Philosophy-Based Mechanisms for Communication between Business and IT Experts

Haim Kilov[1], Ira Sack[2]

[1] Independent Consultant, and Stevens Institute of Technology
haimk@acm.org,
[2] Stevens Institute of Technology
isack@stevens.edu

Abstract. The paper shows how a system of important concepts and approaches proposed by system thinkers (such as philosophers, mathematicians, and computing scientists) has been used to understand and specify various kinds of business and IT systems, and to base the IT work on a solid foundation that can be used for communicating with non-IT experts, thus establishing successful and meaningful interactions between business and IT experts and organizations. These common elegant concepts — such as abstraction, system, structure, relationship, composition, pattern, name in context, etc. — come from exact philosophy and mathematics. They have been stable for centuries, and have been successfully used in theory, in industrial practice (including international standards), and in teaching of business and IT modeling. The essential stable *semantics* of these fundamental concepts and of systems specified using them ought to be clearly separated from the accidental (often IT-industry-imposed excessively complex and rapidly changing) details.

Introduction

The proverbial communication gap between business and IT experts has been a sad reality for quite a while. This has led to substantial problems in information system design and development including significant monetary losses together with loss of customers' trust and patience (Evans, 2002). As noted in *Computerworld* (October 11, 1999), "85% of IT departments in the US fail to meet their organizations' strategic business needs". More recently, almost the same percentage (84%) was cited in *Computerworld* (May 9, 2005) as the percentage of top executive MBA candidates (at the Fisher College of Business at Ohio State University) who have full-time jobs and who "when asked to recall personal experiences related to IT, cited very negative situations". At the same time, not all IT projects fail: in some environments business and IT experts do communicate in a successful manner. Therefore, it would be instructive to determine – and make explicit – some properties of successful business-IT communication.

Thanks to the division of labor, experts can concentrate – and achieve a lot – in their specific areas of expertise. Furthermore, they can, and should, use the achievements of other experts (in their own and other areas) as and when needed. In

order to do that, it is necessary to use a common system of basic concepts for communication. This system of concepts does not belong to any specific area of expertise, but rather is used as fundamental one in all of these areas. When we want to understand "what is there" in a specific area of expertise (that is, when we discover an ontology of that area) and when we communicate our understanding to others (that is, when we represent the ontology in some manner), we have to use such fundamental common concepts. And the importance of understanding and properly using these concepts increases in those areas of expertise that are less specialized, such as the area of information technology, especially, understanding and proper handling of complexity.

Common concepts

Where do these common concepts come from? As Mario Bunge noted, "all factual sciences, whether natural, social, or mixed, share a number of philosophical concepts … and a number of philosophical principles" (Bunge, 2001a). More specifically, technologists "who work on general theories of systems, control theory, optimization theory, the design of algorithms or simulation are applied philosophers of sorts, since they use philosophical concepts, such as those of event and system, and philosophical principles, such as those of the existence and lawfulness of the external world" (Bunge, 2001). This is where business and IT experts can and should find a common ground for understanding and therefore communication.

In order to succeed, the fundamental concepts – rather than various rapidly changing IT-industry-imposed fashionable "new things" (Evans, 2002) – ought to be used in an explicit manner. These concepts are stable, and have been so for centuries. At the same time, proper exactification of some of these concepts became possible only more recently, thanks to the developments in exact philosophy, semiotics, mathematics, and computing science. In particular, it has been necessary for success to abstract away from the (extreme) complexities often imposed by software and service vendors, and go "back to basics" (Kilov, 2001).

We will see that the same fundamental concepts have been used both in exact philosophy and in important IT-based work, such as international standards (for example, RM-ODP – the Reference Model of Open Distributed Processing (ISO/IEC, 1995)). These concepts have been successfully used not only in theory, but also in industrial practice of business system modeling, design and development, as well as in teaching (using information modeling and RM-ODP, with an appropriate very small subset of UML used for representation purposes) in a unit on modeling of a suitable IS course (such as data and knowledge management) for students of management, MBAs, IT, etc.

Many philosophical foundations of these fundamental concepts have been exactified by Mario Bunge. Other thinkers, such as Wittgenstein (Wittgenstein, 1933) and F.A.Hayek (Hayek, 1964; Hayek, 1952), also contributed a lot. Some fundamentals go back to Aristotle: for example, there exists a very important difference between the Aristotelian and a prototypical (example-based) approach to modeling (Madsen, Moller-Petersen and Nygaard, 1992; Kilov and Ross, 1994). The

former provides for the intension of the model; the latter – despite being buzzword-compliant in some popular IT-based methodologies – provides for its probably incomplete extension. When we use the former, we can always find out whether a fact corresponds to the model, while when we use the latter, we often cannot do that. In the same manner as testing of a program can show an error but cannot demonstrate that the program is correct (E.W.Dijkstra), examples are helpful to illustrate an ontology, but are inadequate when we want to create and communicate it. After all, testing of a program, or of an ontology (that is, using examples to check it), differs significantly from creating and understanding it.

It is very instructive to notice that essential concepts and structuring rules defined in the RM-ODP standard are based on the same fundamentals from exact philosophy, as well as on mathematics – "the art and science of effective reasoning" (E.W.Dijkstra). This international standard was created to support the definitions of "the basic concepts to be used in the specifications of the various components which make up the open distributed system" (ISO/IEC, 1995). While such a standard may seem to be very specific and useful only in computer-based IT environments, open distributed systems exist not only – and not even mainly – in computer-based environments: such systems have existed and have been described in all kinds of human endeavor, for example, in a market economy (Bagehot, 1873; von Mises, 1949), or in reasoning about purposeful behavior (Hayek, 1952). As an example, it was very easy and instructive to show how the concepts and structuring rules defined in RM-ODP perfectly apply to a precise specification of a specific open distributed system – a banking clearing house – based on a century-old text (Dunbar, 1901) and still – without changes but perhaps with some refinements – applicable now (Kilov, 1999).

Similarly, we observe that crucial business concepts have been known and explicitly used for a long time and can be found, for example, in works by Adam Smith. And we also observe, with great pleasure, that Smith's – more specialized – *Wealth of Nations* was based on philosophical foundations laid in his *Theory of Moral Sentiments*. This paper shows how the system of important concepts and approaches proposed by these thinkers has been used to understand and specify essential fragments of the ontologies of various business and IT systems, and thus to establish successful communication between business and IT experts.

A (relatively) familiar example: names

The abysmal notion of "meaningful names" has existed in IT for quite a while (Kent, 1978). As a modern-day example, consider relying on "data names" – instead of an explicit information model – in XML within the context of Semantic Web (for a criticism and a semantics-based approach, see (Kudrass, 2001)). The need for semantic information integration based on meaning rather than on "data" and tools (none of which integrates data beyond the syntactic level) has been emphasized for a long time, and is becoming acknowledged in popular industrial publications (Raden, 2005) where, for example, we encounter references to "a hundred different meanings" of such terms as "on-time percentage" or "customer profitability". At the same time,

the inadequacy of “meaningful names” in IT was recognized by Grace Hopper as early as in 1957: “[w]hile the computation of the square root of a floating decimal number remained the same in Pittsburgh, Los Angeles, and New York, the computation of gross-to-net pay obviously did not remain the same even in two installations in the same city” (Hopper, 1957).

Let us try to exactify the concept of a name.

We start with observing that synonyms and homonyms exist: the same thing (or action, or process, or relationship) may have different names, and the same name may denote different things (or actions, etc.). Although many IT-based approaches promote the apparent need to determine and impose the “only correct name” of a thing, other names of that same thing, including nicknames and abbreviations, have been (and will be) used in business anyway, leading to various and often serious problems. It is counterproductive, for example, to try to determine which of the 50 or more somewhat different types of things, all named “patient” by various groups of HMO stakeholders *in different contexts*, is a real patient, and which of them, therefore, are “not real patients”. To distinguish between these different types of things with the same name, we explicitly use contexts, such as “this is what emergency calls a patient”, or “this is what insurance company XXX calls a patient”. (If from an insurance company’s viewpoint a patient is the person who pays insurance premiums then the statistical datum stating that 40% of male patients have been pregnant at some point in time could be understood…)

We continue with observing that in business, the context of a name may not always be made explicit, and may change even within the same narrative – even the same sentence – presented by a single person. Questions about semantics, determined to a large extent by the context, can always be (informally) asked and answered, although in many business situations this is not done because it is wrongly assumed that “everyone knows what XXX is”. Since interactions with computer-based systems require a substantially more disciplined approach to using names, a strict discipline may often be imposed – often implicitly – on using “the same meaningful name” to denote apparently “the same thing”. Such approaches often lead to failures.

There is no need to invent new and better approaches in order to solve the “name problem”. Philosophers have noted that names by themselves do not convey any meaning: as stated by Wittgenstein, “[o]nly the proposition has sense; only in the context of a proposition has a name meaning” (Wittgenstein, 1933). Similarly, semiosis was defined by Charles Saunders Peirce as “an action, or influence, which is, or involves, a cooperation of three subjects, such as a sign, its object, and its interpretant, this tri-relative influence not being in any way resolvable between pairs... If this triple relation is not of a degenerate species, the sign is related to its object only in consequence of a mental association, and depends upon habit.” (Peirce, 1931). The concept of an interpretant is very close to the concept of a context within which a name is being considered (Kilov, 2002). Clearly, the context itself – the set of relevant relationships defining the structure of the relevant domain (that, is, a fragment of the ontology) – has to be defined in an explicit manner. Correspondingly, David Hilbert noted that the content of elementary geometry does not suffer any changes if we replace the words “point”, “line”, and “plane” by the terms “chair”, “table”, and “bar” (Yaglom, 1986).

This philosophy-based approach provided a solid foundation of the *system* of corresponding RM-ODP definitions (ISO/IEC, 1995): a name is "a term which, in a given naming context, refers to an entity", and an identifier is "an unambiguous name, in a given naming context". Clearly, the same entity may have several names and several identifiers in the same naming context. These definitions are based on semantics and apply to any system, independently of whether it does or does not use computer-based information technology. And there has been no need to use IT-specific (or any buzzword-compliant and thus "rapidly changing") concepts for these definitions. They have been distilled, taking into account the appropriate philosophical foundations, from a huge body of industrial data and business modeling experience, including a lot of work in proper handling of data input described, for example, in (Gilb and Weinberg, 1977; Kent, 1978; Kilov and Ross, 1994). It is a pity that the same old mistakes in name treatment and data input handling have still been encountered in many, and varied (failed) present-day IT-related activities (some of them may be described in articles with titles referring to "data quality" or "semantic integration"). To avoid such mistakes, it is essential to base the IT work on a solid foundation that, as we have seen, can also be used for communicating with non-IT experts.

A moderately detailed model of "name in context" based on the definitions discussed above was presented, for example, in (Kilov, 1999). This model – as any other business model, for that matter – substantially uses a very small number of generic relationships, such as composition and subtyping. The *semantics* of these relationships was precisely defined, for example, in RM-ODP using philosophical foundations such as those described by Mario Bunge (Bunge, 2001; Bunge, 2003) or by F.A.Hayek (Hayek, 1952). Thus, business and information modeling does not start with a blank sheet of paper, but rather reuses essential business patterns such as generic relationships. Note that various business patterns based on composition were precisely described and used by such system thinkers as Adam Smith (Smith, 1776, Book One, Chapter VI – the example of a price of commodity as a composition[1] of rent, labor and profit), F.A.Hayek (Hayek, 1964, and elsewhere), Ludwig von Mises, and others.

On systems and relationships

All too often, existing systems or those that are supposed to be built, are represented by "experts" using various kinds of box-and-line diagrams the semantics of which is either unclear or too vague to be of any use. Although some readers of such diagrams may have a warm and fuzzy feeling about them, especially if some names used in the diagrams happen to be familiar, different readers will usually have quite different understanding of the meaning of these diagrams. Often, the narratives describing the diagrams are either unclear or concentrate only on examples (compare with the contrast between the Aristotelian and prototypical approaches to modeling referred to

[1] See the precise definition of the composition relationship below. While Adam Smith did not use this definition explicitly, his treatment of this example clearly demonstrates his excellent understanding of composition semantics.

above). Specifying an existing system or designing a new one on such a basis leads to serious problems and often to failures.

Again, as in the name example, let us try to exactify the concept of a system together with some related concepts. To quote Mario Bunge, "[e]very system can be analyzed into its *composition* (or set of parts), *environment* (or set of objects other than the components and related to these), *structure* (or set of relations, in particular connections and actions, among the components and these and environmental items) and *mechanism* (or set of processes peculiar to it, or that make it tick)" (Bunge, 2001a). Moreover, analysis means "breaking down a whole into its components and their mutual relations" (Bunge, 2003). These approaches were not invented by Bunge – they have been known and used for centuries. For example, Walter Bagehot in *Lombard Street* observed in 1873: "[t]he objects which you see in Lombard Street, and in that money world which is grouped about it, are the Bank of England, the Private Banks, the Joint Stock Banks, and the bill brokers. But before describing each of these separately we must look at what all have in common, and at the relation of each to the others." (Bagehot, 1873). Thus, the concepts of a relation and of composition are of essence in analysis (and system design) work. Observe also that in order to understand individuals we ought to consider relationships between them: "individuals are merely the *foci* in the network of relationships" (Hayek, 1979). Clearly, we need to concentrate on relationship *semantics*, and in doing so we emphasize, in particular, that a line between two boxes is not a relationship since its semantics has not been specified (also, most relationships are not binary).

From the considerations above it follows that the same kind of approach may and should be used to understand business and IT systems, and that the same concepts and constructs may be used to analyze[2] and design such systems. Furthermore, the structure of composite processes may be understood in the same manner as the structure of composite ("whole") things, so that the same kinds of relationships may be used in analyzing (and designing) both things and processes. In this manner, it becomes possible to define and use a small number of concepts and constructs for understanding and describing any kind of system. When the semantics of these concepts is defined in an explicit manner clear to both business and IT experts, these experts can communicate: they will use the same system of concepts for creating their explicit ontologies, in the same manner as different people use the same well-defined system of concepts to create and use various kinds of roadmaps, from those in nicely bound atlases to those scribbled on a napkin. And in the same manner as the semantics of every relationship between things in a roadmap is conceptually clear – or can be easily explained – to any user of a roadmap, *the semantics of every relationship* between items in an ontology *should be conceptually clear* or be easily explainable to every user of the ontology. Fortunately, this is not too difficult since the number of basic relationships is quite small, and their semantics has been clearly defined.

As noted above, a system of such concepts was specified, for example, in international standards such as RM-ODP and the General Relationship Model (GRM

[2] This applies to the analysis both of business systems and of IT systems. The latter often have to be explicitly analyzed because "it does what it does" is not an adequate characterization of an IT system by its vendor (or by anyone else).

(ISO/IEC, 1995a)). These approaches to creation and use of ontologies and to system analysis (and design) in general have been applied both in academia (teaching and research) and in industry – in various business and IT areas including finance, insurance, telecommunications, document management, organizational modeling, managerial decision making, business process change, metadata management, business and IT system architecture, and many others (Garrison, 2001; Kilov and Ross, 1994; Kilov, 1999; Kilov, 2002; Kilov and Baclawski, 2003; Kudrass, 2001; Morabito, Sack and Bhate, 1999; OMG, 2004; etc.). Furthermore, these approaches have substantially influenced various OMG (Object Management Group) standards, such as the Relationship Profile of the UML profile for Enterprise Distributed Object Computing, the Model-Driven Architecture (MDA), etc.

Composition

One of the most important concepts in system thinking is that of a composition relationship. It is defined in RM-ODP as "[a] combination of two or more [items] yielding a new [item], at a different level of abstraction. The characteristics of the new [item] are determined by the [items] being combined and by the way they are combined." (ISO/IEC, 1995). Similar definitions have been provided by philosophers, notably, by Bunge and Hayek. It follows that the concept of *emergent* properties of the composite is the essential one to understand and use composition. Whereas in some cases the values of these properties may be easily determined by a computer-based system (e.g., the total number of pages in a paper document composed of sections), in other – more interesting – cases the values of the emergent properties have to be determined by humans (e.g., the abstract of such a document).

We may want to distinguish between various degrees of novelty of emergent properties. In some cases, we follow Bunge's definition of emergence and look at radical novelty: "[a] property of a system is emergent if it is not possessed by any component of the system. Examples: [...] being alive (an emergent property of cells), perceiving […] and social structure (a property of all social systems)." (Bunge, 2003). In certain other cases, the novelty is not radical at all (the emergent properties may, for example, result from the relationships of the composite to its environment), while in many cases the degree of novelty of emergent properties is between the two extremes: consider the various degrees of social significance in various groups of individuals as described in (Ogburn and Nimkoff, 1947). This degree of novelty need not be fixed: for example, we can differentiate a group of people who met for the first time for some purpose from that group that functions as a team with social and other bonds. When the latter is disbanded (for example, by management), the bonds still remain, and therefore the team as a composite (perhaps of a different kind) still exists.

A composition relationship is in most cases not binary: the invariant that determines the emergent properties of the composite refers to all its components rather than only to one of them. Moreover, in understanding and using compositions we refer to the emergent properties of the composite, and therefore "definitions" of compositions that do not refer to property determination miss the essential semantics of the defined concept. These inadequate definitions lead to failures but still have been widely used in popular IT-based (more specifically, tool-based) modeling approaches because handling various kinds of emergent properties – and of property determination in general – is not easy and more often than not cannot be automated.

A property of any system (including a software system) is emergent if it is not possessed by any of the components of that system. Any complex system can be described in this manner. The observation about *abstraction layers* in software was made as early as in the 1960s, by Dijkstra, Hoare, and others. With respect to information modeling, the same kind of observation about any system was made, for example, in (Kilov and Ross, 1994, pp.30-31). Similarly, as noted above, composition has been defined as a form of abstraction in RM-ODP. Examples of emergent properties of software systems may include reliability, execution speed, complexity, maintainability, and what a system of intelligent agents "knows". Even more specific examples include: a data warehouse as a composition of databases (together with its metadata); a database as a composition of data model, DBMS, schema, files, and code; the Google search engine as a composition of "whatever is inside"; etc. Thus, competitive advantage may be considered as an important example of an emergent property of the Google search engine while most properties of its components (with the exception of the user interface simplicity) are not visible to its users.

In understanding systems we often distinguish between different kinds of composition. For example, the distinction between a traditional and a modern corporation as described in (Drucker, 2001) may be exactified as the distinction between a hierarchical and a non-hierarchical composition of parts of that corporation (Kilov, 2002). In a similar manner, we may distinguish between a traditional and modern industry: the former is composed of industry-specific technologies, and this composition is hierarchical because the technologies pertain only to "their" specific industry, while the latter is composed of various technologies, and this composition is non-hierarchical because many technologies are not specific to that industry and thus are reused as components by various industries (Kilov, 2002). Clearly, such a rough draft of the ontology of an industry or of a corporation – at a very high abstraction level – may be scribbled on the back of a proverbial envelope and successfully used by stakeholders for demonstrably reasonable decision making. Note that the semantics of all elements, and especially of *all relationships* between elements, in these back-of-an-envelope presentations is well-defined and *can be easily explained* to existing and new stakeholders.

Business Patterns

Composition is not the only important concept in systems thinking.

Business and IT modeling activities should never start with a blank sheet of paper. By using business patterns (also known as templates – "specifications of the common features of a collection of [items] in sufficient detail that an [item] can be instantiated using it" (ISO/IEC, 1995)) and discovering where such patterns can be instantiated in various business and IT system contexts, analysts or designers make their work substantially easier and at the same time more challenging: they can handle more complex systems without the need to reinvent basic constructs over and over again. In other words, the essence of analysis and design work can be described as pattern matching in context.

Again, the concept of a (business) pattern is not new at all. It was presented, in a clear and explicit manner, by Adam Smith in his *Theory of Moral Sentiments* (1759):

"When a number of drawings are made after one pattern, though they may all miss it in some respects, yet they will all resemble it more than they resemble one another; the general character of the pattern will run through them all; the most singular and odd will be those which are most wide of it; and though very few will copy it exactly, yet the most accurate delineations will bear a greater resemblance to the most careless, than the careless ones will bear to one another."

Business patterns used in ontologies – and in analysis and design in general – may be classified into fundamental (such as "invariant"), generic (such as "composition"), business-generic (such as "contract") and business-specific (such as "financial derivative", or "foreign exchange option"), and if business-specific (or even business-generic) patterns are not available or not known to the analyst, it is always possible to use the less specialized generic or fundamental ones (Kilov, 1999; Kilov, 2002). In applying patterns, in creating new reusable patterns (and in modeling in general!), it is essential to use abstraction ("suppression of irrelevant detail" to enhance understanding (ISO/IEC, 1995)) and exactification which "consists of replacing vagueness with precision [and] is attained by using, wherever necessary, the exact and rich languages of logic and mathematics instead of ordinary language, which is necessarily fuzzy and poor because it must serve to communicate people of widely different backgrounds and interests" (Bunge, 2001a). The same approach of using abstraction and precision – together with a small number of well-defined concepts – has been successfully advocated and used by such founding fathers of computing science as E.W.Dijkstra (for example, (Dijkstra, 1976)) and C.A.R.Hoare (for example, (Hoare, 1983)).

When we understand a business (be it a traditional business or that of some IT system) and communicate this understanding to others we want to concentrate on a stable foundation – the business domain – discovered by means of pattern matching in context. These patterns do not exist in isolation: to quote Lawvere, one of the founding fathers of modern category theory, "comparing reality with existing concepts does not alone suffice to produce the level of understanding required to change the world; a capacity for constructing flexible yet reliable *systems* of concepts is needed to guide the process". In the same manner as in science we deal with laws of nature, in business we deal with "business laws" – "patterns satisfied by facts" (Bunge). Furthermore, various actions including those to be accomplished by computer-based IT systems (and described in the requirements for such systems) substantially refer to the things and relationships of the appropriate domain and therefore should use the business domain model. In other words, the (relatively stable) ontology of the domain comes first, before discovering and formulating any (relatively volatile) system requirements. And in the same manner as relationships are not the same as semantic-free links connecting relationship elements, ontologies are not the same as data dictionaries.

Conclusion: Communication and decision making

Communication between experts in different domains is only possible on the foundation of a joint ontology, and therefore such an ontology is essential for

successful communication between (traditional) business and IT experts. To create this ontology, it is necessary to use *a common system of concepts and constructs* applicable to and extensible within any specific viewpoint. Exact philosophy defines such a system that was also standardized in RM-ODP and GRM. RM-ODP demonstrates how the same foundation (ISO/IEC, 1995) has been successfully used to define various specification viewpoints. Similarly, Gerald Weinberg in his very pragmatic text on general systems thinking (Weinberg, 1982) states that "the student trained in general systems thinking can move quickly into entirely new areas and begin speaking the language competently within a week or so". He further notes that mastery of a person's native (plus at least one non-native) language together with mastery of mathematics are essential for success in general systems thinking; in E.W.Dijkstra's independent opinion, these are the only two necessary prerequisites of a good programmer.

Since a computer-based system has to be exact, exactification is needed and useful in information management, business and IT modeling, decision making, etc. Moreover, independently of whether computer-based systems are or will be used, decision making in any system ought to be based on clear and explicit foundations: as E.W.Dijkstra stressed, it is pondering (or modeling) that reduces reasoning to a doable amount. Caveats certainly exist, especially in business systems, such as unpredictability of human actions (von Mises, 1949) and of the environment (open systems) (Hayek, 1952), but it may be possible to describe, reason about, and predict at least some essential characteristics ("patterns" (Hayek, 1964)) of such systems. In fact, it may be undesirable or impossible to provide for a very detailed model of a business domain and especially processes, particularly when some actions are accomplished by humans rather than by computer-based systems (or when the unpredictable context, such as the market context, is of substantial importance). At the same time, precise and *abstract* models are possible and desirable: "precise" is not the same as "detailed", and in abstract models irrelevant details are suppressed to enhance human understanding.

Various business patterns – from fundamental to business-generic to business-specific – based on exact philosophical concepts have been used to provide clarity and understandability in business and IT modeling, and thus to communicate between business and IT experts. It was not necessary (nor perhaps desirable) to use the term, "philosophy", to establish communication between, and with, quite a few of these experts, but the semantics of concepts could be explained and immediately used in a relatively straightforward manner.

Future work

While the concepts described in this paper have been around, and have been successfully used both in business and IT, unfortunately this usage has not (yet) become widely accepted, especially in many IT environments. The tendency to write programs before they are designed or even before they have requirements (K.Baclawski) and before the domain within which they (will) act has been understood, is still with us. On a more positive note, standardization developments

(and, implicitly, their underlying philosophical framework) described in this paper, notably, RM-ODP and GRM, have been acknowledged and even popularized in such pragmatic documents as OMG's UML Profile for Relationships (OMG, 2004) and OMG's Model-Driven Architecture, especially its Computation-Independent Model (CIM) that "is sometimes called a domain model". Furthermore, "[i]t is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other." (OMG, 2003).

A lot of research and practical work ought to be done to further the approach and activities described here for understanding, specifying and designing complex systems. At the same time, the appropriate philosophical foundation has been laid, and successfully, by such thinkers as Adam Smith, Friedrich August Hayek, Mario Bunge, and others, while the corresponding computing science and information technology foundation based on mathematics and on work of such thinkers as E.W.Dijkstra, C.A.R.Hoare, D.Bjørner, and others, has also been around for a while. These foundations are conceptually clear. Business patterns referred to in this paper have been described in literature (Kilov, 1999; Kilov, 2002) and successfully used in many industrial projects.

It would be theoretically very interesting and practically useful to provide more complete business domain models understandable – and usable! – to various business and IT stakeholders and *explicitly* based on work of such thinkers as Hayek, von Mises, and Bunge. Some preliminary work in these areas already exists (Kilov, 2002; Kilov, 2002a; Kilov, Sack, 2005). In this manner it would become blindingly obvious that both academic and practical work on ontologies, as well as on business system modeling and design ought to be accomplished not from the tools (or languages) point of view, but rather ought to be based on a sound and explicit philosophical and mathematical foundation that has existed for a long time and that has become explicitly formulated more recently (for some explicitly formulated mathematical foundations, see papers by Joseph Goguen such as (Goguen, 2004; Goguen, 2005)). The underlying concepts and approaches have been successfully taught to, and very positively assessed by, students with backgrounds both in business and information technology (Kilov, Sack, 2003). More work – including that on curricula and textbooks – is certainly needed in this area as well.

When designing and developing a good ontological infrastructure, there is no need to start from a blank sheet of paper. Very interesting work in ontology development can and should be considered, and its appropriate fragments should be discovered, abstracted out, exactified (if needed) and reused. Such formal ontologies as OpenCyc provide an excellent example of an appropriate ontological infrastructure. Of course, OpenCyc does not represent well the semantics of many elements (and especially relationships), for example, of Bunge's materialist ontology, but many of its definitions (such as those in its part-whole vocabulary (CYC, 2002)) can be extended to do so.

In the future an increasing amount of IT development will be done by developers who are not in physical proximity with business analysts and users, and thus the

communications gap between business and IT may become wider and thus more challenging to bridge. It is imperative to exactify business semantics as well as IT semantics using a common approach – especially in those business domains in which business applications are intended to be fully or partially automated. But this challenge is not completely new either: recall that one of the triggers for formulating the concepts required to describe behavioral semantics in such standards as RM-ODP and GRM (Kilov, Redmann, 1993) was the need – imposed by law – to separate between specification and implementation in telecommunications.

Acknowledgments

Many thanks go to our (virtual) teachers some of whom have been mentioned in the paper. Thanks also go to our colleagues and students, as well as to the anonymous reviewers who provided excellent food for thought in their comments. Finally, a lot of thanks go to the organizers and participants of the ONTOSE'2005 workshop (especially, to Miguel-Angel Sicilia and Salvador Sánchez-Alonso) who by means of the online discussions substantially contributed to the clarification and improvement of the presentation.

References

Bagehot, W. (1873). Lombard Street: A Description of the Money Market. Scribner, Armstrong & Co., New York.

Bunge, M. (2001). Scientific realism. (Ed. Martin Mahner). Prometheus Books.

Bunge, M. (2001a). Philosophy in crisis. The need for reconstruction. Amherst, NY: Prometheus Books.

Bunge, M. (2003). Philosophical dictionary. Enlarged edition. Amherst, NY: Prometheus Books.

Bunge, M. (2003a). Emergence and convergence: Qualitative novelty and the unity of knowledge. University of Toronto Press.

CYC (2002). http://www.cyc.com/cycdoc/vocab/part-vocab.html

Dijkstra, E.W. (1976). The teaching of programming, i.e. the teaching of thinking. In *Language hierarchies and interfaces.* (Eds. F.L. Bauer and K. Samelson), *Lecture Notes in Computer Science*, 46, pp. 1-10, Springer Verlag.

Drucker, P. (1988), The Coming of the New Organization. *Harvard Business Review*, January-February. pp. 4-11.

Drucker, P. (2001). The next society. *The Economist*, 361, 8246 (November 3rd-9th).

Dunbar, C.F. (1901) Chapters on the theory and history of banking. (Second edition, enlarged and edited by O.M.W. Sprague). G.P. Putnams Sons, New York and London.

Evans, B. (2002). *Information Week*, February 11.

Garrison, J.S. (2001). Business specifications: Using UML to specify the trading of foreign exchange options. *Proceedings of the 10th OOPSLA workshop on behavioral semantics (Back to Basics)* (Eds. K. Baclawski and H. Kilov). Northeastern University, Boston, pp. 79-84.

Gilb, T. and Weinberg, G. (1977) Humanized input. Winthrop Publishers.

Goguen, J. (2004). Data, schema, and ontology integration. In: *Proceedings, Workshop on Combination of Logics,* Center for Logic and Computation, Instituto Superior Tecnico, Lisbon, Portugal, pp. 21-31.

Goguen, J. (2005). What is a concept? In: *Conceptual Structures: Common Semantics for Sharing Knowledge: 13th International Conference on Conceptual Structures, ICCS 2005*, Kassel, Germany, July 17-22, 2005. Proceedings. (Editors: Frithjof Dau, Marie-Laure Mugnier, Gerd Stumme.) Lecture Notes in Computer Science, Vol. 3596, Springer Verlag.

Hayek, F.A. (1952). The sensory order. Routledge and Kegan Paul Limited, London.

Hayek, F.A. (1964). The theory of complex phenomena. In: *The critical approach to science and technology (In honor of Karl R. Popper)*. (Ed. Mario Bunge). The Free Press of Glencoe, London, pp. 332-349.

Hayek, F. A. (1979). The Counter-Revolution of Science. Studies on the Abuse of Reason. Liberty Press, Indianapolis.

Hoare, C.A.R. (1983). Programming as an engineering profession. In: P.J.L.Wallis (Ed.), *Software Engineering*, State of the Art Report, Vol. 11, No. 3, pp. 77-84.

Hopper, G. (1957) Automatic Programming for Business Applications. In: *Proceedings of the 4th Annual computer applications symposium, October 24-25, 1957*, Armour Research Foundation, Chicago.

ISO/IEC (1995). Open Distributed Processing - Reference Model: Part 2: Foundations (ITU-T Recommendation X.902 | ISO/IEC 10746-2).

ISO/IEC (1995a). Information Technology - Open Systems Interconnection – Management Information Systems - Structure of Management Information - Part 7: General Relationship Model, ISO/IEC 10165-7.

Kent, W. (1978). Data and Reality. North-Holland. Also reprinted by 1stBooks, 2000.

Kilov, H. (1999). Business specifications. Prentice-Hall.

Kilov, H. (2001). Back to basics. *Requirements Engineering*, 6, 3, 200-203.

Kilov, H. (2002). Business Models. Prentice-Hall.

Kilov, H. (2002a). Finding work: An IT expert as an entrepreneur. *Proceedings of the OOPSLA2002 Workshop on behavioral semantics (Serving the customer)* (Eds. H.Kilov, K.Baclawski), Northeastern University, Boston.

Kilov, H., and Baclawski, K. (Eds.) (2003). Practical foundations of business system specifications. Kluwer Academic Publishers.

Kilov, H., and Redmann, L. (1993). Specifying joint behavior of objects: formalization and standardization. Software Engineering Standards Symposium, Brighton, UK, 30 Aug-3 Sep 1993, 220-226. ISBN: 0-8186-4240-8.

Kilov, H., and Ross, J. (1994). Information modeling. Prentice-Hall.

Kilov, H., and Sack, I. (2003). An innovative university course in data management for professionals. UKAIS Conference 2003, University of Warwick (UK), April 9-11. http://www.hear-see-do.com/ukais2003/auto_abstracts.asp

Kilov, H., and Sack, I. (2005). Exploiting reusable abstractions in organizational inquiry: Why reinvent square wheels? In: *Inquiring Organizations: Moving from Knowledge Management to Wisdom* (Eds. James Courtney, David B. Paradice, John D. Haynes), Idea Group, pp. 337-359.

Kudrass, T. (2001). Coping with semantics in XML document management. In: Proceedings of the Ninth OOPSLA Workshop on Behavioral Semantics (Ed. By H.Kilov and K.Baclawski), Northeastern University, pp. 150-161.

Madsen, O.L., Moller-Pedersen, B., and Nygaard, K. Object-oriented programming in the BETA programming language. (Draft. August 11, 1992.)

Mises, L. von (1949). Human Action: A treatise on economics. Yale University Press, New Haven.

Morabito, J., Sack, I., and Bhate, A. (1999). Organization Modeling: Innovative Architectures for the 21st Century. Prentice Hall.

Ogburn, W.F. and Nimkoff, M.F. (1947). A handbook of sociology. Kegan Paul, Trench, Trubner & Co., Ltd., London.
OMG (2003). *MDA Guide, Version 1.0.1*, http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf.
OMG (2004). UML Profile for Relationships. http://www.omg.org/cgi-bin/doc?formal/2004-02-07
Peirce, C.S. (1931). Collected papers of Charles Saunders Peirce, Vols. 1-8. The Belknap Press of Harvard University Press, 1931-1961.
Raden, N. Start making sense: Get from data to semantic integration. *Intelligent Enterprise*, Vol. 8, No. 10, pp.25-31.
Smith, A. (1776). An Inquiry into the Nature and Causes of the Wealth of Nations. Printed for W. Strahan; and T. Cadell, London.
Weinberg, G.M. (1982). Rethinking systems analysis and design. Little, Brown and Company, Boston and Toronto.
Wittgenstein, L. (1933). *Tractatus Logico-Philosophicus*. 2nd corrected reprint. Kegan Paul, Trench, Trubner & Co. Ltd., London.
Yaglom, I.M. (1986). Mathematical structures and mathematical modeling. Gordon and Breach Science Publishers.