## 1 Introduction

Since the dawn of the digital age, the application of digital information and communication technology (ICT) for the support of people in organisations, has suffered from inappropriate conceptions about organisations and about the potentials and limitations of ICT. A typical testimony of this current state of the art, is the persistent failing of requirements engineering, evidently the major cause of ICT application failures. Another testimonial is the fragmented consideration of essentially integrated topics, like business processes, data, and rules.

Transcending this experience is the observation that organisational development in a broad sense suffers from inappropriately understanding the very notion of organisation, and the role of human beings in particular. The Taylorist 'human resource' idea, and the subsequent mechanistic notion of management still prevails. Altogether, the state of the art in organisational development and management, including the development and management of enterprise information systems (EISs), falls short in all three generic goals of enterprise engineering (EE): intellectual manageability, organisational concinnity, and social devotion [Dietz, Hoogervorst et al. 2013].

The ψ-theory contributes to achieving the first and the third goal in particular, leaving the second goal primarily to the β-theory [JDM-8]. Therefore, the primary function of the ψ-theory for enterprise engineers is to be an intellectual instrument in making organisational problems intellectually manageable, and in providing them with proper insights, notably in communication, information, (human) action, and organisation. The secondary function is to offer them the right arguments for empowering employees with the authorities and responsibilities they deserve. This is an ethical duty of enterprise engineers.

The ψ-theory (the Greek letter "ψ" is pronounced as "PSI", which stands for "Performance in Social Interaction") is a theory about the construction and operation of organisations. It explains how and why people cooperate, and in doing so bring about the business of an enterprise. It is rooted in speech act theory [Austin, 1962; Searle, 1969], in social action theory [Habermas, 1981], in information systems theory [Langefors, 1977], and in systemic ontology [Bunge, 1979].

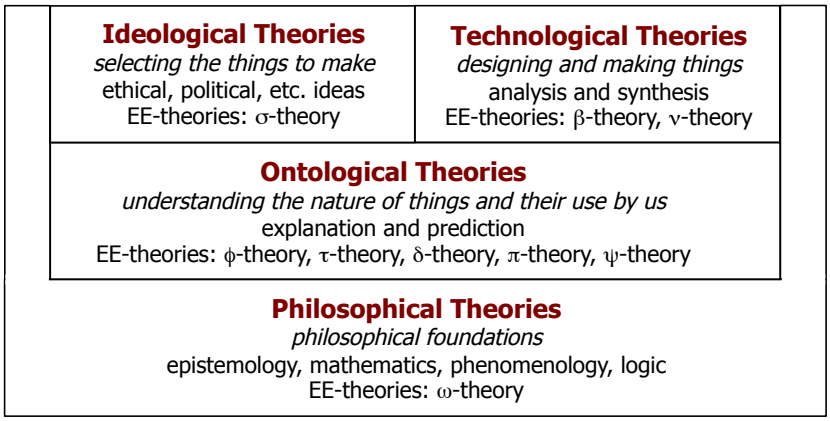



Figure 1.1 Classification scheme for enterprise engineering theories

This extended summary presents and discusses the main subjects and outcomes of the ψ-theory, including its relationship with other enterprise engineering theories, as exhibited in the classification scheme in figure 1.1 (Note: in this scheme, an arrow from A to B means that A is a basis of B). The ψ-theory is an *ontological* theory, which means that it concerns the nature of things, in this case of organisations and information systems. The next core notions are clarified by the ψ-theory: organisation, actor, social interaction, business process, business data, business rule, information system. From other EE-theories, these notions are taken: system, model, construction, function (τ-theory); information, knowledge (ϕ-theory); operation, state, action, event, process (δ-theory).

Because the ψ-theory is quite encompassing, it has been split in two parts: the *general ψ-theory* (also called the human face or front side of the ψ-theory) and the *special ψ-theory* (also called its system face or back side).

## 2 General ψ-theory

Human beings are the gems of every organisation. The general ψ-theory explains that human communication is the root of information and (social) action, and subsequently of organisation. This basic understanding makes organisations primarily social systems, of which the elements are human beings, bestowed with appropriate authority and bearing the corresponding responsibility. The operating principle of organisations is that *subjects* (human beings in their capacity of social individuals) enter into and comply with commitments regarding the products/services that they bring about in cooperation. The general ψ-theory is primarily based on the Theory of Communicative Action [Habermas 1981] and on Speech Act Theory [Austin, 1962; Searle, 1969].

### 2.1 Social interaction

By performing *production acts* (or P-acts for short), subjects bring about *products*. A product is an independently existing, original fact (like "membership #387 is started"). It may have a number of dependent facts with it, called properties of the product. People in the environment of the enterprise, who take primarily the function perspective on the enterprise (Cf. τ-theory [JDM-2]), perceive products as (business) *services*.
By performing *coordination acts* (or C-acts for short), subjects enter into and comply with commitments towards each other regarding products. A product can be material (like manufacturing a bike) or immaterial (like selling a bike).

The generic structure of a C-act is: < performer > < intention > < addressee > < product > < time >

Let us take as an example the request by John (the *performer*) towards Mary (the *addressee*), who is a desk officer at a library, to become member of the library. The *product* is the start of a membership, with John as its member, and the *time* is the requested production time (which is in this case the starting day of the membership). The *intention* represents the 'social attitude' taken by the performer towards the addressee: it is the request by John to Mary to start his membership. In performing a C-act, or any communicative act, the performer raises three *validity claims* [Habermas 1981] towards the addressee: the claim to justice, the claim to sincerity, and the claim to truth. All three of them have to be accepted by the addressee in order to make the coordination act successful. The *claim to justice* (German: Richtigkeit) concerns the validity of a coordination act in the social context of the two actors. The claim is satisfied if the addressee acknowledges the authority of the performer to play the role he/she plays, like the performer acknowledges the authority of the addressee, both in the social context in which the act is performed. In the example, this means that Mary acknowledges John's authority to request for a membership. The *claim to sincerity* (German: Wahrhaftigkeit) concerns the validity of a C-act in the context of the personal relationship of the two subjects. It is a matter of trust. So, the question is: does Mary trust that John is sincere in his request to become member? Clearly, satisfying the claim to sincerity must emerge from the particular situation in which John and Mary find themselves. Lastly, the *claim to truth* (German: Wahrheit) is satisfied if the product does exist or if creating it leads to a lawful state of the 'library world'. In the example, this is guaranteed as long as John requests for a membership, and not for milk or honey, because these products cannot exist in the 'library world', and as long as Mary is able to produce the product.

Performing a C-act is a process in which both the performer and the addressee apply three human abilities, called performa, informa, and forma (Cf. Figure 2; John is on the left side, Mary on the right side). Accordingly, three levels of communication are discerned. The most important level is the *performa level*, which means that John and Mary have to achieve *social correspondence*. It is achieved if in Mary's mind the commitment is evoked that John has exposed, such that she can respond appropriately to John's request. A prerequisite for social correspondence is to achieve *cognitive correspondence*, which occurs on the *informa level* of communication. It is achieved if the right understanding of both the intention and the proposition (product and time) are induced in Mary's mind. Achieving cognitive understanding needs at least two communication acts: one in which John informs Mary, and the other one in which Mary confirms to John that she has understood him. A prerequisite for cognitive understanding is to achieve *notational correspondence*, which occurs on the *forma level* of communication. It is the case if for every communication act at the informa level the sentences uttered by John, are correctly perceived by Mary, and vice versa. A prerequisite for achieving notational understanding, is that these sentences are transmitted between John and Mary without distortion. This is the concern of the *medium level*.

The relationships between the distinct levels of communication can be understood properly by the metaphor of the Russian dolls (matryoshka's), as shown in Figure 2.1. The small blank doll at the top represents the most inner self of human beings. There resides our wisdom and love, from which we decide whether and how to respond to coordination acts of our fellow men. The ψ-theory is only concerned with the levels within the blue frame in Figure 2.1.

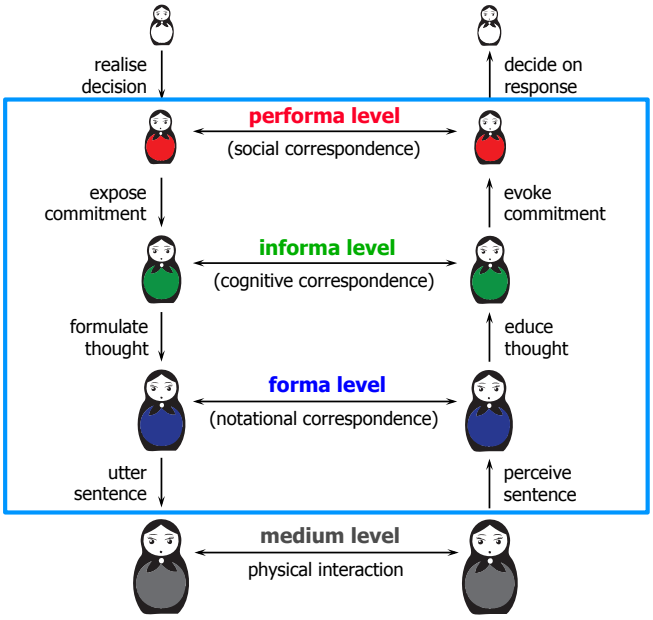



Figure 2.1. Levels of communication in coordination acts

**2.2 Transactions**

C-acts and P-acts appear to occur in universal patterns, called *transactions*. A transaction involves two subjects, one in the role of consumer (*initiator*) and the other one in the role of producer (*executor*). Figure 2.2 exhibits the *basic transaction pattern*. The process starts when the consumer performs the request regarding some product to be produced, e.g. becoming member of a library. Performing the request results into the *transaction status* 'requested'. The producer responds by promising the requested product, which brings the transaction in the status 'promised'. Then the producer produces the product (which is in this case the decision to start a new membership), after which he/she states to have done it, by which the status 'stated' is reached. In response to this event, the consumer accepts the product, which brings the transaction in the final status 'accepted'.

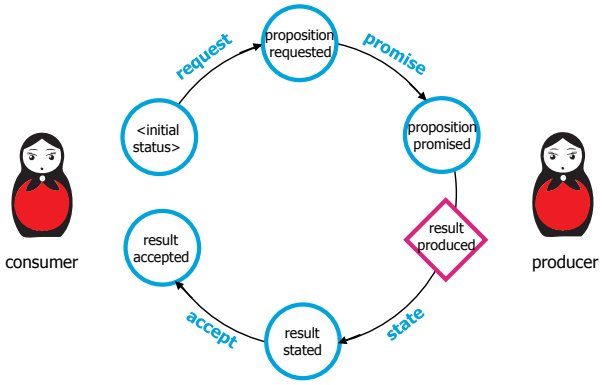



Figure 2.2. The basic transaction pattern

The four basic steps (request, promise, state, accept) must necessarily be performed in order to make a transaction end successfully. The best understanding of a transaction, however, is that it proceeds in three phases: the proposition phase, the execution phase, and the result phase. In the *proposition phase* the two subjects discuss and negotiate in order to come to agreement about the proposition: the product to be produced by the executor, as well as its production time. The proposition phase ends successfully if the (most recently) promised proposition equals the (the most recently) requested proposition. In the *execution phase*, the executor strives to bring about the promised proposition. The initiator is basically ignorant of what is going on in this phase. At some point in time, the executor addresses himself to the initiator by stating the result of his endeavours, which marks the beginning of the *result phase*. In this phase the two subjects discuss and negotiate in order to come to agreement about the produced result, which may differ from the agreed proposition. The result phase ends successfully if the (most recently) accepted result equals the (the most recently) stated result.

Next to the basic steps, a number of 'side' steps can be taken in the proposition phase and in the result phase. In addition, one may revoke any of the four basic steps[1], even repeatedly. The *complete transaction pattern* comprises twenty different C-acts, including of course the basic ones. It is exhibited in Figure 2.3. Every transaction process is a path (possibly including iterations) through the complete transaction pattern. This holds for every enterprise; therefore the complete transaction pattern is considered to be *universal*. Ideally, these C-acts are performed explicitly. Practice shows however that they can be performed implicitly, and even tacitly (which means that there are no observable evidences that the C-act is performed; yet it is). Lastly, instead of succeeding, i.e. to reach the final status "accepted", a transaction may fail, which means ending in an unsuccessful terminal status (quit or stop).

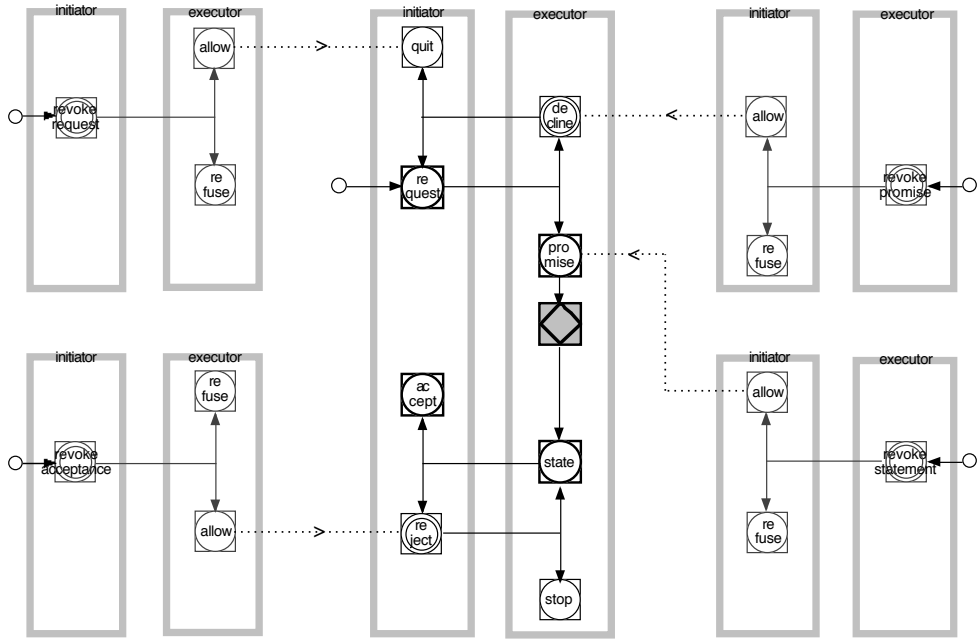



Figure 2.3. The complete transaction pattern

**2.3 Organisational building blocks**

Two additional notions need to be introduced: transaction kinds and actor roles. Every transaction is of some *transaction kind*. Transactions of the same kind regard products of the same kind. For example, transactions of the kind 'membership start' concern products of the kind 'Membership is started', in which 'Membership' is a placeholder for individual memberships (like membership #387). The notion of *actor role* serves to abstract from

[1] Theoretically, every step in a transaction process should be revocable. Practically, the four revocation patterns discussed appear to be sufficient.

the particular subjects that perform acts. An *elementary actor role* is defined as the authority to be the executor of exactly one transaction kind (e.g. membership start). This *authority* is allocated to a subject on the basis of the subject's *competence*. When allocated, the subject is expected to behave in a *responsible* way. Subjects are *accountable* for the way in which they exert their authority. An *actor* is a subject in its fulfilment of an actor role.

A subject may fulfil several actor roles (sequentially in the course of time, or simultaneously), and an actor role may be fulfilled by several subjects. This can also be sequentially in the course of time, or simultaneously or collectively. An example of the latter is the collective authority of a general meeting or of a board of directors. In practice, subjects typically fulfil a dozen of actor roles simultaneously, typically also spread over the three aspect organisations as will be discussed in Section 3.1.

Whereas every transaction kind has exactly one (elementary) actor role as its executor role, it may have many actor roles as initiator role. Therefore, a transaction kind with its executor role is considered to be the *building block* of organisations (see Figure 2.4). In this so-called Actor Transaction Diagram, transaction kinds are represented by a shape that consists of a diamond (standing for production) embedded in a disk (standing for coordination). Actor roles are represented by boxes (of a square or rectangular form). The connecting line from the transaction kind shape to the actor role shape ending in a small black diamond on the edge of the box represents an executor link. So, actor role A1 is the executor of transaction kind T1 (Note: by convention, they get the same number). The grey-lined box on the left side of the transaction kind shape, connected to it with the grey line, represents one of the initiator roles of T1.

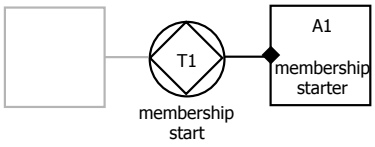



Figure 2.4. The organisational building block

The executor role (A1) of T1 may on its turn be an initiator role of one or more other transaction kinds. The same reasoning holds for the executor roles of these transaction kinds. In this way tree-like networks of transaction kinds and actor roles are built, which constitute ontological models of organisations. An *ontological model* of an organisation is defined as a conceptual model of the organisation in which one abstracts from implementation: from the particular subjects fulfilling the actor roles, as well as from the medium level and the forma level in coordination.

The general ψ-theory therefore provides the basis for an effective notion of Enterprise Ontology (EO), defined as the (constructional) essence of an enterprise's organisation[2]. This notion allows for a considerable reduction of complexity (expressed in the size of diagrams), compared with current (also construction oriented) business process modelling approaches, like BPMN, ARIS/EPC, and Petri Net. The theoretical and practical significance of EO is extensively discussed in [Dietz 2006].

## 3 Special ψ-theory

The general ψ-theory can be viewed as the human face (or 'front side') of the total ψ-theory. It explains and clarifies how cooperation between people, as social individuals, is going about. The focus of the special ψ-theory is on the consequences of the general ψ-theory for the systemic ontology of organisations [Bunge, 1979], i.e. on the intrinsic wholeness of a complete conceptual model of a system, as well as on effective divisions of such a holistic understanding, as well as for information systems theory [Langefors, 1977]. Therefore, the special ψ-theory is said to constitute the system face (or 'back side') of the total ψ-theory. Accordingly, "PSI" is read as "ISP". It has two meanings: *Intelligent System Partitioning* and *Integrated System Perspectives*. Both meanings of ISP will be discussed hereafter. In addition, the formalisation of the special ψ-theory in the so-called CRISP model will be presented and discussed.

---

[2] Recall from the τ-theory [JDM-2] that the term "organisation" refers to the construction perspective on enterprises, and the term "business" to the function perspective.

### 3.1 Intelligent System Partitioning

The three human abilities on which we based the levels in coordination, can also be applied to production. It now leads to considering subjects to take any of three shapes in performing P-acts: the performa shape, the informa shape, and the forma shape. In their performa shape, subjects are able to perform *original* P-acts, like creating things (manufacturing, transporting, observing), deciding, and judging. In their informa shape, they are able to perform *informational* P-acts, like remembering facts, recalling them, and computing derived facts from other (original or derived) ones. In their forma shape, subjects are able to perform *documental* P-acts, like storing, retrieving, copying, and transmitting sentences or documents (containing inscriptions of facts). Based on this distinction, the network of transaction kinds and related actor roles that constitutes the complete ontological model of an organisation can usefully be partitioned in three subnetworks: one containing the original transaction kinds and actor roles, one containing the informational ones, and one containing the documental ones. Consequently, three aspect organisations can be distinguished: the *B-organisation* (B from Business), the *I-organisation* (I from Informational), and the *D-organisation* (D from Documental). They are exhibited in Figure 3.1.

The relationships between them are that the I-organisation provides informational services to the B-organisation, and the D-organisation provides documental services to the I-organisation. By definition, the B-organisation contains all original transaction kinds and actor roles. It may, however, also contain informational and documental ones. This is the case if the business of the enterprise is (also) to provide informational and/or documental services. By definition, the I-organisation contains only informational transaction kinds and actor roles, whereas the D-organisation contains only documental transaction kinds and actor roles.

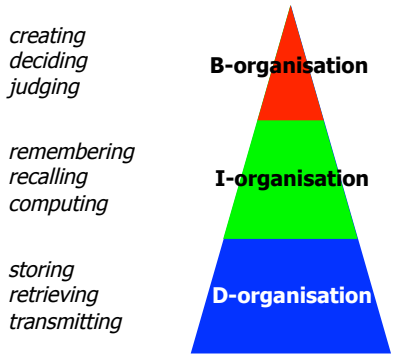



Figure 3.1. Intelligent System Partitioning

### 3.2 The essential model

By focussing on the B-organisation, we can achieve an additional major reduction of complexity. The ontological model of the B-organisation, in which the information sharing services by the supporting I-organisation are modelled as *information links* between actor roles and transaction kinds (now interpreted as conceptual containers of C-facts and P-facts in the B-organisation), is called the *essential model* of (the complete organisation of) the enterprise[3]. The essential model of an enterprise comprises everything that is necessary to understand and reason about the construction and the operation of the enterprise's organisation, only fully abstracted from realisation and implementation (to be elaborated in Sections 3.5 and 3.6).

Every enterprise has exactly one essential model, which is comparable to the *genotype* of a biological organism. It may change over time, when the the enterprise's organisation changes. However, because of their high level of abstraction, essential models are extremely stable. Most changes occur in its realisation, i.e. the supporting I- and D-organisation, and in the implementation of all three aspect organisations.

On the basis of the ψ-theory, a proper and exact definition of an *enterprise information system* (EIS) can be provided: it is some implementation of some realisation of the I-organisation of (a part of) the enterprise, and its

[3] Actually, the I-organisation and the D-organisation also have information links, that represent informational transaction kinds and supporting documental transaction kinds. They are disregarded, however, to avoid eternal recurrence.

supporting D-organisation. This definition shows at once the intrinsic and intensive interweaving of an EIS with the organisation supports. Therefore, a proper metaphor for an EIS is the nervous system of the human body: it is intrinsically and intensively interwoven with the body, and it cannot easily be replaced by another one..

### 3.3 Integrated System Perspectives

The complete ontological model of an organisation is divided into four sub models, each representing a particular perspective or view on the complete model: the construction model (CM), the process model (PM), the Fact Model (FM), and the Action Model (AM), as exhibited in Figure 3.2.

The CM of an organisation consists of the identified transaction kinds and the corresponding actor roles (as executor role and/or initiator role). The PM shows how exactly the identified transaction kinds are connected in tree structures, called *business processes*. The performance of a step in a transaction causes the occurrence of a *business event*, which may be an agendum for an actor to respond to. The FM contains the identified *business fact* kinds (the *business object* classes, and their properties), as well as the *business laws* that determine the state space and the transition space of the production world. The AM is a collection of business rules. A *business rule* is a procedural guideline to an actor for responding to a business event. These business rules are the imperative operationalisations of the applicable business laws. Next to business rules, the AM may contain *work instructions*, which guide actors in performing P-acts.

It is of paramount importance to recognise and understand that the four sub models represent four views on one integral model, in which the views are intrinsically connected. The current practice of separately dealing with business processes, business data (objects and facts), and business rules, often even allocated to different organisational departments, clearly can never lead to real improvement of the total organisation.

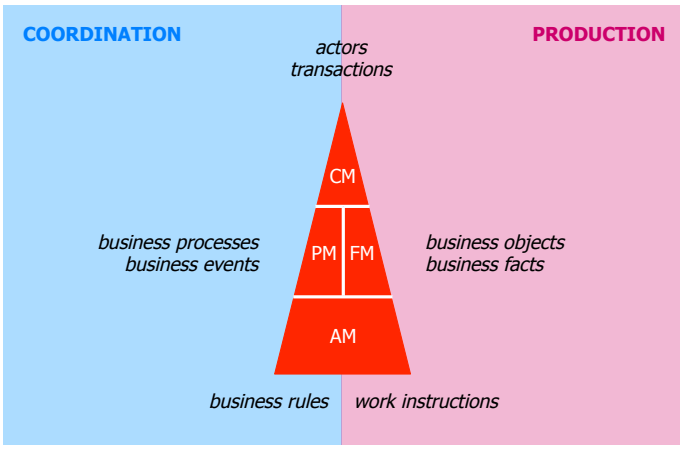



Figure 3.2. Integrated System Perspectives

### 3.4 The CRISP model

The CRISP model is a formalisation of the special ψ-theory, in which the intrinsic connection of the four sub models, as presented in Section 3.3, is clarified. It is based on the π-theory [JDM-7], which on its turn is based on the δ-theory [JDM-3]. The ψ-theory based automata are called *crispies*. Like smarties [JDM-7], crispies operate in a discrete linear time dimension, which means that we assume the existence of distinct points in time and that the difference between any two consecutive points in time is always the same.

At every point in time, the *world* (Cf. the τ-theory [JDM-2]) of a crispie is in some state. A *state* is a set of *facts*, both C-facts and P-facts. The *state base* of a crispie is defined as the set of C-fact types and P-fact types, whose instances may belong to a state.

In order to model the dynamics of a crispie, the notion of *C-act/fact* is introduced. The effect of performing a C-act is the creation of the corresponding C-fact, which is also called the occurrence of a *business event*. The set

of C-fact types, to the instances of which a crispie can respond, is called its *coordination base*. A C-fact regards a product of some kind. The set of product kinds that may occur in C-facts of the crispie is called its *product base*.

The *operation* of a crispie can be explained as follows. At every moment a crispie disposes of a set of *agenda* (things to do). An *agendum* (singular of agenda) is a pair (*c*, *t*) in which *c* is a C-fact (i.e. an instance of a C-fact type in the coordination base), and *t* is a point in time, called the *settlement time*. It is the point in time at which the creator of the C-fact wants the event to be responded to. The set of acts in the agenda with settlement time *t* is called the *action* at time *t*. A crispie responds instantaneously to an action by evaluating a partial function, called the *rule base* of the crispie. The result of the evaluation is a finite set of C-facts, called the *reaction*.

Hereafter, a formal definition of a crispie is presented, fully based on the formal definition of a smartie [JDM-7]. In this definition, the union of the extensions of a set of concept types (act types or fact types) C is denoted as $\underline{C}$, and the power set of a set X is denoted as $\wp X$. Points in time are represented by elements of the set $\mathbb{T}$; the current point in time is denoted by Now; (positive) time durations are elements of the set $\mathbb{D}$.

A crispie is formally defined as a tuple $< \mathbf{C}, \mathbf{R}, \mathbf{I}, \mathbf{S}, \mathbf{P} >$, where

| | | |
|---|---|---|
| **C** | : | a set of C-fact types, called the *coordination base* |
| **R** | : | a set of action rules, called the *rule base* |
| **I** | : | a set of intentions, called the *intention base* |
| **S** | : | a set of C-fact types and P-fact types, called the *state base* |
| **P** | : | a set of product kinds, called the *product base* |

$$\mathbf{R} : \wp\underline{\mathbf{C}} * \wp\underline{\mathbf{S}} \rightarrow \wp(\mathbf{I} * \underline{\mathbf{P}} * \mathbb{T} * \mathbb{D})$$

A crispie is called *elementary* if the C-facts in its coordination base all regard one and the same product kind. Crispies that are not elementary, are called *composite*.

The components **C**, **S**, and **P** have been explained above. The intention base **I** comprises all intentions that are contained in the complete transaction pattern (Cf. Figure 2.3). The rule base **R** can conveniently be represented by its extension, i.e. the set of *action rules* ( or business rules) of the form $< C, S, \mathbf{< i, p, pt, sd >} >$ where:

*C* is the *event* that is going to be responded to; $C \subset \underline{\mathbf{C}}$. Note that $\underline{\mathbf{C}} \subset \mathbf{I}^*\underline{\mathbf{P}}$
*S* is a set of C-facts and P-facts, called the *state*; $S \subset \underline{\mathbf{S}}$.
**< i, p, pt, sd >** is the *response*; it is a set of tuples < *i*, *p*, *pt*, *sd* > where *i* is the intention of the created C-fact ($i \in \mathbf{I}$) and *p* is the product that the C-fact is concerned about ($p \in \underline{\mathbf{P}}$). The production time of *p* is *pt* ($pt \in \mathbb{T}$), and the settlement time of the C-fact *st* = Now + *sd* ($sd \in \mathbb{D}$).

Let us again take the library example for clarifying the CRISP model. We will follow the steps in the basic transaction pattern of the transaction process in which the membership, identified by #387, with starting day 20130401 (Gregorian calendar), is brought about. We assume that the product and the production time remain unchanged during the transaction process.

Then Mary gets this event to respond to: < *request, membership #387 is started, 20130401*, *st-r* >, created by John. The time *st-r* is the settlement time of the request. Let us assume that the current state *S* allows Mary to promise. Her response consists of the tuple < *promise*, *membership #387 is started, 20130401*, *st-p* >, in which *st-p* is the settlement time of the promise. This is an event to which Mary has to respond. Let us assume that the current state *S* allows Mary to produce the product (the execute act, followed by the state act). Then her response consists of the tuple < *state*, *membership #387 is started, 20130401*, *st-s* >, in which *st-s* is the settlement time of the state. This is the event to which John has to respond. Let us assume that the current state *S* allows him to accept. Then his response will consist of the tuple < *accept*, *membership #387 is started, 20130401*, *nill* >, in which *nill* is the settlement time of the accept. Since the accept fact is a terminal state in the transaction process, its settlement time is irrelevant; that's why it gets the value *nill*.

**3.5 The crispienet**

The influencing relationships among a collection of crispies are somewhat more complicated than those among a collection of smarties (Cf. π-theory [JDM-7]). Actually, every step in the complete transaction process (Cf. Figure 4) of a transaction kind is to be conceived as an elementary smartie. These smarties are 'organised' under the 'architecture' of the complete transaction pattern. However, if we take advantage of the fact that all activities in a collection of crispies take place in transaction processes, the mutual influencing relationships can be made quite comprehensible if the collection of crispies is modelled as a crispienet. A *crispienet* is a network consisting of two kinds of components: actor roles and transaction kinds. Figure 3.3 exhibits the symbolic representations of the components of a crispienet diagram, as well as its basic constructs.

An *actor role* represents the transition mechanism of a crispie, consisting of its rule base and its operating cycle. By the *operating cycle* is understood the periodic checking of an actor (at high frequency) whether there are transitions to be performed. Therefore, an actor role is also called the *kernel* of a crispie. The kernel of an elementary crispie is an *elementary actor role*; the kernel of a composite crispie is a *composite actor role*.

A *transaction kind* represents the complete transaction pattern for the product kind that appears in the coordination base of an elementary crispie. Transaction kinds have two interpretations. In the process interpretation they represent *transaction processes*; in the state interpretation they represent *transaction banks* (which are containers of the histories of their transaction processes). An *aggregate transaction kind* is a collection of transaction kinds. It serves only to keep the smartienet diagram concise.

An *initiator link* between an actor role and a transaction kind means that the actor role takes the initiator role in its transaction processes. An *executor link* between an actor role and a transaction kind means that the actor role takes the executor role in its transaction processes. An *information link* between an actor role and a transaction kind means that the actor role has access to its transaction bank for inspecting the histories of the transaction processes. Executor links and initiator links are considered to 'cover' an information link, so the executor and the initiator of a transaction have full access to the history of their transaction.

It is also possible to draw aggregate transaction kinds with composite actor roles as initiators and executors. These are not shown in Figure 3.3, however.

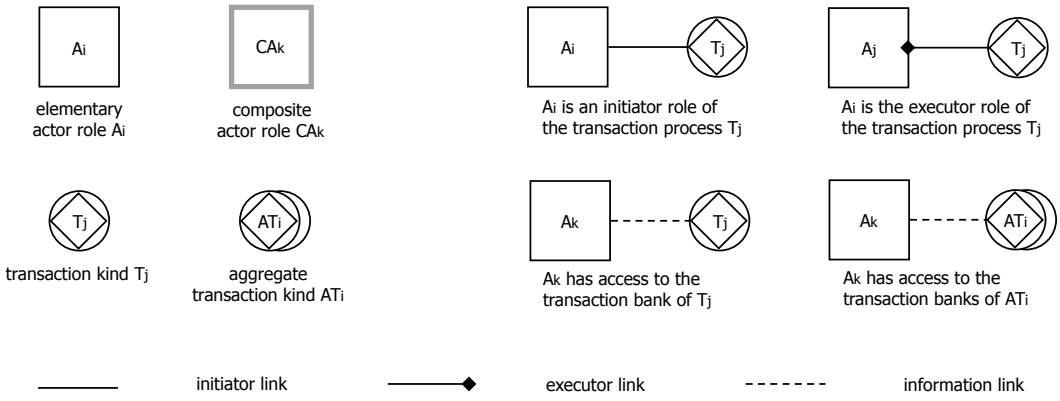



Figure 3.3. Legend of the crispienet

The mutual influencing of actor roles through being initiator and executor in transaction processes is collectively called *interaction*. The mutual influencing of actor roles through being creator and inspector of facts (both C-facts and P-facts) in transaction banks is collectively called *interstriction*[4]. The distinctive difference between interaction and interstriction is that interaction causes crispies to perform transitions, whereas interstriction does not. Consequently, a crispie is not 'aware' of a state change at the time it takes place. Instead, it 'takes notice' of

[4] The noun "interstriction" comes from the Latin verb "stringere", which is also the root of "restriction". The term "interstriction" expresses that smarties restrict each others freedom of action or 'play area'. By changing the state of the world, and subsequently taking these changes into account when being active.

a state change at the next point in time at which it is activated. So, between two adjacent points in time at which a crispie performs a transition, it 'sleeps'. In that period however a number of state changes may occur. (For a deeper discussion, see the δ-theory [JDM-3]).

Figure 3.4 exhibits the representation of (elementary) crispies as crispienet building blocks or *modules*. The lower part shows how the initiator role of T2 (left crispie) and the executor role of T2 (right crispie) are 'clicked' together. In order to make the connected actor roles A1 and A2 a complete ontological model, the other 'loose ends' have to be clicked properly to other crispies. In this way unlimited large networks are constructed of actor roles (kernels of crispies) and transaction kinds. As said before, these networks constitute the ontological models of (networks of) organisations, and thus of societies.

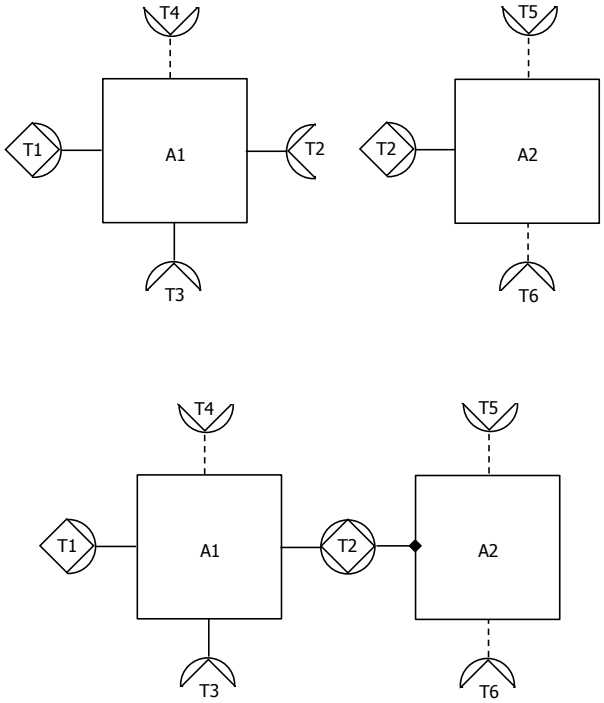



Figure 3.4. Constructs in crispienets

### 3.5 Realising crispienets

Crispienets are *essential models* of concrete systems, that is, they show the (ontological) essence of a system, fully abstracted from realisation and implementation. In order to arrive at a a concrete system, using a crispienet as its blueprint, it has to be realised first. By *realisation* is understood the design of an implementable model, starting from the essential model of the crispienet. The first step is to design an I-organisation that satisfies all informational needs of the B-organisation (remembering facts, recalling facts, computing derived facts). Put differently, this I-organisation realises the information links in the essential model, as well as the state interpretation of the transaction banks in it. The second step is to design a D-organisation that satisfies all documental needs of this I-organisation (storing, retrieving, copying, and transmitting documents). A full account of realising essential enterprise models is provided in [Jong, 2013].

### 3.6 Implementing crispienets

By *implementation* is understood the allocation of suitable technological means to (the elements of) a realised implementable model, so that the crispienet becomes an operating concrete system. In current practice, it means finding an appropriate IT platform on which the compiled crispie model can 'run'. A full account of implementing crispie models, using ICT as the main technology, is provided in [Kervel, 2012].

## 4 Applications

### 4.1 Enterprise applications

The chief application domain of the ψ-theory are enterprises, of any kind and size. Examples of such applications can be found in numerous publications (see e.g. www.ee-institute.org). Small cases for educational purposes can be found in [Dietz, 2006] and [Perinforma, 2013].

### 4.2 Technical applications

The main theory for understanding and analysing technical systems is the π-theory [JDM-7]. However, it appears that many systems that we call technical, are actually (and often originally really) social systems, only technically implemented. Taking this point of view may contribute considerably to a deeper and more appropriate understanding of technical systems. To illustrate this, the elevator control system, which is discussed in [JDM-7] as a purely technical system, will be considered as a technically implemented social system hereafter.

### 4.3 Example: elevator control system

In the period between 1980 and 2000, several 'reference' software problems have gone around in the area of software research and development, for demonstrating the qualities of a development approach, and for comparing appoaches. One of them is the Elevator Control System (ECS). Hereafter, the version of the ECS as elaborated by Edward Yourdon (in his book "Modern Structured Analysis") is copied. The assignment is to produce the essential model of this system according to the ψ-theory, so as a crispienet. This is possible if the ECS is considered as a social system, only technically implemented. The analysis and solution are based on [Dietz, 2003]. In order to save space, each paragraph of the description will be followed by its analysis (in italics) in the π-theory.

**Problem description and analysis**

The general requirement is to design and implement a program to schedule and control four elevators in a building with 40 floors. The elevators will be used to carry people from one floor to another in the conventional way.

The program should schedule the elevators efficiently and reasonably. For example, if someone summons an elevator by pushing the down button on the fourth floor, the next elevator that reaches the fourth floor traveling down should stop at the fourth floor to accept the passenger(s). On the other hand, if an elevator has no passengers (no outstanding destination requests), it should park at the last floor it visited until it is needed again. An elevator should not reverse its direction of travel until its passengers who want to travel in its current direction have reached their destinations. (As we will see below, the program cannot really have information about an elevator's actual passengers; it only knows about destination button presses for a given elevator. For example, if some mischievous or sociopathic passenger boards the elevator at the first floor an then presses the destination buttons for the fourth, fifth, and twentieth floor, the program will cause the elevator to travel to and stop at the fourth, fifth, and twentieth floors. The computer and its program have no information about actual passenger boardings and exits.) An elevator that is filled to capacity should not respond to a new summons request. (There is an overweight sensor for each elevator. The computer and its program can interrogate these sensors.)

*Because we abstract from implementation, there is no limit to the number of floors or elevators. We will use the variables Floor and Elevator to denote floors and elevators. We assign a property "orientation" to each elevator with value "upward", "downward", or "idle". The orientation of an elevator is upward if there are pending destination orders for floors above the current floor or pending summons orders from floors above the current floor with direction "up". The orientation of an elevator is downward if there are pending destination orders for floors below the current floor or pending summons orders from floors below the current floor with direction "down". If neither is the case, the value orientation is "idle".*

*The (simple) scheduling policy we adopt is that an elevator proceeds moving in a direction (up or down) as long as its orientation is upward or downward respectively. If there is a summons order from a floor that will not be passed by one of the moving elevators, an 'idle' elevator will be sent to this floor, as soon as there is one.*

*Modelling the ECS as a social system means that we consider all activities and all communication to take place in transactions between human actors. So, we imagine that on each floor there is a functionary to which passengers can tell that they want to go up or down. This is the first transaction kind we identify: T1 (summons*

*order fulfilment), with product kind "summons order for Floor in Direction is fulfilled". Similarly, we imagine that in each elevator there is a functionary to which passengers can tell to which floor they want to go. This is the second transaction kind: T2 (destination order fulfilment) with product kind "destination order for Elevator to Floor is fulfilled". Next, we imagine that there is a (physically very strong) functionary on top of each elevator shaft whom one can request to move the elevator car up or down or to let it stop at a floor. Therefore we need the third transaction kind: T3 (elevator movement), with product kind "Elevator is set to move in Direction", where Direction can have the values "up", "down", and "stop".*

*The initiator role of T1 is the environmental actor role CA1 (floor passenger), and its executor role is the internal actor role A1 (summons order fulfiller), fulfilled by the functionaries on the floors. Likewise, the initiator role of T2 is the environmental actor role CA2 (elevator passenger), and its executor role is the internal actor role A2 (destination order fulfiller), fulfilled by the functionaries in the elevators. Next, the executor role of T3 is the environmental actor role CA3 (elevator mover), fulfilled by the functionaries on top of the elevator shafts. The question now is: who is the initiator of T3? Clearly, it cannot be A1 or A2 because they do not have the overview of all pending transactions T1 and T2. Consequently, another actor internal role is needed, who does have the overview, and who therefore can decide which elevator must visit which floor, in order to enable the execution of transactions T1 and T2. Let us call this actor role A4 (visits performer), and the transaction kind of which it is the executor role: T4 (visiting), with product kind "Elevator visits Floor with Orientation". So, there is an additional functionary who fulfils actor role A4. The initiator of T4 is a self-activating actor role A5 (visit controller) who is activating itself at a sufficient high frequency.*

*In addition, there are two external (aggregate) transaction banks: AT1 (elevator position) and AT2 (overweight indicator). We don't care about their initiators and executors.*

The interior of each elevator is furnished with a panel containing an array of 40 buttons, one button for each floor, marked with the floor numbers (1 to 40). These destination buttons can be illuminated by signals sent from the computer to the panel. When a passenger presses a destination button not already lit, the circuitry behind the panel sends an interrupt to the computer (there is a separate interrupt for each elevator). When the computer receives one of these (vectored) interrupts, its program can read the appropriate memory mapped eight-bit input registers (there is one for each interrupt, hence one for each elevator) that contains the floor number corresponding to the destination button that caused the interrupt. Of course, the circuitry behind the panel writes the floor number into the appropriate memory-mapped input register when it causes the vectored interrupt. (Since there are 40 floors in this application, only the first six bits of each input register will be used by the implementation; but the hardware would support a building with up to 256 floors.)

*Pressing the destination button of some Floor in some Elevator counts as performing a request of a transaction T2 for this Floor and Elevator to the functionary in the elevator. It will be responded by a promise if the floor is not equal to the current floor, and if it is in the scope of the orientation of the elevator. Otherwise it will be declined.*

As mentioned earlier, the destination buttons can be illuminated (by bulbs behind the panels). When the interrupt service routine in the program receives a destination button interrupt, it should send a signal to the appropriate panel to illuminate the appropriate button. This signal is sent by the program's loading the number of the button into the appropriate memory-mapped output register (there is one such register for each elevator). The illumination of a button notifies the passenger(s) that the system has taken note of his or her request and also prevents further interrupts caused by additional (impatient?) pressing of the button. When the controller stops an elevator at a floor, it should send a signal to its destination button panel to turn off the destination button for that floor.

*The illumination of the destination button of some Floor in some Elevator counts as the being promised of a transaction T2 for this Floor and Elevator.*

There is a floor sensor switch for each floor for each elevator shaft. When an elevator is within eight inches of a floor, a wheel on the elevator closes the switch for that floor and sends an interrupt to the computer (there is a separate interrupt for the set of switches in each elevator shaft). When the computer receives one of these (vectored) interrupts, its program can read the appropriate memory mapped eight-bit input register (there is one for

each interrupt, hence one for each elevator) that contains the floor number corresponding to the floor sensor switch that caused the interrupt.

*These interrupts are modelled as facts of the kind "position of Elevator is Floor", contained in the aggregate transaction kind AT2 (elevator position).*

The interior of each elevator is furnished with a panel containing one illuminable indicator for each floor number. This panel is located just above the doors. The purpose of this panel is to tell the passengers in the elevator the number of the floor at which the elevator is arriving (and at which it may be stopping). The program should illuminate the indicator for a floor when it arrives at the floor and extinguish the indicator for a floor when it leaves a floor or arrives at a different floor. This signal is sent by the program's loading the number of the floor indicator into the appropriate memory-mapped output register (there is one register for each elevator).

*Every time an elevator is approaching a floor, the controller creates a fact of the kind "position of Elevator is Floor". At every moment, there is exactly on such fact for every elevator. These facts are contained in the aggregate transaction bank AT1 (position).*

Each floor of the building is furnished with a panel containing summons button(s). Each floor except the ground floor (floor 1) and the top floor (floor 40) is furnished with a panel containing two summons buttons, one marked UP and one marked DOWN. The ground floor summons panel has only an UP button. The top floor summons panel has only a DOWN button. Thus, there are 78 summons buttons altogether, 39 UP buttons and 39 DOWN buttons. Would-be passengers press these buttons in order to summon an elevator. (Of course, the would-be passenger cannot summon a particular elevator. The scheduler decides which elevator should respond to a summons request.) These summons buttons can be illuminated by signals sent from the computer to the panel. When a passenger presses a summons button not already lit, the circuitry behind the panel sends a vectored interrupt to the computer (there is one interrupt for UP buttons and another for DOWN buttons). When the computer receives one of these two (vectored) interrupts, its program can read the appropriate memory mapped eight-bit input register that contains the floor number corresponding to the summons button that caused the interrupt. Of course, the circuitry behind the panel writes the floor number into the appropriate memory-mapped input register when it causes the vectored interrupt.

*Pressing a summons button at some Floor for some Direction counts as performing the request of a transaction T1 for this Floor and Direction to the functionary on the floor. It will be responded by a promise if the floor is not equal to the current floor. Otherwise it will be declined.*

The summons buttons can be illuminated (by bulbs behind the panels). When the summons button interrupt service routine in the program receives an UP or DOWN button vectored interrupt, it should send a signal to the appropriate panel to illuminate the appropriate button. This signal is sent by the program's loading the number of the button in the appropriate memory-mapped output register, one for the UP buttons and one for the DOWN buttons. The illumination of a button notifies the passenger(s) that the system has taken note of this or her request and also prevents further interrupts caused by additional pressing of the button. When the controller stops an elevator at a floor, it should send a signal to the floor's summons button panel to turn off the appropriate (UP or DOWN) button for that floor.

*The illumination of the summons button at some Floor in some Direction counts as the being promised of a transaction T1 for this Floor and Direction.*

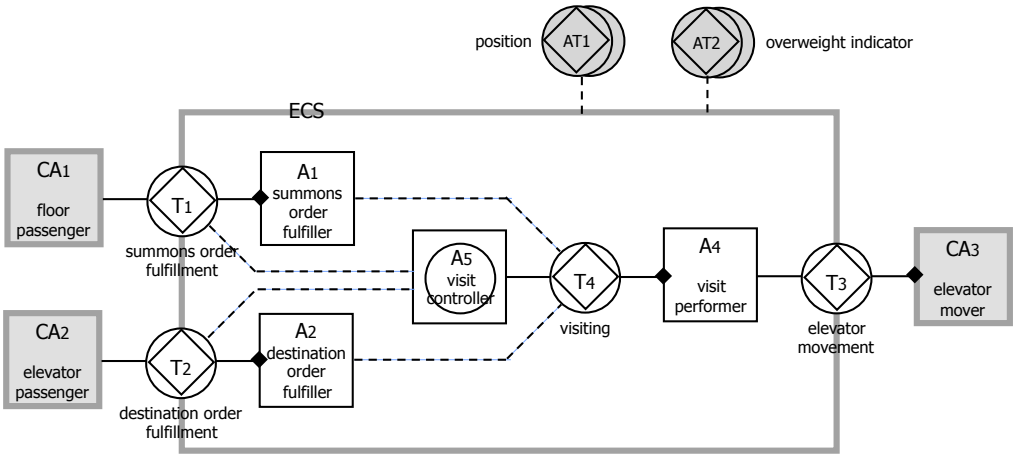
There is a memory-mapped control word for each elevator motor. Bit 0 of this word commands the elevator to go up, bit 1 commands the elevator to do down, and bit 2 commands the elevator to stop at the floor whose sensor switch is closed. The elevator mechanism will not obey any inappropriate or unsafe command. If no floor sensor switch is closed when the computer issues a stop signal, the elevator mechanism ignores the stop signal until a floor sensor switch is closed. The computer program does not have to worry about controlling an elevator's doors or stopping an elevator exactly at a level (home) position at a floor. The elevator manufacturer uses conventional switches, relays, circuits, and safety interlocks for these purposes so that the manufacturer can certify the safety of the elevators without regard for the computer controller. For example, if the computer issues a stop command for an elevator when it is within eight inches of a floor (so that its floor sensor switch is closed), the conventional, approved mechanism stops and levels the elevator at that floor, opens and holds its door open

appropriately, and then closes its door. If the computer issues an up or down command during this period (while the door is open, for example), the manufacturer's mechanism ignores the command until its conditions for movement are met. (Therefore, it is safe for the computer to issue and up or down command while an elevator's door is still open.) One condition for an elevator's movement is that its stop button not be depressed. Each elevator's destination button panel contains a stop button. This button does not go to the computer. Its sole purpose is to hold an elevator at a floor with its door open when the elevator is currently stopped at a floor. A red emergency stop switch stops and holds the elevator at the very next floor it reaches irrespective of computer scheduling. The red switch may also turn on an audible alarm. The red switch is not connected to the computer.

*The actor role A4 (visit performer) can initiate transactions T3 (elevator movement) with product kind "Elevator is set to move in Direction", where Direction is "up" or "down" or "stop".*

**Solution**

Based on the provided analysis, we produce right away the detailed crispienet diagram of the elevator control system, as well as the Transaction Product Table (TPT), in Figure 5.1.




| transaction kind | product kind |
|---|---|
| T1 summons order fulfillment | P1 summons order for Floor in Direction is fulfilled |
| T2 destination order fulfillment | P2 destination order for Elevator to Floor is fulfilled |
| T3 elevator movement | P3 Elevator is set to move in Direction |
| T4 visiting | P4 Elevator visits Floor in Orientation |
| T5 visit control | P5 visit control for Period is done |

Figure 5.1. Detailed crispienet diagram and TPT of the ECS

Only the internal crispie A2 (destination order fulfiller) will be specified hereafter, for the sake of space, just as a demonstration of how a crispie specification looks like.

**Specification of crispie A2**

**C2** = {destination order fulfillment **for** (Elevator, Floor) is requested,
destination order fulfillment **for** (Elevator, Floor) is promised,
visiting **for** (Elevator, Floor, Orientation) is stated,
visiting **for** (Elevator, Floor, Orientation) is accepted}

**I2** = *the set of intentions of the standard transaction patterns (revocations are disregarded)*

**S2** = {position **of** Elevator **is lower than** Floor, position **of** Elevator **is higher than** Floor,
position **of** Elevator **is equal to** Floor,
orientation **of** Elevator **is** up, orientation **of** Elevator **is** down, orientation **of** Elevator **is** idle,
overweight indicator **of** Elevator **is true**, overweight indicator **of** Elevator **is false**}

**P2** = {destination order for Elevator to Floor id fulfilled, Elevator visits Floor in Orientation}

The rule base **R2** is specified in Figure 5.2.

```
when    destination order fulfillment for (Elevator, Floor) is requested
        if      overweight indicator of Elevator is false
        then
                if      position of Elevator is lower than Floor
                then    if      orientation of Elevator is up or orientation of Elevator is idle
                        then    promise destination order fulfillment for (Elevator, Floor)
                        else    decline destination order fulfillment for (Elevator, Floor)
                else-if position of Elevator is higher than Floor
                then    if      orientation of Elevator is down or orientation of Elevator is idle
                        then    promise destination order fulfillment for (Elevator, Floor)
                        else    decline destination order fulfillment for (Elevator, Floor)
                else-if position of Elevator is equal to Floor
                then    decline destination order fulfillment for (Elevator, Floor)

when    destination order fulfillment for (Elevator, Floor) is promised
        request visiting for (Elevator, Floor, Orientation)
                with Orientation is orientation of Elevator

when    destination order fulfillment for (Elevator, Floor) is promised
        while   there is some Orientation
                for which visiting for (Elevator, Floor, Orientation) is accepted
        execute destination order fulfillment for (Elevator, Floor)
        state destination order fulfillment for (Elevator, Floor)

when    visiting for (Elevator, Floor, Orientation) is stated
        then    accept visiting for (Elevator, Floor, Orientation)
```

Figure 5.2. Rule base of crispie A2

**Discussion**

Modelling a technical system as a social system provides a deeper insight in the nature of the system, as was illustrated above for the elevator control system. In addition, the application of the complete transaction pattern reveals that revocations are not dealt with in the original description of the case ECS. So, for example, revoking a destination order is impossible: the elevator will stop at the selected floor, even if this was not the intention of the passenger. In software engineering, revocations are commonly considered as exceptions, whereas there is nothing more 'normal' for human beings than to revoke previous acts. Although surrounded by cultural and legal rules, we accept from each other that we make mistakes and that we like to make them 'undone'.

Moreover, the crispie model of the ECS makes one reconsider, in a very natural way, the whole set-up of elevator control systems: why should a passenger who wants to go from floor A to floor B have to say first to the functionary at floor A that he wants to go up or down, and later to the functionary in an elevator car that he wants to go to floor B? This sounds quite weird. So, a major improvement from the social point of view would be to let passengers be the initiator of a new transaction kind "transportation" with product kind "transport from FloorA to FloorB is performed".

## 5 Conclusions

The ψ-theory provides a way of understanding the construction and operation of organisations that appears to be very useful and applicable, as numerous practical applications have proven (see www. ee-institute.org). The most distinctive feature is undoubtedly that it is able to reconcile two different approaches to organisation and ICT that seemed to be irreconcilable up to now. One of them is the social ('soft') approach that is primarily concerned with people as members of organisations. The other one is the technical ('hard') approach that is primarily concerned with applying technology.

The 'soft' approach is fully accommodated by the general ψ-theory. It goes beyond the current abilities of organisational science, by really putting human beings in the centre of the organisation, and by introducing the notions of transaction and actor, as well as the related notions of authority, responsibility and competence, thereby demonstrating convincingly that an organisation chart does not reflect the construction of the organisation.

The 'hard' approach is fully accommodated by the special ψ-theory. It goes beyond the current abilities of computer science, by really integrating the traditionally separately dealt with areas of business processes, business objects and data, and business rules. In addition, it provides the insight that an enterprise information system is some realisation and implementation of (a part of) the essential model of the enterprise. An important practical implication of this insight is that the requirements engineering problem is solved, once and for all, because the requirements (derivable from the essential model) are guaranteed relevant and complete.

In addition to these advantages, the ψ-theory allows for a 2-step reduction of complexity. The first step is provided by the organisational building block (Figure 2.4), representing the complete transaction pattern, combined with a focussing on the construction model of the organisation. In this way, a reduction of complexity, in terms of size of models, is achieved of about 80%. This figure arises from comparing the 4 basic steps (request, promise, state and accept) with the 20 steps in the complete pattern. The second step is provided by the intelligent system partitioning (Figure 3.1). By focussing on the ontological model of the B-organisation (so the enterprise's essential model), in which the support by the I-organisation (including its support by the D-organisation) is represented by information links (interstriction), another reduction of complexity is achieved of about 85%. This is figure is an reasoned estimation supported by practical evaluations. Summing up, the total reduction of complexity is well over 90%.

## References

Austin, J.L.: *How to do things with words*. Harvard University Press, Cambridge MA 1962

Bunge, M.A.: *Treatise on Basic Philosophy, vol.4, A World of Systems*. D. Reidel Publishing Company, Dordrecht, The Netherlands 1979)

Dietz, J.L.G.: Designing Technical Systems as Social Systems, *Proc. of the 8th International Working Conference on the Language-Action Perspective on Communication Modeling* (LAP 2003), July 1-2, Tilburg, the Netherlands, ISBN 90-5668-119-2.

Dietz, J.L.G.: *Enterprise Ontology*. Berlin, Springer, 2006

Dietz, J.L.G., Hoogervorst, J.A.P. et. al: The Discipline of Enterprise Engineering. In: *Int. J. Organisational Design and Engineering*, Vol. 3, No. 1, 2013, pp 86-114

Habermas, J.: *Theorie des Kommunikatives Handelns*, Erster Band, Suhrkamp Verlag, Frankfurt am Main, 1981

Jong, J. de: *A Specification Framework for Enterprise Ontology based Design of Enterprise Information Systems*, dissertation Delft University of Technology, 2013

Kervel, S.J.H. van: *Ontology driven Enterprise Information Systems Engineering*, dissertation Delft University of Technology, 2012

Langefors, B.: Information System Theory. *Information Systems* **2**, 207–219 (1977)

Perinforma, A.P.C.: *The essence of organisation*, Sapio, 2013

Searle, J.R., *Speech Acts, an Essay in the Philosophy of Language*, Cambridge University Press, Cambridge MA, 1969

**List of TEEMs (Theories in Enterprise Engineering Memorandum)**

TEEM-1: the ω-theory (OMEGA: Organisation's Management, Engineering & Governance Appreciation)

TEEM-2: the τ-theory (TAO: Teleology Across Ontology)

TEEM-3: the δ-theory (DELTA: Discrete Event in Linear Time Automaton)

TEEM-4: the ϕ-theory (FI: Fact and Information)

TEEM-5: the ψ-theory (PSI: Performance in Social Interaction)

TEEM-6: the σ-theory (SIGMA: Socially Inspired Governance and Management Advancement)

TEEM-7: the π-theory (PI: Performance in Interaction)

TEEM-8: the β-theory (BETA: Binding Essence to Technology under Architecture)

TEEM-9: the ν-theory (NU: Normalised Unification)