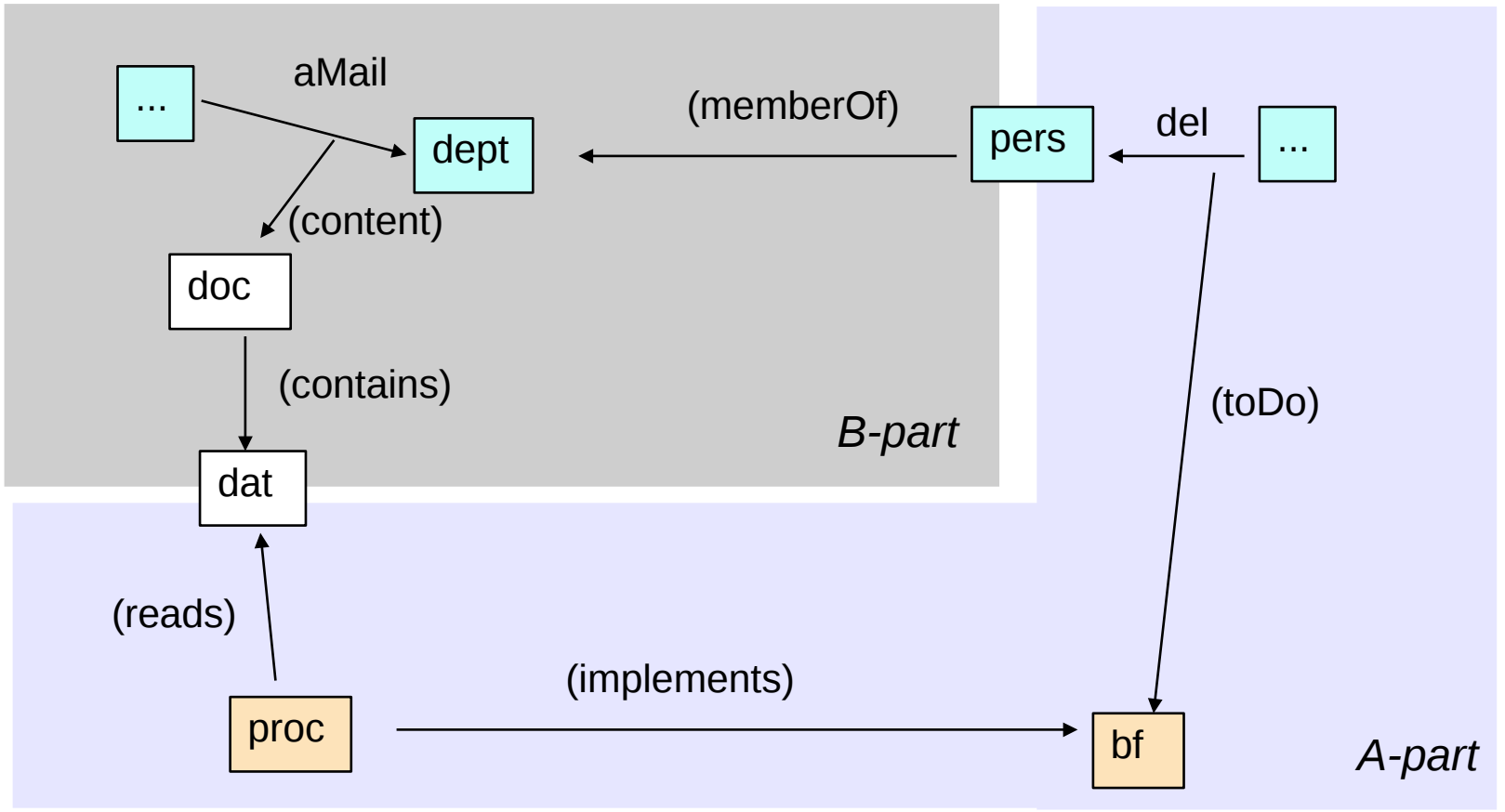
Many balancing rules including the example have the form




```
forall X A(X) ==> exists Y B(X,Y)
```

describes a pattern of objects present in models of A1,A2,.. (“reach y from x”)

demands a pattern of objects present in models of B1,B2,.. (“reach y over another path”)




```
forall del/Person!delegates pers/Person bf/BusinessFunction
       proc/Process dat/Data
   To(del,pers) and (del toDo bf) and (proc implements bf) and
   (proc reads dat)
==> exists dept/Department aMail/Department!mails doc/Document
  (pers memberOf dept) and To(aMail,dept) and (aMail content doc) and
  (doc contains dat)
```

The Zachman Framework

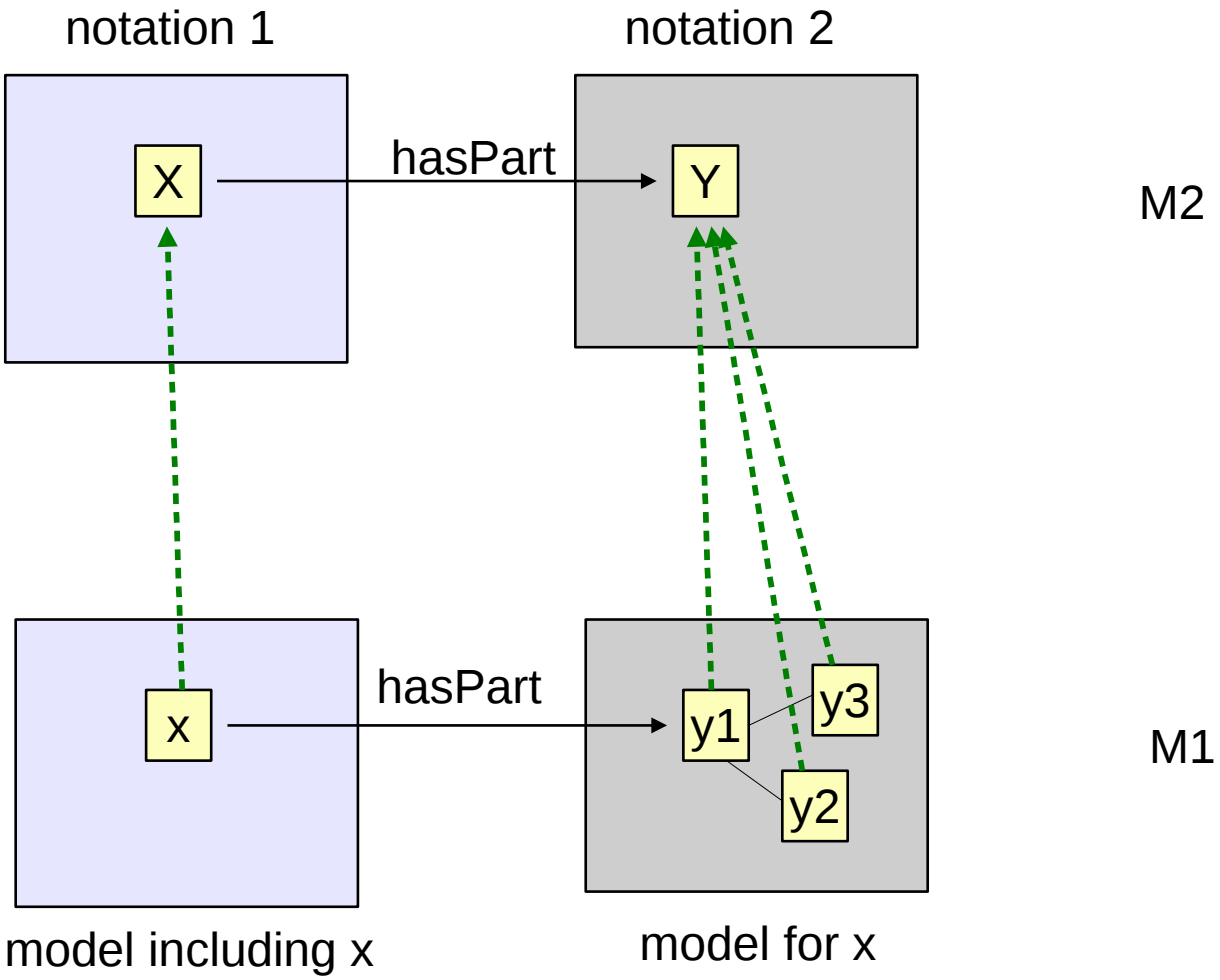
	Why	How	What	Who	Where	When
Contextual	Goal List	Process List	Material List	Organizational Unit & Role List	Geographical Locations List	Event List
Conceptual	Goal Relationship	Process Model	Entity Relationship Model	Organizational Unit & Role Rel. Model	Locations Model	Event Model
Logical	Rules Diagram	Process Diagram	Data Model Diagram	Role relationship Diagram	Locations Diagram	Event Diagram
Physical	Rules Specification	Process Function Specification	Data Entity Specification	Role Specification	Location Specification	Event Specification
Detailed	Rules Details	Process Details	Data Details	Role Details	Location details	Event Details

*) image freely licensed from Wikipedia:Zachman Framework

- Best-known example of a development framework; features 30 (!) modeling languages but many have rather simple incarnations (e.g. goal list=textual list of goals)
- Many balancing rules and mapping rules to be expected but not formalized in the framework

Patterns for cross-notational links

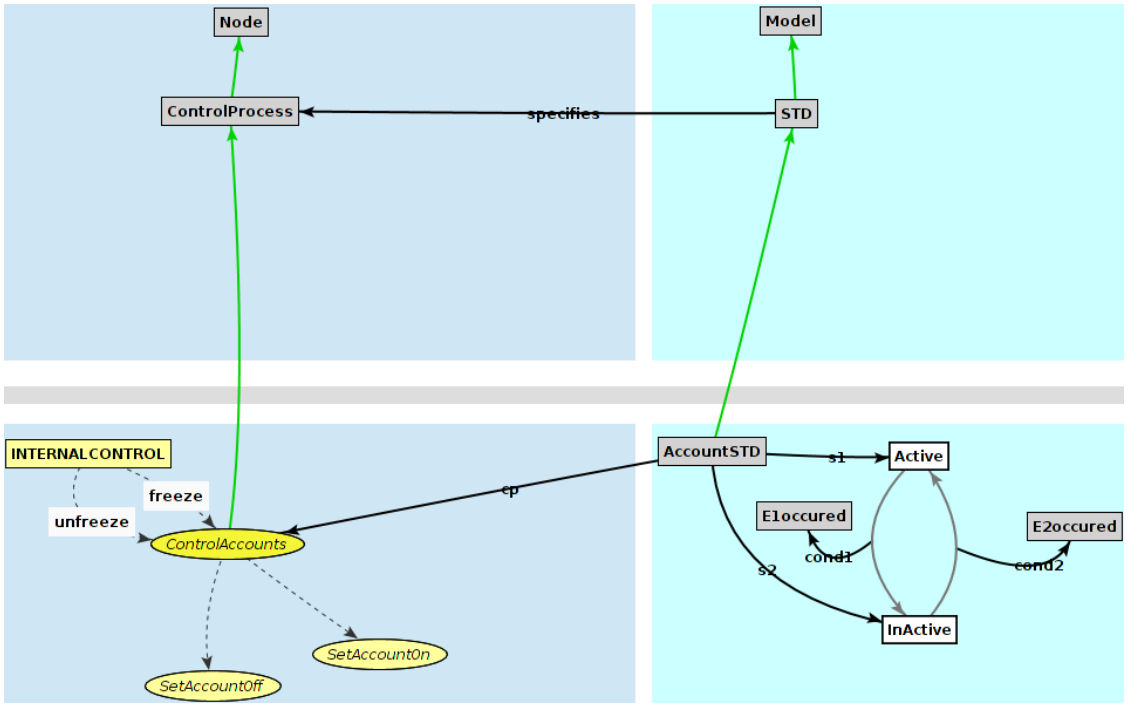
Pattern 1: The hasParts link (called “model explosion” in MetaEdit+)



The model for x defines the element 'x' that is used on the left hand side.

Pattern 1: Examples

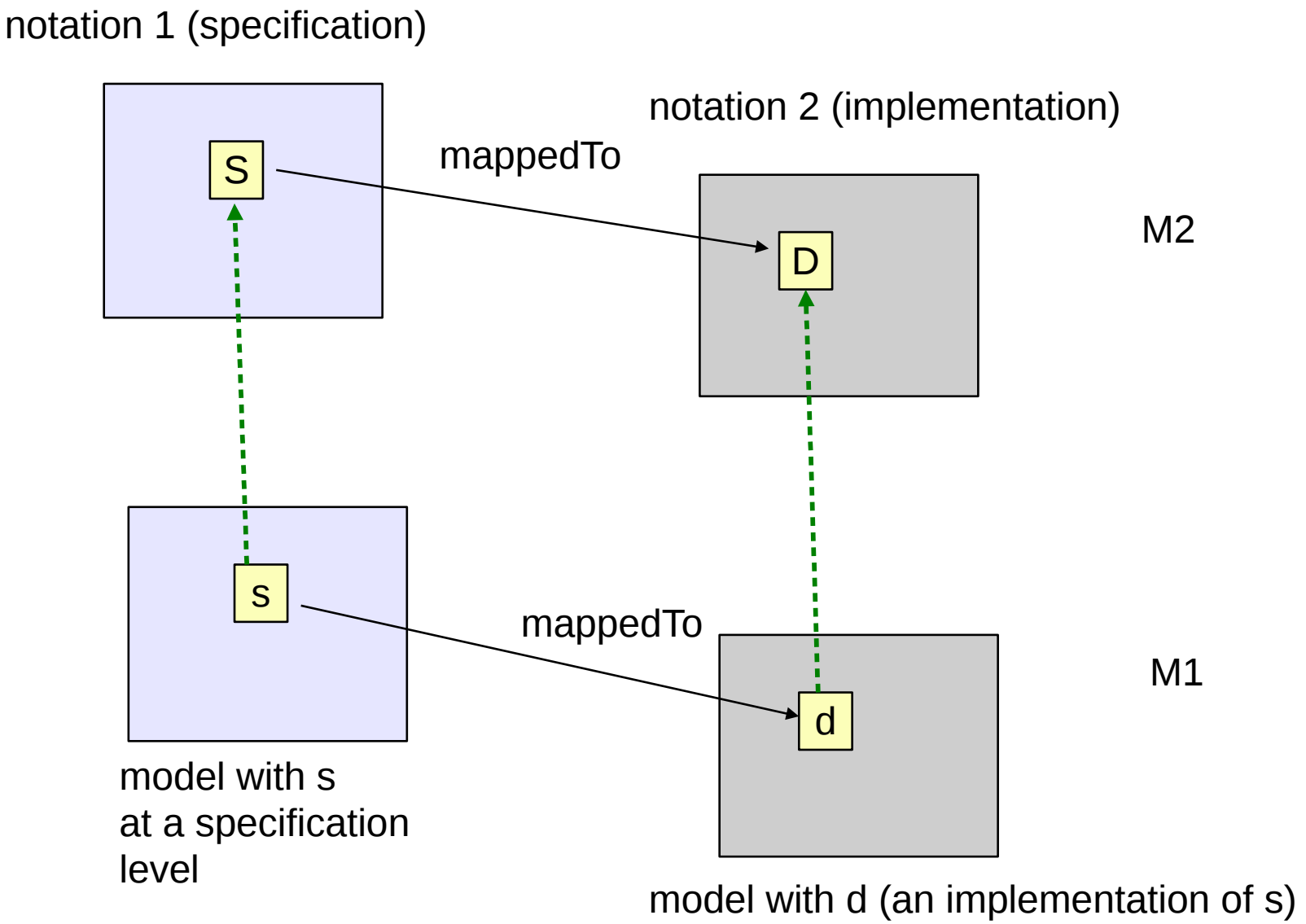
- BPMN: the decomposition of a task from a high-level process model to a lower-level process model
- Relational data model: the schema of a database as a set of table definitions
- The source code of a class occurring in a class diagram
- **Yourdan: the STD being the specification of a control process occurring in a DFD**



http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/3825814

http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d3825827/pattern1.gel

Pattern 2: The mapping to a more concrete representation



Typical: when the whole is mapped, then also its parts (e.g. mapping of an ER model to relational schema implies that all elements of the model are mapped)

Pattern 2: Examples

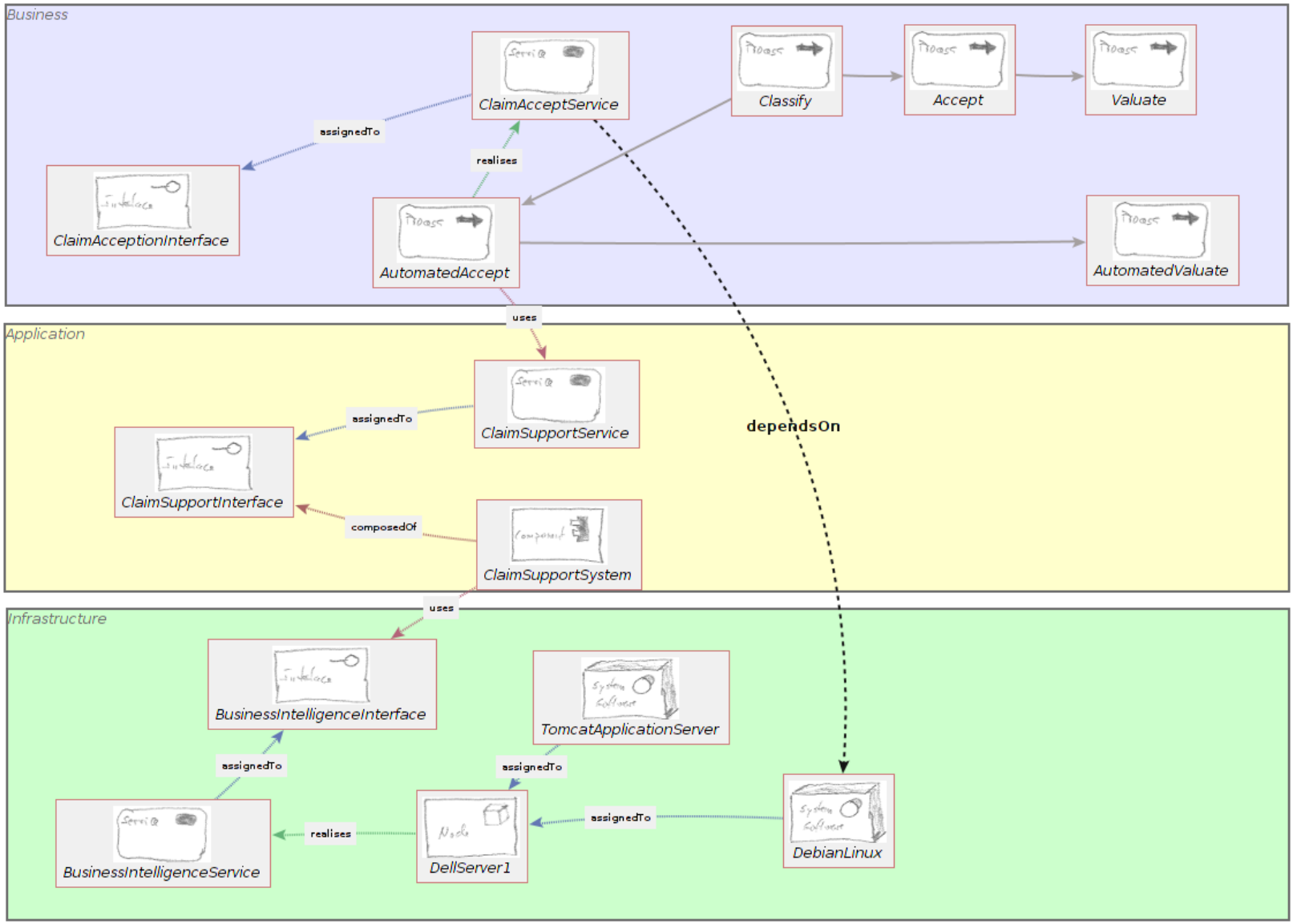
- the relational schema mapped from an ERD model
- a process specification mapped to program code
- **an application system implementing a business task (ArchiMate)**

The target of the mapping (the implementation) is about the same object but at a more concrete level, in particular certain design choices can be made to implement the source (specification).

Besides one-to-one, the mapping can also be one-to-many, many-to-one, and many-to-many.

See also: model transformations

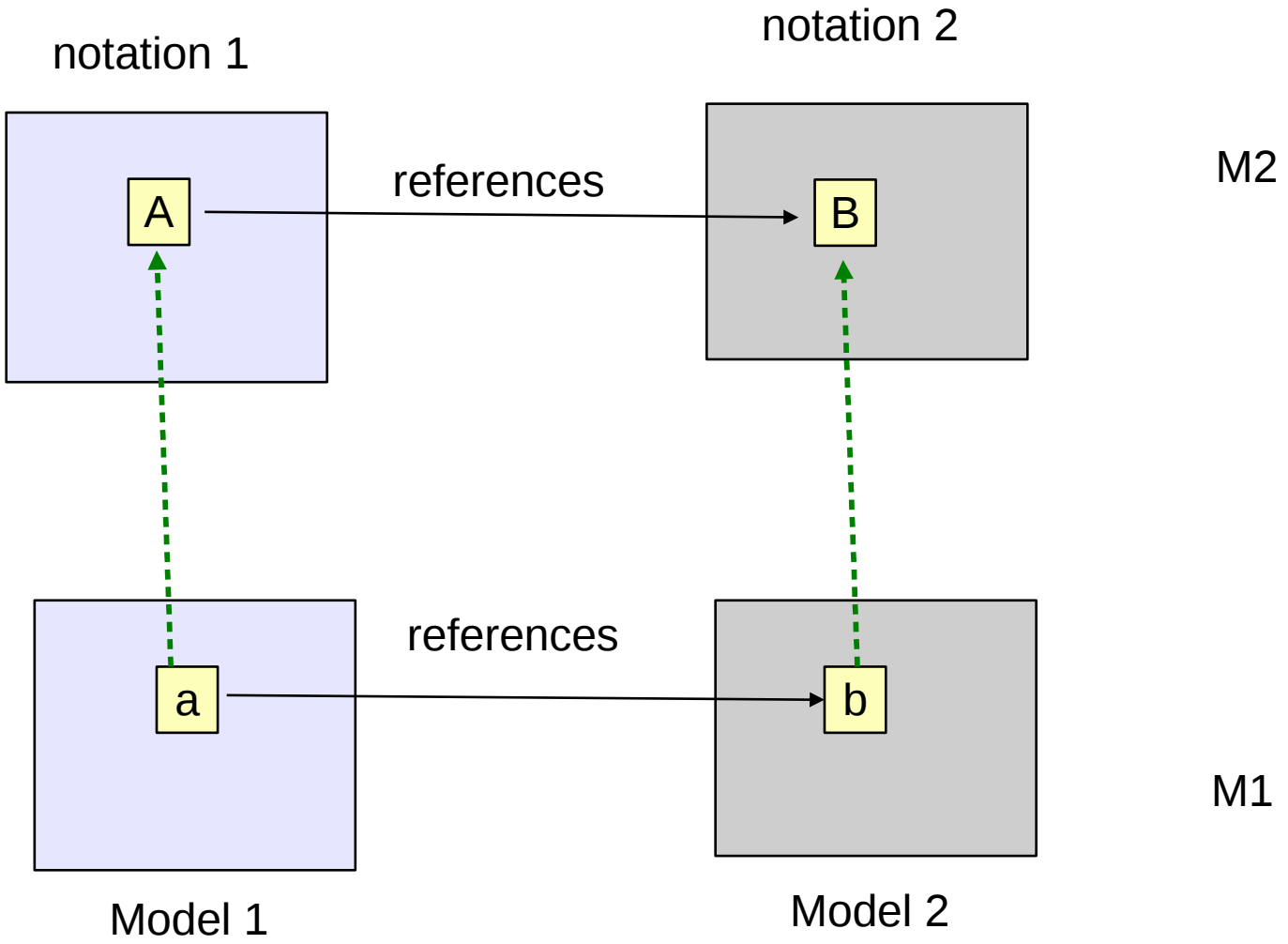
Pattern 2: Demo ArchiMate example



http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d3920090/archisurance.gel

http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/3822537

Pattern 3: A cross-reference to neighbor notation



Typical: Notaion 1 focusses on a limited number of constructs and has outsourced referenrec constructs to a neighbor notation. Both must be seen together and could also form a joint notation (notation 1+2).

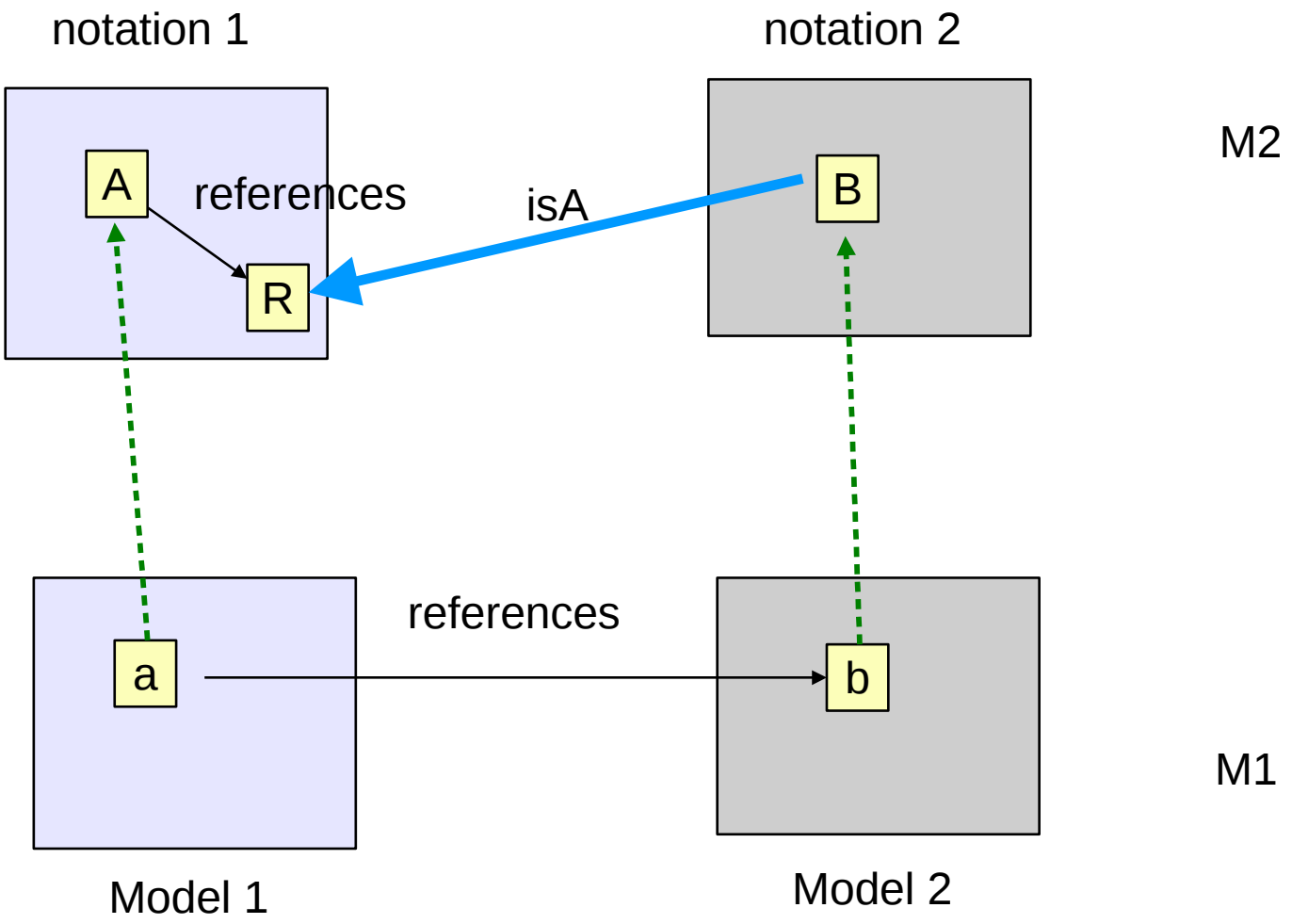
Pattern 3: Examples

- goals (goal notation) reference actors (resource notation)
- tasks (process modeling notation) reference organizational units
- a business service complies to a business rule

The cross-reference typically comes with a multiplicity constraint line 1..1 or 1..* on one or both of the reference ends. For example, a task in a business process must have an associated organizational unit that is responsible for its execution.

The reason to split the modeling task into multiple notations is for pattern 3 sometimes just pragmatic e.g. to limit the number of constructs per notation, or to have only elements in a notation that can be mapped to another notation, e.g. map a process model to a simulation model.

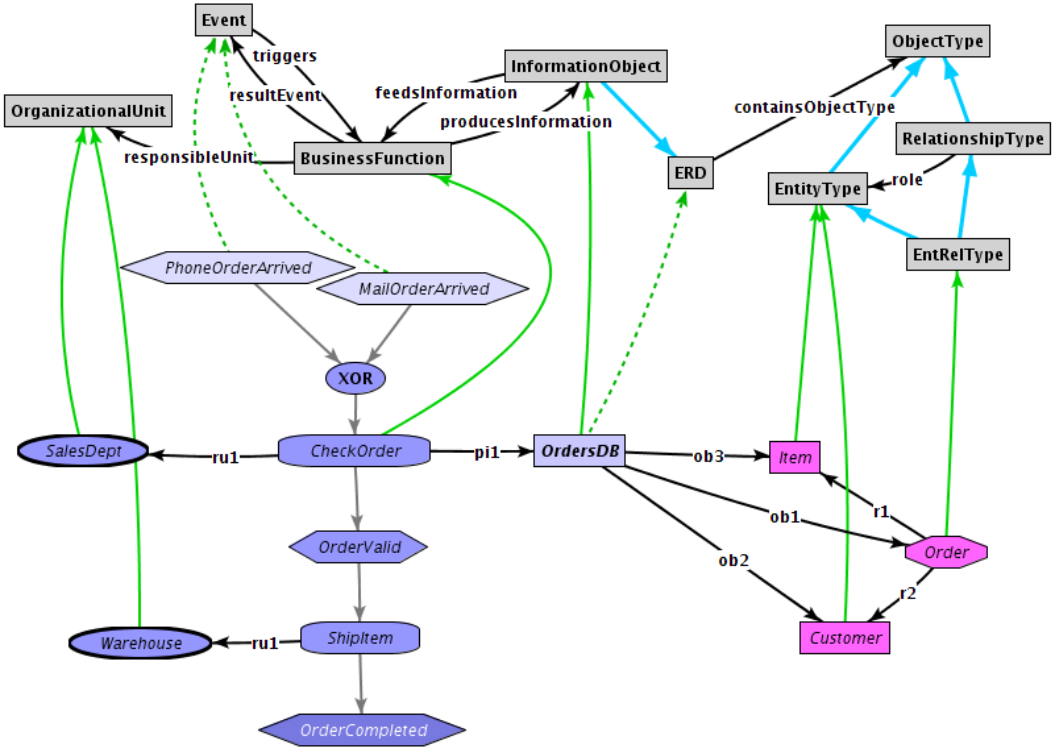
Pattern 4: Interface classes



Special case of pattern 3: The abstract class R is located in notation 1 but has subclasses in other notations such as B in notation 2. There can be several such subclasses distributed over multiple notations. Advantage: Notation 1 is more self-contained. It is also easier to plug-in new constructs like B without needing to change notation 1.

Pattern 4: Examples

- the feedsInformation/producesInformation links from BusinessFunction to InformationObject in



http://conceptbase.sourceforge.net/cross-notations.html

Note that the subclass ERD stands for an Entity-Relationship Diagram. Hence, information objects can be instantiated by ERDs.

- the cross-notational links in EKD/4EM (see next slide)

4EM: linking perspectives via subclasses

4EM (formerly also named EKD) is set of enterprise modeling notations that mostly focus on the business layer. Its particilar strength is the strong linkage between the perspectives.

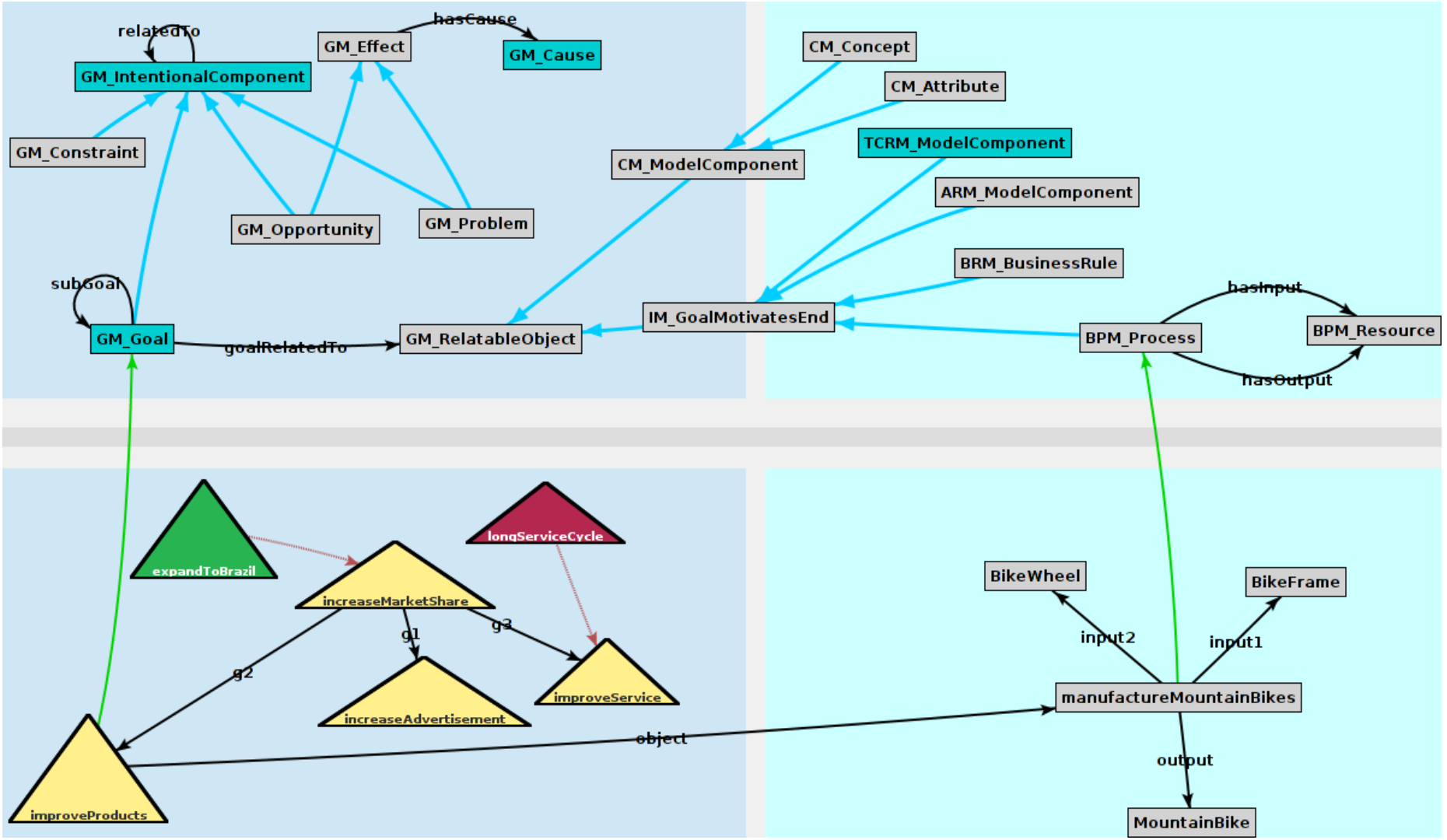
4EM Perspectives:

1) GM Model: Goals, Cause/Effect, Threats, Problems, ...
2) BPM Model: Processes, Resources, ...
3) ARM Model: Actors, Roles, ...
4) CM Model: Concepts, Attributes
5) TCRM Model: Technical components, IS components, requirements
6) BRM Model: Business rules

All perspectives are related. In particular GM and BRM can link to most other perspectives.

See also:
Sandkuhl, K., Stirna, J., Persson, A., Wißotzki, M. : Enterprise Modeling - Tackling Business Challenges with the 4EM Method. Springer, 2014.

4EM: demo its specification in ConceptBase



This is an example of pattern 4 (interface classes)

http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d3827551/goal-relations.gel
http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/3821866

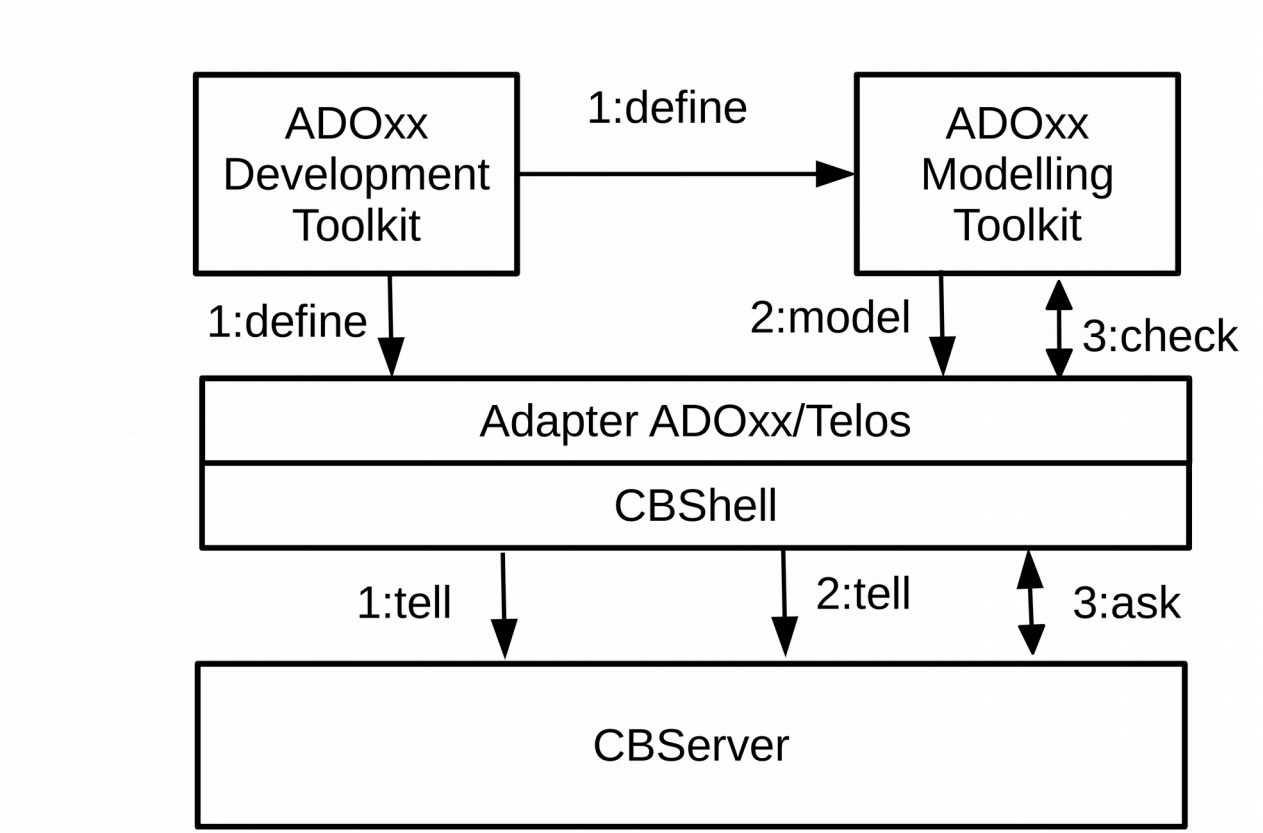
Analyze the 4EM specification

Since the 4EM specification in ConceptBase is just “data”, it can be queries. For example, to find out which constructs from other modeling notation are used in a given notation, e.g, to which foreign constructs does the goal modeling notation link to:

```
ImportedConstruct in GenericQueryClass isA NodeOrLink with
  computed_attribute,parameter
    notation : Notation
  constraint
    isOutside: $ exists n/NodeOrLink
                   (notation includes n) and (n link_sym this)
                   and not (notation includes this) $
end
```

any link

Integration with ADOxx



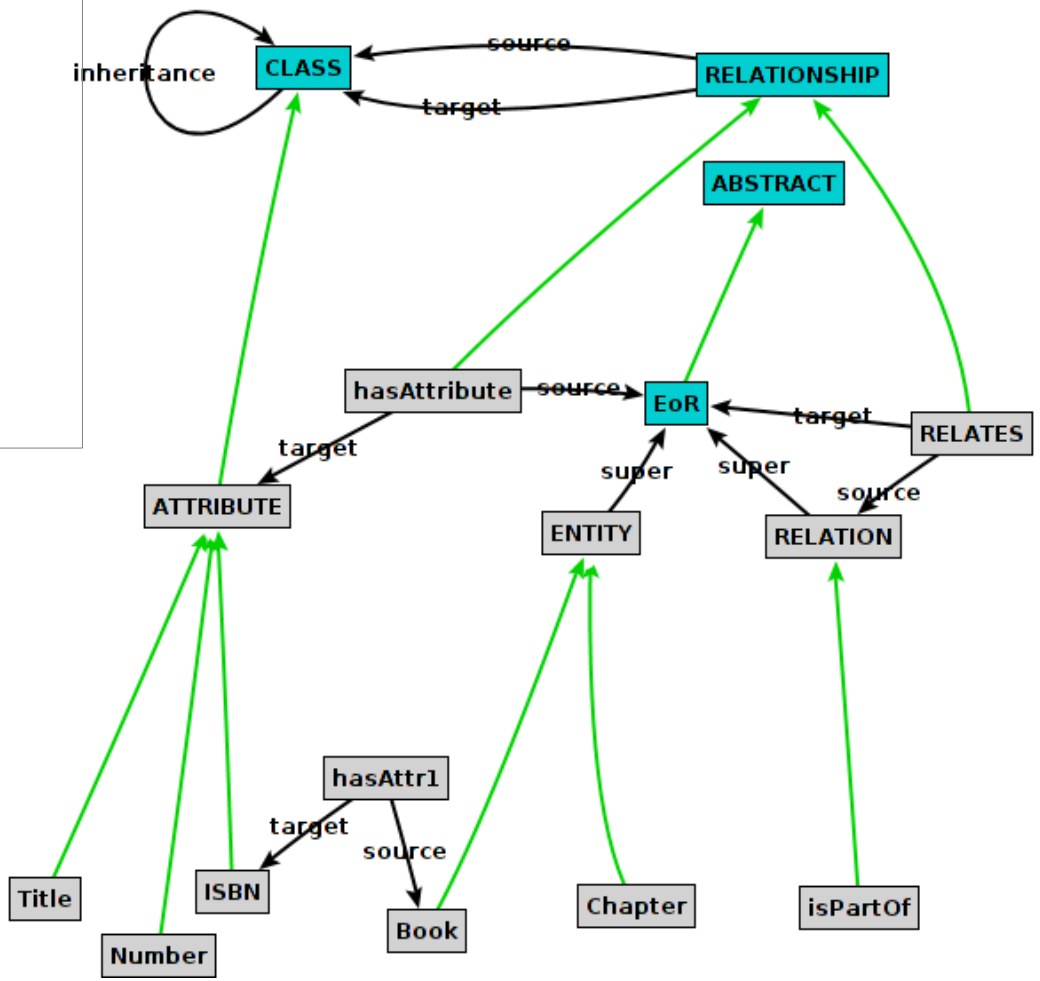
- ConceptBase used as back-end to ADOxx
- Integrity constraints can be defined for Meta^2 model (does not change), meta model (per modeling language e.g. ER, BPMN, ...), and even for domain models

Fixed Meta^2 Model of ADOxx translated to ConceptBase

```
CLASS with
  attribute,single,transitive,reflexive,antisymmetric
    inheritance: CLASS
end
RELATIONSHIP with
  attribute,single,necessary
    source: CLASS
end
RELATIONSHIP with
  attribute,single,necessary
    target: CLASS
end
InheritanceRule in Class with
  rule
     r1: $ forall C,D/CLASS x/Proposition
  (C inheritance D) and (x in C) ==> (x in D) $
end
```

Defined Meta Model of ER from ADOxx translated to ConceptBase

```
EoR in ABSTRACT,CLASS end
hasAttribute in RELATIONSHIP with
   source
     source: EoR
end
hasAttribute in RELATIONSHIP with
   target
     target: ATTRIBUTE
end
ATTRIBUTE in CLASS end
ENTITY in CLASS with
  inheritance
     super: EoR
end
RELATION in CLASS with
  inheritance
      super: EoR
end
RELATES in RELATIONSHIP with
  source
    source: RELATION
end
RELATES in RELATIONSHIP with
  target
    target: EoR
end
```



This meta model is created in step “1:define” by ADOxx Development Toolkit

More details:
http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/3869501

Thank You!

Herzlichen Dank!

Dank u wel!

Tack så mycket!

Manfred Jeusfeld
conceptbase.cc/mjf