Conceptual Modeling in Social and Physical Contexts

Roel Wieringa

University of Twente, Enschede, The Netherlands
roelw@ewi.utwente.nl
www.cs.utwente.nl/~roelw

Abstract. The history of the computing sciences shows a shift in attention from the syntactic properties of computation to the semantics of computing in the real world. A large part of this shift has been brought about by the introduction of conceptual modeling languages. In this paper I review this history from the early 1970s and identify the elements of real-world semantics that these notations have been used for. In the physical domains typical of control systems, conceptual modeling is always combined with causal modeling in order to register and control behavior in the domain. Because causal relationships are domain-specific, conceptual modeling languages in physical domains can be expected to evolve into domain-specific languages used by engineers. By contrast, in social domains causal modeling plays a minor role. In social domains conceptual models are shared by the people in the domain, and therefore constitute the domain. This creates a different mechanism for registration and control, in which events can be made to occur by means of social convention. Because conceptual models constitute the social world, we can expect conceptual modeling languages to evolve into domain specific languages here too, but in contrast to conceptual modeling languages in physical domains, they will be used as means of communication between engineers and members of the social domain. This paper ends with a plea for more specialization and less standardization in conceptual modeling.

1 Introduction

Thinking about the history of conceptual modeling introduces a problem: When did conceptual modeling start? When Chen introduced the entity-relationship notation in the mid-1970s [1]? When Von Neumann wrote his first computer program to sort data in the 1940s [2]? When Lady Lovelace designed a program to compute Bernoulli numbers in the mid-19th Century? When European merchants started to design procedures and accounting techniques for long-distance trade in the late middle ages [3]? Or when Sumerians in the second millenium BC designed computations to calculate the size and location of fields after they had been flooded [4]?

This raises a more fundamental question: What is conceptual modeling anyway? In the above cases, structures in reality were analyzed and symbolic structures were designed that can represent them, and that can be manipulated formally. But are all of the above activities, and more like them, included in what we usually call "conceptual modeling"? It is meaningless to discuss the "true" meaning of a word, and we should be content with defining an *interesting* meaning, that can provide us some insight into our world. In this paper, I will define *conceptual modeling* as the activity of capturing conceptual structures of a domain in notations that can be manipulated by a programmable computer. A *domain* is a part of the world that some stakeholders are interested in for some purpose. The concept of stakeholder arises because we deal with engineering: Producing artefacts useful for some stakeholders.

This puts the start of conceptual modeling in the early 1970s, when several methods foir representing domain structures were developed. In section 2 I will follow some highlights of the history of conceptual modeling until today. This history reveals a large diversity of conceptual modeling notations that however turn out to contain some shared underlying structures, which we will analyze in section 3. It turns out that the notations focus on a limited number of aspects of the domains and systems that they can describe, namely the decomposition of these systems and domains into parts, and the behavior and communication of these parts. Furthermore, each notation has been designed with a primary purpose in mind, such as specifying a software system or specifying a domain, but it invariably turned out they could also be used for other purposes. This means that all notations are conceptual modeling notations, but no notation is exclusively a conceptual modeling notation.

We will then move from the notations to their semantics and analyze the difference between physical and conceptual models, and between physical and social domains in section **??**. In physical domains, conceptual modeling is intertwined with causal modeling, and stands in the service of measurement and control. In social domains, by contrast, conceptual models have a constitutive function because conceptual models are shared by people in the domain, and have the function of defining the domain for a group of people, allowing these people to exercise control through conceptually defined activities. These activities in turn change the conceptual models that defined them. In this dynamics too, conceptual modeling stands in the service of measurement and control but the mechanisms are not causal, but created by social convfention. In section 4 I will conclude from this that further development of disciplines of conceptual modeling should proceed to design and study techniques for measurement and control in these two domains. I will also argue that progress will require specialization according to domain, where the distinction between physical and social domains is only the most general one. Conceptual modeling is an engineering activity, that has to include conditions of practice if it is to deliver relevant results, and this means that we must move from general claims about all conceptual modeling activities to specific claims about particular conceptual modeling activities.

2 Conceptual modeling notations since the 1970s

2.1 Entity-oriented approaches

In databases, the abstraction from computational structures to domain semantics was made in two steps. In 1970, Codd introduced the relational model to abstract from pointer structures in data storage [5]. This introduced a logical view on data in which storage structures had been abstracted from and what remained were variables, called attributes, and cartesian products of these variables, called relations. Codd concentrated on the mathematical structure of these cartesian products and was more concerned with abstracting from storage structures than with adding domain semantics. The second step was taken by Chen who added domain semantics by attaching attributes to entities, and recognizing relationships between entities [1]. In a conceptual data model, a cartesian product of attributes (a relation) could represent an entity with its attributes, or it could represent a relationship between entities. The resulting three-level view of databases has been described by the ANSI/SPARC in a reference architecture [6], in which the storage structure view is called “physical”, the relational view is called “logical” and the Er view “conceptual”. Usually, a fourth level is recognized, at which user views on the data are defined, one for each user group.

We should recognize that there is nothing physical about the “physical” database view. Pointers and other computation structures are abstractions from a physical computer, and all view mentioned above are conceptual. The difference is in the domains they describe: The “physical” view describes computational structures, the logical view describes abstract mathematical structures, and the conceptual and user views describe domains of interest to stakeholders of the system.

Numerous extensions to the ER notation have been defined, notably extensions of cardinality constraints and extensions that dealt with time and history [7,8,9]. If we view an ER model as a representation of a conceptual domain structure at an arbitrary instant of time, then cardinality constraints restrict how many entities a given entity can be related to at an arbitrary instant of time. Temporal extensions of the ER notation offer ways of expressing constraints on how entities, their relationships and their cardinality constraints can evolve over time. Historical extensions allow one to express constraints on the history of entities or relationships.

Several other proposals for expressing domain structures were made in the 1970s, notably a proposal by Smith & Smith to express taxonomy and aggregation structures [10], and a less well-known proposal by Hall to clearly distinguish a domain entity from its surrogate in a computing system [11]. Modeling taxonomic domain structures was often referred to as *semantic data modeling* [12,13]. By the early 1990s, ER approaches had stabilized and textbooks on ER modeling started to appear [14].

In parallel with the development of the ER approach, an approach was developed variously referred to as “Nijssen’s Information Analysis Method”, “Natural language Information Analysis Method” (both NIAM) and Object-Role Modeling (ORM) [15,16]. The core idea of NIAM/ORM is to describe a domain in elementary sentences of subject–verb-object structure and represent this as a set of binary relationships. NIAM/ORM recognizes taxonomic structures too.

A third approach to modeling taxonomic domain structures arose in knowledge representation, using notations motivated from linguistics such as semantic networks [17,18]. This line of research has always been closely linked to various logic specification languages and today, ontology languages based on description logic have been developed to describe the semantics of data found on the World Wide Web. Ontology languages allow one to define conceptual models of domain concepts and their taxonomic relations and they facilitate automated reasoning about taxonomic structures. [19]. Analysis and design of these languages tends to ignore the conceptual modeling approaches listed above but uses results from philosophical logic and formal logic [20,21].

The conceptual modeling approaches listed so far all focus in one way or another on entities and relationships that can be found in domains, where anything that can be classified and counted can be treated as an entity—people, products, services, processes, events and organizations, money, etc. I will call them *entity-oriented.* Some people will restrict conceptual modeling to entity-oriented approaches but I think that this view is too restrictive. There are more conceptual structures in domains than entities that can be classified and counted.

2.2 Communication-oriented approaches

A totally different approach to abstract from computational structures originated in the early 1970s with the ISAC approach to developing information systems proposed by Langefors, Lundeberg and others [22,23,24]. ISAC identified the change needs in an organization, analyzed business processes to be supported by an information system, and then identidfied the information needs of these processes before it turned to specifying the information system itself. I will call ISAC a *communication-oriented* conceptual modeling technique because it focussed on charting the communication needs of an organization. Where entity-oriented approaches focussed on what stakeholders communicated *about*, communication-oriented approaches focus on who talks to whom and what for.

Another communication-oriented approach came to be known as structured analysis, proposed in different variants by Ross [25,26], DeMarco [27], and Gane & Sarson [28]. Structured analysis was proposed as a method to identify the desired functions of software systems independently from its internal structure. The data flow diagram (DFD) notation used in structured analysis represents tasks to be performed in an organization, data that is communicated from one task to another, either synchronously or asynchronously (through data stores).

2.3 Behavior-oriented approaches

In the 1980s structured analysis was extended by Ward & Mellor [29] and Hatley & Pirbhai [30] with notations for behavior, in particular state transition tables and diagrams, to represent actual or desired behavior in domains to be controled by software. This introduced a third group of notations that are *behavior-oriented.* State transition diagrams defined by Mealy and Moore had been in use since the 1950s to design discrete hardware systems, but since their addition to structured analysis they can be be used in the design of control software too. Harel combined and extended these notations into a very powerful notation called statecharts, that allows specification of complex reactive behavior [31,32]. This could be behavior of a software system or behavior in a domain to be controled by a software system.

The distinction between domain and software is not always made in the methods described so far and in some methods, like structured analysis, the domain is present only as "external entities". Jackson System Development (JSD) is unique in making this distinction very sharply [33]. It models domain entities and events, allocates events to entities and then designs the system starting from an initial model, which duplicates domain structures, and then adds system components to register and control domain events according to the desired functions of the system. In retrospect, this can be viewed as an extension of Hall's [11] idea to distinguish entities from their surrogates in the system. JSD too offers a graphical behavior-oriented notation, based on regular expressions.

In the telecommunication area the System Description Language (SDL) was developed, which allowed decomposition of a system into blocks of which the behavior could be specified by finite state machines [34,35,36]. Block diagrams are communication-oriented, finite state machines are behavior-oriented. Communication scenarios could be illustrated using message sequence charts, a notation that became popular in object-oriented modeling.

Many of the foregoing developments found their way in the Unified Modeling Language (UML), which started out as a notation for documenting small Java programs assembled from notations used by Booch [37], Rumbaugh [38] and Jacobson [39]. The UML however contains mechanisms such as stereotypes and profiles that allow unlimited extension so that today, the UML should be viewed as a family of languages that continues to grow. And where it originally was used to model the conceptual structure of Java programs, it is now also used to model business processes and other domain structures.

3 Analysis

3.1 Not Yet Another Method

A detailed analysis of the above developments reveals that there are, scattered over the literature, a large number of excellent ideas for conceptual modeling and software specification that are in danger of getting lost or reinvented periodically, both of which is a waste of energy. I have created a framework with a number of dimensions according to which software specification notations can be classified and analyzed [40,41], and assembled the results into something that, to indicate that it is *not* a revolutionary new approach, I called Not Yet Another Method

(NYAM) [42,43]. Details are given at length elsewhere [43], and here I only describe the structure of NYAM.

Conceptual modeling notations are usually designed to express one of the following aspects of a software system or a domain:

– *Classification and counting.*
  For example, ER diagrams allow the analyst to represent the classes of entities that can exist in a domain, taxonomic relationships between these classes, and cardinality properties of these classes. ER diagrams and diagrams like it can be used to represent anything that can be classified and counted, including data and software in a computer.
– *Functionality.*
  The functionality of an artefact consists of useful external interactions of the artefact. In many cases this is described in natural language, but we saw that there are notations for two particular aspects of functionality, behavior and communication.
  ∗ *Behavior* is the way that actions can be sequenced, and behavior descriptions usually allow the analyst to indicate at what point choices between different futures are possible. We saw that state transition diagrams, in particular statecharts, are popular behavior notations.
  ∗ *Communication* is the way different entities exchange messages. DFDs and SDL block diagrams are well known notations to express communications.

  Some notations express more than one aspect, much as UML sequence diagrams (behavior over time, communication between actors) and UML use case diagrams (communication, allocation of functionality to actors).

NYAM consists of a set of well-known notations for these aspects, and the description of NYAM consists of a description of the syntax of each notation in it, and a compendium of guidelines for how to use the notations. These guidelines are given at length elsewhere [43]. Here, I want to point out that guidelines for using a notation are motivated by what the notations are supposed to be used for. For example, if we describe behavior in a domain, the concept of encapsulation of behavior in an object does not make much sense, because events tends to be shared by many objects. But in an object-oriented software system, the concept of encapsulation makes a lot of sense. And the notion of composite object is different in software from what it means in the physical world [43, page 234], and both differ from what we mean by composition in the social world. This shows that an old slogan once popular in the philosophy of language is valid in the design of conceptual modeling languages:

$$\boxed{\text{meaning} = \text{use}}$$

The meaning of a notation must be appropriate for the intended use of the notation. It follows also that the guidelines that we give for the use of a notation must match the meaning of the notation. And it is dangerous to unleash on software engineers a collection of notations for which *any* use is stated to be appropriate, as the UML. When such a notation is used, it is simply that: a notation. It means whatever the writer wants it to mean and there is no resource other than the writer's mind to figure out what an expression in it means. Conversely, once writer-independent semantics is defined, for example in standards, we must fix an intended use for the notation, such as software specification, business modeling, or data modeling, and refrain from using the notation for something else.

3.2 Three domains

To better understand what a notation is used for we should distinguish three worlds. The *physical world* is the world of time, space, energy and mass measured by kilograms, meters, second, Amperes, etc. The *social world* is the world of conventions, money, commercial transactions, business processes, job roles, responsibility, accountability, etc. structured in terms of conceptual models shared by people.

We may additionally distinguish the psychological world of mental states, which would be relevant for applications that affect mental states. Alternatively, we may view the psychological world as a part of the social world. We will ignore this issue in this paper.

At the interface between the social and physical worlds we have the *digital world* which consists of symbols that have a meaning for people. The digital world is physical, because symbol occurrences are physical. Paper, ink, memory states signals travelling through a wire, etc. are physical. But at the same time these symbol occurrences have a meaning that is, ultimately, defined by a convention chosen by a group of people. A photo printed on paper has a meaning to, but this is not defined by an arbitrary convention. By contrast, a JPG file describing a photo is a (highly structured) symbol occurrence. The characteristic feature of a symbol is that there is a dictionary that defines the meaning, and that the meaning would have been different had the dictionary been different.

Computers are physical symbol processing machines [44]. For the symbols that they process, meaning is encoded in formal manipulation rules. The digital world consists of symbols with physical occurrences but socially defined meaning. And it is the role of conceptual models to define this meaning.

This brings us back to the question what is specified by conceptual models. I do not intend this to be read as an empirical question ("looking around us, what do we see people using CM notations for?") but as a normative question ("What should we use CM notations for?") The answer to the normative question that I propose is that we use conceptual model notations for physical, digital and social domains only, and not for cross-world domains. So we should not make a conceptual model of a domain that is partly physical and partly social, or partly digital and partly physical. If we model a computer application, we should model this as a network of domains, each of which is either physical, digital, or social. This is similar to Jackson's view on problem modeling [45,46].

This simple guideline can be surprisingly difficult to apply. We habitually talk about employees and widgets to be "in" a computer, thus placing social and physical entities inside a digital domain. This creates many conceptual modeling errors: An employee may exist in the social domain before a record of it exists in a physical domain. And where the cardinality property that a person cannot borrow more than 10 books at the same time is a norm in the social domain, which can be violated, it can be a true property of objects in the digital domain, where we can set things up in such a way that it can not be violated [47].

3.3 Modeling digital domains

The confusion arises because the digital domain is used to represent other domains—themselves digital, or physical, or social. The importance of Hall's distinction between entities and surrogates [11] and of Jackson's distinction between domain and initial model in JSD [33] lies in the support thisn gives in avoiding conceptual modeling errors. The distinction is fundamental because by definition, digital domains come with a meaning convention. If we do not distinguish domain entities from the surrogates in a digital domain that represent them, we will not be able to describe the meaning convention. And to repeat, the role of conceptual modeling is to define this meaning.

If we model a digital domains, we should not only define the meaning of the symbols in the domain, but also specify the formal manipulations of the computer that will manipulate these symbols. This is the province of software specification and here I need only point out that symbol manipulations are *formal* if they are *physical*. If we manipulate symbol occurrences on paper formally, a physical symbol occurrence such as an ink blot on paper is manipulated by considering its form only; and the manipulations consist of erasing or creating a symbol occurrence. By extension, computers are physical machines that can manipulate symbol occurrences in their physical memory based on rules that consider the physical properties of these occurrences only. At rock-bottom, the symbol occurrences are states of hardware and the manipulations are physical processes. Formal manipulation is the replacement of meaning by physical form.

3.4 The engineering argument

Computers exercise their functionality by interacting with other domains (physical, social, or digital). The dynamics of this interaction is that manipulations in the digital domain D are influenced by other domains, i.e. by the environment E, and return the favor by having effects on these other domains. The rationale of the interactions is that by connecting the digital domain to its environment the resulting system gets closer to a goal state G:

(1) $E\&D \rightsquigarrow G$,

where E and D are the environment and a digital domain, G is a goal state of the environment, and $\rightsquigarrow$ indicates causality. Gunter et al. [48] formalize this logically as the requirements formula $Spec_E \wedge Spec_D \models Spec_G$ where now $Spec_E$ is a description of the environment, Spc_D a specification of the digital domain and $Spec_G$ a specification of the"real requirements", i.e. a description of environment goals to be satisfied. This view lends itself to model checking whether a software specification satisfies goals with respect to an environment. We can also give an alternative formalization that lends itself to theorem-proving [49]. Here I stick to (1) because I would like to keep open the possibility that $E\&D$ get somewhat closer to G but do not fully reach it.

Description (1) provides us the structure of the argument that an engineer who designs D must produce to justify the design, which I call the *engineering argument*: If you insert D into E goals G are achieved [43, page 37]. It also defines the shape of the *design hypothesis* to be verified by the engineer, namely the hypothesis that conjoining E and D is sufficient to create effect G [50].

Usually, achievement of G requires not only conjoining D to E but also involves changes to E, such as the implementation of new business processes, user procedures, devices, support staff, etc. We will discuss some of these changes later, but for the time being, we ignore them.

Structuring the engineering argument in the form of description (1) allows us to distinguish conceptual modeling, which is finding a description of E and G in order to be able to write description of D, from software specification, which is finding a description of D that will make (1) true.

Viewed this way, conceptual modeling consists of modeling an existing environment and software specification of a future software system. Conceptual modeling is a descriptive activity, software specification a prescriptive activity. This means that in conceptual modeling, we may have omitted relevant information about domains that exists even if we did not describe it, whereas in software specification, we describe a formal manipulation system that should behave as specified, and should not do what we do not describe. Conceptual modeling works with an open world assumption, software specification with a closed world assumption.

3.5 Conceptual modeling

Developing a software system is creating a domain D of digital phenomena so that (1) is true. In order to do this we must also make a description of the environment E of the software, and of the goals G to be achieved of the software. This is conceptual modeling. The conceptual models will provide meaning to the symbol manipulations, i.e. to the digital phenomena exhibited by the software.

The majority of conceptual modeling notations reviewed in section 2 is concerned with classifying things that can be counted. It is well known from philosophical logic that classification and identification are two sides of the same coin [51,52]. Much of the history of the entity-oriented conceptual modeling notations reviewed in section 2.1 replays developments in philosophical logic. The ER notation rediscovered the fact that the world can be partitioned into natural kinds (entity types) that have properties (attributes). It also repeated Frege's insight that not only can we identify subject-predicate structures (entity-attribute structures) in the world, but that some properties are n-place relationships with $n > 2$ [53]. The discovery that relationships (links) can mean many things was a basic insight of semantic networks [18] and of the NIAM notation [15]. The discovery of different kinds of taxonomic structures [17,52] is based on insights obtained long ago in philosophical logic [21] and insights about part-whole relationships in non-digital domains [54] replay insights gained earlier in linguistics and logic [55].

We can draw two conclusions from these observations. First, translating insights from linguistics and logic into conceptual modeling techniques requires us to make those insights even more precise than they already are in those other disciplines. This is required because of the demands of formal manipulation, i.e. in order to provide the engineering argument (1) the models of the environment E and the goals G must be very precise. This may force the modeler to drop generality in favor of applicability. Uncertainties and ambiguities in our best linguistic and logical insights must be replaced by choices made in the interest of computability.

Second, conceptual modeling will never advance beyond the general insights of linguistics and logic. Put differently, if we remain on a general level, developing modeling techniques for any domain whatsoever, we will continue to develop variations of techniques already discovered in general linguistics and logic. The way ahead, therefore, is in specialization: Developing modeling techniques for particular domains, such as health care, insurance, automobile manufacturing, etc. This is in fact what consultancy companies do: Develop reference models that define structured sets of concepts for particular domains.

3.6 Causal modeling

A conceptual model is a set of concepts and relationships between them. It is what research methodologists call a set of *constructs* [56], or variables as they are often called. In terms of the ER model, constructs are attributes. Empirical science is about the identification of causal relationships, or at least statistical correlations between constructs. A scientific theory consists of a conceptual model plus claims about relationships among constructs, which are statistical correlations or causal relations [57,58]. A causal relationship is a correlation between events in which the cause precedes the effect and in which there is no plausible explanation of the effect other than the cause [59, page 5–7]. Other things being equal, if the cause would not have happened, the effect would not have happened either.

In conceptual modeling we model the existing environment of computers, and if we extend this to causal modeling we will worry about whether or not other things are equal or not. Closing the valve will cause the fluid to stop flowing through the pipe—other things being equal. But if the valve leaks, other things are not equal, and the fluid will not stop flowing. This is similar to what software engineers worry about: performing an action A should cause a variable X to change value—other things being equal. But with many actions performed in parallel, unexpected interactions may occur and the variable does not change its value as intended. So the software engineer tries to set up the computing system in such a way that other things, even if they do occur, cannot impact X can therefore be treated as equal as far as the effect of A on X is concerned.

The difference with causal modeling of the environment is that every action in the computing system occurs because some software engineer somewhere said it should occur.

Research in the engineering sciences is concerned with identifying causal relationships that can be used to verify design hypotheses of the form (1) indicated earlier. Engineering research typically results in theories with restricted scope. They are not valid for the entire universe, nor are they intended to be, but for the particular class of artifacts of interest to the engineer, such as a furnace or a propeller, and even for one particular artifact, such as a particular furnace under construction. At the same time, engineering theories typically include a much larger class of variables than is customary in natural science [60,61]. The reason for this is that engineers have to deal with conditions of practice and and with performance goals and constraints that are motivated from the intended context of use, from which they cannot abstract [62, page 149–154]. I will call theories with limited, non-universal scope *local.*

Causal modeling—the constructions of models that exhibit the behavior as described by a causal theory—can be done in any kind of domain. Causal models with universal applicability are more widespread in physical domains than in social domains, even though social scientists spend considerable effort in finding universal models of social domains. In the discipline of complex system dynamics, by contrast, there is a large number of local models of social systems such as production processes, auto recycling and public transport [63]. These can best be viewed as engineering models: They are local, they incorporate a large number of variables to account for the local conditions of practice, and they are used for engineering arguments of the form (1): if this solution is inserted in this environment, we get closer to stakeholder goals. In complex system dynamics, this is called policy analysis.

In digital domains too, causal modeling is local and needs to include all relevant conditions of practice. A software engineer debugging a program is building a local theory of the cause of aberrant behavior. Performance modelers build theories of software systems that explain timing behavior. The discipline of software measurement defines indicators of important constructs such as maintainability and reliability of software [64].

Does causal modeling open new research directions for conceptual modeling? One promising direction is in the definition of constructs and indicators for particular domains. This should bring us beyond universal structures like classification, identification and aggregation, which can be found in any domain, to particular structures that can only be found in some domains. Defining indicators for constructs is the province of measurement theory. In digital domains this has been applied to the measurement of software quality attributes [64]. We can find definitions of relevant constructs and indicators in any domain, and the first question for conceptual modeling research should be to investigate how this is actually done, e.g. by case study research. Next, we could try to provide methodological support to formulate constructs and indicators in a way that allows proper formulation and verification of the engineering argument (1) in situations where where the middle domain, the one that should bring a solution, is digital, and therefore forces a degree of formalization on the definition of indicators and constructs that is absent from non-digital domains.

A second promising direction is the study of the use of languages to state the engineering argument (1) in. Causal relationships in continuous domains are usually stated in the language of the differential calculus. In system dynamics, a graphical representation of partial differential equations called causal loop diagrams (CLDs) has been developed [63]. In requirements engineering, some language proposals such as Tropos [65] and i* [66] have some resemblance to CLDs but are more oriented to describing goals than to describing phenomena in a domain. State transition diagrams, such as Mealy, Moore and Harel (statechart) diagrams allow one to express exactly how a discrete variable changes by which event. Communication diagrams allow one to specify which entity has causal impact on which other entity. Particular domains tend to have their own, domain-specific notations to represent domain structures. The diversity of languages is so large that the engineering question how to *improve* these languages is not meaningful; rather, we should ask the research question how, in a particular domain, these languages are actually used to state the engineering argument (1) in. If we understand their use, we may as a next step be able to improve some of these languages.

3.7 Modeling social domains

Social domains are of special interest to the conceptual modeler because they contain conceptual models shared by the participants of the domain. If a domain contains people, then these people structure their world by means of shared models of job roles, tasks, responsibilities, processes, departments, committees, clients, suppliers, accounts, norms, standards, etc. etc. The conceptual modeler must learn the relevant parts of these models from domain inhabitants and use them as shared background when communicating with the domain inhabitants.

This is different from sharing a conceptual model of a physical domain with domain experts. The shared conceptual model of social domain structures *constitutes* the domain, i.e. some domain structures exist because they are represented to exist. Entities like companies, job roles, business processes, customers, suppliers, money, debts, accounts, standards, norms, etc. etc. exist because a group of people considers them to exist and acts accordingly [67,68]. Physical entities exist even if no one is aware of them, but the social constructions that interest us here cannot exist if no one is aware of them.

It is possible that some social structures exist even if no one has become aware of them. For example, a group of people may interact more frequently among each other than with other people without anyone being aware of this. These are not the social structures defined by conceptual models.

The conceptual modeler may encounter various difficulties, such as that not every domain inhabitant may share the same conceptual model of that domain. Different groups of domain inhabitants may view entities and events in the domain differently. And even if there is one shared conceptual model among domain inhabitants, it may not be described explicitly, and the definitions that have been made explicit may contain ambiguity and vagueness. It is the task of the conceptual modeler to eliminate vagueness, reduce ambiguity, negotiate agreement about concepts referred to by the digital domain that will help stakeholders get closer to their goals, and help ensure that the resulting model is shared by all domain inhabitants.

3.8 Performative dynamics

There are two kinds of dynamics in this process, neither of which occurs in physical domains. The first one is a *performative dynamics* in which people perform speech acts to change the state of a social domain. People can act by talking. For example, people can act by describing what happens, by telling people what to do, by committing themselves to do something, by declaring that something exists and by expressing their feelings [69,70]. An example of a declarative speech act is the declaration, by the chairperson of a meeting, that the meeting is opened. By saying this, the meeting is in fact opened and so the statement is made true by the chairperson's uttering the statement. Declarative speech acts have their intended effect if they are uttered by people in the right kind of circumstances: E.g. by the chairperson (not someone else) at the start of a meeting (not in the middle or at the end). This is an example of performative dynamics.

The relevance of this for conceptual modeling is that computers can be made to do these things too. This has been observed by several authors in the 1980s [71,72,73,74,75] and various attempts have been made to capture this in deontic logic-based conceptual modeling languages [76,77,47]. It remains to be seen whether performative dynamics can be formalized sufficiently well to be of use in conceptual modeling. Formalization of performative dynamics is however not needed in order to make computers act by means of speech acts. The requests, commands, questions, promises, announcements, declarations, and expressions about its own state made by a computer are simple actions that can be programmed without formalization of performative dynamics. And the definition of their effect on the social environment does not require formal logic either. People understand what a command is without being presented with a logical axiomatization of it.

What concerns us here is what the delegation of speech acts to a computer means for engineering arguments of the form (1). Take for example a library information system that registers that a book is borrowed from a library. The social domain to which this system is conjoined can be structured in such a way that the act of borrowing is defined to take place when and only when the registration takes place. In that case the term "registration" is misleading. The information system does not register a borrowing event in the same sense that a control system can register data about the current temperature in a heating vat. If the system registers a domain event there is a causal chain of events from some physical domain event to the registration event. In this chain, cause precedes effect. But if the borrowing event takes place at the exact point in time when the library information system "registers" the event to take place then the borrowing event cannot be the cause of the registration event. It is simultaneous with it. Rather, the "registration" event is a declaration event. It is a declarative speech act by the information system that someone borrowed a book. In terms of engineering argument (1), we can define a conceptual model of a social domain E in which an information system D is conjoined with E so that borrowing events can take place (G). But this means that the appropriate social structures must be present in E, such as when the system is allowed to do this, how it is supposed to do this, how the borrower is to be informed of this, who is accountable for this event, what norms are applicable, etc. The conceptual modeler must work with the domain inhabitants to ensure that these structures are present. This too is part of conceptual modeling of social domains [75].

3.9 Structuration

A second kind of dynamics in social domains is that conceptual modeling may change the performance dynamics of the domain. It is as if by changing the conceptual model of a physical domain one could change the causal relationships between events. This is of course not possible in physical domains: If we change a causal model of a physical domain, the model becomes incorrect but causality is unaffected. But in social domains, if the conceptual model shared by domain inhabitants changes, then the domain changes, and among the things that could change are performative dynamics. Conceptual modeling of social domains is part of what Giddens [67] calls *structuration,*, which is a process in which part of the conceptual domain model may be changed while the rest remains unchanged. People act against the background of a shared conceptual model that constitutes their social world, and therefore defines the meaning of what they are doing/saying, but which conversely is affected by what they are doing/saying [67].

This makes conceptual modeling potentially a power game. During meetings each participant of the meeting may try to impose his or her definition of the situation on the others, with a view to making certain courses of action seem the natural thing to do. The use of information as a source of power in organization is a long-standing research topic in management information systems [78,79]. Forcing access to or exclusion from information is but one tool in the struggle for power; enforcing one's preferred conceptual model to define the meaning of the situation is another, very effective tool. This is part of problem framing in social contexts [80].

Conceptual modeling can be part of a power game whether or not it is part of introducing software solutions. A second phenomenon of structuration is more commonly associated with automation, namely the reduction of flexibility of conceptual models once they are embedded in software systems. This is a well-known phenomenon in ERP implementation [**?**]. Inflexibility may simply be a consequence of the size of ERP systems; in that case, making these systems smaller would be the only solution to the problem. However, a lack of flexibility seems to be associated with administrative applications in general and there is room for more technical research into improving the flexibility of conceptual models.

4 Summary, conclusions, discussion

CM languages have settled on classification and identification, taxonomies, aggregation, communication and behavior as universal structures found in all conceptual models. Notations for these structures are provided by the UML, but due to some mechanisms built into the UML, and due to a lack of prescription of what the UML is intended for, the UML tends to grow into a set of notations and guidelines that can be used for many purposes by many people. A much narrower approach is taken in ontology languages such as OWL, which restricts itself basically to an entity-oriented conceptual modeling approach but then tries to formalize this fully and provide tool support for reasoning about taxonomic structures. Just as in the case of he UML, much work in ontologies focusses on standardization of modeling languages.

Standardization however is not an activity that befits researchers. Standardization is commercial politics [81], and the criteria of scientific validity do not play an overriding role there. Even if researchers manage to define an international *de jure* standard, such as the telecommunication specification language Lotos [82], the standard will not lay a significant role outside the community of researchers that produced it.

To get beyond the general insights of classical logic and linguistics, we should not move into standardization but into specific domains. The first distinction to make is that between physical and social domains. In physical domains, conceptual modeling is a prelude to causal modeling. Conceptual modeling defines the variables among which causal modeling posits causal relationships. Variables must be operationalized, that is they must be measurable and/or controllable to be of use in causal modeling for engineering applications. Variables are measured and/or controlled in order to achieve certain goals, such as bringing certain goal variables into desirable ranges, where goal variables are among the measurable and controllable variables. These goal variables are often called quality attributes, meaning that some stakeholders have desires concerning their value. Measurability and controllability of variables (quality attributes) are interesting research areas for conceptual modeling.

In social domains, causal modeling is relevant too, and the relevant variables are often indicators of the performance of an organization or some other social system. In addition to causality, there is a second kind of dynamics in social domains, which takes place by means of speech acts. People change their social reality by means of commands, questions, declarations, commitments, expressions, etc. Computers can do this too, if they are inserted into a social domain in a way that makes human inhabitants of the domain accept the speech acts of a computer as valid. Two kinds of dynamics take place by means of speech acts: Performative dynamics is the change the state of a social domain by means of speech acts in accordance with a conceptual model shared by inhabitants of the domain, and structuration is the change of a conceptual model itself, also by means of speech acts. Structuration does not take place according to fixed rules and may easily be associated with power games. Automation tends to introduce rigidity in conceptual models and there is room for improvement of flexibility of conceptual models.

From a scientific point of view, study of the phenomena of performative change and structuration can be very interesting. However, from an engineering point of view more mileage can be had by specializing to particular social or physical domains and studying operationalization, measurement, and control of particular variables relevant for some stakeholders with particular goals for those domains. Engineering research cannot abstract from conditions of practice and is therefore specific [60,61]. This holds for software engineering and conceptual modeling too [83,84], and I think the way ahead is to specify and analyze conceptual models of particular domains.

References

1. Chen, P.S.: The entity-relationship model – Toward a unified view of data. ACM Transactions on Database Systems **1** (1976) 9–36
2. Knuth, D.: Von Neumann's first computer program. ACM Computing Surveys **2**(4) (1970) 247–260
3. Lopez, R.: The Commercial Revolution of the Middle Ages, 950–1350. Cambridge University Press (1976)
4. McNeill, W.: The Rise of the West: A History of the Human Community. University of Chicago Press (1963, 1991)
5. Codd, E.: A relational model of data for large shared data banks. Communications of the ACM **13** (1970) 377–387
6. ANSI: Ansi/x3/sparc study group on DBMS's interim report. SIGMOD FDT Bulletin of ACM **7**(2) (1975)
7. Elmasri, R., El-Assal, I., Kouramajian, V.: Semantics of temporal data in an extended er model. In Kangalasso, H., ed.: Entity-Relationship Approach: The Core of Conceptual Modelling, Elsevier (1991) 239–154
8. Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., Snodgrass, R., eds.: Temporal Databases. Benjamin/Cummings (1993)
9. Thalheim, B.: Fundamentals of cardinality constraints. In Pernul, G., Tjoa, A., eds.: Entity-Relationship Approach –ER'92, Springer (1992) 7–23 Lecture Notes in Computer Science 645.
10. Smith, J.M., Smith, D.: Database abstractions: Aggregation and generalization. ACM Transactions on Database Systems **2** (1977) 105–133
11. Hall, P., Owlett, J., Todd, S.: Relations and entities. In Nijssen, G., ed.: Modelling in Database Management Systems. North-Holland (1976) 201–220
12. Hull, R., King, R.: Semantic database modeling: Survey, applications, and research issues. ACM Computing Surveys **19** (187) 201–260
13. Peckham, J., Maryanski, F.: Semantic data models. ACM Computing Surveys **20** (1988) 153–189
14. Batini, C., Ceri, S., Navathe, S.: Conceptual Database Design: An Entity-Relationship Approach. Benjamin/Cummings (1992)

15. Nijssen, G., Halpin, T.: Conceptual Schema and Relational Database Design. Prentice-Hall (1989)
16. Vermeir, D., Nijssen, G.M.: A procedure to define the object type structure of a conceptual schema. Information Systems **7**(4) (1982) 329–336
17. Brachman, R.: What IS-A is and isn't: an analysis of taxonomic links in semantic networks. Computer **16**(10) (1983) 30–36
18. Woods, W.: What's in a link: Foundations for semantics networks. In Bobrow, D., Collins, A., eds.: Representation ans Understanding: Studies in Cognitive Science. Academic Press (1975) 35–82
19. W3C: OWL Web Ontology Language Overview. (2004)
20. Almeida Falbo, R, d., Guizzardi, G., Duarte, K.: An ontological approach to domain engineering. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 02), ACM Press (2002) 351–358
21. Guizzardi, G.: Modal aspects of object types and part-whole relations and the *e re/de dicto* distinction. In Krogstie, J., Opdahl, A., Sindre, G., eds.: Proceedings of the 19th International Conference on Advanced Information Systems Engineering, 19th International Conference (CAiSE 2007, Springer (2007) 5–20 LNCS 4495.
22. Langefors, B.: Information systems theory. Information Systems **2** (1977) 207–219
23. Lundeberg, M., Goldkuhl, G., Nilsson, A.: A systematic approach to information systems development - I. Introduction. Information Systems **4** (1979) 1–12
24. Lundeberg, M., Goldkuhl, G., Nilsson, A.: A systematic approach to information systems development - II. Problem and data oriented methodology. Information Systems **4** (1979) 93–118
25. Ross, D.T.: Structured analysis (SA): A language for communicating ideas. IEEE Transactions on Software Engineering **SE-3**(1) (1977) 16–34
26. Ross, D.T., Schoman, K.E.: Structured analysis for requirements definition. IEEE Transactions on Software Engineering **SE-3**(5) (1977) 6–15
27. DeMarco, T.: Structured Analysis and System Specification. Yourdon Press/Prentice-Hall (1978)
28. Gane, C., Sarson, T.: Structured Systems Analysis: Tools and Techniques. Prentice-Hall (1979)
29. Ward, P.T., Mellor, S.J.: Structured Development for Real-Time Systems. Prentice-Hall/Yourdon Press (1985) Three volumes.
30. Hatley, D., Pirbhai, I.: Strategies for Real-Time System Specification. Dorset House (1987)
31. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming **8** (1987) 231–274 Preliminary version appeared as Technical Report CS 84-05, The Weizmann Institute of Science, Rehovot, Israel, February 1984.
32. Harel, D.: On visual formalisms. Communications of the ACM **31** (1988) 514–530
33. Jackson, M.: System Development. Prentice-Hall (1983)
34. Belina, F., Hogrefe, D.: The CCITT-specification and description language SDL. Computer Networks and ISDN Systems **16** (1988-1989) 311–341
35. Færgemand, O., Olsen, A.: Introduction to SDL 92. Computer Networks and ISDN Systems **26** (1994) 1143–1167
36. Sarraco, R., Tilanus, P.A.J.: CCITT SDL: Overview of the language and its applications. Computer Networks and ISDN Systems **13** (1987) 65–74
37. Booch, G.: Object-Oriented Design with Applications. Benjamin/Cummings (1991)
38. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-oriented modeling and design. Prentice-Hall (1991)

39. Jacobson, I., Christerson, M., Johnsson, P., Övergaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Prentice-Hall (1992)
40. Wieringa, R.: Requirements Engineering: Frameworks for Understanding. Wiley (1996) Also available at http://www.cs.utwente/nl/∼roelw/REFU/all.pdf.
41. Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Computing Surveys **30**(4) (1998) 459–527
42. Wieringa, R.: Postmodern software design with NYAM: Not yet another method. In Broy, M., Rumpe, B., eds.: Requirements Targeting Software and Systems Engineering. Springer (1998) 69–94 Lecture Notes in Computer Science 1526.
43. Wieringa, R.: Design Methods for Reactive Systems: Yourdon, Statemate and the UML. Morgan Kaufmann (2003)
44. Newell, A.: Physical symbol systems. Cognitive Science **4** (1980) 135–183
45. Jackson, M.: Software Requirements and Specifications: A lexicon of practice, principles and prejudices. Addison-Wesley (1995)
46. Jackson, M.: Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley (2000)
47. Wieringa, R., Meyer, J.J.C., Weigand, H.: Specifying dynamic and deontic integrity constraints. Data and Knowledge Engineering **4** (1989) 157–189
48. Gunter, C., Gunter, E., Jackson, M., Zave, P.: A reference model for requirements and specifications. IEEE Software **17**(3) (2000) 37–43
49. Mader, A.H., Brinksma, H., Wupper, H., Bauer, N.: Design of a plc control program for a batch plant - vhs case study 1. European Journal of Control **7**(4) (2001) 416–439
50. Roozenburg, N., Eekels, J.: Product design: Fundamentals and Methods. Wiley (1995)
51. Wieringa, R., de Jonge, W.: Object identifiers, keys, and surrogates— object identifiers revisited. Theory and Practice of Object Systems **1**(2) (1995) 101–114
52. Wieringa, R., Jonge, W.d., Spruit, P.: Using dynamic classes and role classes to model object migration. Theory and Practice of Object Systems **1**(1) (1995) 61–83
53. Kneale, W., Kneale, M.: The Development of Logic. Clarendon Press (1962)
54. Barbier, F., Henderson-Sellers, B., Le Parc, A., Bruel, J.M.: Formalization of the whole-part relationship in the unified modeling language. IEEE Transasctions on Software Engineering **29**(5) (2003) 459–470
55. Chaffin, R., Herrmann, D., Winston, M.: An empirical taxonomy of part–whole relations: Effects of part–whole relation type on relation identification. Language and Cognitive Processes **3**(1) (1988) 17–48
56. Cooper, D., Schindler, P.: Business Research Methods, 8th edition. Irwin/McGraw-Hill (2003)
57. Gregor, S.: The nature of theory in information systems. MIS Quarterly **30**(3) (2006) 611–642
58. Kaplan, A.: The Conduct of Inquiry. Methodology for Behavioral Science. Transaction Publishers (1998) First edition 1964 by Chandler Publishers.
59. Shadish, W., Cook, T., Campbell, D.: Experimental and Quasi-experimental Designs for Generalized Causal Inference. Houghton Mifflin Company (2002)
60. Küppers, G.: On the relation between technology and science—goals of knowledge and dynamics of theories. The example of combustion technology, thermodynamics and fluid dynamics. In Krohn, W., Layton, E., Weingart, P., eds.: The Dynamics of Science and Technology. Sociology of the Sciences, II. Reidel (1978) 113–133
61. Layton, E.: Mirror-image twins: The communities of science and technology in 19th century America. Technology and Culture **12**(4) (1971) 562–580

62. Polya, G.: How to Solve it. A New Aspect of Mathematical Method. Second edn. Princeton University Press (1985) First edition 1945.
63. Sterman, J.: Business Dynamics: Systems Thinking and Modeling for a Complex World. McGraw-Hill (2000)
64. Fenton, N., Pfleeger, S.: Software Metrics: A Rigorousn and Practical Approach. Thomson (1997) Second edition.
65. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Modelling early requirements in tropos: a transformation based approach. In Ciancarini, P., Wooldridge, M., Weiß, G., eds.: Agent-Oriented Software Engineering II, Springer (2001) 151–168 LNCS 2222.
66. Yu, E.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97), IEEE Computer Science Press (1997) 226–235
67. Giddens, A.: New Rules of Sociological Method: A Positive Critique of Interpretative Sociologies. Basic Books (1976)
68. Searle, J.: The Construction of Social reality. The Free Press (1995)
69. Austin, J.: How to Do Things with Words. Harvard University Press (1962)
70. Searle, J.: Speech Acts. An Essay in the Philosophy of Language. Cambridge University Press (1969)
71. Auramäki, E., Lehtinen, E., Lyytinen, K.: A speech-act-based office modeling approach. ACM Transactions on Office Information Systems (1988) 126–152
72. Kimbrough, S., Lee, R., Ness, D.: Performative, informative and emotive systems: The first piece of the PIE. In Maggi, L., King, J., Kraenens, K., eds.: Proceedings of the Fifth Conference on Information Systems. (1983) 141–148
73. Kimbrough, S.: On representation schemes for promising electronically. Decision Support Systems **6** (1990) 99–121
74. Lehtinen, E., Lyytinen, K.: Action based model of information systems. Information Systems **11** (1986) 299–317
75. Wieringa, R.: Three roles of conceptual models in information system design and use. In Falkenberg, E., Lindgreen, P., eds.: Information System Concepts: An In-Depth Analysis. North-Holland (1989) 31–51
76. Lee, R.: Bureaucracies as deontic systems. ACM Transactions on Office Information Systems **6**(2) (1988) 87–108
77. Meyer, J.J., Wieringa, R., eds.: Deontic Logic in Computer Science: Normative System Specification. Wiley (1993)
78. Pettigrew, A.: Information control as a power resource. Sociology **6** (1972) 187–204
79. Markus, M.: Power, politics, and MIS implementation. Communications of the ACM **26**(6) (1983) 430–444
80. Schön, D.: The Reflective Practitioner: How Professionals Think in Action. Arena (1983)
81. Shapiro, C., Varian, H.: Information Rules. A Strategic Guide to the Network Economy. Harvard Business School Press (1999)
82. Bolognesi, T., Brinksma, E.: Introduction to the ISO Specification Language LOTOS. Computer Networks **14** (1987) 25–59
83. Jackson, M.: Specialising in software engineering. IEEE Software **16**(6) (1999) 119–121
84. Zave, P.: Formal methods are research, not development. Computer **29**(4) (1996) 26–27