Chapter 1

INTRODUCTION

Numerous researchers and practitioners are currently inspired by the goal of proposing innovative ideas and solutions about a better *utilization of advanced IT by enterprises*. That goal is reflected in the *evolution of business processes*: Considered as an essential enterprise asset, *business processes* used to receive much attention, for the sake of improving the enterprise performance, decreasing the enterprise costs, increasing the satisfaction of customers, and so on. Hence, it was widely agreed that by improving business processes, enterprises could substantially increase their value. Many years ago, improving business processes was a matter just of enterprise engineering – then the big challenge was how to organize ordering, accounting, shipping, etc., such that all the different tasks are in synch while the business processes are as simple as possible, leading to effectiveness and efficiency in serving the customer. Nevertheless, changes in business processes came *when computers first appeared* on the scene and it was possible to replace paper streams by databases, to re-use content, and to quickly find needed information – then the big challenge already was how to make better use of computers heavily dependent, in turn, on corresponding *software*: this was a matter also of software engineering (next to *enterprise engineering*). Hence, *enterprise engineering* and *software engineering* had to be brought together [54]. However, those two disciplines developed separately because the so-called 'computerization' was simply about *automation* – the same tasks realized by human entities had to be 'given' to computers. *Automation* indeed allowed many companies to tremendously bring down their workforce but the quality of the IT support delivered to *enterprises* used to be low exactly because of the mentioned 'separation': *Enterprise engineers* would only superficially re-design their *business processes* (when bringing in computers) because they lacked deep IT knowledge while *software engineers* would only partially respond to the original business *requirements* because they lacked deep domain knowledge. This was labeled as a '*mismatch (or gap) between enterprise modeling and software design*'. Since the new millennium, we have been witnessing more and more efforts directed toward *bringing together enterprise engineering and software engineering*, for the sake of bridging the above-mentioned *gap*. This would mean de facto bringing together:

(a) social theories, such as *enterprise ontology*, *organizational semiotics*, *theory of organized activity*, etc. (see Chapter 3);
(b) computing paradigms, such as *component-based software development*, *service-oriented computing*, *model-driven engineering*, etc. (see Chapter 4).

However, this appeared to be a non-trivial task because:

- Within the scope of *enterprise engineering*, as according to [19], used to be the creation of enterprise models capable of usefully restricting the software system-to-be, but this only reached the level of software functionality specification, leaving ambiguity with regard to the implementation choices, platform choices, networking choices, and their impact with regard to the business processes.
- Within the scope of *software engineering*, as according to [66], used to be the development of software, based on computation-independent models and/or the composition of software services (considered at high level and pointing at underlying technical complexity), but all those issues stemmed from a view on the software itself, not assuming an enterprise-modeling-driven derivation of software.

Hence, not bridging that *gap* has led and is currently leading to *failures* of many (current) software projects as well as to *projects going over budget*, and we often observe evidence of *low levels of customer satisfaction* with regard to software applications and/or (enterprise) information systems [8].

Further, the above-mentioned *gap* is pointing not only at the *creation of software* as a way of allowing enterprises to utilize advanced IT but also at the *integration of already created (legacy) IT systems in enterprises*. We observe that many software systems being developed need to be adequately *integrated* in their enterprise context and sometimes already running software applications are 'part' of that enterprise context. For this reason, it is essential to have *alignment* and *traceability* between the enterprise level and the software level, and therefore it seems logical to try identify *enterprise systems* and *software systems*, and bridge the two on that basis [54]. As is well-known, when speaking of a system, we are interested in what the system components are (*composition*), how they are related to each other (*structure*), how they are related to the *environment*, and what the principles are that guide the system evolution. We need an integrated view of the system under consideration and for an enterprise this would point at a coherent whole of principles, methods and models that are used in the design and realization of the enterprise's structure, processes, (possibly) information systems, and infrastructure. Even though *structure*, *processes*, and *data* are essential for software development as well, *more complexities occur when developing software* (coming through analysis, design, and implementation) - what lags behind is managing system complexity expressed in terms of *dependencies between system elements*. Finally, current enterprises and software applications both need to be *adaptable* because of the constantly changing real-life environment to which they should conform. This all raises a number of **`challenges`**, the most important of which are:

- Identifying the enterprise system(s) and/or the software system(s) to be considered;
- Building aspect models accordingly, including models that reflect structure, processes, data, and so on;
- Establishing inter-model consistency;
- Capturing the granularity levels that feature the enterprise models and the (corresponding) software models, acknowledging that it is possible that the particular enterprise models and software models point at different granularity levels;
- Establishing alignment and traceability between enterprise models and corresponding software models;
- Addressing possible dependencies between system elements;
- Allowing for ways to model adaptability.

Referring to [8], we are inspired to consider several **`solution directions`** relevant to the above-mentioned challenges:

- **Modeling viewpoints and overall consistency**: No matter if one would model an enterprise or software, different *modeling viewpoints* can be applied – one could look at structure, behavior, data, and so on. Hence, only if *overall consistency* is achieved, such models would have sufficient value because modeling structure with no grasp on behavioral aspects or modeling behavior with no grasp on data issues (for example) would be of limited use. Further, projecting this also over the enterprise-software 'bridge' would add value. This would mean emphasizing the similarities between enterprise systems and software systems, despite their specific differences, such that it is possible *to create enterprise models and software models which are 'written in the same language'*. This would be the basis for bridging *enterprise modeling* and *software specification* – only when the specification of software is properly restricted by a corresponding enterprise model, it would be possible to develop software that adequately meets the original *business requirements*. For this reason, *consistency* is to be aimed not only 'within a system' *(among the different aspect models characterizing the system under study*) but also 'across systems' (in our case – *between models featuring enterprise systems and models featuring software systems*).
- **Integrating data analytics in enterprise modeling and software development**: Current information system development assumes *increasing importance of data analytics*. Data has always been important functionally and non-functionally in both modeling enterprises and specifying software: at design time, we use (statistical) data for making our models more realistic, while at run time, we use 'incoming' environmental data for adapting the system behavior accordingly. A question however is: what does make difference today, compared to several years ago and why is the emerging discipline of *data science* receiving so much attention currently? Is a reason for that the current *abundance of (sensor) data* showering us every day and if yes, how are we coping with this abundance – distinguishing between the really useful data and the data that may be ignored? Further, it is important to know how we translate

low-level (sensor) data into *higher-level information* that is a basis for *reasoning*, and as well how we know which data to *trust*. Currently, those questions are even more important than in the past. The *integration of data analytics* in *enterprise modeling* and *software development* is hence not only about establishing the context situation or providing a statistical data modeling background but it is also about other issues, such as quality-of-data, occurrence probabilities, data formats, and so on. Hence, in aligning enterprise modeling and software specification, it is important to take those issues into account.

- **Supporting adaptability**: When developing an information system, we usually aim at making it *adaptable* with regard to environmental changes. At the same time, there are restrictions which are two-fold: *(i)* the system behavior 'patterns' through which adaptability would be realized, need to be foreseen and 'prepared' at design time; *(ii)* environmental changes are not always trivial to 'sense'; hence, it is important to know to what and how and information system should *adapt*, and also is this concerned with pre-defined (at design time) scenarios and/or with the run-time behavior of the information system. Further, if we assume that *adapting* means *adjusting behavior to a changing environment*, it would be interesting to also consider how we capture those changes (probably through *sensors*) and how we process and interpret this information (see above). Finally, all those issues point at context-awareness, assuming that *the system 'switches' to a particular behavior based on the appearance of a particular context state*. Thus, *context-awareness* is to be considered in the enterprise perspective, in the software perspective, and also with regard to the alignment between *enterprise modeling* and *software specification*.
- **Considering re-usable modeling patterns**: It is widely agreed that if possible, modeling should be based on *re-usable modeling building blocks*, such that the modeling itself is more *effective* and *efficient* [54]. For this reason, in aligning *enterprise modeling* and *software specification*, it would be good to identify corresponding modeling building blocks and their in turn corresponding mapping – this would allow for bridging the gap between *enterprise modeling* and *software design* in a *component-based way*. Nevertheless, it is still a question how to methodologically identify enterprise modeling patterns and reflect them in corresponding software components, as it will be further discussed in the current book. We still miss exhaustive guidelines on how to realize this, taking into account *granularity concerns*, *traceability concerns*, and *re-use concerns*. Further, a shift to service-based systems is often the case since more and more current software applications provide support to their users through *services* (running software instances), as will be discussed in Chapter 4. This in turn leads to questions because developers are often with limited or no control over the software components which are running the services that are used, but developers should still keep this aligned with corresponding enterprise models and particularly - the modeling patterns related to them. Hence, those concerns need to be reflected in the way we look at the enterprise-software relationship. This would help developers in their succeeding *to align enterprise modeling and software specification in a component-based way*.

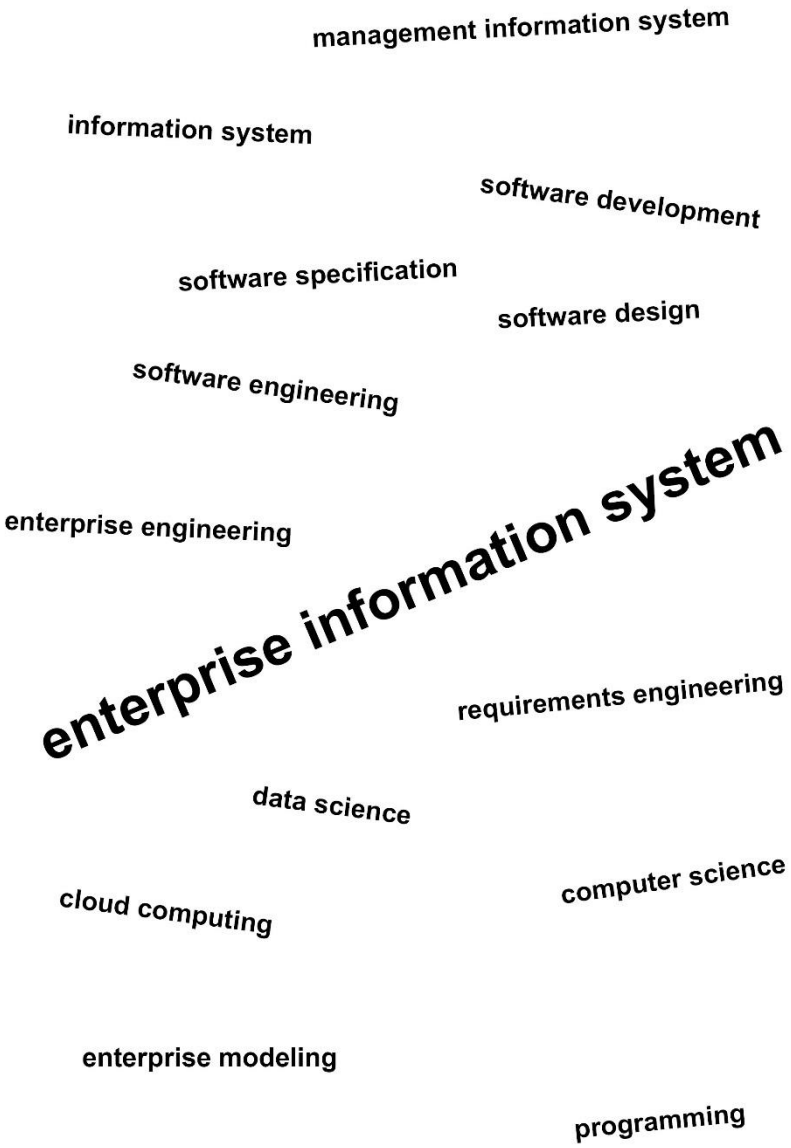

Fig. 1.1. Labels of disciplines and areas.

- **Enabling service-orientation**: As mentioned above, it is often that customers utilize IT/software, by composing web *services* (or '*services*', for short) – this allows for letting the *users* consider *services* at high level, not being burdened by their underlying complexity, while at the same time, *developers* consider the corresponding software components running the *services*. That is how *services* relate to both *enterprise engineering* and *software specification*. For this reason, in aligning *enterprise modeling* and *software specification*, it is important to assume the possibility that the resulting software would be *service-oriented* and if necessary – re-design the enterprise accordingly.

We argue that the above-presented solution directions are realistic, balanced, and relevant to the identified challenges. Nevertheless, those solution directions need to be positioned conceptually, such that it is possible to consider them from a scientific perspective, minding numerous labels that point to (overlapping) disciplines, areas, terms, etc. (as illustrated in Figure 1.1), related to information systems that support enterprises.

As it is seen from the figure, there are labels pointing at different relevant disciplines or areas. Nevertheless, even though some of those labels are more widely accepted than others, we argue that more precision is needed in this regard and we put forward several questions as justification for that observation:

- Is 'computer science' covering only software-development-related issues or is it also covering enterprise modeling that may be relevant to the development of software?
- What is the difference between 'computer science' and 'data science', and is 'computer science' not covering data analytics or is 'data science' focused on data aspects only, not touching upon other related issues?
- Should we consider 'requirements engineering' as a part of 'enterprise engineering', if we stress upon the original business requirements or should we consider 'requirements engineering' as a part of 'software engineering, if we stress upon the user-defined technical requirements that straightforwardly concern the specification of software?
- Is 'cloud computing' only about the utilization of cloud resources or is 'cloud computing' also about the software-related issues concerning this?
- Is the label 'management information system' referring to the management of information systems, assuming a technical-independent view?
- and so on.

Those are just several questions that are not 'expecting' answers. Instead, we use those questions to inspire a discussion on how to position and label our work that concerns enterprises and the software support they are utilizing. We realize that there are two disciplines essentially underlying the issues discussed above:

- ENTERPRISE ENGINEERING;
- SOFTWARE ENGINEERING.

Enterprise engineering is about analyzing, modeling, and (re-)designing an enterprise without considering anything in a technology-specific perspective. Said otherwise, we are interested in the entities (observed within the enterprise under study), their relations, and corresponding processes, no matter if the entities are human beings

or technical devices (we may consider technical devices but we abstract from their internal technical complexity).

As for software engineering, firstly, it should have a focus – there maybe software developed for cars, or software developed for hospital equipment, or software embedded in devices, and so on; we particularly focus on enterprise software. Further, the software engineering scope is the software system-to-be. Finally, with regard to the software system-to-be, we take a technology-specific perspective. Said otherwise, we are interested in the technical complexity inside the software system-to-be.

- Our bringing together enterprise engineering and software engineering would point at what we label as:

ENTERPRISE INFORMATION SYSTEMS.

We therefore make a clear distinction between issues that concern the enterprise-engineering aspects of *enterprise information systems* and issues that concern the software-engineering aspects of such systems. For this reason, any relevant discipline or area of interest, as the ones presented in Figure 1.1, is to be 'positioned' with regard to either enterprise engineering or software engineering. Bridging the two is a matter of a dedicated approach, as it will be further studied in the current book.

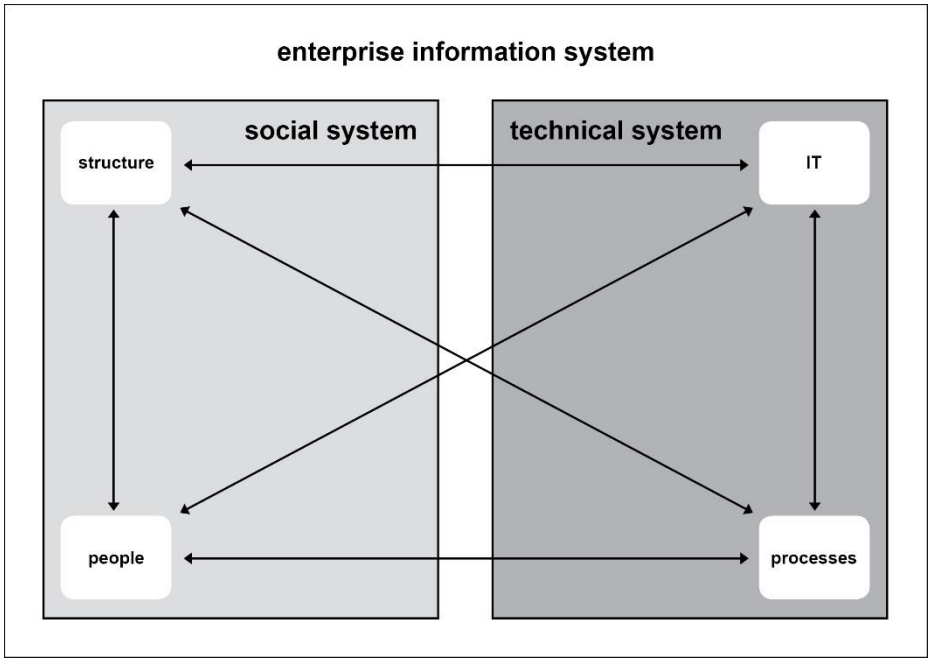

Fig. 1.2. Viewing an enterprise information system as a socio-technical system.

Further, enterprise engineering concerns *enterprise systems* while software engineering concerns *software systems*:

- The former we consider as SOCIAL SYSTEMS;
- The latter we consider as TECHNICAL SYSTEMS.

This inspires our viewing enterprise information systems as socio-technical systems – Figure 1.2 [36] and taking an abstract perspective accordingly.

In line with this and as suggested by the figure, we may distinguish four primary components that must be balanced and 'work together' such that the information processing functionalities required by an enterprise to fulfill its information needs are adequately delivered. There are also corresponding 'driving forces' among the four components, as the figure shows. Hence:

- The HUMAN ELEMENT of an enterprise information system concerns the people and corresponding (organizational) structures;
- The TECHNICAL ELEMENT of an enterprise information system concerns the IT resources + services as well as corresponding (software) processes.

Thus, *IT services* and *technical processes* are supporting not only particular *human entities* but also *organizational units* as such. At the same time the *human entities* are functioning within corresponding *organizational units*. Further, *IT services* and *technical processes* are essentially 'fueled' by actions realized by particular *human entities* and also by collective actions realized by particular *organizational units*; in this the *IT services* and the corresponding *technical processes* are to be in synch.

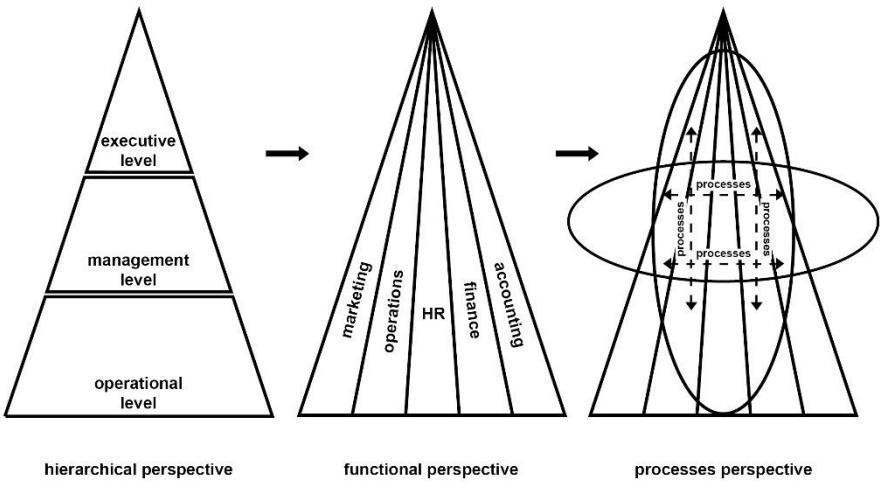

Fig. 1.3. Hierarchical, functional, and processes organizational perspectives.

Since the application area concerning *enterprise information systems* is the area of *enterprises* and *enterprises* in turn represent ORGANIZATIONS, we need to have a good *overall organizational perception* and inspired by [36], we consider accordingly three essential perspectives, as depicted in Figure 1.3.

As the figure suggests:

- The HIERARCHICAL PERSPECTIVE (assuming a centralized organization) features three primary levels in an organization where specific to each level of activity and decision making events take place: *(i)* At the operational level, short-term, highly structured activities are performed and the objective is an efficient processes under a limited degree of uncertainty (hence, recurring operations allow to be conveniently automated, assuring in this was speed, accuracy, and precision in their execution); *(ii)* At the management level, semi-structured (decision-making) activities are performed, mainly related to functional areas, and focused on the execution and control over processes, based on adopted patterns and proven models (hence, the typical IT support in this context would come through decision-support systems that are founded on enterprise-internal operations and resources); *(iii)* The executive level handles all strategic planning and ad hoc circumstances, prioritizing long-term and wide-range decisions (hence, they typical IT support in this context would come through executive information systems that are capable of collecting, analyzing, and synthesizing organizational and external trend data).
- The FUNCTIONAL PERSPECTIVE (assuming a decentralized organization) features business entities based on distinct functional areas such as *marketing*, *operations*, *human-resources*, *finance*, *accounting*, and so on.
- The PROCESSES PERSPECTIVE utilizes *top-down* methodology to achieve *internal business integration*, *activities rationalization*, and *duplications elimination* across functional areas and managerial levels.

Further, the two left-to-right arrows in the figure suggest an evolution over time from *hierarchical* organizations through *functional* organizations to *process-oriented* organizations, each of which has advantages and limitations. Still, we consider the *processes perspective* as most appropriate with regard to *enterprise information systems* because structures of processes are appropriate as basis for utilizing software support.

Finally, even though we acknowledge the *socio-technical components* and corresponding *driving forces* (as featured in Figure 1.2) and the *organizational perspectives* (as featured in Figure 1.3), we claim that a sound approach to *enterprise information systems* should assume a reference to the underlying disciplines (namely: *enterprise engineering* and *software engineering*) and corresponding theories / paradigms (namely: social theories and computing paradigms), as exhibited in Figure 1.4.

As it is seen from the figure, we observe both *human entities* and *technical entities* not only within any *enterprise information system* but also within its *environment*. Further, *human entities* as well as their relations and behavior are to be addressed through *social theories* (as it will be discussed in Chapter 3); *technical entities* and their operation are to be addressed through *computing paradigms* (as it will be discussed in Chapter 4).

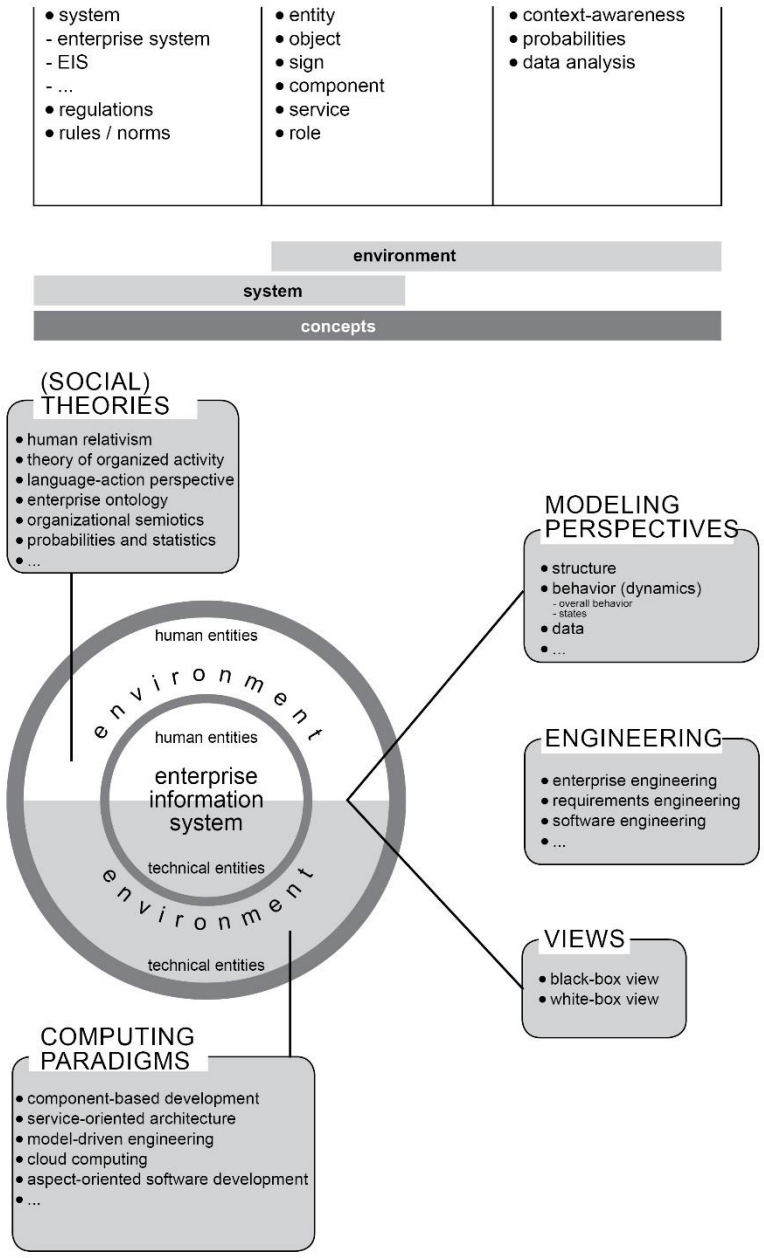

Fig. 1.4. Enterprise information systems – a modeling approach.

Inspired by [54] and acknowledging the *gap* between *enterprise modeling* and *software specification* (as discussed already in the current chapter), we consider a *modeling approach* toward *enterprise information systems*, grounding it in the disciplines and corresponding theories / paradigms, as mentioned above and adding further elaboration in terms of *modeling viewpoints*, as follows:

- Enterprise engineering is instrumental with regard to real-life enterprise processes while software engineering is instrumental with regard to related technical (IT) issues; requirements engineering concerns both since there are not only (original) business requirements but also technical (user-defined) requirements.
- Especially (social) theories are to be considered, touching upon human entities and corresponding real-life behavior, and in particular:
  - Human relativism (featuring human-centricity in enterprise modeling);
  - Theory of organized activity (useful in modeling human behavior);
  - Language-action perspective (useful in modeling language-driven communicative acts);
  - Enterprise ontology (useful in modeling coordination and production);
  - Organizational semiotics (useful in modeling signs and business rules);
  - Probabilities and statistics (useful in modeling surrounding context).
- Especially computing paradigms are to be considered, touching upon technical entities and their operation, and in particular:
  - Component-based development (useful in specifying component-based software applications);
  - Service-oriented architecture (useful in specifying web services);
  - Model-driven engineering (useful in modeling based on abstractions);
  - Cloud computing (useful in modeling the utilization of distant resources);
  - Aspect-oriented software development (useful in modeling crosscutting non-functional concerns.
- Several modeling perspectives are to be considered, no matter if regarding enterprise engineering or regarding software engineering, namely:
  - Structure (how are different entities related to each other);
  - Dynamics:
    - What is the overall behavior of the considered entities;
    - What are the states an entity comes through;
  - Data.
- Depending on the purpose of modeling:
  - A functional (black-box) view would be appropriate if establishing what the system should do;
  - A constructional (white-box) view would be appropriate if establishing how the system should realize its functioning.
- In considering all this, a systemics approach is to be followed (see Chapter 2), such that the modeling focus is put:
  - Either on the system itself;
  - Or on the system environment.

- The concepts to be considered in this regard (in line with the study presented in Chapter 2) are:
  - Concepts relevant to the system scope:
    - System:
      - Enterprise system;
      - Enterprise Information System (EIS);
    - Regulations;
    - Business rules (also labelled 'norms').
  - Concepts relevant to the environment scope:
    - Context-awareness;
    - Occurrence probability;
    - Data analysis.
  - Concepts relevant to both:
    - Entity;
    - Object;
    - Sign;
    - Component;
    - Service;
    - Role.

Hence, taking a *modeling approach* with regard to *enterprise information systems* requires *interdisciplinary* efforts and *multiple perspectives* that are to be applied in synch.

This book tells you how to *bring together enterprise modeling and software specification*, such that an *enterprise-engineering-driven software generation is achieved* – this is considered crucial with regard to the development of *enterprise information systems*.

The remainder of the book is structured as follows:

CHAPTER 2 will introduce our systemics views, touching upon systems and their composition, and also the system environment and context of users, extending this to enterprise systems and enterprise information systems, and introducing a number of concepts accordingly.

CHAPTER 3 will present relevant social theories (as according to Figure 1.4), including: human relativism, theory of organized activity, language-action perspective, enterprise ontology, and organizational semiotics.

CHAPTER 4 will present relevant computing paradigms (as according to Figure 1.4), including: component-based development, service-oriented architecture, model-driven engineering, cloud computing, and aspect-oriented software development.

CHAPTER 5 will introduce the SDBC approach, presenting its foundations, outline, and notations, driven by the goal of proposing a way to bring together enterprise modeling and software specification for the sake of bridging the enterprise-software gap (as discussed already in the current section).

CHAPTER 6 will feature one case study and two illustrative examples, in order to demonstrate how enterprise engineering and software engineering can be brought together, supported by SDBC and enriched by an explicit consideration of user-defined requirements, and also how this can be extended to accommodate service-orientation and middle-out modeling.