# Enterprise Ontology driven Software Generation

Jan L.G. Dietz

# Outline

Model Driven Engineering

System Design ($\tau$-theory)

Enterprise Ontology ($\psi$-theory)

DEMOP

Conclusions

# Outline

## Model Driven Engineering

System Design ($\tau$-theory)

Enterprise Ontology ($\psi$-theory)

DEMOP

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 conceived in the $\tau$-*theory*.
- To start with, let us recognize the important and fundamental differences between the *function* perspective and the *construction* perspective on systems.

# Outline

Model Driven Engineering

**System Design ($\tau$-theory)**

Enterprise Ontology ($\psi$-theory)

DEMOP

Conclusions

# The τ-theory

The Greek letter τ is pronounced as TAO, standing for Technology, Architecture, and Ontology.

The τ-theory is a theory about the design of systems (of any kind). It explains the design process and it clarifies the notions of *technology*, *architecture* and *ontology* in the design process.

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

An important notion in the τ-theory is the notion of *system category* (physical, biological, social, etc.), which is determined by the nature of the elements and their interaction.
Only systems in the same category can interoperate.

# About construction (1)

The *construction* of a system is an *objective* notion. Therefore, one can rightly say that a system is its construction.

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

The *ontological* model of a system is the construction model that is fully independent of the system's implementation. Therefore, it is said to constitute the *essence* of the system.

*Examples of white-box models:*
The DEMO Process Model of a business process
A BPMN model of a workflow
A UML Activity Diagram of a software system

# About construction (2)

*the mechanic's perspective*

**construction** :
the components
and their interactions

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

***constructional (de)composition***

**car**

# About function (1)

The *function* of a system is a *subjective* notion. Function is not a system property but a relationship between a system and a stakeholder. Consequently, a system may 'have' as many functions as there are stakeholders.

Note: regarding artefacts, one may speak of their intended function (intended by the designer).

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

*Examples of black-box models:*
A value-based model of a business process
An IDEF0 model of a workflow
A DFD of a software system

# About function (2)

*the driver's perspective*

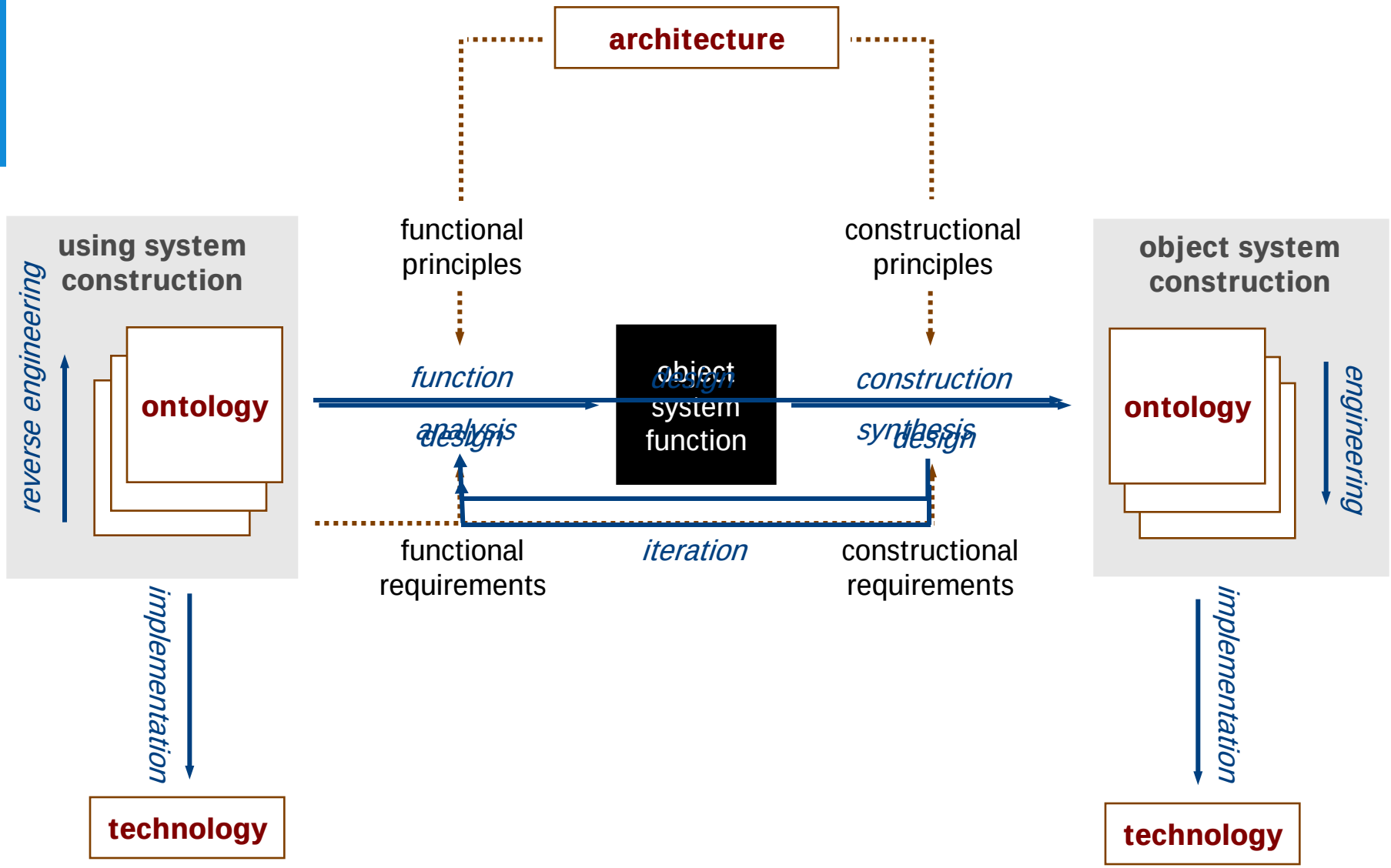
**function** :
relationship between
input and output

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

***functional (de)composition***

**car**

# The Generic System Development Process

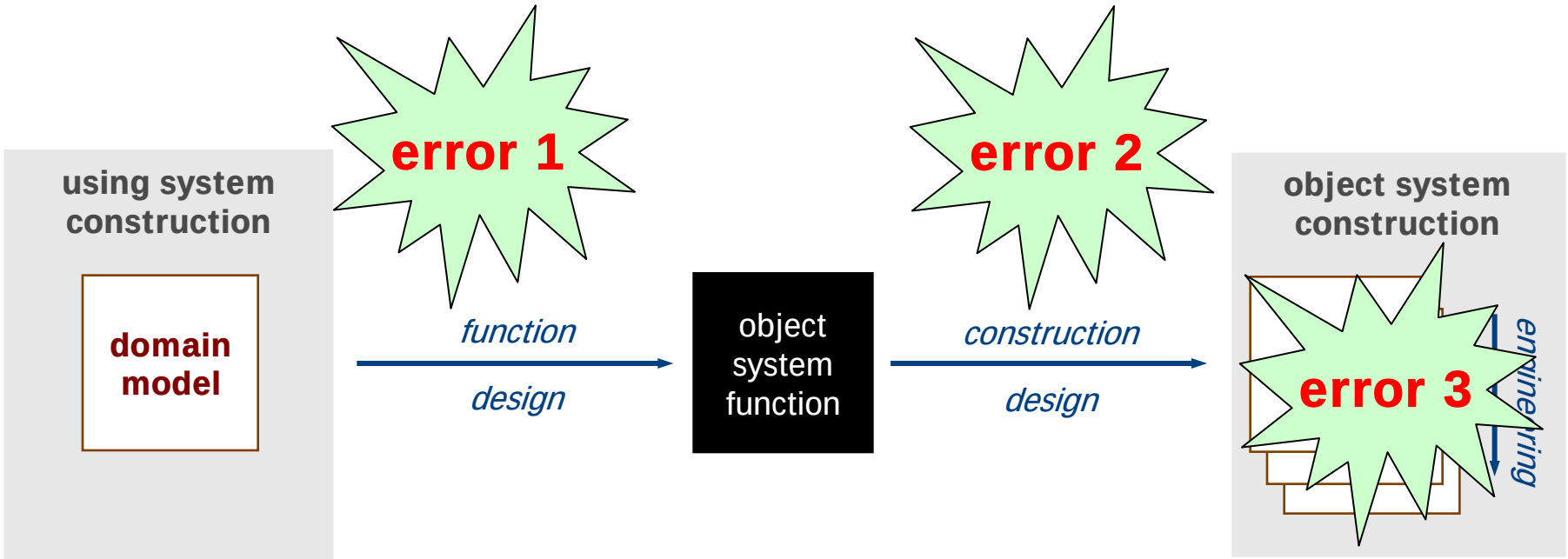


# The persistent errors in software engineering

Errors of type 1: missing or irrelevant requirements in the OS function

Errors of type 2: missing or irrelevant functions, as brought about by the OS construction

Errors of type 3: inconsistency between constructional models



# Strengths and weaknesses of MDE

- STRENGHTS
  - It offers the possibility to generate software from high-level models.

- WEAKNESSES
  - The models produced during the system design process are not formally defined. Hence, it is impossible to *verify* them, that is to check them against each other.
  - Because of the *using system* models (domain models) are not truly ontological, it is impossible to *validate* the requirements, nor the resulting system comprehensively.

- IMPROVEMENTS
  - The weaknesses can be removed to a large extent by putting MDE in the framework of the τ-theory.

# Outline

Model Driven Engineering

System Design ($\tau$-theory)

**Enterprise Ontology ($\psi$-theory)**

DEMOP

Conclusions

# The ψ-theory

The Greek letter ψ is pronounced as PSI, standing for Performance in Social Interaction

The ψ -theory is a theory about the construction and operation of organizations. It explains the *operating principle* of organizations and it defines the notion of *enterprise ontology*.

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

# 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 process*. This is a structure of *coordination acts*, concerning one *production fact*, between two actors. One is the *initiator* (consumer); the other one is 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*

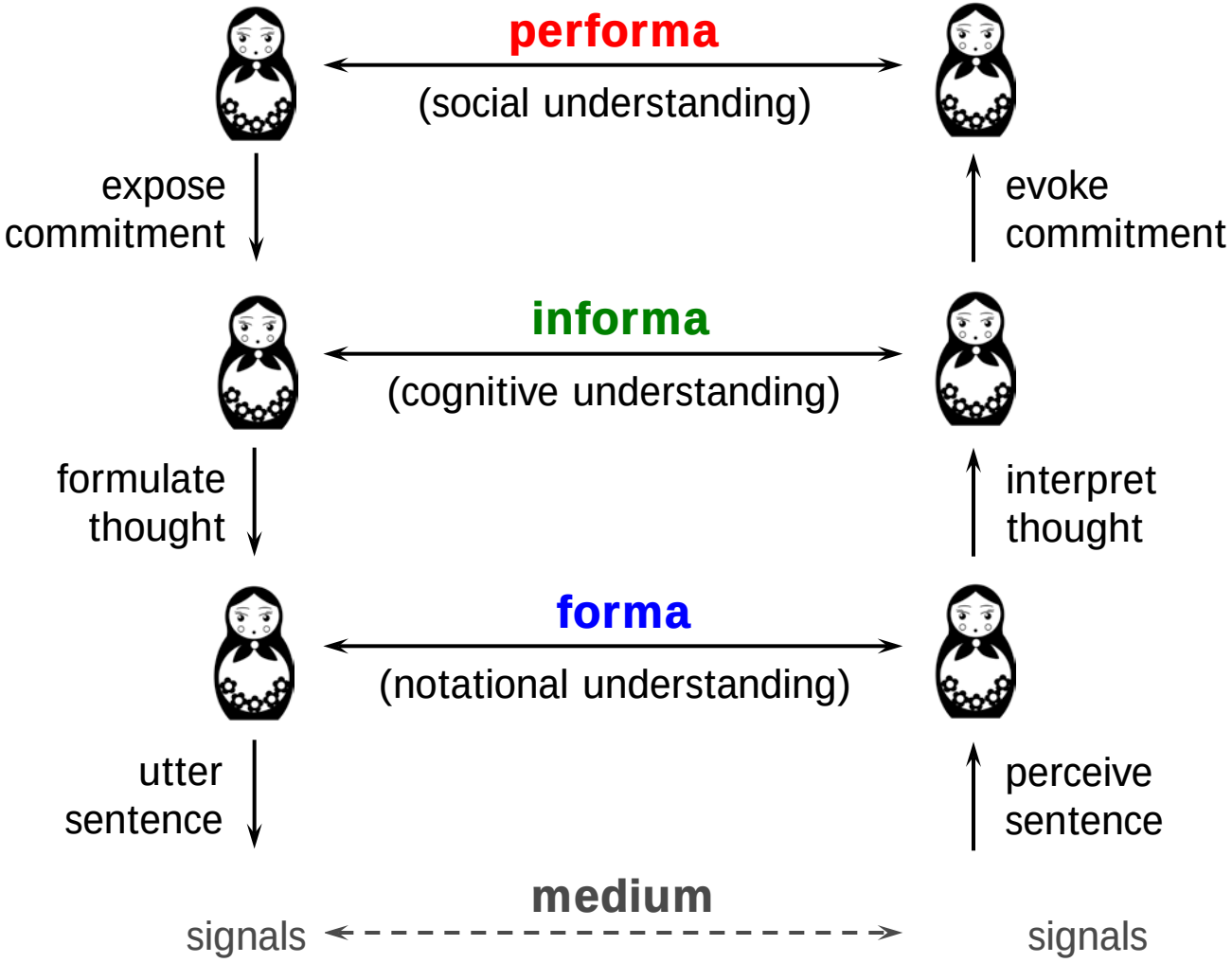
**Celestine : state : Alicia : order 387 is fulfilled**

Alicia: *Thanks*

**Alicia : accept : Celestine : order 387 is fulfilled**

result

# The ψ-theory (2)

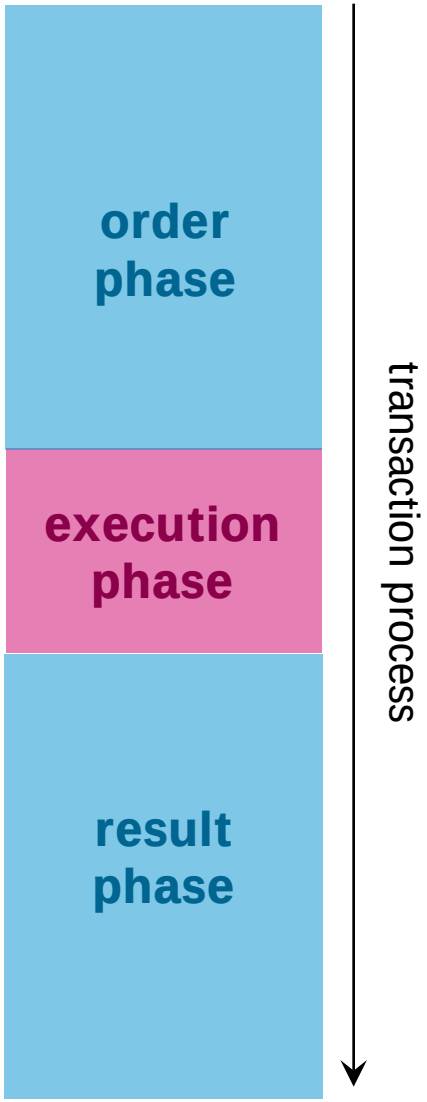


# The transaction process

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

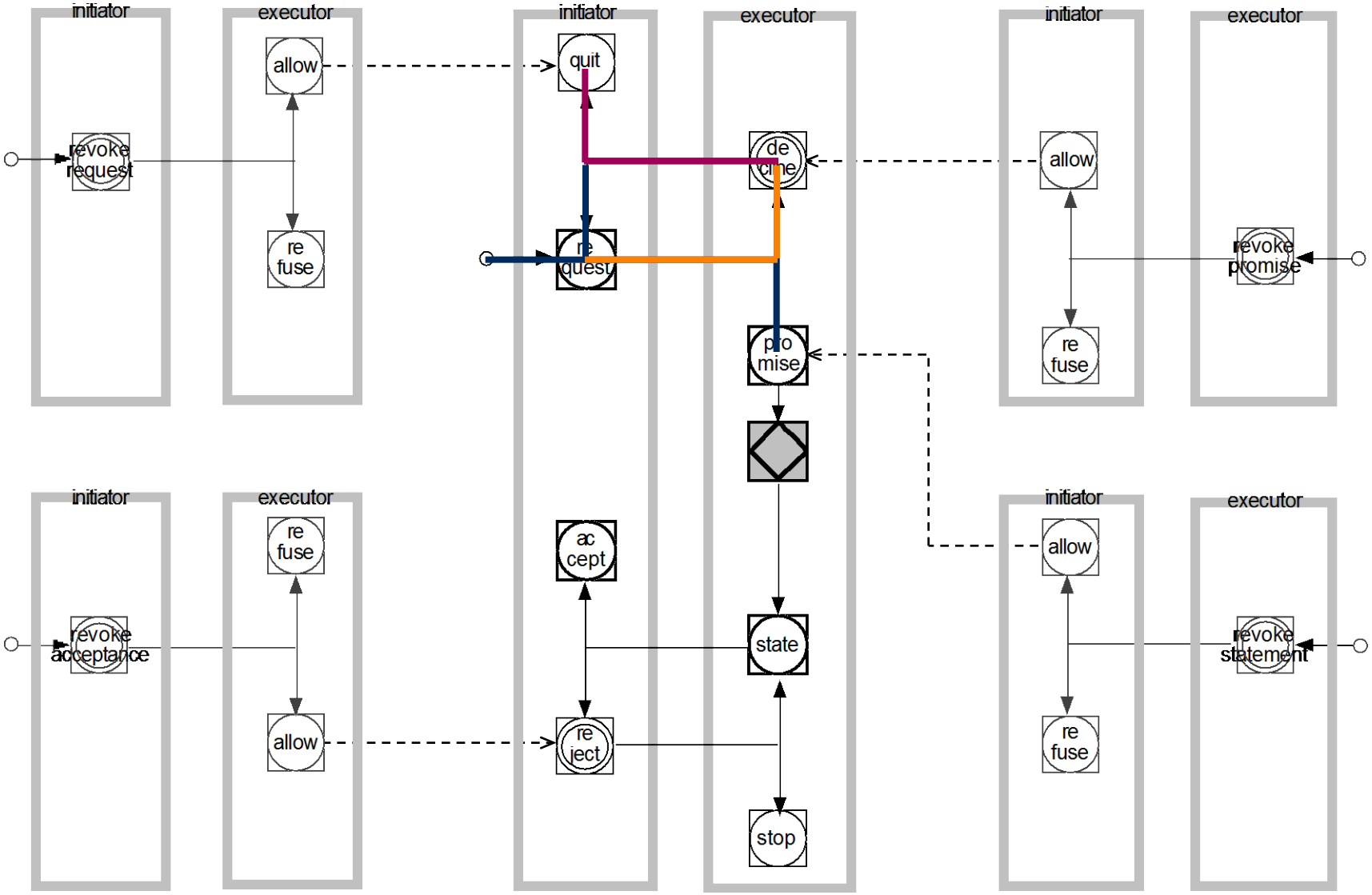



Asking for flowers
Booking a hotel room
Applying for membership
Booking a car rental

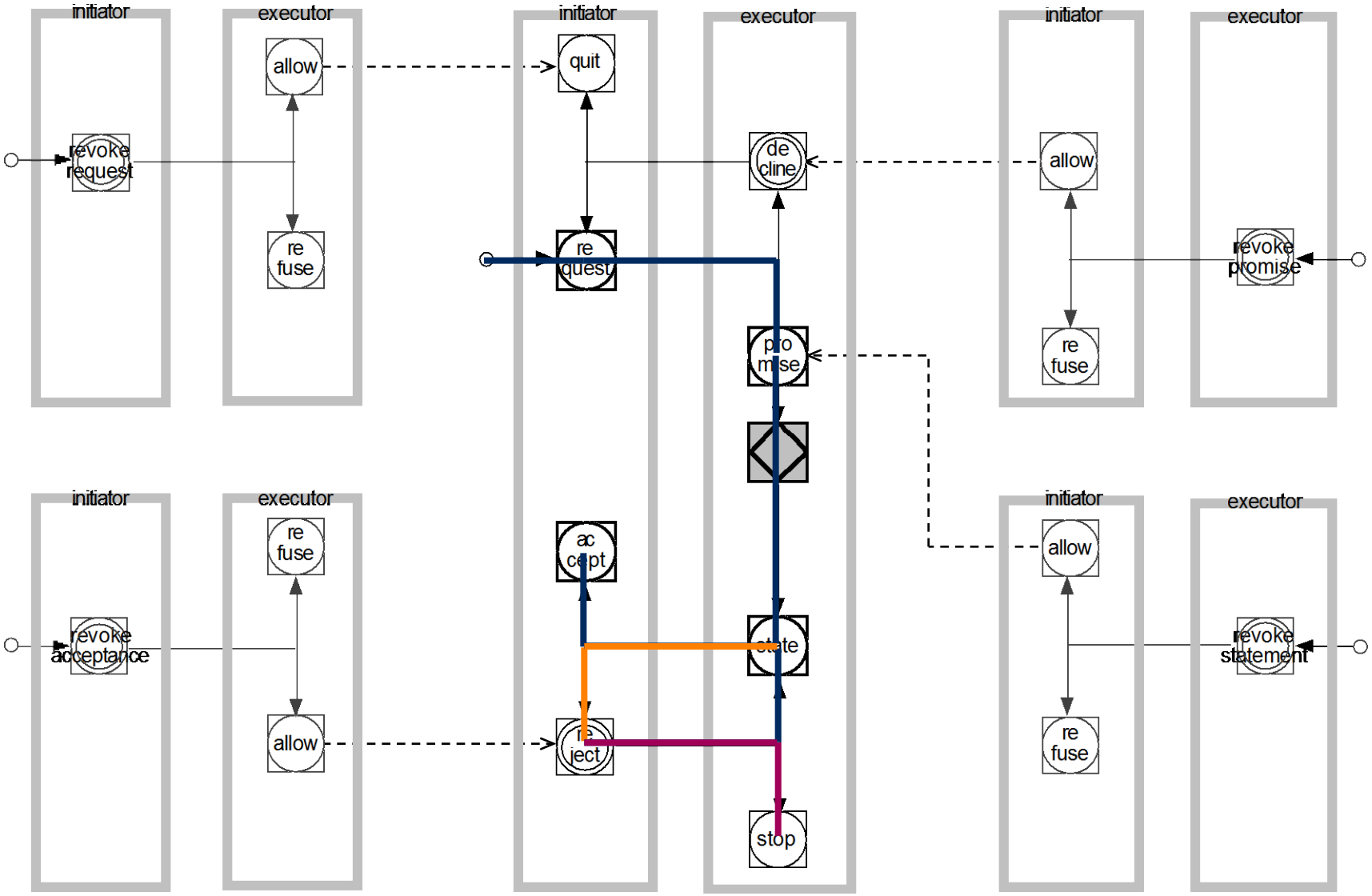
Creating
Deciding
Judging

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

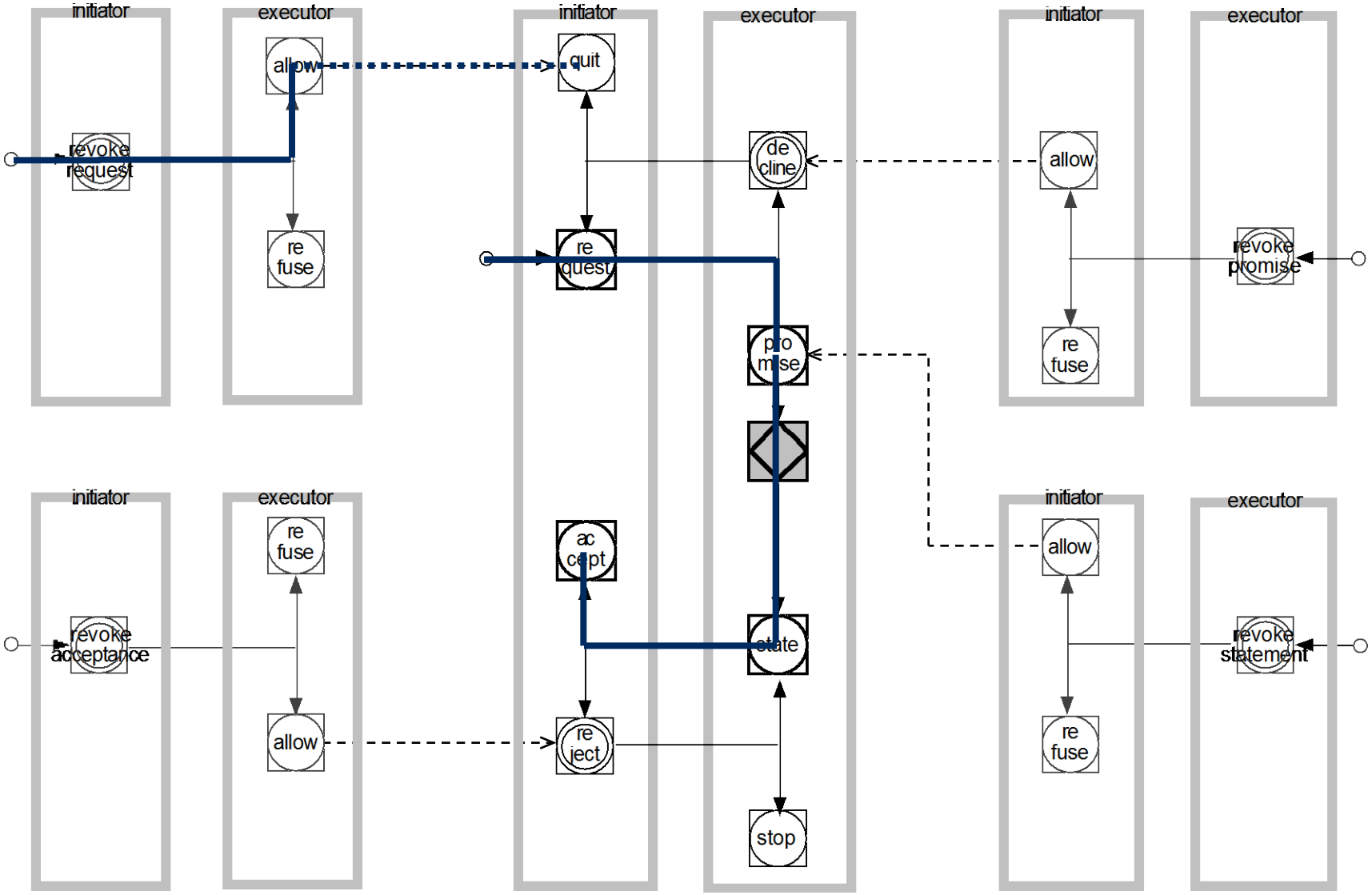
# The universal transaction process



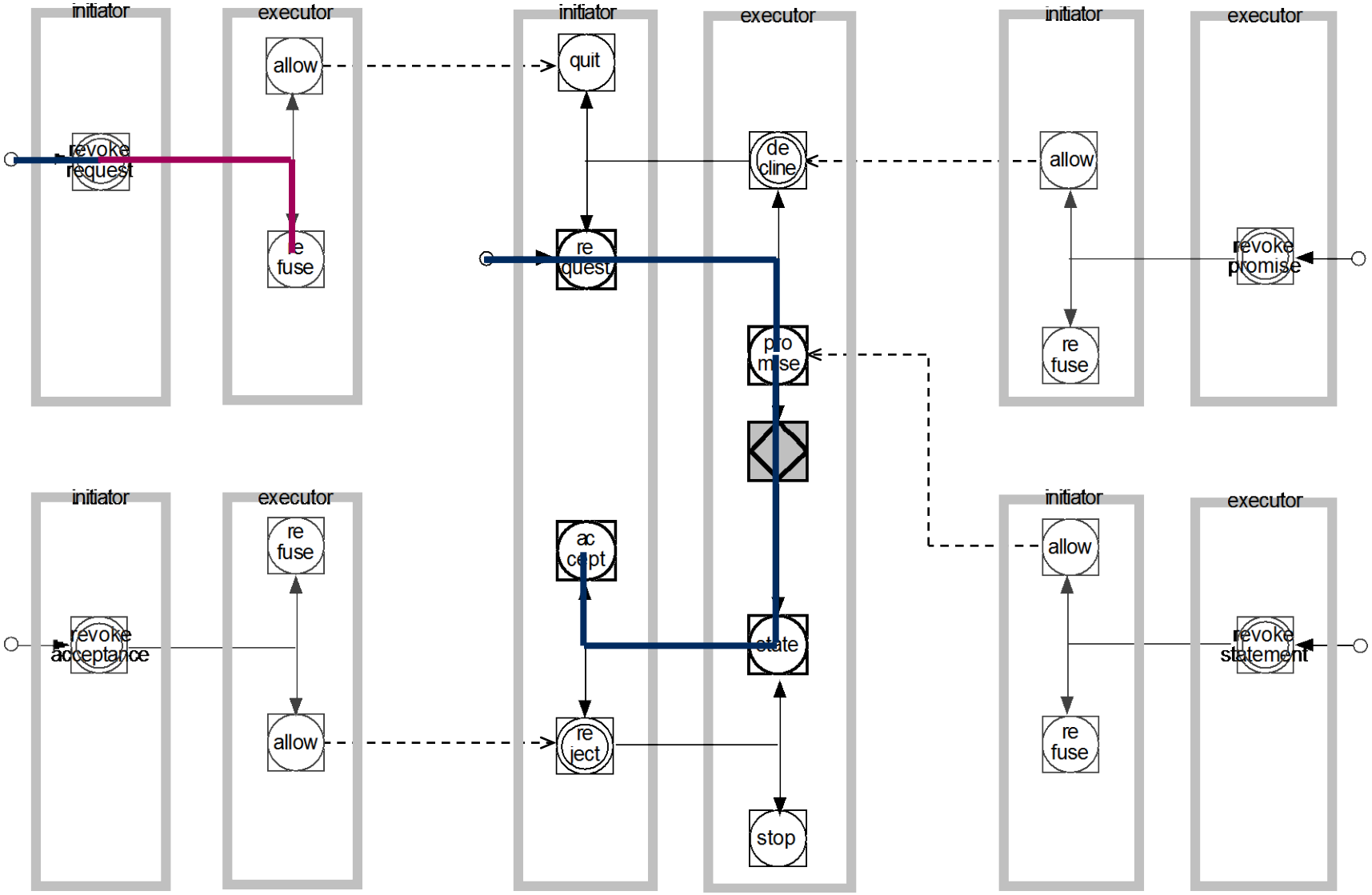
# The universal transaction process



# The universal transaction process



# The universal transaction process



# 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*

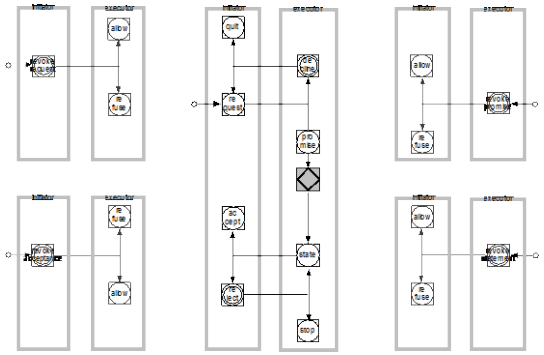
**<tacit act>**

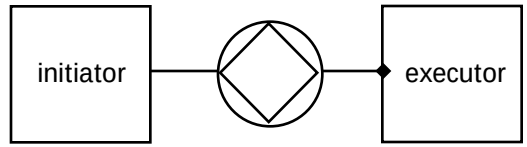
**Alicia : accept : Celestine : order 387 is fulfilled**

result

# The 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.

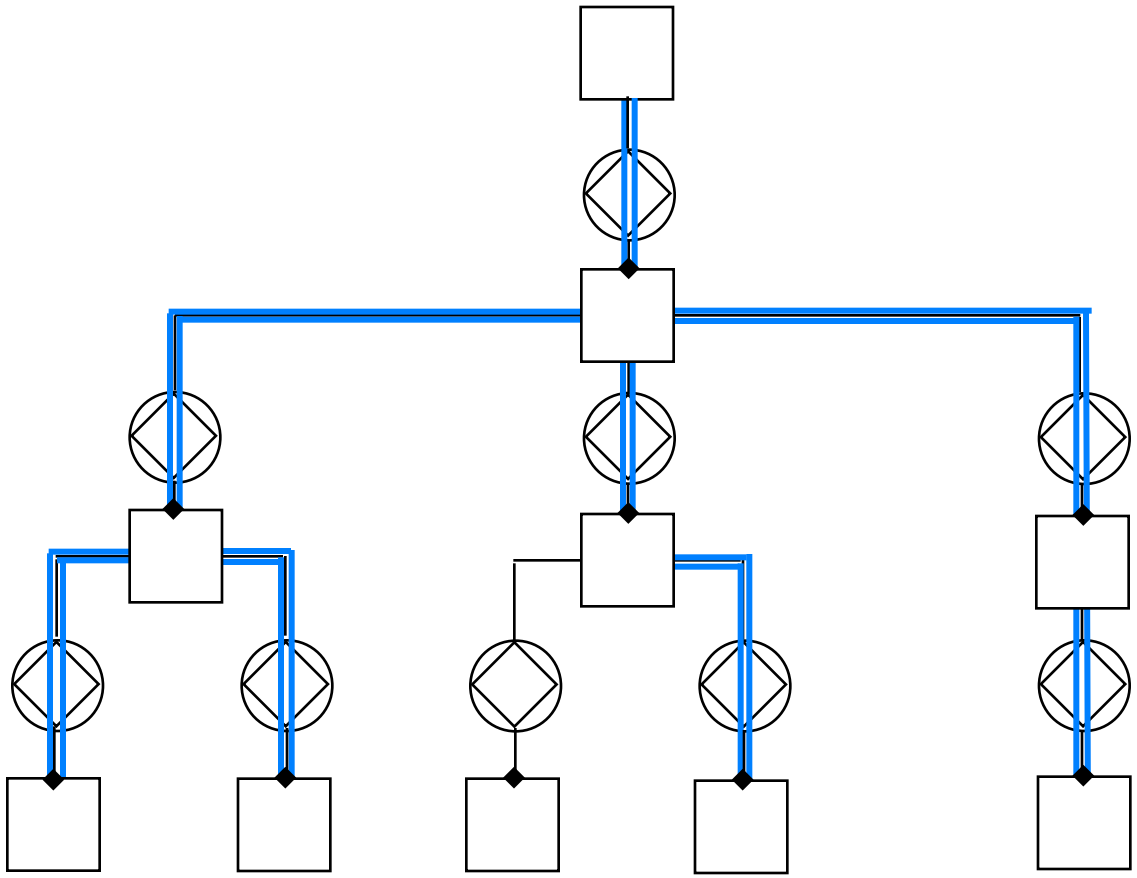





Next to the *process* interpretation of the transaction symbol, there is the *state* interpretation:

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**.

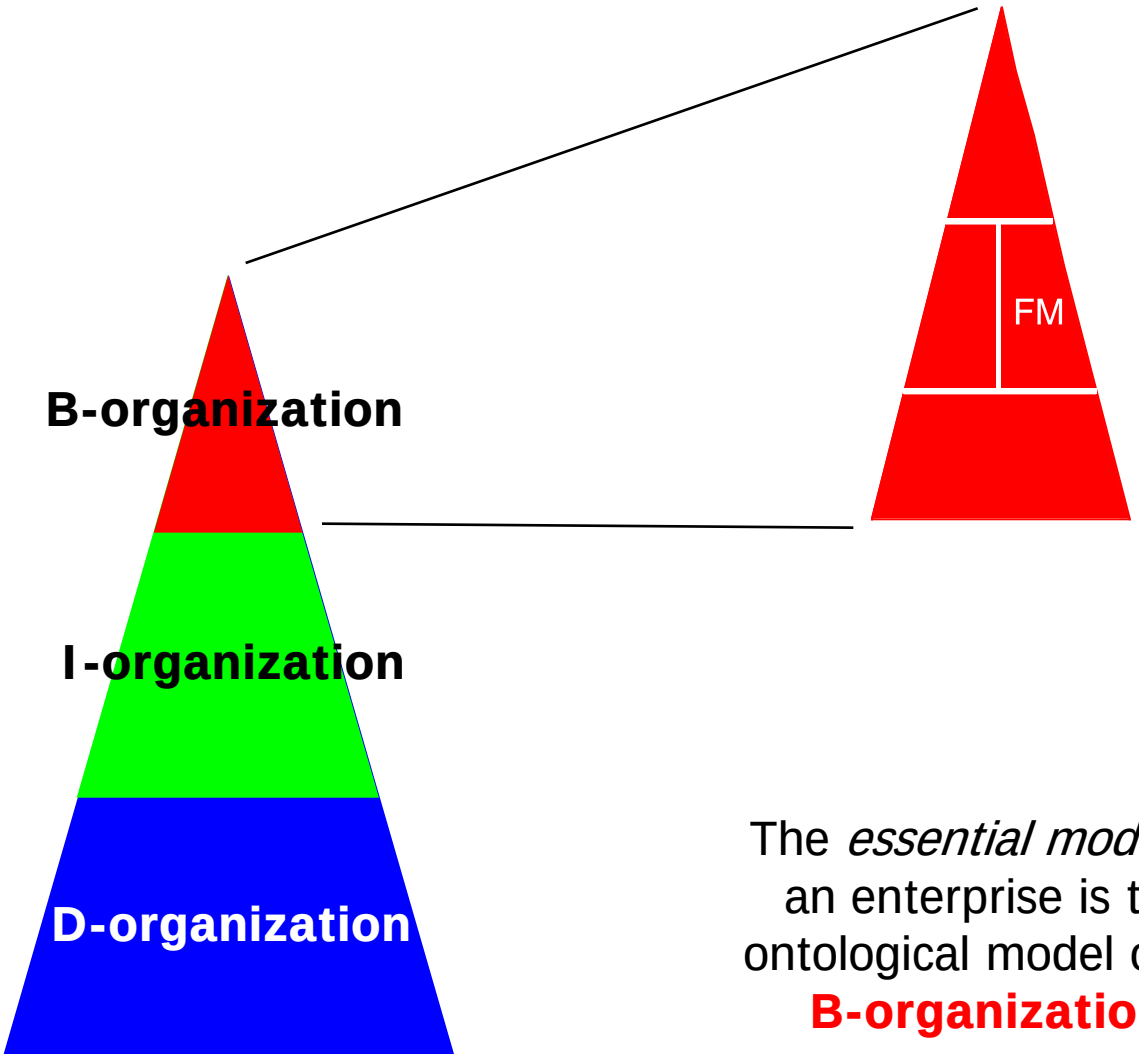
## *Informa*

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)



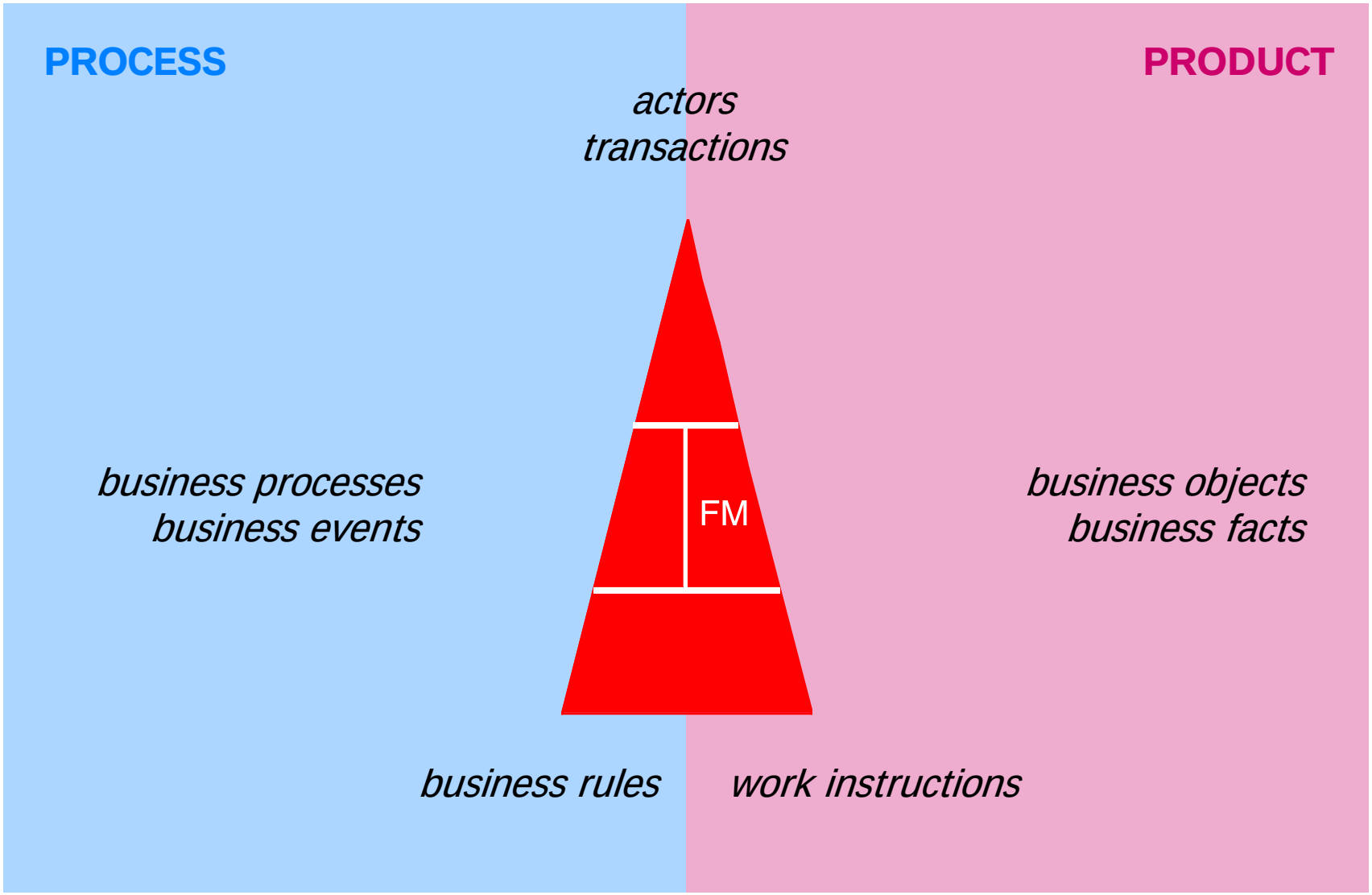

*creating*
*deciding*
*judging*

*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)



# Outline

Model Driven Engineering

System Design ($\tau$-theory)

Enterprise Ontology ($\psi$-theory)

**DEMOP**

Conclusions

# DEMO: Design and Engineering Methodology for Organizations

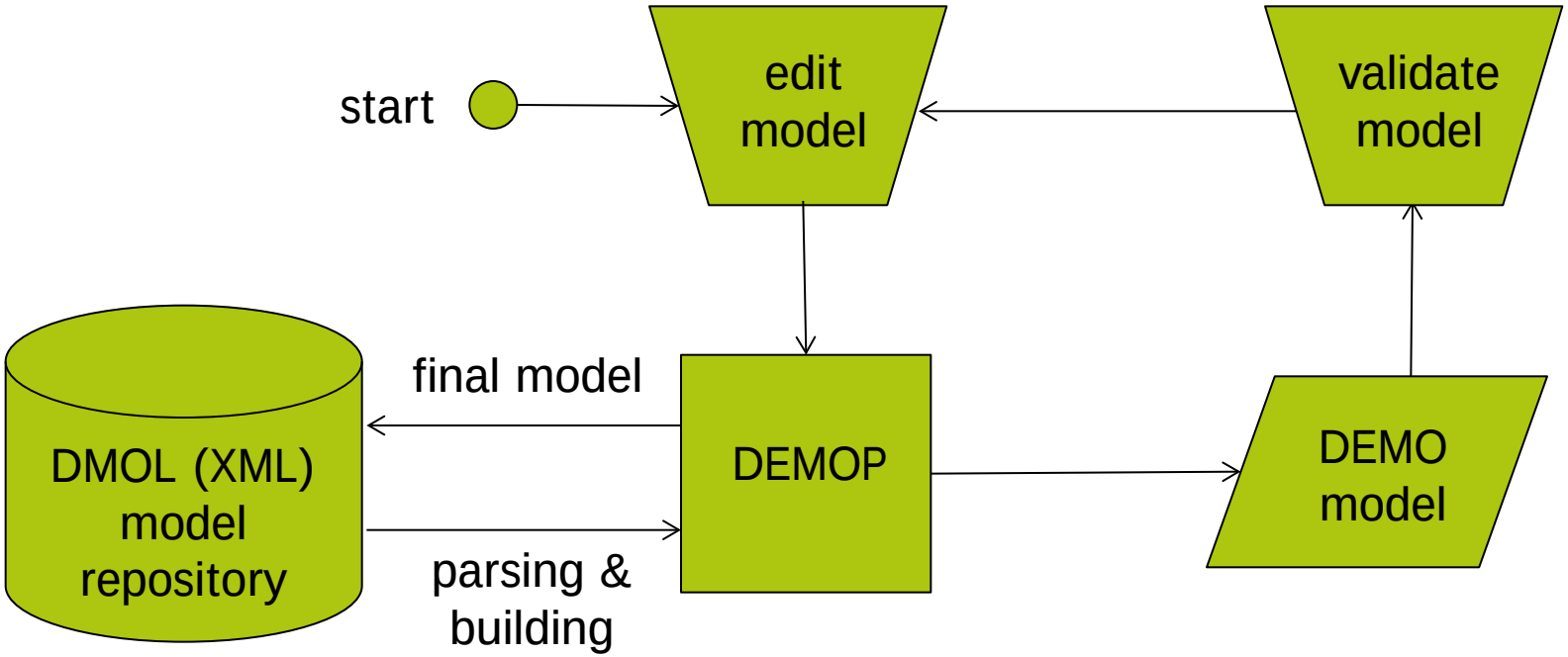
DEMO is the pioneering methodology of Enterprise Engineering.

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.

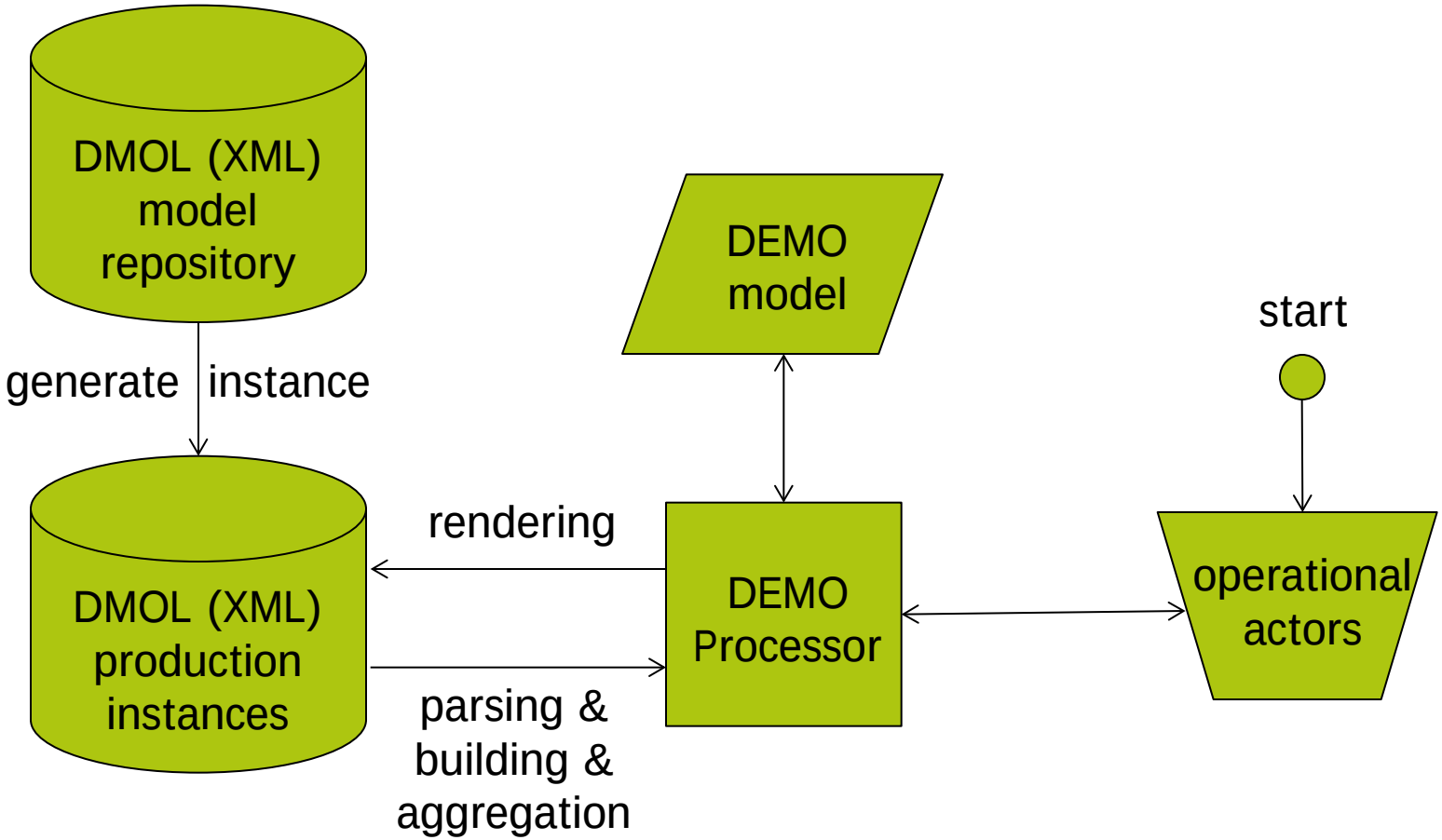
Every Enterprise Information System is *some* implementation of the essential model (DEMO model) of *some* enterprise.

*DEMOP* is a DEMO based generator for Enterprise Information Systems, developed by *Steven van Kervel* (director of ForMetis)

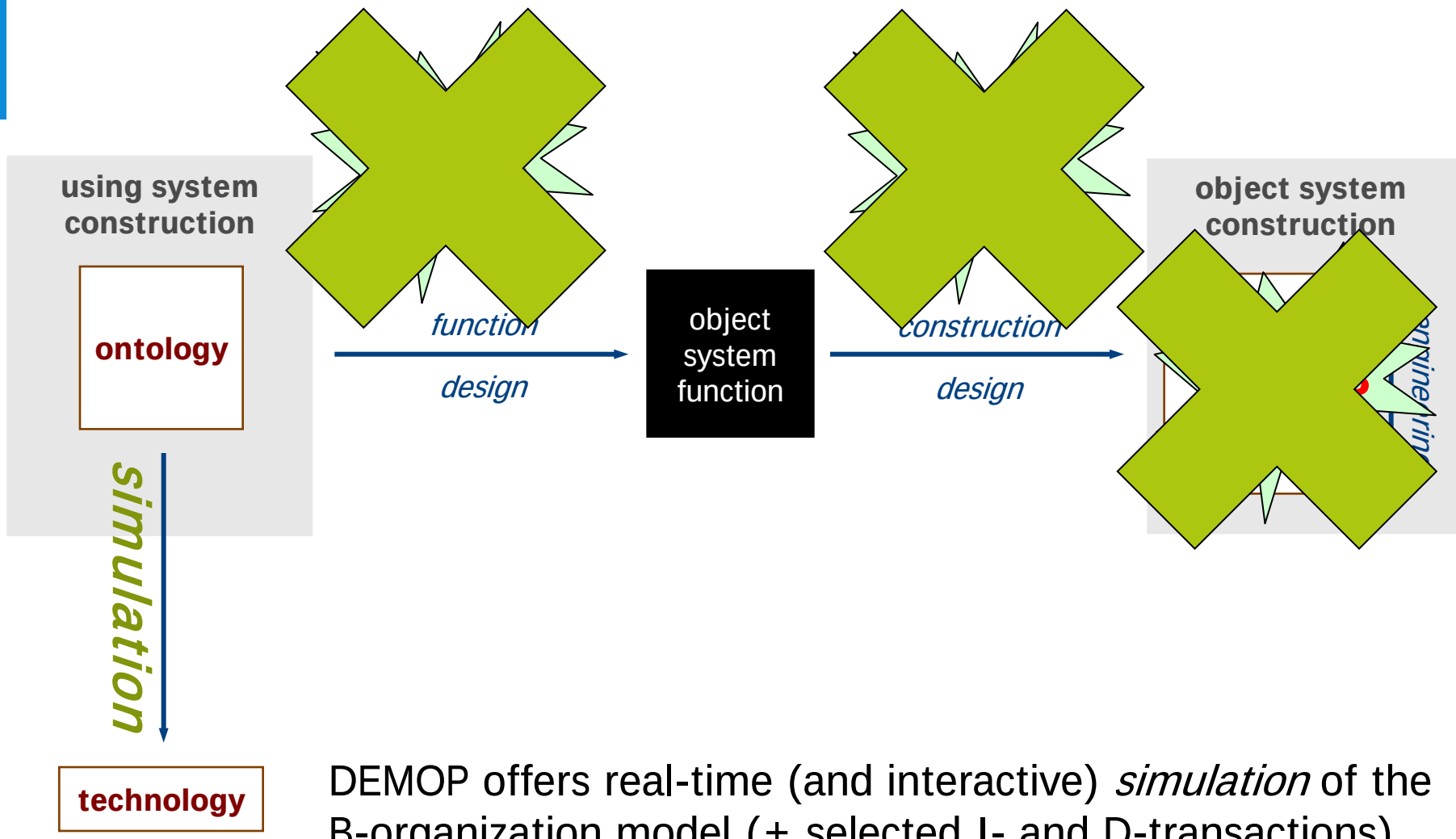
# DEMOP – modeling mode



# DEMOP – production mode



# What does DEMOP achieve?




DEMOP offers real-time (and interactive) *simulation* of the B-organization model (+ selected I- and D-transactions).

# Outline

Model Driven Engineering

System Design ($\tau$-theory)

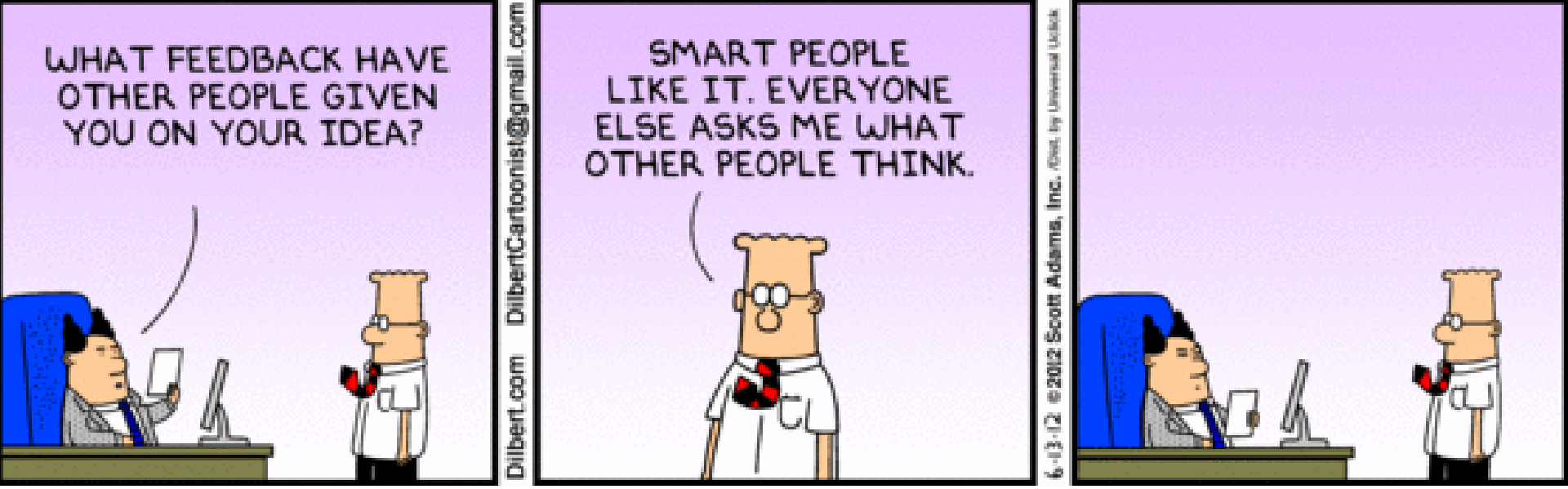
Enterprise Ontology ($\psi$-theory)

DEMOP

**Conclusions**

# Conclusions

- Current approaches to MDE are quite error prone.
- Because of its being fully rooted in the ψ-theory, DEMO delivers *coherent*, *consistent*, *comprehensive*, and *concise* '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 EIS (including ERP and WFM systems) could be.




My TU Delft address: j.l.g.dietz@tudelft.nl

My Sapio address: jan.dietz@sapio.nl

The CIAO! Network: www.ciaonetwork.org

The EE Institute: www.ee-institute.com