Using RM-ODP to bridge communication gaps between stakeholders

Haim Kilov
Independent Consultant, and *Stevens Institute of Technology*
haimk@acm.org

Abstract

The proverbial communication gap between business and IT experts has led to substantial problems in interoperability between different stakeholders of different (business and IT) systems, leading, in turn, to significant monetary losses together with loss of customers' trust and patience. The paper demonstrates not only the problems but also the solution — a clear separation between the business and IT domains based on an explicit usage of a system of concepts common to all domains and understood by all stakeholders. These elegant concepts come from exact philosophy, mathematics, programming and systems thinking and have been described in an international standard, the Reference Model of Open Distributed Processing The paper shows how a system of exactified concepts and approaches has been used to understand and specify the semantics of non-trivial industrial business and IT systems, thus establishing a basis for successful communication between business and IT experts, that is, for semantic (and sometimes syntactic) interoperability.

1. Introduction

In thinking about and discussing interoperability, we observe that the systems that have to interoperate need not be computer-based. Specifically, we may — and probably ought to — look at business-based and IT-based stakeholders as components of several systems that often have serious difficulties in communicating.

The proverbial communication gap between business and IT experts has led to substantial problems in interoperability (both syntactic and semantic) between different stakeholders of different (business and IT) systems, leading, in turn, to significant monetary losses together with loss of customers' trust and patience. The paper demonstrates not only the problems but also the solution — a clear separation between the business and IT domains based on an explicit usage of a system of concepts common to all domains and understood by all stakeholders. These elegant concepts come from exact philosophy[1], mathematics, programming and systems thinking. They have been successfully used not only in theory but also in industrial practice, in international standards such as the Reference Model of Open Distributed Processing — RM-ODP, and in teaching of business and IT modeling. The paper shows how a system of exactified concepts and approaches, especially such concepts as system, type, relationship, composition, pattern, name in context, etc., has been used to understand and specify the semantics of non-trivial industrial business and IT systems, thus establishing a basis for successful communication between business and IT experts, that is, for semantic (and sometimes syntactic) interoperability.

2. Conceptual requirements for interoperability

Let us start with a description of two familiar kinds of interoperability. "Syntactic interoperability is all about parsing data correctly. Semantic interoperability requires mapping between terms, which in turn requires content analysis. This requires formal and explicit specifications of domain models, which define the terms used and their relationships. Such formal domain models are sometimes called ontologies." [32] This description may be used not only for dealing with computer-based information systems but also — and perhaps more importantly — for dealing with *human stakeholders communicating with other humans or with computer-based systems*.

This reference to domain models and relationships is not new. Walter Bagehot, one of the founders of modern money markets, suggested in 1873 the same approach in order to understand and communicate about the objects in the money world: "[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." [2].

Recognition of *importance of domain models* has not been always forthcoming in the context of computer-based

[1]Our interest in and usage of exact philosophy may be explained, for example, by Bunge's observation that concepts and hypotheses are philosophical because they occur in a large number of fields of inquiry [4]. This was recognized in the curricula of (at least) the early universities. Business analysts as generalists who work on analyzing any systems, and (information) system designers belong to the esteemed class of people who use and sometimes develop such concepts and hypotheses.

information systems. However, in any engineering discipline, domain understanding and specification is essential before system requirements can be understood and formulated, and of course, a system can be developed only on the basis of well-understood requirements. This was emphasized, for example, by Dines Bjørner: "domain, requirements and software design are three main phases of software development". As we all know, sometimes IT system requirements exist only "in the collective heads" of the developers, leading to more or less serious failures described, for example, in Peter G. Neumann's "Risks to the Public" column in *Software Engineering Notes*. Therefore interoperability between stakeholders during domain modeling, as well as during requirements understanding and specification, is essential for success in development of any systems, including software systems.

The communication gap between business and IT experts can be exactified as the absence of interoperability. Firstly, often there is no *syntactic* interoperability because business stakeholders are often unable to parse data used by IT stakeholders. Of course, business experts cannot (and should not!) read code, but they also often cannot read specifications written by IT experts using notations[2] (or terminology) which are overly complicated or alien to business. Here is an example from the OOPSLA'99 keynote (on e-business) by Stu Feldman: "We need to understand the domain before addressing software. … Business models are the basis of an organization's entire activity. They are to be understood by CEO and CFO, not just by CIO; and therefore explained without 'method calls will have an XML representation'."

Secondly, often there is no *semantic* interoperability between business and IT experts and also between different business experts, as well as between different IT experts. This happens because the same data (such as "meaningful names") may be and often are interpreted differently by different stakeholders, and in the absence of an explicit domain model these differences may not be discovered until it is too late. Grace Hopper observed in 1957 that "[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" [9]. The same kind of problems still exists today, and we still often encounter statements like "everyone knows what a 'patient' is", or "everyone knows what a 'trade confirmation' is". Many tool vendors avoid these issues, and for a good reason: understanding and solving semantic interoperability problems requires human intervention that cannot be replaced with any tools.

3. Obstacles for interoperability

Interoperability problems are not specific to software systems and have been acknowledged by many business and IT experts. Solution attempts have existed for quite a while, but often they resulted in not much more than warm and fuzzy feelings during meetings. Many if not most failed attempts were based on various more or less fashionable information technology tools and methodologies instead of clear and explicit domain ontologies. On the one hand, most specifications have relied on (a lot of) tacit assumptions which are clearly different for different specification readers. On the other hand, even explicit fragments of specifications presented using box-and-line diagrams (as all too often various architectures and even business plans have been shown), or in natural language, lead to serious problems because different people will interpret the specification differently. Indeed, in order to transmit a message from one person to another without loss of meaning, the author and recipient(s) of the message have to use the same ontology and the same notation. Using the same natural (colloquial) language as a notation is not sufficient since in order to preserve meaning we need also the same context, the same language experience, language norms, cultural tradition, and so on [21], and these properties of different people are often implicit and (very) different. At the same time, a restrictive artificial language *with precisely defined semantics* that does not have contexts, cultural traditions, and so on, can guarantee an adequate transmission of a message's *semantics*, provided, of course, that it can be adequately represented in that language. As a somewhat crude approximation of such a restrictive language, we may consider "legalese" in which, for example, laws (providing "the same context") and contracts are written.

More recently, the recognition of importance of ontologies (see, for example, an overview in [37]) could have become a much-needed (social) innovation — not even a radical one — used to solve the interoperability problems.

However, ontology development today is in a poor state. Often, it has been replaced with an emphasis on (a large number of) logic and ontology languages, in the same manner as programming has been replaced with an emphasis on various "baroque programming languages" blurring our vision "by the wealth of their mutually conflicting 'powerful features' " (Dijkstra), or in the same manner as analysis has been replaced with an emphasis on "Undefined modeling languages" (Parnas). As a (or the) result, the complexity of the domain and problems has been replaced with the complexity of the language (as

[2]Business experts have no time or desire to study — for 5 days, 8 hours a day — the many facilities of a notation using toy examples, as often happens when popular "powerful" notations are taught.

Dijkstra observed, such languages were used to express in a funny way the usually given algorithms). Furthermore, in the context of modeling and ontology languages, methodologies are being created for the sake of understanding how to use the usually complex tools. Because of the lack of any generally accepted processes and methodologies, the tools exist independently and have little support for or concern with interoperability (Ken Baclawski). Also, most conceptual modeling activities have proceeded without the benefit of theory [37]. These often buzzword-compliant approaches are tinkering (Bunge) rather than engineering ones.

The primary goal of a programming language is accurate communication among humans. Clearly, the same is true about a modeling (or ontology) language. With many currently popular languages, this goal has not been reached.

As a result, even in cases when a specification exists and has been agreed upon by the relevant stakeholders, it may not guarantee interoperability of compliant systems, computer-based or otherwise. For example, if one vendor of a supposedly compliant product interprets the specification one way and another vendor interprets it another way, then both will claim compliance yet the products won't interoperate, and nobody can say for certain which interpretation is "the right one". For another example, different business experts may agree on the apparent validity of the business specification, but may interpret tacit underlying assumptions in different ways, and therefore their understandings of the specification — composed of the explicit parts and the tacit assumptions — will differ leading to serious (IT and) business problems. As an illustration, we may recall the "dot-com" epoch when people "suddenly realized that they had invested a fortune based on a few beautiful graphics that were laughably called a business plan" [22].

4. A system of common concepts: The way to address obstacles to interoperability

4.1. The need for effective patterns of reasoning

Information management systems are becoming more complex and non-trivial since they have to serve the needs of complex, non-trivial and rapidly changing businesses. In order to succeed in understanding, specifying, designing and developing information systems we should do better than use rigid methodologies combined with step-by-step approaches, or, alternatively, specification-free approaches[3].

This appears to be difficult (but is not) and perhaps unusual. As E.W.Dijkstra observed a while ago, "many students don't want to be shown effective patterns of reasoning, they want to be told what to do. ... They expect a so-called 'complete methodology'... and complain when they don't get what only the quack can provide. (We just addressed a bunch of industrial computing scientists, and the above phenomenon was alarmingly pronounced.)" [5]. Instead of using "junk food" — the metaphor for simplistic methodologies we owe to Paul Clermont — we should better use effective patterns of reasoning that help at all stages of information management, as well as in business management, independently of any possible computer-based realization of its fragments.

4.2. Where do we get these patterns of reasoning?

We certainly do not *want* to invent them for each and every project or stage of a project. Fortunately, we do not *need* to do that either: an excellent and well-structured system of common concepts on which patterns of reasoning may be based, already exists. It was defined in an abstract, precise and concise — elegant! — manner as an international standard: the *Reference Model of Open Distributed Processing (RM-ODP)* standardized by the International Standards Organization (ISO) in 1995 [25, 26].

4.3. Semantics

RM-ODP specifies semantics in a manner that is syntax-, methodology-, and tool-neutral. It provides *precision without programming*[4]. The Foundations of RM-ODP are very short — only 18 pages. Every concept there is precisely defined in clearly structured English within the context of other precisely defined concepts. In other words, the reader

[3]Here is a fragment of the description of the event "Using formal methods to understand requirements better" at the Imperial College London (November 6, 2002) [27]: "**Anthony Hall** : [...] by being precise, early, you could discover problems and reduce trouble during development.[...] Precise requirements could be defined in Z, a logic notation. [...] Z found as many errors as unit testing, but was five times cheaper. It found errors early too: many of the errors introduced by specification were removed during architecture or detailed design, and in a recent project only one specification error survived through to operations. Modestly but with some pride he quoted an authentic customer statement: "The system behaves impeccably as expected." **Axel van Lamsweerde** said that he agreed 1000%, but what of people who advocated agile methods? Anthony Hall replied by quoting John Barnes: "Ada is not meant to make programming quick. It's meant to make programming slow. Slow is good". There was laughter."

[4]This expression was used by Anthony Hall. Some IT experts claim that "business people cannot understand precision" and that therefore business experts should be provided only with various narratives and pictures instead of precise specifications. These condescending claims are obviously wrong since business experts have understood precision and made precise decisions for millennia.

does not have to figure out what a particular term means, and neither does the reader have to rely on tacit assumptions left undefined since "everyone knows what this means" (but do they know and mean the same things?).

RM-ODP eliminates much of the work that would otherwise be required by each organization to develop its own similar but proprietary guidelines — patterns of reasoning used to understand and to specify businesses as well as to specify, design and develop information systems for these businesses. RM-ODP (hopefully) will also provide the incentive to many business and IT organizations to follow similar approaches in developing specifications, leading to industry standards. RM-ODP has already been substantially used in creating other important standards, such as ISO standards — General Relationship Model and Trader, and OMG standards — UML Profile for Enterprise Distributed Object Computing, Model-Driven Architecture, and others, as well as industry-specific (vertical) business-specification standards.

4.4. Is this a radical novelty?

There is nothing radically new here. The need to elucidate the definitions of things, actions, and especially the *structure (relationships)* of a system for understanding of that system has been noted by many authors, both in modern-day information management and systems thinking, and much earlier (recall the quote from Bagehot, for example). Moreover, the concepts essential to understand and specify the semantics of system components and structure have been formulated and discussed in IT, mathematics, philosophy, and system analysis for a while. RM-ODP and the international standard used to describe relationships in more detail — the General Relationship Model (GRM) [8] — define a *system* of these concepts and are based on these ideas. This system includes such concepts as system, abstraction, viewpoint, level, object, action, state, behavior, type, subtype, template, composition, refinement, contract, name, context, invariant, pre- and post-condition, failure, error, and conformance. These *semantics* concepts are not associated with a particular technology approach (such as object orientation) and are neutral with respect to representational or tool-related issues.

Many of these concepts are quite familiar to good programmers and analysts. In particular, most have been around in programming since the mid-1960s. Many have also been successfully used in various modeling approaches within the framework of the three-schema database architecture. Some of them have been used in engineering and other areas of human endeavor (such as business and law) for centuries. The RM-ODP definitions are theoretically sound (based on mathematics, as demonstrated, for example, in the Architectural Semantics part of the standard) and have been successfully used in practice. In the same manner as businesses in the US rely on the standard Uniform Commercial Code, system specifiers ought to rely on the standard RM-ODP.

4.5. Is RM-ODP really useful?

Here is an assessment of using RM-ODP to specify systems (from the European Air Traffic Management System Architecture Workshop, held at EUROCONTROL headquarters in Brussels on June 11-13, 1996): "The application of RM-ODP provided insight into a number of interoperability issues... The RM-ODP... quickly showed the importance of correct focusing, scope, interpretation and representation within the descriptions. The RM-ODP approach impels the analyst to state clearly the focus and scope at the beginning of the process... The model provides a common, consistent and incremental approach for describing the goals, objectives and behaviour of systems in detail. Hence, it offers support for life-cycle management and for strategy studies. In a similar way, it also offers support for COTS procurement – both for the procurement specification and for the suppliers' system specification and design specification."

Look at this evidence. Not only was RM-ODP useful at the specification stages; it was valuable at all stages of information management including procurement and evaluation of suppliers' systems. This is especially important in environments where "virtually all users are configuring systems, not developing them from scratch" [28].

There exists a lot more evidence of this kind, not only in air traffic control but also in less exotic areas of telecommunications, finance, insurance, document management, medical, pharmaceutical and other industries, as well as in management and strategic consulting. For example, RM-ODP (together with GRM) was used:

- to elucidate and describe various architectures in a large international financial institution,
- to provide a successful communication mechanism for stakeholders in business process reengineering projects (and to describe these projects from specification to realization),
- to specify COTS software components in a simple and understandable manner so that the semantics of these components became clear to their users,
- to create and use simple and elegant (and complete) business specifications of various financial domains (such as accounting, trading, exotic options, etc.) used by large financial firms in their work, both for information system creation and for business process change,
- to specify products by healthcare software vendors

transforming creation of these specifications "from craft to a formal science",
- to develop a complete human resources model for DoD,
- to formulate a clear model of document management separating the concerns of content, logical design, and physical presentation,
- to provide a foundation for business decisions related to mergers and acquisitions,
- to elucidate and formulate fragments of the UML metamodel,
- to describe business strategies,
- to describe various business patterns,
- to develop ontologies of various business domains,
- to build the industry library in the pharmaceutical industry,

and so on. Some of these applications were described by satisfied clients in literature [16 (several papers), 6, 18, 13, 10, 15, 7, 31, 23, 34, 24].

4.6. Separation of concerns based on a common foundation

RM-ODP makes it possible (although not trivial!) to formulate understandable specifications in a disciplined manner. Specifications will be read by people who are non-experts in specifications. This especially applies to *business specifications* [11, 12, 14] all too often reduced to the elusive "business rules" (which are "in the code").

Discipline means precision and abstraction. *Precision* means (among other things) that a developer will not have to invent business rules that have not been described at all or have been described in an ambiguous or incomplete manner. *Abstraction* means (among other things) that a subject matter expert will not waste time and effort trying to understand business rules in terms of a particular computer-based implementation. Business rules (and a business enterprise in general) should be specified using abstract and precise concepts understandable to any good subject matter expert, analyst, developer, or (even) a non-IT manager. (Recall the quote from Stu Feldman, above.) A system of these concepts is the same for all kinds of specifications — thus providing *an excellent foundation for interoperability* — and has been formulated and described in RM-ODP and standards based on it, such as GRM.

The basic concepts defined in RM-ODP and GRM can be — and have been — used not only to describe any kinds of traditional businesses, but also to describe the essentials of any existing or to-be-created information systems (computer-based or not). In this manner, the business and IT stakeholders are able to use a common system of concepts and therefore to communicate in a meaningful manner. Of course, the syntactic representations used by different stakeholders to represent the same semantics may and often does differ, and also of course, different stakeholders may be interested in different levels and viewpoints when describing the same system, but the underlying semantic framework still remains the same.

While using the same system of concepts for all kinds of specifications, we should explicitly separate business from IT system specifications because traditional business and IT ontologies are different. (As a well-known example, a patient is not the same as one of patient's records.) Within each specification we should separate concerns as soon as we see that the specification (including a program — a specification for a computer system) becomes too complex for human understanding and is in danger of having "too much stuff". ("Precise" is not the same as "detailed", and therefore being abstract does not mean being imprecise. Good specifiers, in the same manner as good engineers, postpone decisions and do not get drowned in details. The higher the level of abstraction the more important it is to be precise!)

5. Well-structured specifications

5.1. Relationship semantics

A specification should be well-structured since only in this manner it can be carefully considered, read and understood. The structure of a system is "the collection of relations among its components or among these and items in its environments" [4]. Therefore precisely defined *relationship semantics* is essential for reading, understanding, and creating any specification. The semantics of a relationship is defined by means of its *invariant* referring to (collective) properties of relationship participants. Fortunately, it has been possible to specify the structure of a large number of diverse business and IT systems — such as financial derivatives, insurance underwriting, telecommunications systems, document management, UML metamodels, messaging, and IT system architectures — using only three kinds of generic relationships: *composition*, *subtyping* and *reference*. Semantic definitions of these relationships have been around for some time (see, for example, the exactifications in RM-ODP and GRM, as well as the modeling texts [19, 11, 14]), and it is important and instructive to observe that these definitions are based on *property determination*. In particular, the existence of emergent properties of a composite is the defining characteristic of the composition relationship[5] (see [11, 14, 4]

[5]A composition is a relationship between a "whole" (composite) and its "parts" (components). The type of the whole corresponds to the

for many examples). Thus, it also becomes blindingly obvious why an often encountered statement "this [named] line between these two boxes formally represents the relationship between these two things" does not convey anything at all about the relationship semantics, and therefore why box-and-line diagrams are inadequate for understanding and decision making.

5.2. Abstraction levels and viewpoints

Precision (exactification) is not sufficient for human understanding. Indeed, hundreds or thousands of pages of precise material are (almost) useless if the material is not well-structured. In other words, understanding requires abstraction — "suppression of irrelevant detail" [25], so that essential (for a specific level or for a specific viewpoint) aspects of a specification are clearly separated from accidental ones. Clearly, this is not a new approach, but it has not been stressed in most syntax- , methodology- or tool-oriented texts. And it is instructive that the concepts of abstraction, levels and viewpoints are among the first ones to be described in RM-ODP.

RM-ODP uses abstraction within the context of *levels*. In particular, it notes that "fixing a given level of abstraction may involve identifying which elements are atomic". For a more specific example, the concept of composition is defined using abstraction levels: "[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."

RM-ODP also uses abstraction within the context of *viewpoints* — "form[s] of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system". All viewpoints are based on the same system of basic concepts. RM-ODP specifies five basic viewpoints — enterprise, information, computational, engineering, and technology. It is possible to define correspondences between viewpoints (RM-ODP shows how to do that and provides some examples), but often, one viewpoint cannot be defined in terms of another. Of course, the five basic viewpoints are not the only ones that may be used to describe a system. In accordance with the definition of a viewpoint, any reasonable set of architectural concepts and structuring rules may be chosen to focus our attention on particular concerns within a system; and therefore we often use a *business viewpoint* and an *information system viewpoint*[6]. In this manner, for example, we can exactify the slogan "no requirements in terms of solutions" since requirements and solutions belong to different viewpoints.

types of the parts, and an instance of the whole corresponds to zero or more instances of each type of the part. There are two kinds of the properties of the whole those that are determined by the properties of the parts and the way these parts are combined; and those that are independent of the properties of any parts. A composition also satisfies the general relationship invariant that implies, in particular, that an instance of the whole cannot have itself as a part. This definition is based on the definition of composition in RM-ODP, GRM, on the definition of composition used in systems thinking (e.g., by F.A.Hayek) and in exact philosophy e.g., by Mario Bunge), and on usage of composition by such classics as Adam Smith in his *Wealth of Nations* (see [14]).

6. Business patterns and modeling

6.1. Only primitives?

RM-ODP provides a small but powerful system of interrelated definitional primitives that you can use to build your own specification. These primitives drastically reduce the number of base things and relationships and, hence, the complexity and size of a specification. More importantly, they *reduce the number of concepts* that the readers of a specification have to master in order to understand it.

The primitives need to be expanded in actual specifications. RM-ODP helps here in the form of "structuring rules" specifically designed to allow RM-ODP primitives to be used to develop more complex and/or specific definitions of various *business patterns*. These specialized definitions can be successfully mixed with the original primitives to create increasingly rich systems of definitions. This is similar to the way in which mathematicians create arbitrarily rich theorems from other theorems and well-understood basic axioms, or similar to how engineers create arbitrarily large and complex structures from common subassemblies.

The business patterns, of course, have to be discovered and explicitly formulated — and this is what business domain modeling is for. It discovers and specifies *deep analogies between seemingly different things, relationships, and processes*. In this manner, organizations can understand and deal with "always-changing" requirements as variations of a small number of conceptually simple patterns, leading to substantial savings in intellectual effort, time, and money. Among other things, it becomes possible to be demonstrably *proactive* rather than reactive in solving business problems because a clear and crisp business model may by itself provide a substantial competitive advantage for the modeled enterprise. To quote a satisfied client from a large financial firm, "[i]t has changed the way the client views software development and this single effort will serve as the foundation for other planned software development initiatives.

[6]This distinction between business and system viewpoints was made explicit, among others, in the distinction between computation-independent and the other two (platform-independent and platform-specific) viewpoints of the OMG's Model-Driven Architecture.

This business specification, written for software development, has potential application in other areas. Portions of the specification can be incorporated in corporate policy manuals; regulatory compliance documents, and serves as a basis for business process review." [7] In other words, various business and IT decisions could be based on a solid and explicit foundation rather than on handwaving, eloquence of gurus, or lemming-like considerations.

6.2. Patterns published

The reference list includes books and papers with fragments of business and IT system specifications based on RM-ODP and GRM. None of these are toy examples. Some of them are illustrative and fun but not trivial — like those modeling fragments of domains described by Lewis Carroll — while others are fragments from the generic parts of industrial specifications created for (and with active participation of) business and IT customers. These generic business and IT specifications may be, and have been, reused as business patterns in various customer engagements: after all, *pattern matching in context* is an essential part of successful analysis (and design). Since generic business patterns — such as, at different levels of genericity, invariant, composition, or contract— can be used in any application area, a good analyst can become a contributor in an entirely new area (and successfully interoperate with its stakeholders) within a very short timeframe. The conceptual foundation together with the generic business patterns let the good analyst to ask proper questions and "begin speaking the language [of the entirely new area] competently within a week or so" [35]. Thus, we may contrast Bunge's use of Claparede's criterion of intelligence as the ability to solve new problems [4] with an unfortunately typical help wanted advertisement for a business analyst stating that "knowledge of XXX is a must"

Business-specific business patterns can be discovered and formulated by reusing and exactifying their existing definitions, especially since some of them have been around for centuries, see, for example, Adam Smith's *Wealth of Nations*, rather than by rewriting and redefining them as "requirements" for each project. In other words, ontology reuse — and concept reuse in general — is much more valuable than code reuse.

Let us recall that Peter Naur proposed in 1968 [29] to use the work of Christopher Alexander long before it became fashionable to refer to it as a source of ideas about attacking the software design problem. Naur justified his choice by the fact that Alexander was concerned with the design of large heterogeneous constructions. Indeed, Alexander emphasized in *The Timeless Way of Building* that "...a pattern defines an invariant field which captures all the possible solutions to the problem given, in the stated range of contexts... the task of finding, or discovering, such an invariant field is immensely hard... anyone who takes the trouble to consider it carefully can understand it... these statements can be challenged because they are precise" [1].

Creating and elucidating a domain model cannot be automated. There is no algorithm (or tool) to do that. Such models are created and elucidated by teams consisting of domain SMEs (subject matter experts) and analysts. Even when the SMEs are experienced in formulating ontologies of their domain in an abstract (that is, understandable) and precise manner, analysts are still essential to discover, elucidate and exactify tacit assumptions common to all, or some, SMEs. *Ignorance* — real or perceived — of the subject matter is needed to make the tacit assumptions explicit. As early as 1969, P. Burkinshaw urged: "Get some intelligent ignoramus to read through your documentation; [...] he will find many 'holes' where essential information has been omitted. Unfortunately intelligent people don't stay ignorant too long, so ignorance becomes a rather precious resource." [30].

It is not sufficient to discover, formulate and use (fundamental, basic, and more specific) concepts and structures essential for a good model. We ought to communicate these discoveries, both for understanding of that model and for their usage in other, often apparently very different, models. In the modeling context, a concise and elegant system of basic concepts described in RM-ODP provides a foundation for such a language. Sometimes the *specifics* of this language or, more often, its fragments ought to be created (collectively, by the modelers together with the subject matter experts) for successful communication of a model's semantics. Models formulated in such a manner (of course, based on concepts and structures common to most systems) establish a common background used by all stakeholders of an organization and its relevant environments (e.g., clients and subcontractors) in understanding and business decision making.

7. Ontologies and invariants

7.1. Domain semantics

When we want to use an existing system (component) or plan to use a new one, we need to specify (for existing IT systems it often means "reverse engineer") the semantics of its interfaces, since only precise and explicit semantics make interoperability possible. This applies to any kind of system, independently of whether it is (or will be) computer-based. Clearly, in order to understand and define (the requirements for) interface semantics, we need to be *explicit*

and precise about the business domain in which the system works or will work. This is true for all kinds of businesses — be they traditional ones such as trading, or the business of creating an information management system, or of creating and using a relational database, or of asynchronous messaging, or of a particular general ledger legacy program. In other words, understanding and documenting of "what is there in the business domain" (also known as ontology) comes first. Within this explicitly specified framework, we can discuss the systems (and their interfaces) that work or are planned to work in this domain (after all, the descriptions of these systems refer to the things, relationships and actions of the domain!). And a data dictionary is *not* such a framework: as noted by many and as we know from practice, the same name may mean substantially or somewhat different things in different contexts, and the *structure* of these contexts — expressed in the relationships — is essential for understanding of the things we deal with.

Although ontologies have become fashionable rather recently, the underlying concepts for ontology understanding, development and use in business and IT system analysis are not new. They have been known from exact philosophy (notably, Bunge's work [3, 4]), database, object and information modeling work (for example, [37, 19]), systems thinking (F.A.Hayek[7], von Mises and others), and, of course, such international standards as RM-ODP and GRM. In addition to various kinds of subtyping, the concept of composition is among the most important in domain modeling, and its definition — based on levels of abstraction and on emergent property determination — is essentially the same in RM-ODP, GRM, and in Bunge's work.

7.2. The importance of elegance

With the introduction of standards like RM-ODP and GRM, today the analysis situation resembles the one that existed in programming in the second half of the 1970s. At that time, in E.W. Dijkstra's words, programming was in the process of moving from a craft to a scientific discipline. Most observations made by Dijkstra then for programming are reinvented now for analysis. We also see that in analysis, as in programming (an observation made by Dijkstra as early as 1962), elegance is of utmost importance; elegant specifications are liked by all and, thus, successfully used and reused. As Dijkstra said, "in the design of sophisticated digital systems, elegance is not a dispensable luxury but a matter of life and death, being a major factor that decides between success and failure", and a good specification ought to be convincing in the same manner as a good program is. For a specification to be of use (and to be produced in a reasonable manner, as any artefact produced by professional engineers), it needs to emphasize semantics rather than syntax, and to do that in an abstract and precise manner. This is precisely what RM-ODP and GRM have been designed to do.

7.3. "Learn to abstract: try not to think like a programmer" (J.Wing)

Both in programming and in analysis, the fashionable approaches have often encouraged practitioners to start their work in the middle using an operational approach. Jeannette Wing provided an excellent example of an approach to be avoided: "'If you do this and then that and then this and then that, you end up in a good state...' This [...] process quickly gets out of control. The problem is related to understanding invariants." [36]. *Invariants* are essential for defining the ontology of the domain of interest, be it a business or an information system one. In particular, they define the types of things, actions and relationships. They also determine what actions (and under what circumstances) can and cannot be executed in the context of the domain ontology. To quote a recent eloquent paper by Turski, one of the founders of computing science, a fundamental breakthrough in programming happened when "[i]nstead of preoccupation with a dynamic process ('what happens next'), we concentrated on a piece of text ('what does it say')" [33]. Indeed, the same kind of difference exists between various buzzword-compliant operational approaches to analysis that quickly get out of control, and elegant approaches that lead to understanding of businesses and information systems.

These elegant approaches start with and are based on ontologies. We do not want to start in the middle (that is, with a possible solution, as it too often happens) or even with a specific problem (that is, with requirements). Rather, *we start with the stable (that is, invariant) basics* and proceed from there. (The Information viewpoint of RM-ODP explicitly states that static and dynamic schemata are "subject to the constraints of any invariant schemata" [26].) Clearly, a problem and its solution cannot be understood and specified without the basics because the interrelated concepts used in describing the problem and the solution are defined by (and in) the basics.

7.4. Discover different ontologies

Substantial and especially somewhat different ontologies of different stakeholders (especially tacit ontologies) may make semantic interoperability difficult to achieve. In order to solve this problem, the existence of different ontologies

[7]For example, in accordance with Hayek's observations, prices constitute an essential emergent property that makes possible the functioning of a market economy and that embodies more information than each participant of a market economy directly has.

has to be made explicit. Ontologies — business domain models — are the framework for interrelating the existing vocabularies of different stakeholder communities instead of throwing these vocabularies away, thus providing interoperability solutions that really work, because they involve business *meanings*, while purely technical (or syntactical) solutions fail. Explicit ontologies make it possible to discover and specify mappings between concepts and relationships used in different ontologies (and often to enrich some of them). As a result, different stakeholders can communicate and interoperate: it will become clear which different context-specific terms have the same meaning and which identical context-specific terms have different meanings.

7.5. "Requirements always change"?

In this context, it is instructive to elucidate the (all too familiar) statement "requirements always change, and therefore it is useless to formulate them". Indeed, business *processes* often change. Such changes may lead to a competitive advantage for a business or they may even be perceived as necessary for the business to survive. Similarly, decisions about using IT systems to automate certain business processes may also change (for example, due to perceived opportunities). At the same time, the basics of a business — its ontology — have usually remained the same for centuries, if not for millennia: for example, banking and financial texts published in the early 20th century (or earlier, such as fragments from Adam Smith's *Wealth of Nations*) have been successfully used to understand and specify the corresponding business *domains*. The changes due to modernity are minimal and are mostly additions to or refinements of the existing classical models.

These considerations apply to any kind of business modeling as well as to requirements discovery and specification, independently of whether a computer-based IT system will be created or bought to automate some business processes or steps. As noted earlier, a crisp business model is used *to make demonstrably effective business decisions* only some of which are IT-related. And the concepts, constructs and standards used in creating such models are based on mathematics, "the art and science of effective reasoning" (Dijkstra).

8. Notations

Both in traditional programming and in modeling, we know from the works of Dijkstra and other founders — as well as from the experience of the best practitioners — that the inherent complexities of the problems and their solutions should not be exaggerated by imposing on the readers of programs or specifications a complex notation "with a plethora of *ad hoc* facilities of dubious value and unquestionable ugliness" [33]. In traditional programming, program readers and writers are usually humans with roughly the same background in the notation used. Contrariwise, in modeling (and specifications — the result of modeling), readers and writers often have drastically different backgrounds in the notations used. Therefore in order for business experts to use a precise notation to communicate about business models, it is absolutely essential to explain the basics of the notation on the 'back of an envelope'. This is important to get and retain the attention of the business people. Imposing a complex notation will move the reader's (and modeler's) attention from the complexity of the problems to the complexity of their representation. In other words, the attention will be moved from the essential semantics to the accidental syntax. As a result of such semiotic pollution, communication about problems suffers. This explains why many business system specifications are write-only, at least from the viewpoint of their main customers — the business decision makers.

Business (and any system) modeling notations should not be restricted by the artefacts existing or easily implementable in currently used IT systems. Thus, a modeling notation ought to be able to express multiple and dynamic subtyping, multiple decompositions of the same individual (thing or action), non-binary relationships with well-defined semantics, and other concepts defined in RM-ODP and GRM. For example, it should be possible to specify that a banking industry is a composite in a composition of banks, a federal regulator, customers, and a lot of something else (like clearing houses), and to say that in this composition both the composite and its components ought to exist together.

Turski stresses the need to limit ourselves in the process of understandable program construction to the systematic use — known as "structured programming" (invented by Dijkstra) — of a small collection of programming language instructions having "a clean and well-defined meaning". Further, he emphasizes that a high quality specification has to have a model not only in the programming language domain, but also in the language "used for the description of (a part) of the reality of interest", that is, understandable to the business subject matter experts. Following the ideas of structured programming, we may want to limit ourselves in the process of understandable specification construction to the systematic use of a *small collection of modeling constructs having a well-defined meaning*. And this collection, in fact, this system, already exists and was described in RM-ODP.

When we have to choose or use a specific notation or tool in our programming or modeling activities, we should, first and foremost, look at whether and how the semantics

of concepts we use in programming or modeling is supported by the tool. If the notation or tool is overwhelming then everything need not be lost: it may be possible to choose a (very) small subset of that notation in order to represent concept semantics. This approach is not new at all: it is well-known, for example, that various small subsets of PL/I have been created and used exactly for this purpose. Similarly, a very small subset of UML for business modeling [14] has been created and used in various engagements such as those described in [7, 12] and in several papers in [16], as well as for specifying relationships in [6]. This subset has been represented on one page.

At the same time, an important caveat is in order. When a subset of a notation is being chosen to represent concept semantics, it is essential for the semantics to have an exact representation in the notation. Since many important aspects of UML semantics have not been well-defined, it became necessary to provide for precise definitions of UML constructs used to represent the semantics of such concepts as composition, subtyping and reference relationships. This approach was accepted by OMG in the UML profile for EDOC [6].

9. Conclusions

It is now well understood that attempts to comply with a specification of any system having incomplete or unclear semantics will not guarantee interoperability because important information will be lost. The system of concepts defined in RM-ODP makes it possible to completely and precisely define the essential aspects of the universe of discourse, be it to describe a business or an information management system. These specifications are based on the semantics of the appropriate domain rather than on existing products or solutions. Since the system of concepts and constructs used for these specifications has been itself defined in a clear and crisp manner, the specifications can be *read*, understood, and thus agreed or disagreed upon by all stakeholders. Moreover, specifications of existing products or systems, including legacy systems, can and should be based on the same approach thus leading to demonstrably justified user decisions about acquiring such systems.

An RM-ODP-based specification provides a top-level precise (not semi-precise!) road-map of the appropriate fragment of a business or IT system, or of a product, and with its refinements down to the level(s) we are interested in. These specifications define what should always be true about the things and relationships of the business or IT domain as well as what should be true about each process (step, operation). All defaults are made explicit: in particular, the business expertise does not disappear when the business expert leaves the meeting room, and everything the developers need to know about the business domain and "were afraid to ask" is in the specification. Such specifications may be, and have been, used for making demonstrably justified strategic, tactical, and operational decisions in all kinds of business and IT system environments.

10. Acknowledgments

Many thanks go to the colleagues and customers the interactions with whom helped to distill the concepts and approaches described here. Some of their names are in the reference list below, and many names are in books and papers from this list. Also, specific thanks go to anonymous reviewers for very useful comments and to Rashid Bashshur for useful discussions about the structure of the paper. A draft of this paper was presented at the Telemedicine symposium in Ann Arbor, Michigan, in May 2004.

11. References

[1] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[2] W. Bagehot. *Lombard Street: A Description of the Money Market.* Scribner, Armstrong & Co., New York, 1873.

[3] Mario Bunge. *Philosophical dictionary.* Prometheus Books, 2003.

[4] Mario Bunge. *Emergence and convergence.* Toronto University Press, 2004.

[5] E.W. Dijkstra. *Management and Mathematics.* EWD966, The University of Texas at Austin, 14 June 1986.

[6] A UML Profile for Enterprise Distributed Object Computing Joint Final Submission Part I. 18 June 2001. OMG Document Number: ad/2001-06-09. 3.6. The Relationship Profile.

[7] James S. Garrison. 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, 2001, pp. 79-84.

[8] GRM. ISO/IEC JTC1/SC21, Information Technology. Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model, 1995. ISO/IEC 10165- 7.2.

[9] Grace Hopper. *Automatic Programming for Business Applications.* In Proceedings of the 4th annual computer applications symposium, October 24-25, 1957, Armour Research Foundation, Chicago.

[10] J.Hassall, J.Eaton. Applying ISO RM-ODP in the specification of CORBA interfaces and semantics to general ledger systems. *Behavioral specifications of businesses and systems* (Ed. by H.Kilov, B.Rumpe, I.Simmonds), Kluwer Academic Publishers, 1999, pp. 91-104.

[11] H. Kilov. *Business Specifications*. Prentice-Hall, 1999.

[12] H.Kilov. *Anticipating the market: The value of business models.* Distributed Enterprise Architecture Advisory Service Executive Report. Cutter Consortium, 2001.

[13] Thomas Kudrass. Coping with semantics in XML document management. In *Proceedings of the 10th OOPSLA workshop on Behavioral Semantics — Back to basics.* (Tampa, Florida, 15 October 2001), pp. 150-161.

[14] H.Kilov. *Business models.* Prentice-Hall, 2002.

[15] H.Kilov, A.Ash. How to ask questions: Handling complexity in a business specification. *Proceedings of the OOPSLA'97 Workshop on object-oriented behavioral semantics* (Atlanta, October 6th, 1997), ed. by H.Kilov, B.Rumpe, I.Simmonds, Munich University of Technology, TUM-I9737, pp. 99-114.

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

[17] H.Kilov, M.Guttman. *ISO Reference Model for Open Distributed Processing: an informal introduction.* Cutter Consortium (Distributed Computing Architecture Advisory Service) Executive Report, Vol. 2, No. 4 (April 1999). ISSN 1523-5912.

[18] H.Kilov, H. Mogill, I. Simmonds. Invariants in the Trenches. *Object-Oriented Behavioral Specifications* (Ed. by H. Kilov and W. Harvey). Kluwer Academic Publishers, 1996, pp. 77-100.

[19] H.Kilov, J.Ross. *Information modeling.* Prentice-Hall, 1994.

[20] H.Kilov, K.P.Tyson. *Semantics Working Group Green Paper One.* OMG Semantics Working Group. OMG Document Number ormsc/97-06-10r.

[21] Yuri M. Lotman. *Universe of the mind: a semiotic theory of culture.* Translated by Ann Shukman. Introduction by Umberto Eco. London and New York: I.B.Tauris & Co. 1990.

[22] Herbert W. Lovelace. The medium is more than the message. *Information Week*, July 15, 2001.

[23] http://informatics.mayo.edu/index.php?page=11

[24] K.Riemer. An analysis of RM-ODP viewpoints and system life cycles. In: *Proceedings of the 8th OOPSLA workshop on behavioral semantics* (Ed. by K.Baclawski, H.Kilov, A.Thalassinidis, K.Tyson). Northeastern University, 1999.

[25] RM-ODP-2. ISO/IEC JTC1/SC21. Open Distributed Processing - Reference Model: Part 2: Foundations (ITU-T Recommendation X.902 | ISO/IEC 10746-2).

[26] RM-ODP 3. ISO/IEC JTC1/SC21. Open Distributed Processing - Reference Model: Part 3: Architecture (ITU-T Recommendation X.903 | ISO/IEC 10746-3).

[27] *Requirenautics Quarte*rly (The Newsletter of the Requirements Engineering Specialist Group of the British Computer Society), Issue 28 (February 2003). .Ian F.Alexander. Using formal methods to understand requirements better., 4-6.

[28] Ian Sommervile. MDA revisited (Letter to the Editor). *IEEE Software*, July/August 2004, pp. 9-10.

[29] Software Engineering. Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968. (Chairman: Professor Dr. F.L. Bauer, Co-chairmen: Professor L. Bolliet, Dr. H.J. Helms; Editors: Peter Naur and Brian Randell). January 1969.

[30] *Software Engineering Techniques*. Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969. (Chairman: Professor P. Ercoli, Co-Chairman: Professor Dr. F.L. Bauer, Editors: J.N. Buxton and B. Randell) April 1970.

[31] Lawrence E. Sweeney, Enrique V. Kortright and Robert J. Buckley. Developing an RM-ODP-based architecture for the Defense Integrated Military Human Resource System. *Proceedings of the WOODPECKER-2001 (Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation) in conjunction with ICEIS'2001*, Setubal, Portugal, July 2001, pp. 110-123.

[32] [Semantic web] Hewlett-Packard. Introduction to Semantic Web technologies. http://www.hpl.hp.com/semweb/sw-technology.htm

[33] W.Turski. It was fun. *Information Processing Letters*, 88(2003), 7-12.

[34] A.Thalassinidis and I.Sack. Building the Industry Library — Pharmaceutical. In: *Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications).* (Eds. H. Kilov and B. Rumpe), Technische Universität München, TUM-I19813, June 1998, pp. 245-253.

[35] Gerald Weinberg. *Rethinking systems analysis and design.* Little, Brown, and Company, 1982.

[36] Jeannette M. Wing. Hints to specifiers. In: *Teaching and learning formal methods.* (Eds. C. Neville Dean and Michael G. Hinchey). Academic Press, 1996, pp.57-77.

[37] Yair Wand, Ron Weber. Reflection: Ontology in information systems. *Journal of Database Management*, 15, 2 (April-June 2004), iii-vi.