Enterprise Ontology

driven

Software Generation

Jan L.G. Dietz

Outline

Model Driven Engineering

System Design (τ-theory)

Enterprise Ontology (ψ-theory)

The DEMO Processor

Conclusions

Outline

Model Driven Engineering

System Design (τ-theory)

Enterprise Ontology (ψ-theory)

The DEMO Processor

Conclusions

What is Model Driven Engineering?

Model-driven engineering (MDE) is a software development methodology which focuses on creating and exploiting domain models (that is abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts.

The MDE approach is meant to increase productivity by

- maximizing compatibility between systems (via reuse of standardized models)
- simplifying the process of design (via models of recurring design patterns in the application domain), and
- promoting communication between individuals and teams working on the system (via a standardization of the terminology and the best practices used in the application domain).

How must MDE be understood?

- Regardless the way in which you apply MDE, you have to cope with the intrinsic characteristics of *system design*.
- So, let us have a look at what system design is about, as understood in the *τ-theory*.
- To start with, let us recall the important and fundamental differences between the *function* perspective and the *construction* perspective on systems.

Outline

Model Driven Engineering

System Design (τ-theory)

Enterprise Ontology (ψ-theory)

The DEMO Processor

Conclusions

The τ-theory

τ (is pronounced as TAO): Technology - Architecture - Ontology

The τ -theory is rooted in systemics, ontology, and design theory.

It explains the process of *system design*.

It clarifies the notion of *technology*, *architecture* and *ontology*.

About construction (1)

The *construction* of a system is something *objective*. A system is its construction.

Because constructional models of systems show ‘openly’ their construction, they are called *white-box* models.

System ontology regards the, implementation independent, essence of a system’s construction.

Examples:
A DEMO model of an enterprise’s organization
A BPMN model of a work flow
A UML Object Diagram of a software system

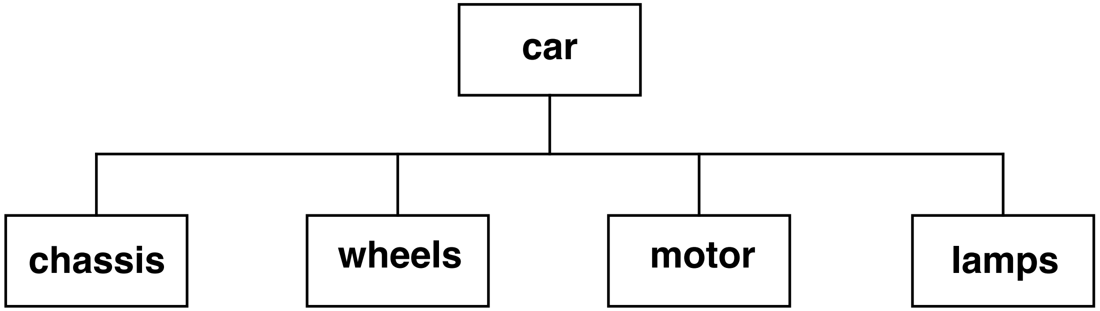
About construction (2)

the mechanic's perspective

construction :
the components and their interaction relationships

operation :
the manifestation of the construction in the course of time

constructional (de)composition



About function (1)

The *function* of a system is something *subjective*. It is not a system property but a relationship between a system and a stakeholder.

Function is in the eye of the beholder.

Because functional models of systems ‘hide’ their construction, they are called *black-box* models.

Examples:
An economic model of an enterprise’s business
An IDEF0 model of a work flow
A DFD of a software system

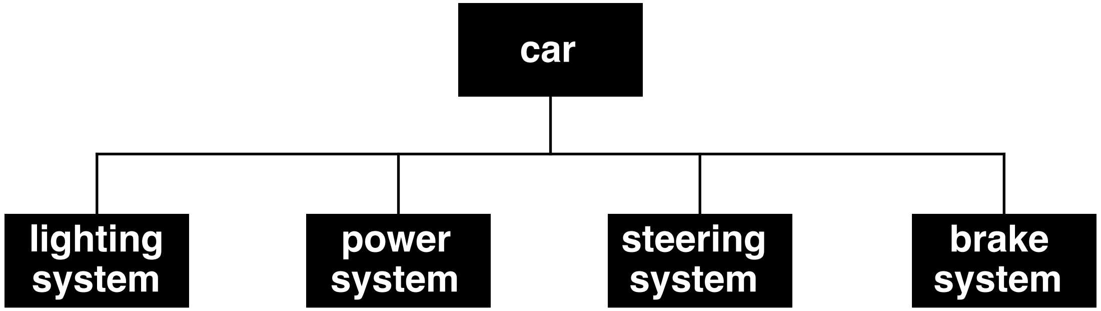
About function (2)

the driver's perspective

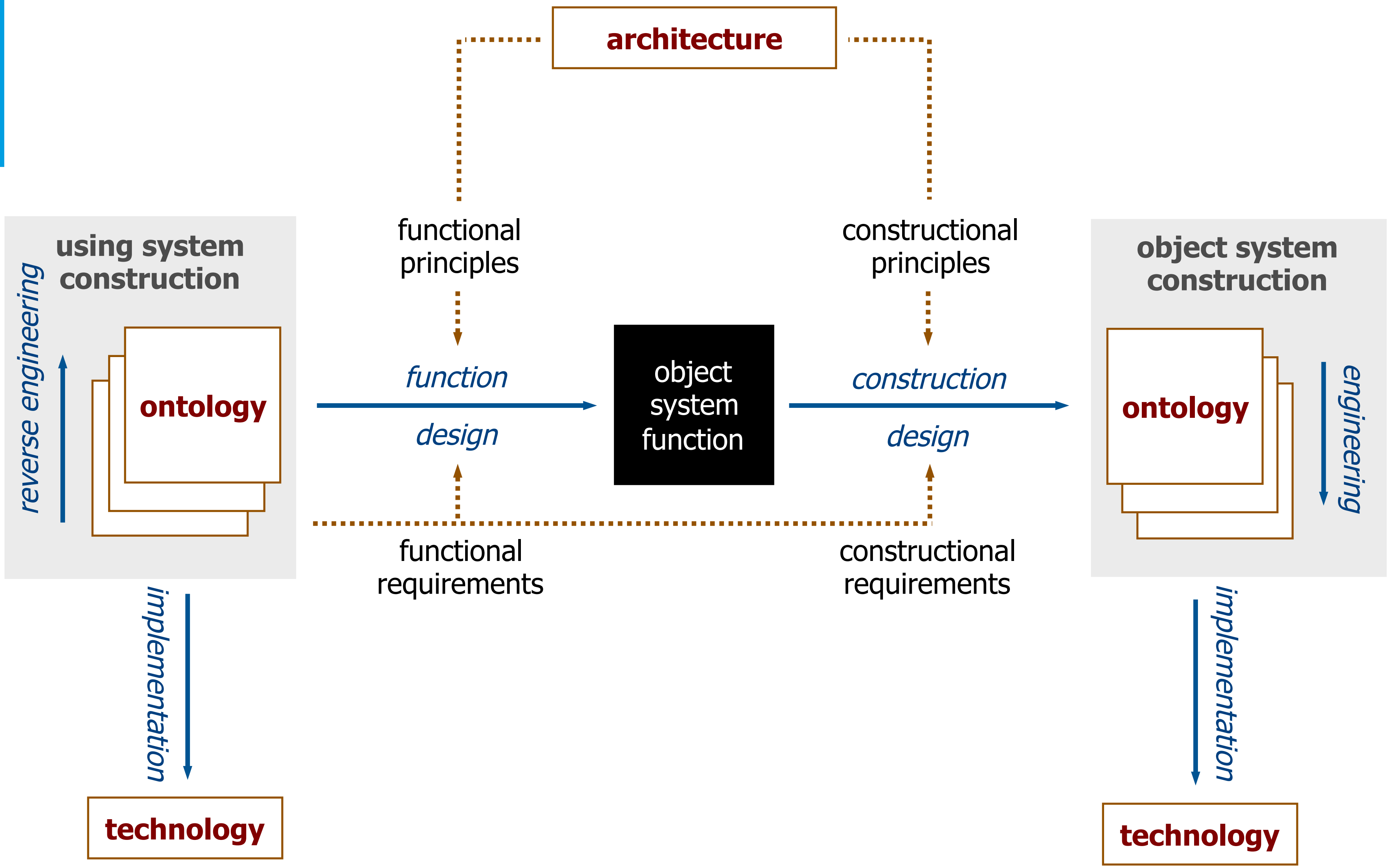
function :
relationship between
input and output

behavior :
the manifestation of the
function in the course of time

functional (de)composition



The Generic System Development Process



What goes wrong with MDE?

- MDE is unable to deliver *using system* models (domain models) from which correct functional requirements can be determined. Hence, it is impossible to *validate* these requirements objectively.

- The models produced during the system development process are not formally defined. Hence, it is impossible to *verify* these models, that is to check them against each other.

Validation

- Validation answers the question *"Will I build the right system?"*
- To answer the question, you have to check the given requirements with the 'real needs' of the users.
- Although it seems to be a good idea to have the users validate the system, it is not, because they do not know their 'real needs'.
- The only way out is to start requirements engineering from the *enterprise ontological model* of the using system (the domain model).

Verification

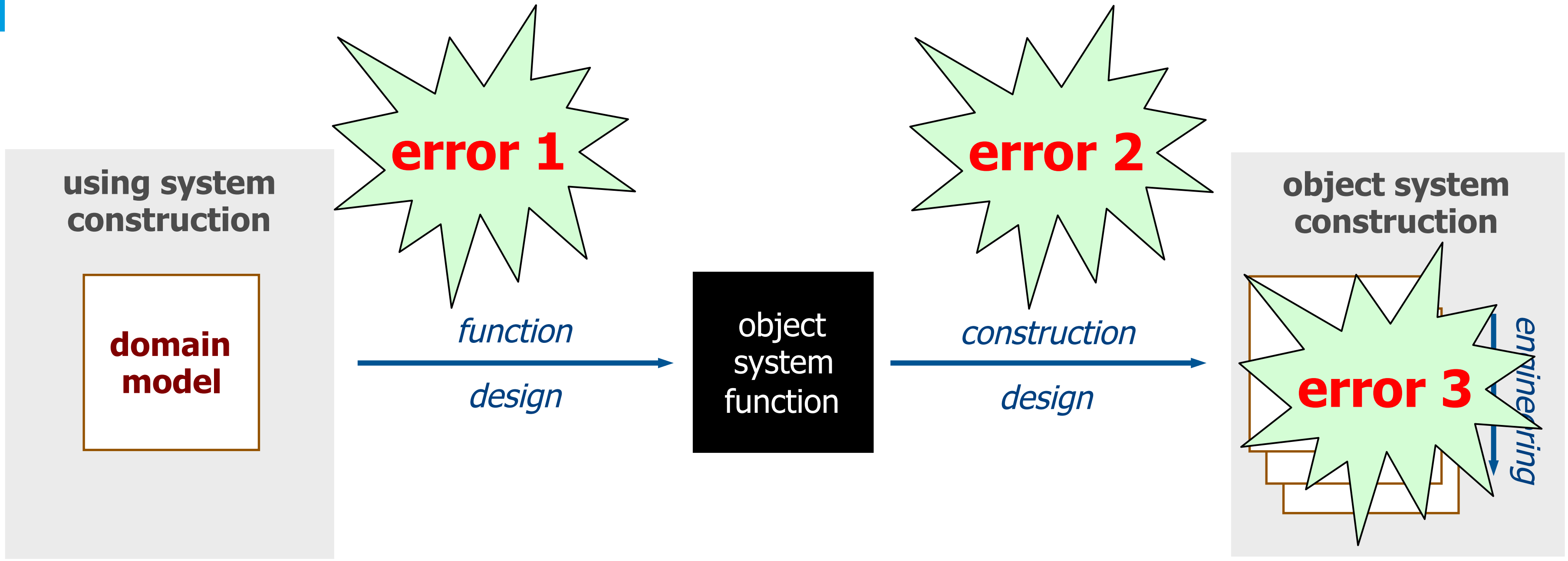
- Verification answers the question *"Did I build the system in the right way?"*

- To answer the question, you have to make sure that every model of the system is a correct 'successor' of the previous model, starting from the (ontological) domain model.

- This can only be achieved if the models are formally defined, which is mostly not the case.

- Moreover, functional models can, by nature, never be formalized.

The persistent errors of MDE



Outline

Model Driven Engineering

System Design (τ-theory)

Enterprise Ontology (ψ-theory)

The DEMO Processor

Conclusions

The ψ-theory

ψ (is pronounced as PSI): Performance in Social Interaction

The ψ -theory is rooted in semiotics, language philosophy, systemics, and social action theory.

It explains the *construction* and *operation* of organizations.

It defines the notion of *enterprise ontology*.

The ψ-theory (1)

- The operating principle of organizations is that *human beings* enter into and comply with *commitments* regarding the production of things. They do so in *communication*, and against a shared background of cultural norms and values.

- Commitments occur in processes that follow the *universal transaction pattern*. This is a structure of *coordination acts*, concerning one *production fact*, between two actors. One is the *initiator* (consumer) and the other one the *executor* (producer).

- An organization is a network of actors and transactions. Every actor has a particular *authority*, assigned on the basis of *competence*. Actors are assumed to exercise their authority with *responsibility*.

Examples of coordination acts

Alicia: *I'd like to have a bouquet of red tulips*

Alicia : request : Celestine : order 387 is fulfilled

Celestine: *Just a moment*

Celestine : promise : Alicia : order 387 is fulfilled

proposition

Celestine: *Here you are*

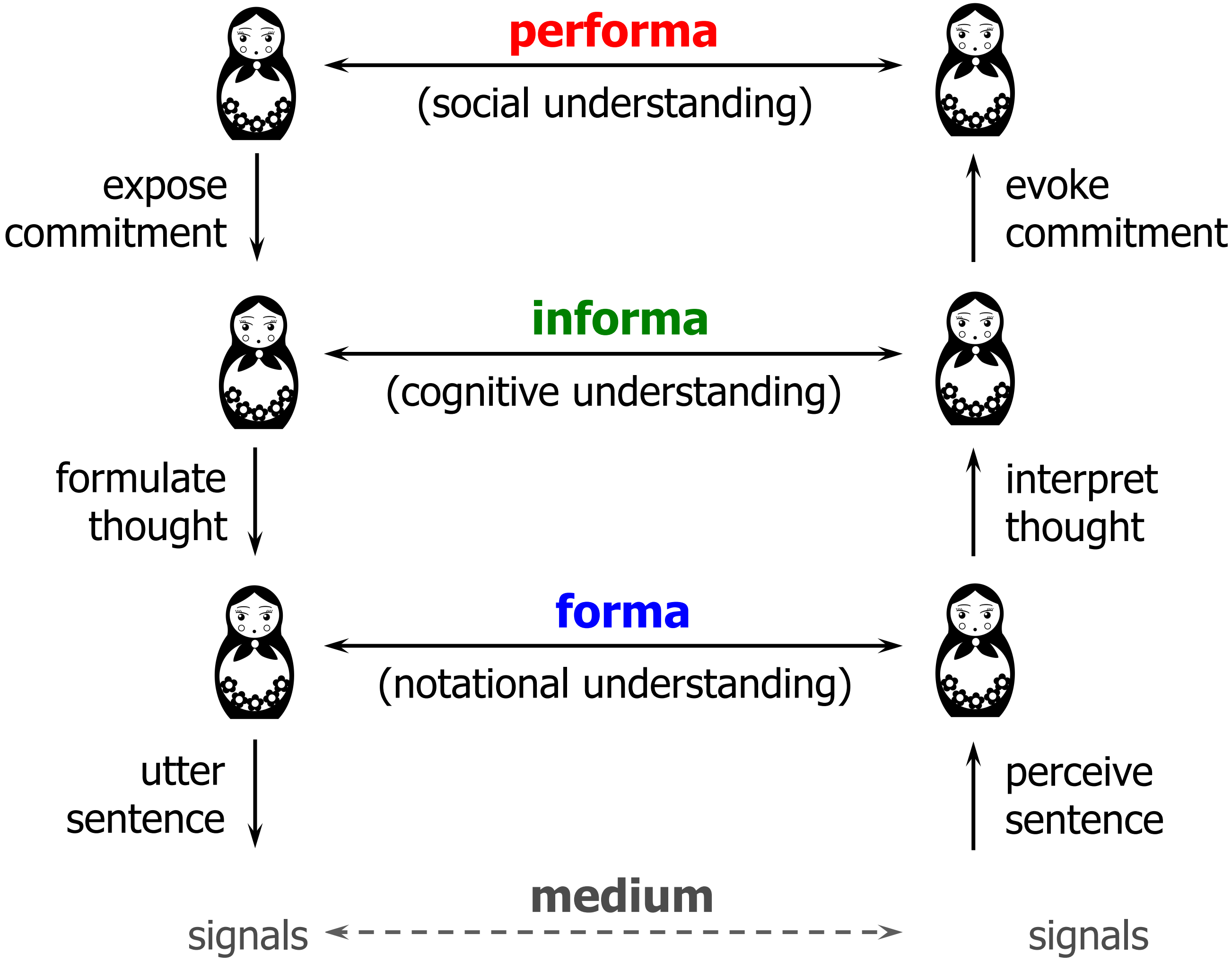
Celestine : state : Alicia : order 387 is fulfilled

Alicia: *Thanks*

Alicia : accept : Celestine : order 387 is fulfilled

result

The ψ-theory (2)



performa

(social understanding)

expose commitment

evoke commitment

informa

(cognitive understanding)

formulate thought

interpret thought

forma

(notational understanding)

utter sentence

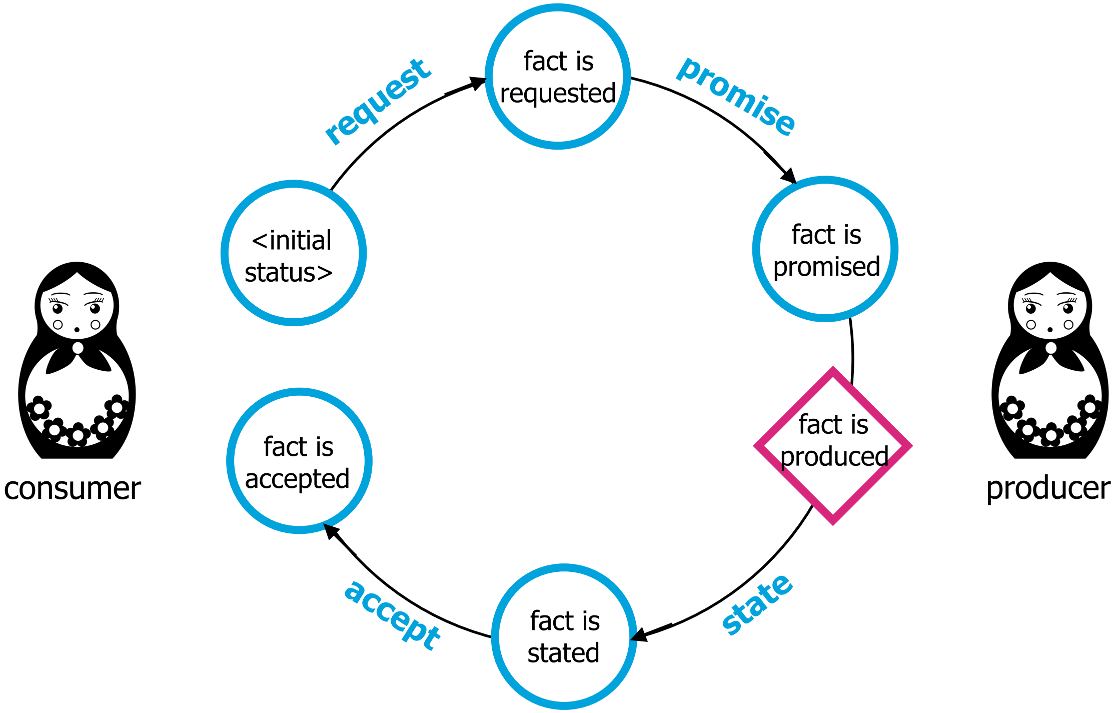
perceive sentence

medium

signals

signals

The basic transaction pattern

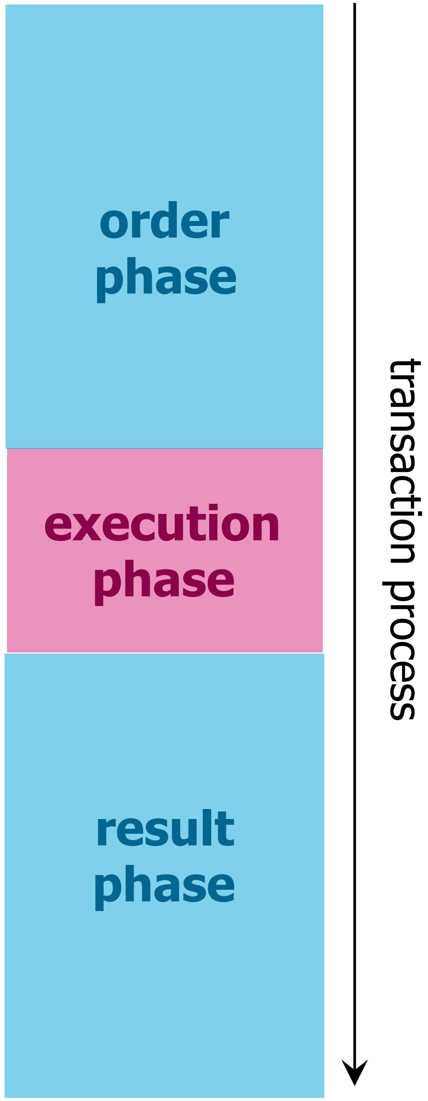


The transaction process

In the **order phase**, the actors discuss the ***fact** to be produced*, and try to come to agreement

In the **execution phase**, the executor *produces some **fact***

In the **result phase**, the actors discuss the ***fact** that has been produced*, and try to come to agreement



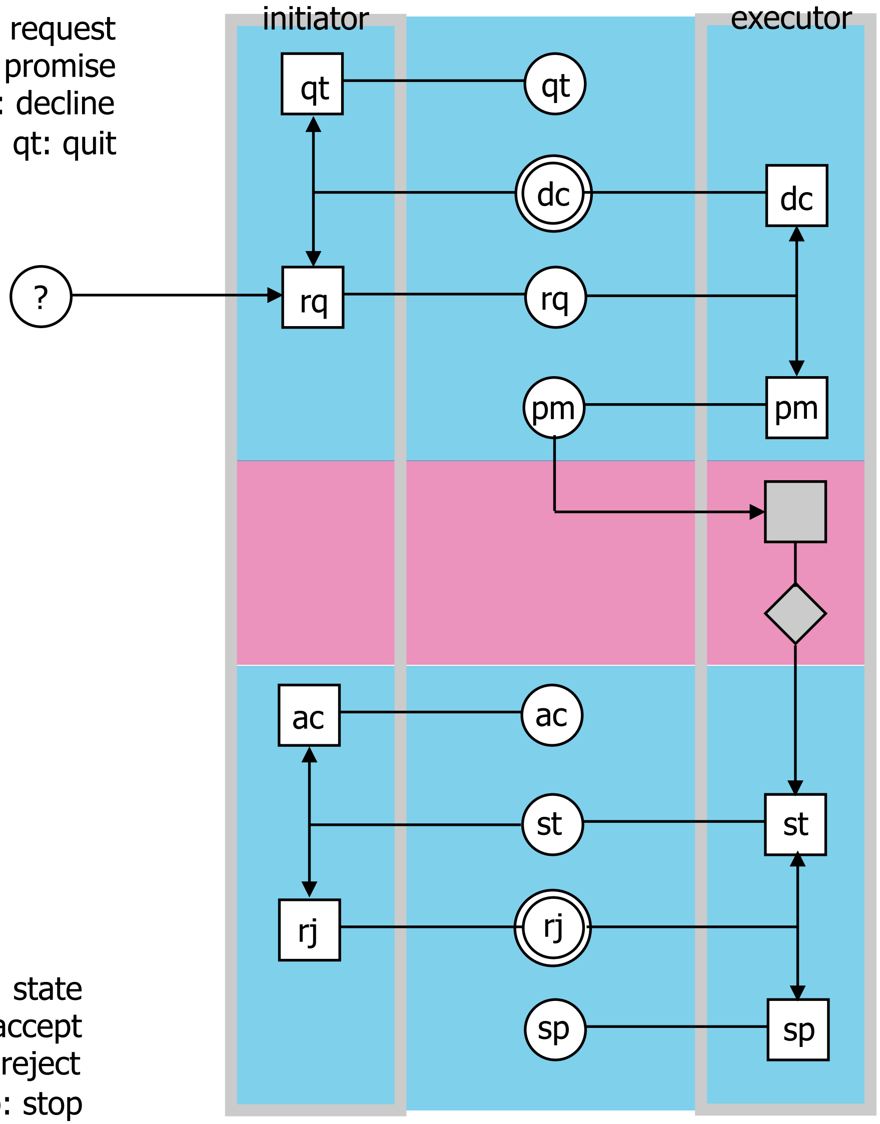
Asking for flowers
Booking a hotel room
Applying for membership
Booking a car rental

Creating
Deciding
Judging

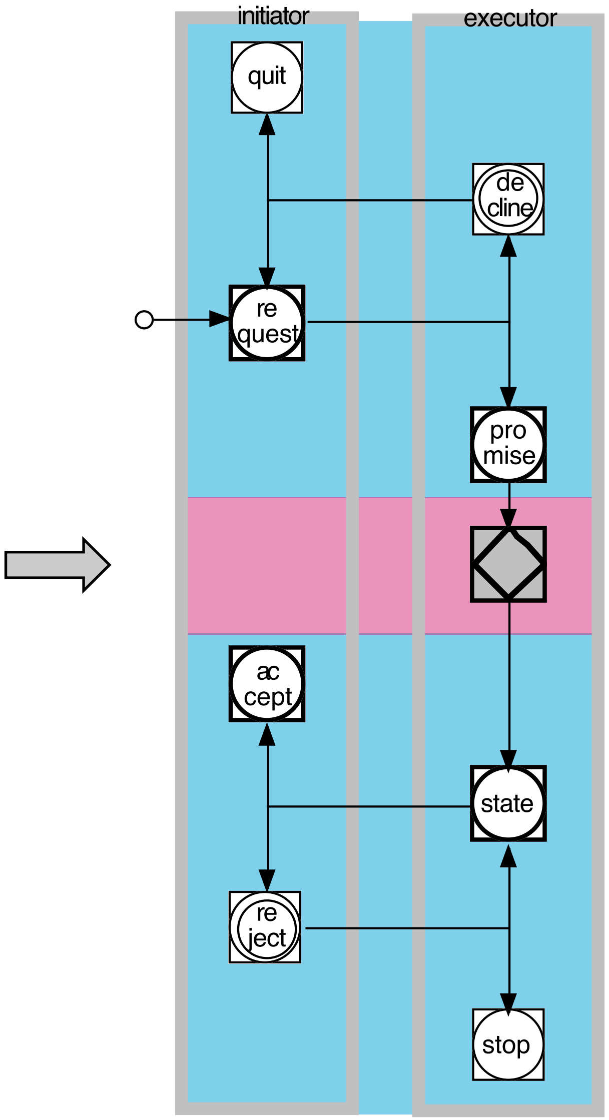
Receiving the flowers
Having stayed in he hotel
Having become member
Having rented a car

The standard transaction pattern

rq: request
pm: promise
dc: decline
qt: quit



st: state
ac: accept
rj: reject
sp: stop



Non-verbal and tacit communication

Alicia: *I'd like to have a bouquet of red tulips*

Alicia : request : Celestine : order 387 is fulfilled

Celestine: *Just a moment*

< tacit act >

Celestine : promise : Alicia : order 387 is fulfilled

proposition

Celestine: *Here you are*

< handing over the bouquet >

Celestine : state : Alicia : order 387 is fulfilled

Alicia: *Thanks*

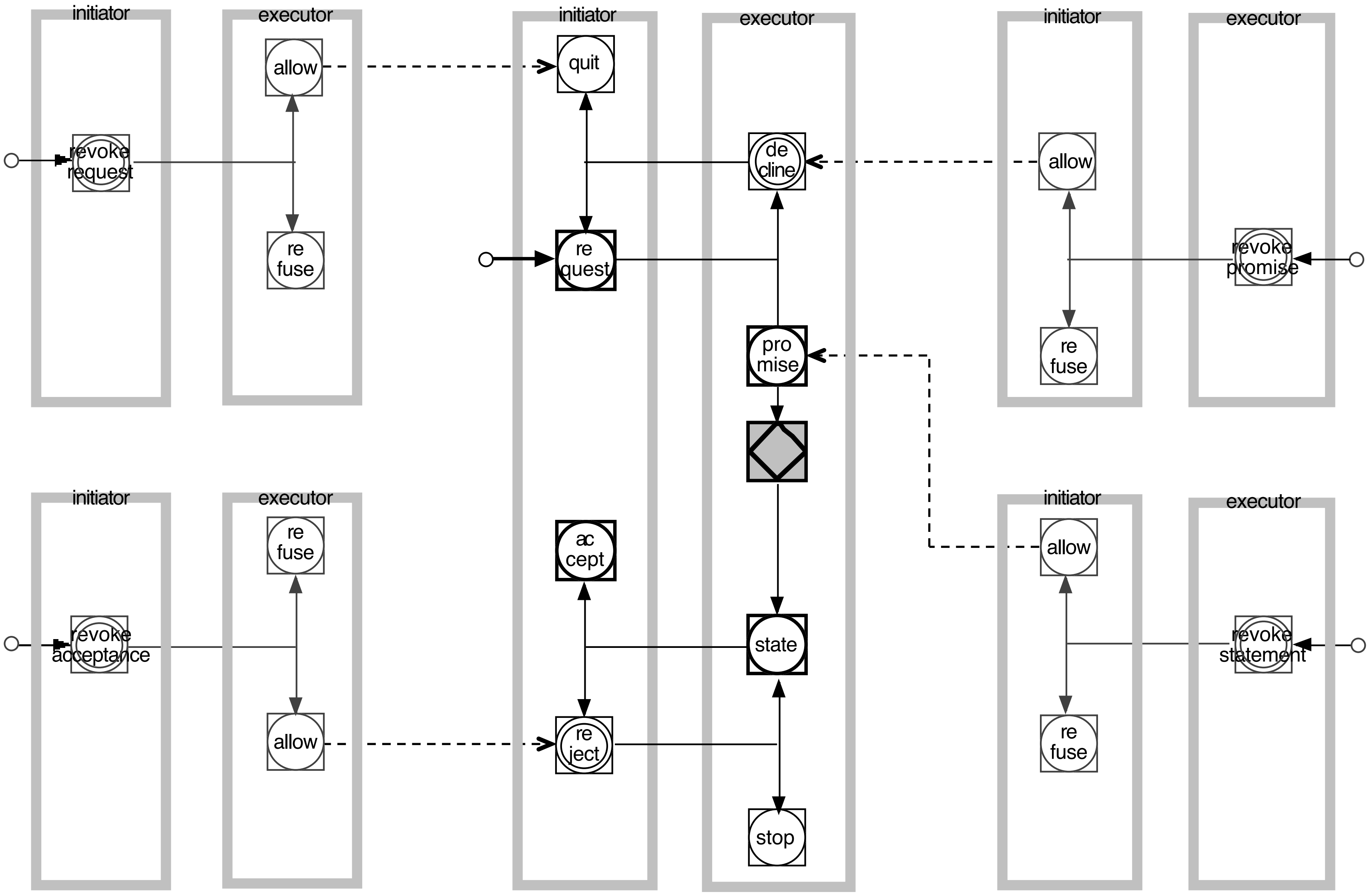
< tacit act >

Alicia : accept : Celestine : order 387 is fulfilled

result

The ψ-theory

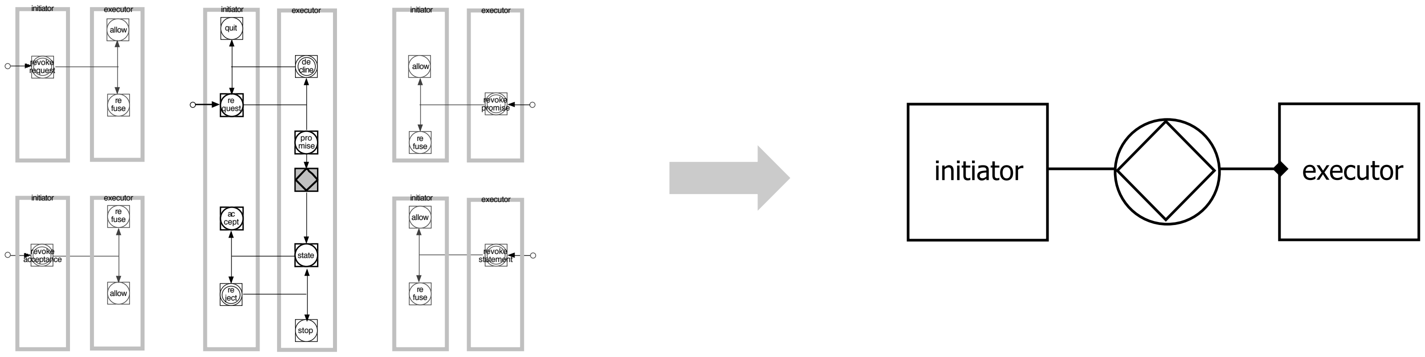
The universal transaction process



The ψ-theory

The building block of organizations

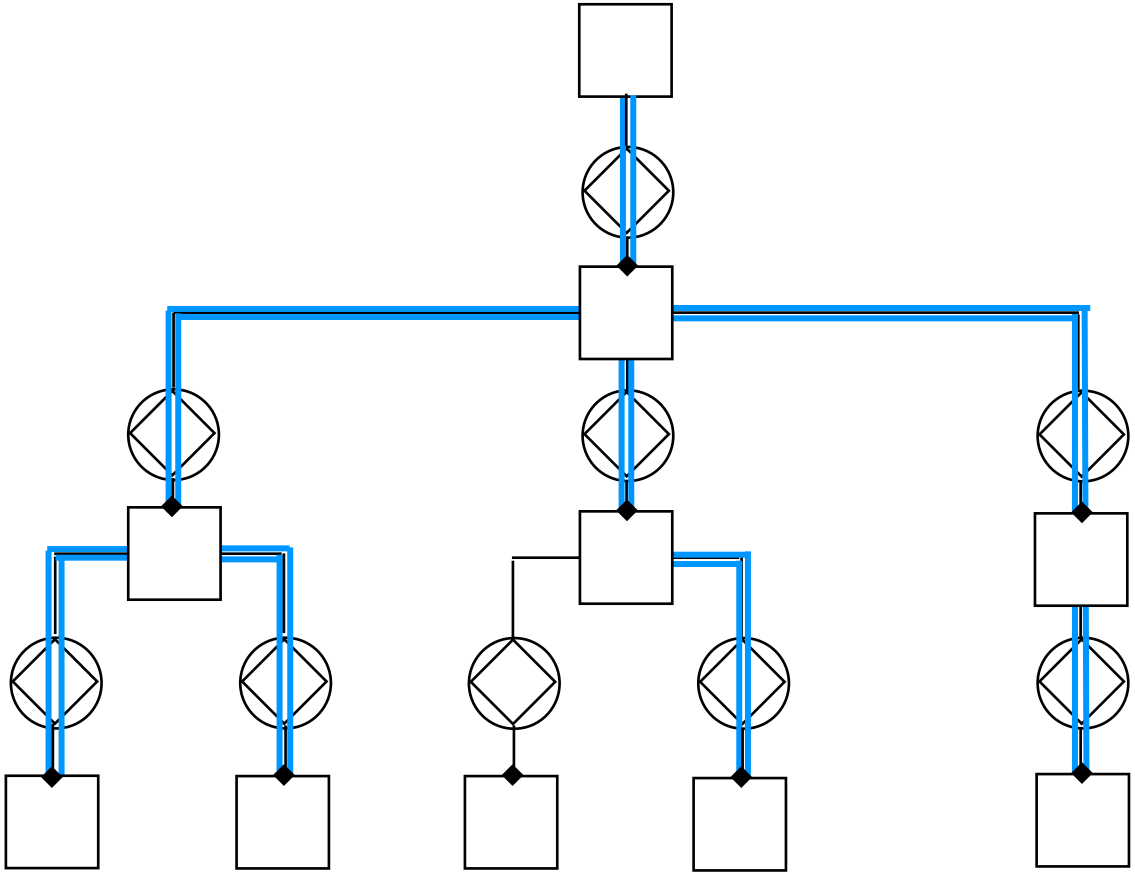
Every (elementary) actor role is the executor of exactly one transaction kind, and initiator of 0, 1 or more transaction kinds.



Next to the *process* interpretation of the transaction symbol, there is the *state* interpretation:

it represents a *production bank* (containing production facts) and a *coordination bank* (containing coordination facts)

A business process is a tree of transactions



Note. Component transactions may also be carried out in parallel

The ψ-theory (3)

The three human *abilities* also apply to *production*:

Performa

The ability to perform *original* production acts, such as to **create** (**manufacture**, **transport**, **observe**)**,** **decide**, and **judge**.

Informa

The ability to perform *informational* production acts, such as to **remember**, **recall**, and **compute**.

Forma

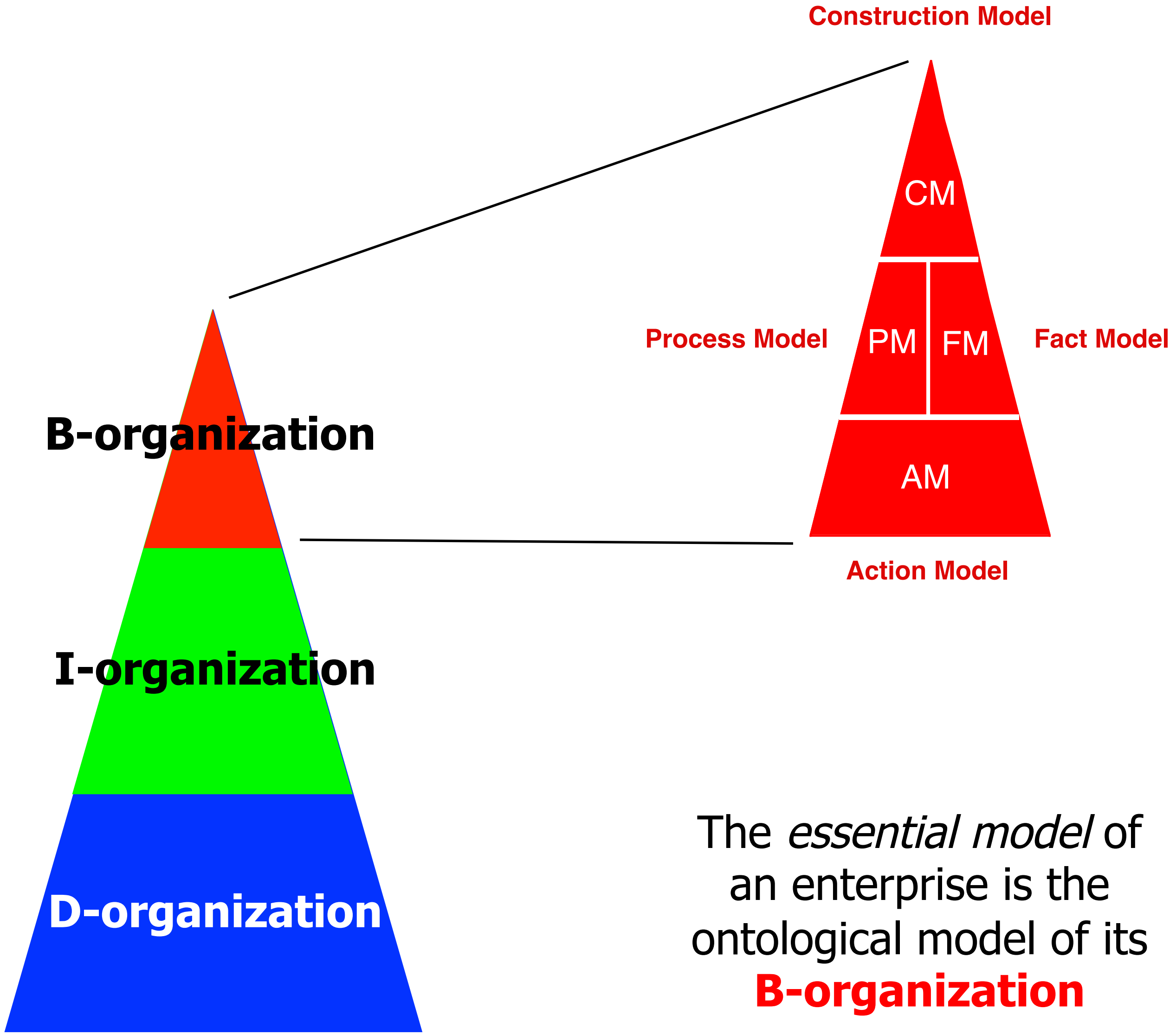
The ability to perform *documental* production acts, such as to **store**, **retrieve**, **transmit**, and **copy** sentences and documents.

The essential model (1)

creating
deciding
judging

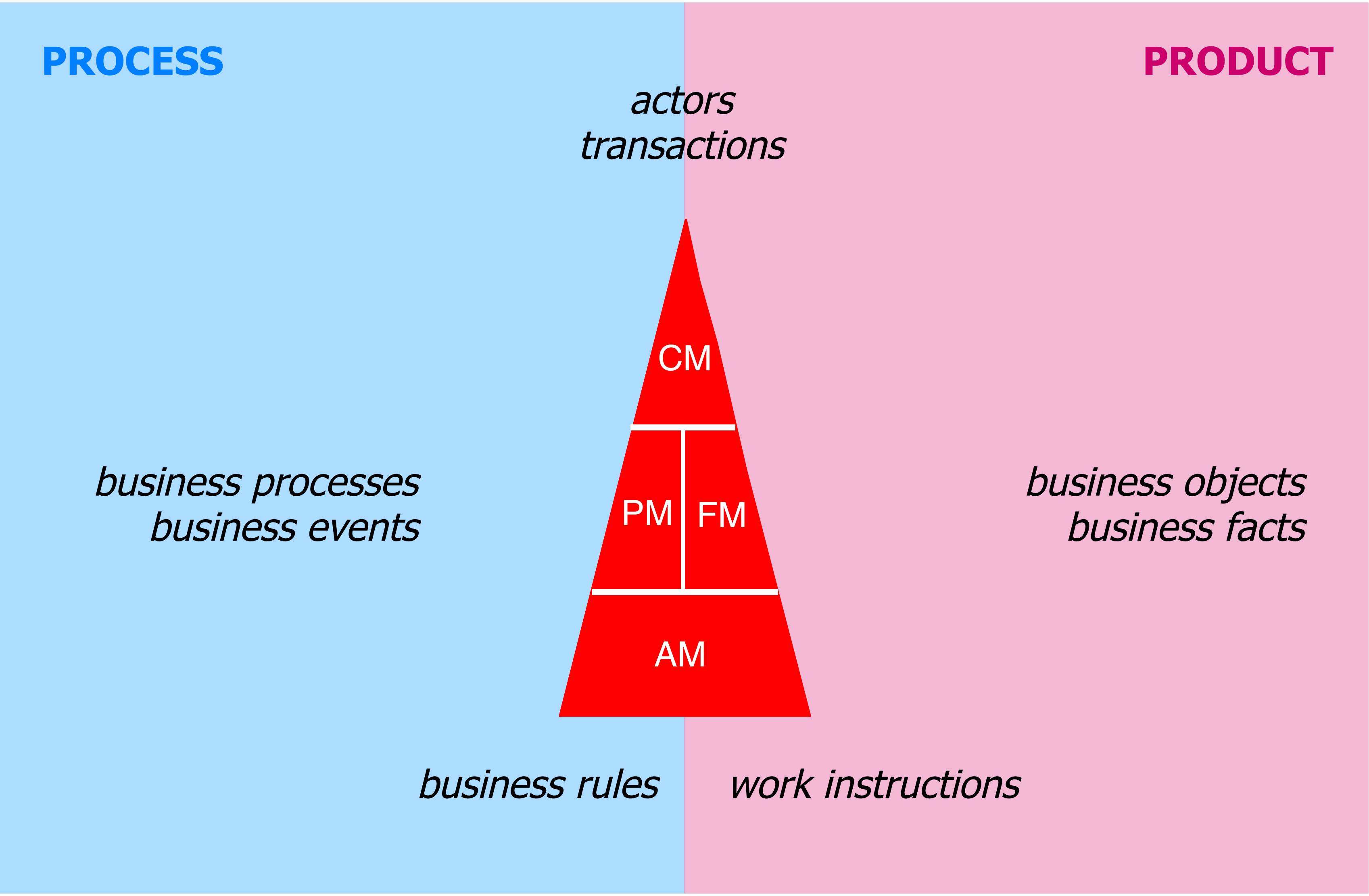
remembering
recalling
computing

storing
retrieving
transmitting
copying



The *essential model* of an enterprise is the ontological model of its **B-organization**

The essential model (2)



PROCESS

PRODUCT

actors
transactions

business processes
business events

business objects
business facts

business rules

work instructions

Outline

Model Driven Engineering

System Design (τ-theory)

Enterprise Ontology (ψ-theory)

The DEMO Processor

Conclusions

DEMO: Design and Engineering Methodology for Organizations

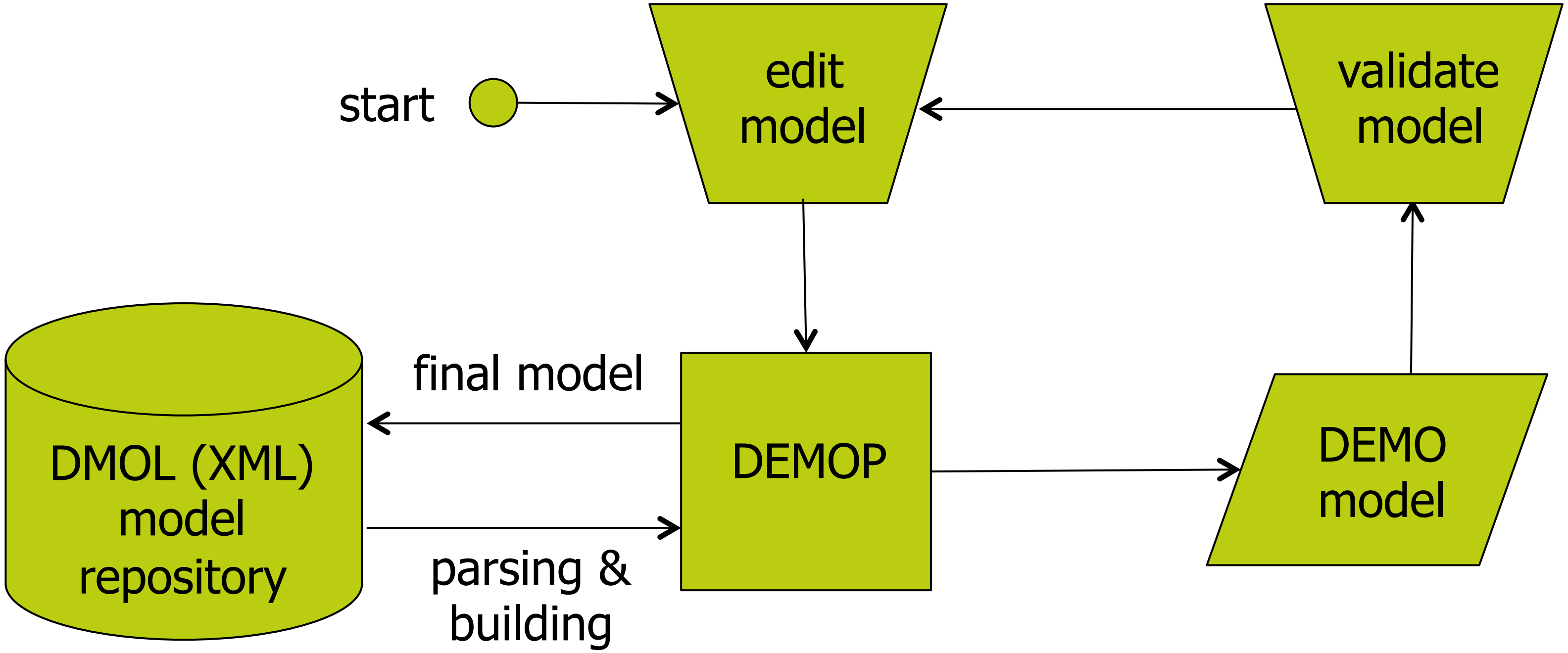
DEMO is the pioneering methodology of Enterprise Engineering.

Enterprise Engineering is the emerging discipline that addresses changes (of all kinds) in enterprises in an integrated way.

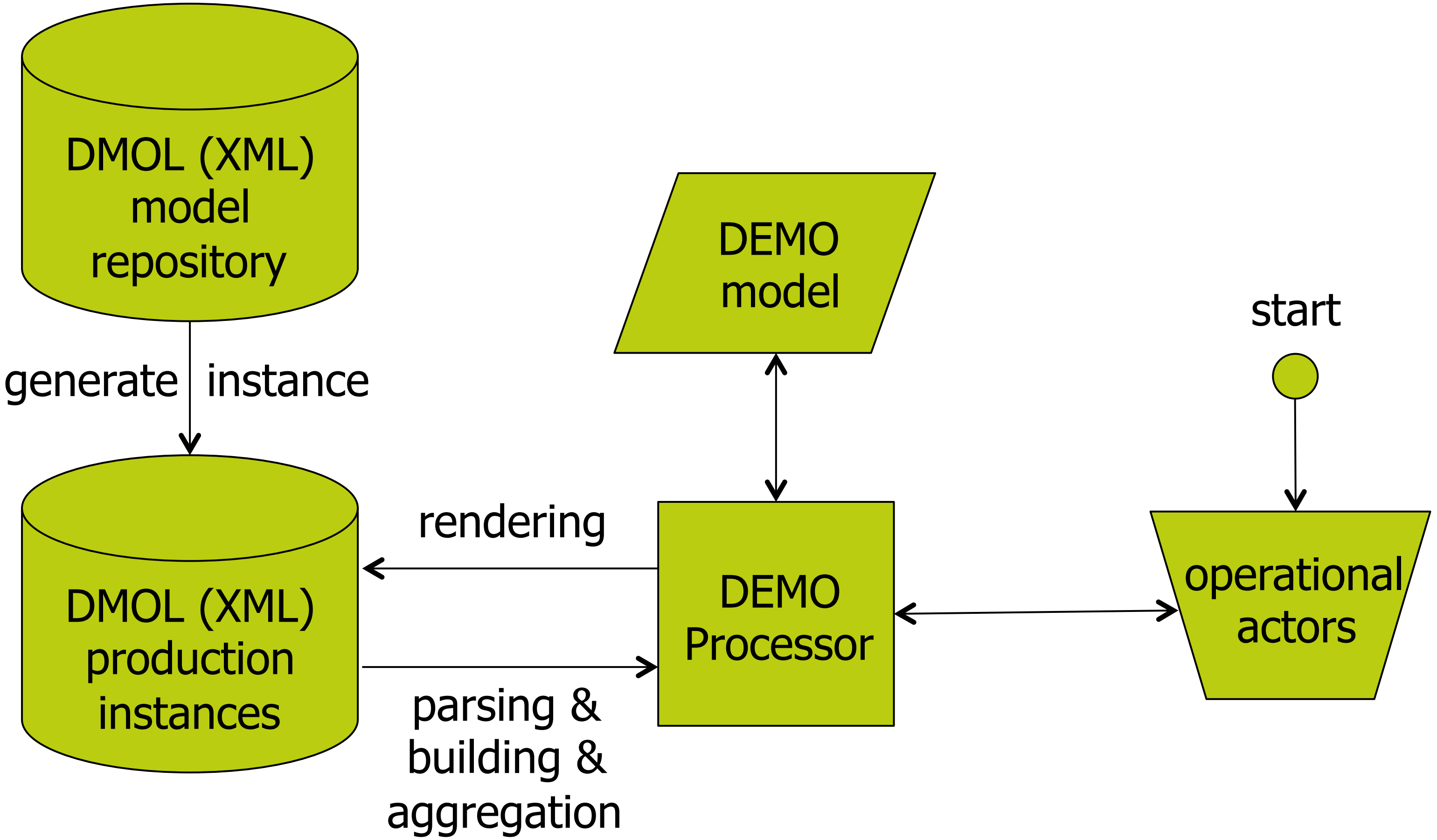
The *paradigm* of Enterprise Engineering is that enterprises are *designed systems*, and thus can be re-designed and re-engineered in order to bring about changes as and when needed.

Every Enterprise Information System is *some* implementation of the essential model (DEMO model) of *some* enterprise.

DEMOP – modeling mode



DEMOP – production mode


DMOL (XML) model repository
generate instance
DMOL (XML) production instances
DEMO model
rendering
parsing & building & aggregation
DEMO Processor
start
operational actors

Outline

Model Driven Engineering

System Design (τ-theory)

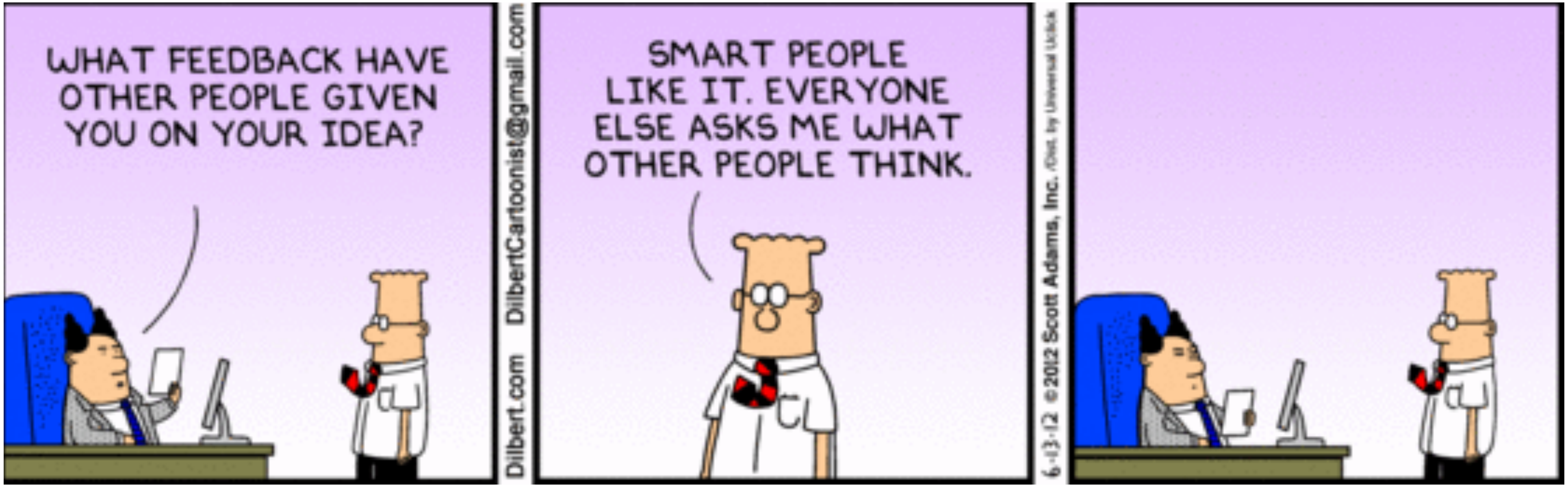
Enterprise Ontology (ψ-theory)

The DEMO Processor

Conclusions

Conclusions

- Current approaches to MDE are quite error prone.
- Because of its being fully rooted in the ψ-theory, DEMO delivers *coherent*, *consistent* and *comprehensive* ‘domain models’.
- DEMOP eliminates three crucial kinds of design errors:
  - Function design errors
  - Construction design errors
  - Implementation design errors
- DEMOP shows what the next generation ‘ERP’ might be.


WHAT FEEDBACK HAVE OTHER PEOPLE GIVEN YOU ON YOUR IDEA?
SMART PEOPLE LIKE IT. EVERYONE ELSE ASKS ME WHAT OTHER PEOPLE THINK.
Dilbert.com DilbertCartoonist@gmail.com
6-13-12 ©2012 Scott Adams, Inc. /Dist. by Universal Uclick

j.l.g.dietz@tudelft.nl

steefk22@telenet.be

www.ciaonetwork.org

www.ee-institute.com