15

Business patterns: reusable abstract constructs for business specification

H. Kilov   I.D. Simmonds
Insurance Research Center
IBM T J Watson Research Center
30 Saw Mill River Road, Hawthorne, NY 10532, U S A
{kilov, isimmond}@watson.ibm.com

'Alice had never been in a court of justice before, but she had read about them in books, and she was quite pleased to find that she knew the name of nearly everything there. "That's the judge," she said to herself, "because of his great wig." ...

"And that's the jury-box," thought Alice; "and those twelve creatures," (she was obliged to say "creatures," you see, because some of them were animals, and some were birds,) "I suppose they are the jurors."'

Lewis Carroll. *Alice's Adventures in Wonderland.*

Abstract

Business patterns are powerful, high-level constructs that provide a natural and well-structured way of both understanding and specifying businesses and their rules. To be of use, business specifications have to be presented in an abstract and precise manner, and this paper shows how to do just that. Key concepts include business invariants and operations, and a higher level notion of "business pattern". Specifications built in this way can be parameterized and reused in various business contexts. Business patterns in particular promise to be extremely helpful as a basis for systematic business analysis and subsequent implementation of the results of this analysis. We have successfully used these concepts and constructs in our engagements with (insurance) customers (Kilov et al., 1996).

The motivation for our work is to allow the production of complete, rigorous business specifications understandable by both business users and system developers. These specifications require rigorous expressions of semantics — that is, assertions — rather than loose, "intuitive," descriptions. We present different kinds of reusable and abstract specification fragments — patterns — such as "action" and "module" patterns, which have different characteristics. We include examples of both elementary patterns — such as "composition" — and nonelementary patterns — such as "information gathering" and "joint ownership". Un-

like typical programming constructs, instantiations of business patterns are inherently interactive and so must adapt to their changing environment.

Keywords
Precision, abstraction, reuse, change, business pattern, generic relationships, collective behavior

1. INTRODUCTION

1.1 Motivation

There are countless examples showing that system development almost inevitably fails in the absence of complete, clear and rigorous specification of the business in business terms. For example, "The [US] Treasury Department has acknowledged that its decade-long effort to modernize the Internal Revenue Service's [...] computers is 'badly off the track' and must be rethought from top to bottom. [...] The Deputy Secretary of the Treasury, Lawrence Summers, told a House appropriations subcommittee that the [$20 billion] project [...] was driven by what technology was available rather than what would make the best tax collection system." (*International Herald Tribune*, March 16-17, 1996, page 3).

When a customer wishes to address a business goal with a partially automated solution, specifications are required of fundamental business needs (the "problem specification" or "business specification"), of a desired business strategy (the "business design"), of a system providing appropriate functionality (the "system specification"), and of how this system is to be produced (the "system implementation"). Whilst these four concerns are naturally and clearly separated in activities other than information management (eg in traditional engineering (Parnas, Madey, 1995; Parnas, 1995)), the practice of information management often fails to separate them, leading to excessive complexity, inconsistency, incompleteness, and therefore to failure of the development project.

The goal of a business analyst is similar to the goal of a traditional architect when understanding and eliciting a customer's wishes: it is to provide a business (rather than software) specification that will be unambiguously understood by both business users and system developers. The business specification can then be used as a basis for projects that enhance or put in place new business processes or products, some of which are to be partially automated; it is also a sound basis for defining the scope of such a project. In order to be useful for these purposes, the business specification must be abstract (with no implementation details), complete (having no gaps for developers to fill), precise (requiring no guesses over ambiguities), simple and concise (so that it will be read and understood by all).

Swatman (1994) and others have noted that the strengthening of the analysis-design boundary will be of increasing importance as companies outsource more and more of their software and system development and maintenance. Each participant — service consumer and provider — in the outsourcing relationship should have well-defined (contractual (Meyer, 1988)) obligations and responsibilities. The business specification and business design are a natural responsibility of the service consumer.

Our general goal is to ensure the widespread production and use of rigorous business specifications that can be **unambiguously understood and used** by subject matter experts (SMEs), business analysts, and system designers and developers. We believe that business specifications are an essential, and alarmingly undervalued, basis for the development of solutions to business problems. They are undervalued because too often they are not produced, and when they are, they are incomplete and insufficiently rigorous. Alternatively they are presented in terms of possible solutions and system constructs which, for reasons of lack of conceptual familiarity, may be difficult for business users to understand and thus check for completeness or accuracy. Finally, they may be too complex due to artificial restrictions imposed by using constructs based on current implementation technology.

More specifically, we have the feeling that fragments of business specification are often reinvented. Consider, as a specific example, the description and thus implementation of insurance underwriting for different kinds of insurance (life, health, etc) which have important non-trivial concepts and constructs in common, such as the insurance application folder which itself is a composition of insurance application, requests for changes to insurance application, and an underwriter's decision. (The concept of a composition is defined below.)

We want to understand and precisely and unambiguously capture these fragments, and make them available for reuse. This has been done elsewhere — and successfully — by "Real Engineers" (for example, Ohm's Law of electrical resistance[1] (Parnas, 1995a)), architects (in Alexander's "Pattern Language" (Alexander, 1979)) and even programmers (Backus-Naur form for specifying context-free grammars). As can be clearly seen from the previous sentence, formulating such a pattern ranks like a scientific discovery (and the discoverer's name is assigned to the pattern)!

A first level of generic reusable business specification fragments is presented in (Kilov, Ross, 1994; ISO, 1995a; ISO, 1995b). Experience of using these fragments — generic relationships — is presented in, for example (Kilov et al., 1996; Redberg, 1996). This paper is our first attempt at achieving a second — and perhaps more business-specific — level. Therefore the paper points to areas of future research, and contains only a few ready-made and reusable solutions.

Specifications exist at all stages of information management and, therefore, there is an opportunity to find patterns at all stages. In particular, our first level generic patterns have been successfully applied to the specification of both entire domains of business (eg medical insurance) and details of technological infrastructure (eg long transactions).

1.2 Patterns elsewhere

One definition of "pattern" given by the Concise Oxford Dictionary, is a "regular or logical form, order, or arrangement of parts". The value of a pattern when used in a specification or design is that someone familiar with it may either recognize an occurrence of it (and so better and more quickly understand the overall design), or recognize that a pattern is applicable within a particular situation (and so reuse the pattern as a known and proven specification or design strategy). The same pattern is encountered in many different contexts, and therefore can be invented once and reused afterwards rather than being separately reinvented for all contexts it is encountered in. To be reusable, a pattern must be specified in a precise, explicit, and unambiguous manner, so that it should be immediately clear whether or not it is applicable in a particular situation.

[1] "No Electrical Engineer confuses the "dot" notation for derivatives with the fact that an ideal resistor obeys Ohm's law. They also understand that the existence of resistors that are not ideal does not mean that the mathematics they have learned is irrelevant."

At the time of writing, the notion of design pattern has come to be considered as a key element of good software implementation practise, and especially when using an object-oriented programming language. A catalogue of generic and widely reusable design patterns has been published (Gamma et al., 1995) and others seek to find more specific patterns for application to more specific software design and implementation problems, such as for achieving data persistency or concurrency control.

Design patterns for software were themselves inspired by Christopher Alexander's work in architecture and civil engineering (Alexander, 1979). Alexander noted many common patterns that had been frequently reused in architecture, many over millennia and across several continents. Alexander's patterns are rigorously defined and make explicit use of specification notions such as invariants. In particular, Alexander states that "... a pattern defines an invariant field which captures all the possible solutions to the problem given, in the stated range of contexts"; that "the task of finding, or discovering, such an invariant field is immensely hard..." and that, nevertheless, "anyone who takes the trouble to consider it carefully can understand it". Our goal in discovering and formulating business patterns is to do just that, while being at the appropriate level of precision, so that "these statements can be challenged, because they are precise"! For example, "each room has light on two sides"; "establish ... marketplaces... made up of many smaller shops that are autonomous and specialized"; "traffic accidents are far more frequent where two roads cross than at T junctions" (Alexander, 1977).

Even within architecture, Alexander's work on patterns remains controversial. Nonetheless his work has originated in a mature and ancient discipline. Both his work and that of patterns for software design satisfy another (reuse-related) definition of pattern from the Concise Oxford Dictionary — "a model or design [...] from which copies can be made."

2. REUSABLE BUSINESS SPECIFICATION FRAGMENTS

2.1 Basic concepts and approach

Business analysis should be done before, and separately from, the design of any imagined automated support, with coding of such a system a distant third. Business analysis involves understanding the business: elicitation of all written and unwritten business rules, and ensuring that rules are documented precisely and in a manner that will be both read and unambiguously understood by all interested parties. It tames complexity by identifying only pure business "things", relationships, rules and constraints, and the behaviors of collections of these "things". It does not refer to computers, screens, databases, tables, etc., because these are both unnecessary details and may confuse many (non-specialist) readers. It relies upon explicitly defined concepts and semantics rather than meaning presented only implicitly, either in names or informal descriptions. Implicit semantics result in the need for each reader to "interpret" — that is, invent a meaning for — the named or described concepts; each reader inevitably invents a different meaning, and the resulting system fails to meet the needs of the business, which were never explicitly captured. The major part of a business specification — the deliverable of business analysis — is a structured representation of all rules that govern the business.

Our analysis approach insists on being simple, abstract and concise. Its concepts address primarily: collections of related things (objects); what you can do to these collections (operations); and what is always true, no matter what you do to them (invariants). Most operations and invariants involve several interrelated things (example: milk a cow, buy a house, take money from customer's account). A contract for an operation (elsewhere called a use case) should state: what "things" are relevant (signature), when you have to do it (triggering conditions), when you can do it (preconditions), what is achieved (postconditions), and what doesn't change (relevant invariants only). Contracts do not say how they are to be fulfilled: this will be specified during their refinement, ie design and development.

2.2 First level - basic reusable patterns

As mentioned above, a business specification consists primarily of **operations** (specifications of changes to collections of things) and **invariants** (specification of unchanging properties of collections of things). Most invariants are about constraints on the properties of collections of several objects rather than constraints on individual objects. Given this, it was natural to look for powerful, reusable and abstract constructs — patterns of invariants, as it were — for expressing invariants (and their properties) in the relationships between objects. It appeared that a very small number of generic relationships — such as composition (see below) and dependency — are encountered in all applications and thus — if and when precisely defined — provide an excellent basis for reuse. As noted in (Mac an Airchinnigh, 1994), "abstractions are, of course, the ultimate in reusability!"; thus abstractions such as generic relationships (and invariants!) are essential for saving intellectual efforts, time, and money.

Most high-value reusable patterns are likely to refer to collections of objects since most of business is about **collective behavior**: that is, properties that are not about a single "thing" but, rather, about a "set of related things". A pattern defines a collection of interrelated things together with available operations (like the **module** described more formally in an object-oriented extension of Z (Alencar, Goguen, 1992)), so that these things are not considered in isolation. Therefore, in specifying a business pattern it will not be necessary — as implied by many legacy OO languages and thus many OO "analysis and design" methods — to overspecify and make unnecessary choices by attaching the collective state (invariant) or collective behavior (operation's pre- and postconditions) to a particular object referred to in the invariant or in the operation. An emphasis on collective-behavior-oriented (rather than isolated-object-oriented) approach has been used in such ISO standards as (ISO, 1995a, ISO, 1995b) and is essential for successful business specifications; in particular, the constructs available to the subject matter expert and the business analyst are not restricted to the ones available in a particular family of implementations.

Specifications whose invariants are expressed in terms of generic relationships can be extremely compact and readable when compared to other specification approaches, while remaining complete and precise (several examples are shown later in this paper). Nevertheless, there still remains a need to present summary views of each specification on a small number of pages, covering both operations and invariants. An important international (ISO) standard — the Reference Model for Open Distributed Processing (ISO 1995b) — provides concepts for doing just that. Moreover, certain business-specific aspects of such concepts are referred technically as the enterprise viewpoint, and, at the time of writing, are a research topic for an ISO standardization technical committee. Our hope is that business patterns will be a (however partial) contribution to providing an enterprise viewpoint for business specifications.

The generic relationships of (Kilov, Ross, 1994; ISO, 1995a) can be seen to be "elementary molecules" or "elementary business patterns", in the sense that some or all of them are encountered in all business specifications. They may be considered as building blocks from which both complex and complete business specifications, and higher-level, reusable yet still generic "nonelementary business patterns" can be constructed. Some simple nonelementary patterns were shown in (Kilov, Ross, 1994a). A "notification" is a typical example widely used in telecommunications and elsewhere: it results in the creation of an instance of a thing (such as a "traffic violation ticket") only if a triggering condition ("notification criteria") is satisfied for a particular state of a particular instance of a "monitored object". A notification is composed out of two elementary generic (reference) relationships. Another example of a nonelementary pattern — a composition of symmetric relationships — will be used in the joint ownership pattern below.

2.3 Second level

First level, basic, reusable patterns, like the generic relationships mentioned above, can be used not only for business specifications, but also for specifying their refinements, including business design and system design and development. Moreover, they can be used (and have been successfully used) to specify a "meta-model" of the information management lifecycle itself. This can happen because concepts underlying generic relationships — such as invariants and collective behavior — are encountered at all stages of information management lifecycle. Thus, these concepts are of (re)use at all stages of information management; they provide a great help as powerful generic constructs used to understand the business and specify this understanding. Composition is a good example of a first-level basic construct. Interestingly enough, others have already termed these generic relationships "patterns" (Ayers, 1996).

In moving to a second level of reusable constructs, we must formulate these patterns in terms of the solid and powerful foundation provided by the basic patterns. In particular, the patterns must continue to promote abstraction and precision, and therefore both their discovery and formulation should be free from unnecessary and irrelevant — at this level of abstraction — details, to "enhance understanding" (ISO, 1995b)

When we try to identify "second-level" concepts and constructs for business specifications, they will inevitably define and refer to common business notions. As such they will not immediately be useful in implementation, for example, although we will indicate that many business patterns may have components relevant at other stages of information management lifecycle. While a second-level pattern may refer to business notions, it will still be generic enough to be of (re)use for all kinds of businesses. Obviously, first level, basic constructs can and should be used to specify the second-level ones. Information gathering is a good example of a second-level construct.

2.4 Composing patterns to form a specification

To quote Alexander, "each pattern helps to sustain other patterns [...] the individual configuration of any one pattern requires other patterns to keep itself alive" (Alexander, 1979).

When creating a business specification, we want to construct a unified view of a business by composing fragments, specified as viewpoints (Harrison et al., 1996) each of which is a composition of patterns. An entire viewpoint (eg Underwriting) may itself follow a pattern.

Composition is essential both laterally and vertically. Laterally, different — peer — parts of a specification which refer to some common "things" must be brought together, at which point the invariants of these components must be conjoined (and reconciled, as necessary). This is a highly non-trivial problem (outside the scope of this paper); however, we want to note that business patterns provide at least a good way to specify the problem. Vertically, a high-level and, of necessity, less detailed component must be composed with its details, again requiring a conjoining of invariants. Correspondingly, at a software level, we must learn how to compose software frameworks which operate at differing levels of abstraction. Indeed, we expect that business patterns — and, correspondingly, software frameworks — may be instantiated at different levels of abstraction within a single business specification, and so may need to be composed both laterally and vertically with other instances of themselves! This most certainly happens with abstract patterns like invariants and generic relationships.

2.5 Where to look for business patterns

It seems reasonable to presume that we can identify and develop business patterns more or less on paper, at least for the early stages of the information management process such as business specification and business design. Thorough understanding of business specification and design concepts can be — indeed, is best — achieved without resort to software design considerations. After all, businesses existed and flourished for a long time without any software implementation, and business specifications existed and were even taught (eg for banking, insurance, accounting, law, etc.) without any reference to computer systems. As an example, consider a complex specification for life insurance including parameterizable contracts with customers published in 1835 (Proposals, 1835). Probably, it was not the first such specification (it appears that the concept of life insurance dates back to the 16th century, and insurance for merchant voyages goes back over millennia).

There may be a substantial amount of work in designing and implementing each business pattern. Typically, each business pattern permits several refinements depending in part upon the business and system environments; and these refinements may also be precisely formulated as reusable patterns. In this way we will incrementally build up an asset base of business patterns with corresponding software frameworks.

2.6 A more technical foundation

As mentioned earlier, our approach is concerned with collections of things ("objects"), what you can do to these collections of things ("operations" specified in terms of a "signature", "precondition", "postcondition" and "triggering condition"), and what is true about the things no matter what you do to them ("invariants"). Careful consideration of (dynamic) triggering conditions will probably require dealing with obligations (Meyer, Wieringa, 1993) and related issues which may be better discussed in the framework of a workflow specification (and thus provide precise specifications of important fragments of the workflow framework). It is common practice to factorize out system invariants — conditions that cannot be violated at any time except, perhaps, as an intermediate step in an operation passing between two valid states.

A number of highly reusable "generic relationships" (Kilov, Ross 1994; ISO, 1995a) greatly simplify the specification of many, if not most, invariants. These include "composition", "dependency", "symmetric" and "reference", some of which appear in the examples given below. Each generic relationship has been formally defined (in terms of its invariant) for reuse, and comes with specifications of associated basic CRUD (Create, Read, Update, Delete) operations applied to its participants. Specializations of the generic relationships (Kilov, Ross, 1994) (such as different mutually orthogonal kinds of composition) augment the invariants of the most generic relationships. These specializations are still generic.

Quite a few things are known about each pattern. Some of these things are captured as invariants, which are typically expressed in terms of generic relationships. Many of the things included in these invariants represent formal parameters — "slots" to be filled as the pattern is instantiated. A "slot" is a "parameter", "formal" (in the generic specification) or "actual" (in its instantiation). For example, in the generic Composition pattern below, the formal parameter (see below) "composite type" may be filled by the type "Underwriting case folder", and the formal parameter "component types" may include "Application for insurance" (with exactly one instance), "Changes to application" (with a possibly empty set of instances) and "Underwriting decision". For a perhaps more often encountered example of a Composition, the "composite type" formal parameter may be filled in by the type "Croque Monsieur", with the "component types" formal parameter filled in by an (ordered!) set {"Toasted white bread", "Mustard", "Ham", "Melted Swiss Cheese"} (with at least one instance of each of these component types).

3. FROM A WARM AND FUZZY FEELING TO A PRECISE SPECIFICATION

3.1 An example of a first-level (business) pattern: Composition

Even for very abstract and generic patterns, such as composition, it is possible to say very many things — the invariant for a composition tells a lot, and reusable CRUD (Create, Read, Update, Delete) operations are very well defined, taking several pages (Kilov, Ross, 1994). Let us consider this pattern (a generic composition) in more detail.

Firstly, "everyone knows what a composition is", thus having a warm and fuzzy feeling about this construct. A car is a composition of an engine, a body, and four wheels; a book is a composition of pages; and an insurance contract is a composition of "contract components", right?

Secondly, we observe that using examples is not sufficient: a new construct may not **exactly** correspond to any existing one; and we need to abstract away from details specific only to particular examples.

What do these constructs — presented by examples above — have in common? There exists a "composite" and "components"; and there may be several components of different types in the same composition (in Alexander's example of a marketplace composition above, the autonomous and specialized shops — components — are certainly of different types). Also, not all component instances may be present in the composition (is a car without a wheel still a car? probably, yes). Moreover, a composite and a component in the same composition should be of different types, that is, they should have distinguishable properties. We may also observe (or reuse the observation of others (Wand, 1989)) that a composite has two kinds of properties: independent of properties of any component in a composition (eg the author(s) and title of a book) and those determined by properties of components (eg the number of pages of a book, or —more interestingly — the book's abstract, or the price of Croque Monsieur in the example above).

After some effort we discovered (abstracted out) the **invariant of a composition** (Kilov, Ross, 1994): a composite type corresponds to one or more component types, and a composite instance corresponds to zero or more instances of each component type. The application-specific types for the composite and each of its components should not be equal. There exists at least one resultant property of a composite instance (determined by the properties of its component instances), and at least one emergent property of a composite instance (independent of the properties of its component instances)[2].

The composition and its invariant have thus been expressed in terms of formal parameters: composite type, composite instance, set of component types, sets of component instances for each component type, set of resultant properties of the composite, and set of its emergent properties. This list of formal parameters constitutes the signature of a composition.

Even with all this we had not captured everything interesting about a composition. We noticed that there were several different generic kinds of composition, and specified them. Indeed, a composition may be:

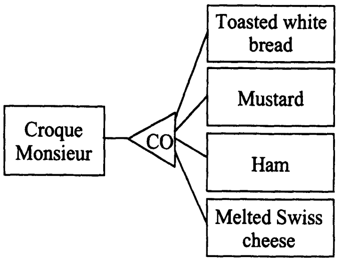
- ordered or not
- hierarchical or not
- fixed or not (an example of a fixed, ordered, hierarchical composition is the composite layering of a highway)
- and, finally, it is possible to determine whether a composite or a component can exist independently of the existence of a composition.

Thus, we described four **mutually orthogonal generic subtypes** of composition. These subtypes are defined in a more precise manner, using invariants, in (Kilov, Ross, 1994). Moreover, we may and probably should define generic operations (like "add a new component of this type to that composition") for each of these subtypes. Such an operation will be defined precisely, just as any other operation, by using pre- and postconditions, etc., as noted above; some of these conditions are implied by the invariant of the appropriate subtype of the generic composition (eg ordered composition).

When we (re)use composition or its subtype, we do not need to repeat its invariant: it is quite sufficient to refer to it by using the name, as in:

```
Fast French Food: Ordered composition (croque monsieur,
{Toasted white bread, Mustard, Ham, Melted Swiss cheese}),
```

or, graphically, as in:



[2] It is interesting and instructive to note that this invariant corresponds to Alexander's description of an invariant (see above).

If we are still concerned about the precision of this specification (and we may well be), the next step will include using a *formal* specification language such as Z (Nicholls, 1995), perhaps with extensions. If we do that (Kilov, 1993) and translate the result back into stylized English — such as the one used above in the specification of the invariant — then this result will provide important insights and perhaps will point at still existing omissions and ambiguities, as indicated, for example, in (Mac an Airchinnigh, 1994).

Do you still think that "everyone knows what a composition is"?

3.2 An example of a second-level business pattern

As with many patterns that we have noticed reoccurring and have subsequently explicitly abstracted, described and "named," information gathering is an abstraction of a wide variety of incredibly common business operations performed in many, if not all businesses. Moreover, and frustratingly, we know that we have reinvented this pattern many times over the years, without having ever before explicitly preparing it for reuse. Examples of information gathering include: get a credit report, establish birthdate, interview candidates for a job opening, find a job, find a spouse, find a divorce lawyer.

Informally, the information gathering pattern is about obtaining some information from some environment. Gathering the information may be a non-trivial task, requiring several requests from several external sources. The information eventually obtained may not be what was originally requested, and the expectation of the requester may change over time. In general there should be a set of "quality" criteria specified in the information gathering operation's postcondition stating whether an obtained value is acceptable. There may be several sufficiency criteria, with a relation "is better than," and a way to decide when to "stop" in requesting more and more acceptable information, perhaps based on some cost-benefit assessment. This description is better than "everyone knows what a composition is," but still too imprecise to be of substantial practical use. Let us make it more precise.

Less informally, the overall operation to request information starts with some source information, some desired information (eg type of resulting value) and some sufficiency criteria, together with believed sources of further information (if any). After the operation, a result has been obtained and is deemed acceptable, although the result may not be what was originally desired. However, this obtained result will (have to) have at least some of the properties (that is, be a supertype) of the desired result. Moreover, the sufficiency criteria may be changed between the start and the end of the operation.

Inherent in information gathering is the fact that you may not receive what you requested, at which point you must resort to suboperations that decide the need for better information, seek to obtain that better information, change sufficiency criteria, and even change assumptions about what result may be the best obtainable in the situation.

Note that already we have considered aspects of both business specification — what outcomes we will accept — and business design — how we might change our sufficiency criteria based on successive responses to our requests (however, the definitions of these "change" operations belong to the business specification). Additionally in business design, we might want to assign each operation and suboperation to a particular job role within the enterprise, or decide what parts of the information gathering process may be wholly manual, wholly automated, or supported interactively, and so on.

4. TECHNICAL DETAILS

4.1 Naming of patterns and their components; precision

We have been using the term "business pattern" to cover that concept which includes "slots" for all information management stages. Alternatively, we could — probably should — choose separate stage-specific names for those fragments of a pattern that are fully instantiated at the end of each stage. Since we have yet to fully comprehend all relevant concepts, find satisfactory names for these specific notions, or fully define the boundaries between these "stages", it seems a little premature to finalize (or even suggest) names. This approach of using viewpoints, or stages, is consistent — in intention if not in name — with the ISO Reference Model for Open Distributed Processing (ISO 1995b).

Patterns and their slots must themselves be given names. These names should be carefully chosen to help people build an intuition — a "correct" warm and fuzzy feeling — as to when and how to consider applying the patterns. For example, we feel that minimal explanation and documentation can render both "information gathering" and "composition" sufficiently plain and intuitive that the potential applicability of these patterns to appropriate situations will be evident. Here we are more fortunate than our colleagues who are seeking names for design patterns, since the language and abstractions of business and society with the corresponding "intuitive understanding" have emerged together over centuries if not millennia rather than the few decades of mostly isolated maturation of the computer industry. But even for business patterns, **names and feelings are certainly not sufficient**.

Inventing names — for patterns, slots, and so on — does not, by itself, define these things. This is because however intuitive they may seem, different people will interpret the same, however "meaningful" or "natural," names in different ways. In contrast, the usage of precise specifications, including invariants and pre- and postconditions, does define things, their relationships, behavior, and so on, in a complete and unambiguous way.

4.2 Completeness/ instantiation

The (re)use of a pattern is termed instantiation. A complete pattern instantiation has to provide values for all formal parameters. However, at any one time only some of the formal parameter slots may have been filled. It is a very deliberate decision, on our part, to allow this, making explicit our concern to enable rigorous capture of incomplete knowledge and, more generally, to enable independence in decision making; that is, to actively encourage separation of concerns when capturing business knowledge. Note that a formal specification language Z (Nicholls, 1995) permits the specifier to do just that, but in a formal — rather than just rigorous — manner. For example, it permits specifying abstract, generic, and often incomplete constructs of open distributed processing, thus leading to better understanding and reuse (Kilov, Johnson, 1996).

A parameter may be, and often is, a nonelementary thing, such as a set, a relation, a sequence, and so on (as shown above in the composition examples). As such, and in the same vein as the previous paragraph, a slot may be partially filled (for example, only some members of the set may be provided), and an explicit decision must be made that the slot is complete. For example, as noted above, a composition may be "fixed", implying that all components have been enumerated and there will be no more additional components; or, more often, a composition may be variable. Note that making a decision that something is "complete" is a different concern from declaring that the decision will not be changed at some time in the future.

4.3 Stages of completion / instantiation

As suggested above, the instantiation of a pattern usually occurs in a number of stages, during business specification, business design, system design and implementation.

For each slot we can state in which (possibly many) stages it is possible to add (or refine existing) values to the slot. For example:

- most information specifically related to business processes, job roles, organizations and so on must be supplied during business design
- only those job roles (for example) that are mandated by laws, regulations and so on should be considered part of business specification

No processes, job roles or organizations should be either invented or modified during system design and system implementation, whose goals are simply to produce a system that satisfies a specification and scope of automation as identified in business specification and business design. However, a stage-specific (for example, implementation-specific) refinement may be required, probably resulting in a (for example, implementation-specific) predicate being conjoined to an existing pattern's invariant.

Some patterns may be primarily used to complete slots of other patterns at a stage beyond business specification. As such, they have no slots that are completed during business specification. An example might be a pattern to specify relationships between a business's sub-organizations; since internal structuring of a business — even if subject to external regulation — is a matter of business design, this pattern would probably have no business specification-level aspect.

5. GENERAL FORM OF PATTERNS

This section is concerned with the content and documentation of patterns. We discuss generic aspects of the "form" of patterns. In documenting a pattern, we seek to ensure that the pattern is rigorously expressed and both easily taught and appropriately presented for reference when applied in a particular situation. Also, it should be clear whether a particular business pattern is or is not applicable in a particular situation, and thus an understandable and unambiguous definition of each pattern is essential.

Inventing (discovering) a new pattern is hard, and does not need to be done by everyone involved. As reading and thus reusing is substantially easier than writing, reusing existing patterns is substantially easier than discovering new ones; and leads to reuse with substantial savings of intellectual effort, and thus time and money.

All patterns "exist" in some context. Obviously, providing all details about a context will lead to a substantial, often unacceptable, increase in the size and complexity of the specification. Therefore, only the top-level information (resultant properties of the "composite", see "composition" pattern above) about the context should be provided — and referred to — in a business pattern specification. This information (as well as any other) should be provided explicitly.

5.1 Classification of patterns into "modules" and "actions"

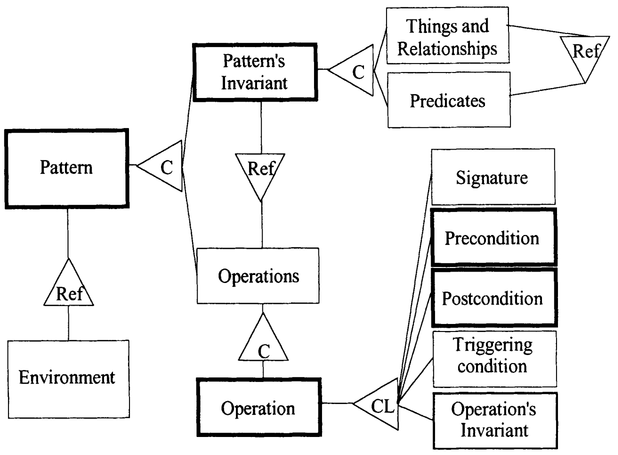
We have broadly classified all patterns that we have so far encountered as being either "actions" or "modules". This classification is related to the overall intent of the pattern. If the intent is to provided a reusable structuring of an operation and its suboperations, albeit with reusable invariants for the values of its slots, the pattern is an "**action**" (like "information gathering"). If the intent is to provide a reusable way of expressing a complex invariant, albeit with related operations consistent with and supporting the invariant, the pattern is a "**module**" (Alencar, Goguen, 1992) (like "composition" or "joint ownership").

We would like to follow our intuition and believe that many or even most patterns are either "modules" or "actions". Modules can be seen to be predominantly about high-level, reusable invariants. Actions are predominantly about high-level, reusable pre- and postconditions. In other words, a module describes constructs that exist "all the time", whilst an action — constructs that "start" and "end" (eg "repayment of mortgage"). This description is quite informal. Sometimes it is difficult (and may not be necessary) to distinguish between modules and actions: a marriage, for example, has a start and an end, but, unlike a mortgage repayment, is not about one particular operation. This should not prevent us for specifying a marriage as a business pattern — after all, both an action and a module exist in some context and have an invariant! In a similar vein, other possible broad categories could conceivably be those that are predominantly about "obligations", "triggering conditions", or "permissions". This classification of business patterns may thus be somewhat arbitrary; but the specification of a business pattern should in any case be precise, explicit, and therefore unambiguous.

Example actions might include: assessment, information gathering, negotiation, receipt of a communication and transfer of ownership. Example modules might include: ownership, joint ownership, officially registered event, bill of materials and, more abstractly, composition (see above) and reference from (Kilov, Ross, 1994; ISO, 1995a). Another example of a module has been discussed at some length by Wegner in (Wegner, 1995) — a marriage contract. This example includes obligations and permissions of the kind described in (Meyer, Wieringa, 1993), and often includes operations leading to the change of some invariants — quite typical for an open (externally interacting) system. In fact, as noted in (Wegner, 1995), this distinguishes an open system from a closed one.

Note that all of these patterns — and particularly the actions — may be applicable at several levels of granularity, while still remaining at the business specification stage of information management. For example, "in verifying the correctness of the transfer of ownership of a house I must: verify that the purported seller is the current owner (which may require that I gather information about the present ownership of the house ...) and verify that funds are available for the transfer (which may require that I gather information about the availability of funds for the transfer ...) ..." and so on. Here the actions verification — which is a special case of assessment — and information gathering are each used several times, with verification being applied and composed with itself at different levels of granularity. Both interact with module patterns related to ownership. The phrases written in brackets could have been omitted to present a high-level view of the overall verification action.

Let us describe the invariant for a general (business) pattern.



This picture is a graphical representation of the invariant of a pattern (in other words, it represents a substantial part of the precise specification of a pattern). It uses the composition pattern described above, as well as reference — another generic relationship pattern. It shows that any pattern is composed of the pattern's invariant and a composition of operations. Every operation, in turn, is composed of its signature, precondition, postcondition, triggering condition, and the operation's invariant. This composition is a list (Kilov, Ross, 1994), meaning that neither the composite, nor the components may exist independently of the composition. An invariant is composed of things and relationships, and predicates (about these things and relationships). The reference relationships show that some properties of "maintained" things — such as Operations in the `Pattern's Invariant - Operations` relationship, or Pattern's Invariant in the `Environment - Pattern's Invariant` relationship — are determined by the properties of their "reference" things — such as Pattern's Invariant in the former, or Environment in the latter relationship.

Obviously, this specification may be considered partial. It does not include, for example, such informal but useful pattern components as the description of the goal, the name of the author, etc. Also, we may refine this specification and show, for example, that there are two different subtypes of an operation — one that does not change the environment, and another that does (eg an operation in an open system may change the pattern's invariant, see below). There are other, equally interesting and important, ways to refine this specification.

For an action pattern (that defines an interesting and potentially reusable operation applied to a collection of things), the composition of operations ("subactions") shown in the picture above will be partially ordered (some operations may be executed in parallel). For a particular kind of action you may reuse many typical subactions (eg "assessment" typically involves "information gathering" subactions).

For a module pattern, usually no ordering of operations is defined. Also, new operations may be typically added (especially for modules); and operations themselves may be considered as parameters.

6. CHANGES

6.1 Inevitability

Business pattern instantiations are often interactive. In other words, they exist in an open world which permits unpredictable inputs during the "lifetime" of an instantiation of the pattern. As a result of these inputs, the environment of the pattern instantiation may change, resulting in changes of its invariant, as well as pre- and postconditions of its operations. Usually only fragments of a pattern may be considered "closed" and "atomic" and so considered to have an unchanging environment as in conventional "short" transactions (Wegner, 1995).

We have seen that these considerations apply to a module pattern, such as a marriage contract. The same, however, is true for an action pattern, for example, a mortgage repayment contract. In the United States (an environment), a mortgage repayment contract includes in its invariant the existence and properties of an escrow account held by the mortgage company to settle possible changes in taxes and insurance. When the laws and regulations about the maximum possible amount of the escrow account changed (thus providing some help to mortgage holders who did not want to have excessive amounts of money on their escrow accounts), new operations were added to the business pattern (eg "credit" — another business pattern of a particular action). Some old, existing, operations in the business pattern for a mortgage repayment contract were also changed as a result in this change of laws and regulations.

6.2 Changes to invariants

A pattern's invariant is relatively stable, but can change, especially when laws and regulations change. We have seen a marriage contract example above. As another example, "joint ownership" — perhaps of shares — may have tax consequences. When tax laws change, the invariants that model the laws — and the concepts in terms of which they are expressed — will change. Again, for example, there is currently a notion of "capital gains tax" in the US; if a proposal to restructure the tax system as a "flat tax" were adopted then the notion of "capital gains tax" may disappear completely, leading to changes to (fragments of) the invariants modeling taxation and joint ownership. These changes will imply appropriate changes to operation definitions: some operations will become unavailable because their preconditions will not be satisfiable (they even may refer to concepts no longer available!), and some other operations will have to change their pre- or postconditions.

Why will operations have to be changed? As shown above, an invariant of a pattern is referenced by (in a reference relationship (Kilov, Ross, 1994) with) all operations of that pattern. The invariant of a generic reference relationship states that some properties of the maintained object (in this case, the pre- and postconditions of the module's operations) are determined by some of the properties of the referenced object (the module's invariant). As such, when the invariant changes, all operations of that pattern must be reassessed because the precondition for each operation assumes that the invariant is also satisfied, as does its postcondition.

An example of an invariant that is, at the time of writing, changing for many businesses is: `actual year=1900+system year, where 0≤system year≤99`, be it within the business's information systems or preprinted forms (including cheques). This invariant will soon be invalidated, at which point many operations will be invalidated and will need to be changed to satisfy the new invariant (stating how "century" will be captured and affect the outcome of operations). As the pre- and postconditions of most of these operations are at best implicit in the code of some system, many billions of dollars may be spent in more or less successful attempts to resolve this problem, either in advance or as part of a cleanup.

Note that when such changes to invariants occur, they are first formulated as a change proposal before being adopted, and on adoption both old and new formulations must be maintained, at least for some time. In particular, this allows processing errors to be corrected when, for example, a letter containing a cheque arrived but "fell behind the back of a filing cabinet". In such a situation a correction must be calculated in terms of the rules in effect at each point in time between error occurrence (losing the letter) and error discovery and correction, implying the availability of each **version** of the rules. It is obviously essential to have an explicit and precise specification of the operations to be executed in such erroneous situations.

6.3 Changes to pre- and postconditions of operations

As for invariants, in most cases we assume that the pre- and postcondition stated when an operation "starts" do not change during the operation. For atomic, short-lived, non-interacting operations this is true, because there is simply no opportunity for the postcondition and invariants to change. However, for prolonged business operations — for example, some kinds of assessment such as the underwriting of a nonstandard insurance application, or selection of an appropriate treatment for a medical condition — obviously the definition of an operation (including the pre- and postcondition as well as the invariant) may change.

Once an operation has started, we lose all interest in its triggering condition and whether or not it continues to be satisfied. However, we are interested in the pre- and postcondition and the invariant until the operation is concluded, and should not (and cannot) assume that they remain constant throughout the operation.

We have already seen examples of changes to operation specifications triggered by changes in invariants (new laws and regulations about mortgage escrow accounts; or eventual changes in capital gains tax laws). Examples of changed pre- and postconditions for a (long) operation are common in a bureaucracy (an organization that tries to make the lives of its "subjects" more difficult). If a subject needs to obtain a permission from the bureaucracy and satisfies the preconditions for doing so at the moment of requesting the permission, and if the process (operation) of obtaining this permission is lengthy, then it is quite possible for the bureaucracy to change — "on the fly" — the pre- and postconditions of "obtaining permission" in such a way that the new preconditions would be more difficult to satisfy. As another example, in the midst of underwriting a particular application for health insurance, new regulations may appear stating that an underwriter must approve health insurance for persons with some specific preexisting conditions.

7. EXAMPLE OF AN ACTION PATTERN: INFORMATION GATHERING

This and the following section present examples of second-level business patterns. These examples show both the content (semantics) of the patterns, and illustrate how patterns might be presented for reuse. The language and layout used for presenting the patterns are perhaps nearer to those suitable for inclusion in a patterns catalogue (Alexander et al., 1977) than in a general introductory text about (business) patterns (Alexander, 1979). We tried to be as precise as possible, and already see some ways of improving the presented patterns.

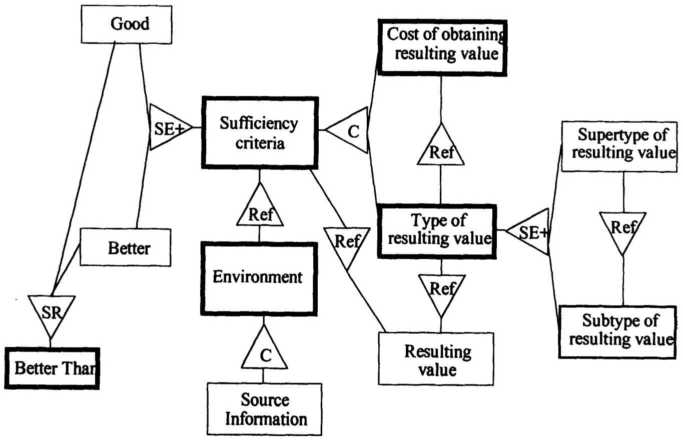
7.1 Informal introduction

Information gathering is an action to obtain some information using some environment (eg an applicant for insurance seeks to determine whether his application will be approved, declined, postponed, approved with substandard conditions; or a bureaucracy wishes to determine a person's birthdate supported by acceptable evidence).

The pattern recognizes that requested and obtained information are often different (indeed, are of different types; in the birthdate example above, it may be impossible to provide any birth certificate for an immigrant, and other supporting evidence could be used). An operation to obtain information of a particular type may retrieve less rich information (a supertype) or, conceivably, better information (a subtype). In cases where less good than requested information is received, the obtained value should be of a supertype of the requested type of the requested value.

Also, as the action proceeds there may be a change in expectations on what results can be retrieved at a reasonable cost to the requestor. Consequently, a gathering action may consist of one or more operations that request information, together with operations for making decisions to obtain more information or to change expectations for the gathering action. The pattern's invariants refer to expectations (sufficiency criteria) and types of resulting (ie requested and obtained) values.

7.2 Invariant



7.3 Operation: request for information

Signature

The signature of this operation consists of the things highlighted in its pre- and postconditions, together with the things referred to in the pattern's invariant above (which is this operation's invariant).

Precondition
- `source information` exists
- requested `type of resulting value` exists
- `environment` (element of the set of environments) together with corresponding `sufficiency criteria` exists.

Postcondition
- `resulting value` exists and satisfies one of the `sufficiency criteria` for the `environment`
- the `type of resulting value` is one of the `supertypes of resulting value`
- `cost of obtaining resulting value` is known

Note: sufficiency criteria may be changed between the start and the end of the operation.

7.4 Sub-operation: obtain better information ("further request")

Precondition

The following exist:
- `source information`
- old `resulting value`
- old `type of resulting value`
- requested `type of resulting value`
- old `environment` with corresponding `sufficiency criteria` one of which has been satisfied by the old `resulting value`
- sum of old `costs of obtaining resulting value`
- `environment` to be used to obtain better information.

Postcondition
- new `resulting value` exists and satisfies one of the `sufficiency criteria` for the new `environment` which is a `sufficiency criterion`, `better than` the old one
- new `type of resulting value` is a `subtype` of the old type `of resulting value` and a `supertype` of the requested type `of resulting value`
- new `cost of obtaining resulting value` is known.

7.5 Sub-operation: change sufficiency criteria

Precondition
- old `sufficiency criteria` for the `environment` exist.

Postcondition
- new `sufficiency criteria` for the `environment` exist and are different from the old `sufficiency criteria`.

7.6 Sub-operation: change relation "is better than" for sufficiency criteria

Precondition
- old relation `is better than` exists.

Postcondition

• new relation `is better than` exists for the same set of `sufficiency criteria` and is different from the old `is better than.`

7.7 Sub-operation: decide need for better information

Precondition

The following exist:

• `source information`
• old `resulting value`
• old `environment` with its `sufficiency criteria` one of which has been satisfied by the old `resulting value`
• sum of old `costs of obtaining resulting value`
• new `sufficiency criteria.`

Postcondition

For the given sum of old `costs of obtaining resulting value` and old `environment`, one of the following:

• `resulting value` satisfies new `sufficiency criteria` (information has been obtained)
• `resulting value` does not satisfy new `sufficiency criteria` (triggers "obtain better information")

Note: this assessment may be done by a human; and may include "override criteria" (compare with the considerations below).

7.8 Possible extension

We may also consider the nonmonotonic case where it will be discovered that something that you thought was true is in contradiction with the newly established information — in other words, that the "current" state of the system is about to become inconsistent. This situation is analogous to the one with mutually inconsistent obligations, or with mutually inconsistent viewpoints that nevertheless need to be composed (Harrison et al., 1996). One way of dealing with these problems is provided by explicit partial ordering of the importance of information (obligations, viewpoints), so that the "less important" information will be overridden by "more important" one.

7.9 Informal guidelines on how to instantiate

Information to be gathered — and the procedure for gathering it — may be arbitrarily simple or complicated. For example, a telephone number is relatively simple while a medical history of a candidate for health insurance (and her ancestors) may be extremely complex. Some businesses (such as a credit bureau) may exist simply to gather and supply information.

We suggest that the pattern is instantiated in roughly the following order:

• types of gathered (requested and obtained) values
• source information and environment(s)
• relative acceptability of gathered values in different circumstances
• sub-operations for obtaining better information
• sub-operations for deciding need for better information

• other sub-operations
• costs associated with obtaining a resulting value

The values to be gathered should be specified first since it is difficult to understand how to gather something that is itself not understood. Likewise, it is useful to understand the environment from which the information is being gathered sooner rather than later. Since it is often the case that the value actually obtained in an initial attempt may not be the value requested (perhaps the requested value is simply unobtainable as in the birthdate example above), it is important to specify all possible sources of information and the different kinds of information obtainable from each source.

We specify relative "quality" of value types in terms of subtyping relationships:
• a value is strictly "richer" than another value if it contains more information
• that is, it has all of the properties of the "less rich" value plus some additional properties (which is precisely the definition of a subtyping relationship)

In general, an operation to obtain information of a particular type may retrieve less rich information (a supertype) or, conceivably, better information (a subtype). In defining the postcondition of a specialized "obtain better information" operation, you must state what types of resulting value the operation must recognize, perhaps including a value of "unacceptable". In cases where less good than requested information is received, the obtained value should be of a supertype of the requested type of the resulting value.

For example, instead of obtaining a birth certificate for a person, it may be possible just to establish a date (or even the year) of birth from some other documental sources: thus, "obtaining a birth certificate" will have a type "establish a birthdate using the best possible document", and its supertype will be "establish some birthdate information using a documental source". The invariant includes a (partial) ordering of acceptability criteria, with a relation "better than", and a way to decide when to "stop" in requesting more and more acceptable information should be provided.

Additional details, such as job roles, precise ordering of requests, and what types of gathering to outsource, are considered to be beyond business specification and best handled during business design.

8. FRAGMENTS OF A MODULE PATTERN — JOINT OWNERSHIP

8.1 Informal introduction

The joint ownership pattern is about most kinds of property ownership. It deals with several owners of a property, leaving the property itself atomic (ie ignoring its composition). It is applicable in many cases, including situations when a property typically has only one owner, but may conceivably have several.

Examples are well-known: joint ownership of a company (shareholding), of a house, of a car (by the bank and the owner), of a racehorse (by the members of a syndicate), and so on. Another interesting example is "joint tenancy", which happens when the composition in the invariant is fixed, and can be changed only with the death of a co-owner.

At least two specializations of joint ownership exist (but will not be considered here): one in which a co-owner can dispose of his share in the property without consulting other co-owners, and the other in which this is not allowed.

We have not included all possible operations. For example, in addition to the operations presented below, there are also such operations as "redistribute the shares of interest in property" (with the operation invariant "the collection of owners is unchanged"), "transfer share of interest in property to a new party", and so on. The specification of these operations is probably straightforward.

8.2 Invariant

The `property` belongs to a non-empty `collection of owners`; and a `share of interest in property` for each `owner` has been established. (A composition of symmetric relationships, a generic molecular pattern described in (Kilov, Ross, 1994a), can be used to rigorously formulate this invariant).

Note: If the share of interest has not been established (ie it is not yet known to the business), in most cases the only operation that can be applied for this pattern is "establish the share of interest"; this operation involves information the pattern of which was presented above.

8.3 Acquire interest in property

Invariant

`Property` exists

Precondition

- `joint ownership` is not a `joint tenancy`
- the (old) `collection of owners` has been established
- it has been established that `co-owners` of the old `collection of owners` agree to the acquiring `interest in property` by the `collection of parties` to acquire `interest in property`
- the `collection of parties` to acquire `interest in property` has been established.

Postcondition

- the `collection of owners` is equal to the union of the old `collection of owners` and the `collection of parties` to acquire `interest in property`
- the new `share of interest in property` for each `co-owner` has been established.

8.4 Lose interest in property

Invariant

`Property` exists

Precondition

- the (old) `collection of owners` has been established
- the party to lose `interest in property` has been established.

Postcondition

- the `collection of owners` is equal to the difference between the old `collection of owners` and the `owner` to lose its `share of interest in property`
- the new `share of interest in property` for each `co-owner` has been established

- the `share of interest in property` of the `owner` to `lose interest in property` has been extinguished.

9. CONCLUSION AND FUTURE RESEARCH

The existence of many business patterns can be inferred from the language of business. Generic business concepts — "negotiation", "ownership", "receipt" and so on — are a rich source of relatively abstract patterns. However, to be of value, these patterns have to be(come) precisely specified. Some of them probably are precise — in economics and other professional texts such as (Alexander et al., 1977) — but are perhaps incomplete. Requirements for specification completeness are quite different for different audiences: business analysts and developers of information systems are not experts in the subject matter of the business they will partially automate, and thus require explicit specification of all "common knowledge" of subject matter experts. As this common knowledge may (and usually does) rely upon quite different default assumptions for different subject matter experts, the value (and understandability) of business patterns can be substantially improved by specifying these common-knowledge assumptions explicitly. Indeed, often a pattern can only be used once these assumptions have been made explicit. Such specifications will also lead to the discovery of different specializations (subtypes) of these patterns, and of additional parameters essential for successful pattern reuse and instantiation.

More specific language — "term life insurance policy" — will lead to more concrete, although still generic, business patterns of more focused reuse value. Given this assertion, we expect there to be considerable reuse value in action patterns that capture common business actions such as "negotiation", "assessment", "information receipt", "information recording - solicited" and "information recording - unsolicited", and module patterns such as "ownership", "joint ownership", "liability", and so on. Having said this, we intend to construct and refine a catalogue of patterns as they are encountered and reused — by us and others — during business analysis.

Note that we (almost) did not use the buzzword "Object-Orientation"in this paper. The most important of the concepts we use are **abstraction** and **precision** (leading, for example, to understanding and reuse), which have been around a lot longer than OO, and were warmly embraced and emphasized by the best OO advocates. These concepts may be used both for understanding the business and for providing technical solutions based on this understanding. Moreover, our approach yields business specifications that can form the basis for system development in a variety of paradigms including both OO and the more "traditional" ones.

An interesting test of the validity of patterns with common language roots is whether their "natural" value appears to be preserved when their documentation is (carefully!) translated into languages and cultures other than, for example, American English. It may even be that some patterns have limited applicability, being tied to specific cultures.

10. REFERENCES

Alencar, A.J., and Goguen, J.A. (1992) OOZE. In: *Object orientation in Z* (Workshops in Computing), ed. by S.Stepney, R.Barden and D.Cooper. Springer Verlag, 79-94.

Alexander, C., Ishikawa, S., Silverstein,M., Jacobson, M., Fiksdahl-King, I., Angel, S. (1977) *A pattern language*. Oxford University Press.

Alexander, C. (1979) *The timeless way of building*. Oxford University Press.

Ayers, M. (1996) Book review of "Information modeling — An object-oriented approach" by Haim Kilov and James Ross. *Software Engineering Notes*, Vol. 21, No. 2, 91-92.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995*) Design patterns: elements of reusable object-oriented software*, Addison-Wesley.

Harrison, W., Kilov, H., Ossher, H., and Simmonds, I. (1996) From dynamic supertypes to subjects: a natural way to specify and develop systems, *IBM Systems Journal*, Volume 35, Number 2, to appear.

ISO (1995a) ISO/IEC JTC1/SC21, Information Technology - Open Systems Interconnection - Management Information Systems - Structure of Management Information - Part 7: General Relationship Model, ISO/IEC 10165-7.

ISO (1995b) ISO/IEC JTC1/SC21/WG7, Open Distributed Processing — Reference Model: Part 2: Foundations (IS 10746-2 / ITU-T Recommendation X.902, February 1995).

Kilov, H. (1993) Information modeling and Object Z. In: *Proceedings of the Conference on Next Generation Computer Technology and Systems*, Haifa, Israel (June 1993), 182-191.

Kilov, H., Johnson, D.R. (1996) Can a flat notation be used to specify an OO system: using Z to describe some RM-ODP constructs. In: *Proceedings of FMOODS'96: IFIP WG 6.1 Conference on Formal Methods in Open Object-based Distributed Systems*, Paris, March 1996, 407-418.

Kilov, H., Mogill, H., Simmonds, I. (1996) Invariants in the trenches. In*: Object-oriented behavioral specifications*, ed. by H.Kilov and W.Harvey, Kluwer Publishers, to appear.

Kilov, H., Ross, J. (1994) *Information Modeling: an Object-oriented Approach*. Prentice-Hall, Englewood Cliffs, NJ.

Kilov, H., Ross, J. (1994a) Generic concepts for specifying relationships. In: *Proceedings of NOMS'94 (IEEE)*, Orlando, 207-217.

Mac an Airchinnigh, M., Belsnes, D., and O'Regan, G. (1994) Formal methods & Service specification. In: *Towards a Pan-European Telecommunication Service Infrastructure* (Lecture Notes in Computer Science, Vol. 851). Ed. by H.-J.Kugler, A.Mullery, N.Niebert, Springer Verlag, 563-572.

Meyer, B. (1988) *Object-oriented software construction*. Prentice-Hall.

Meyer, J.-J.Ch., Wieringa, R.J. (1993) *Deontic logic in computer science*. John Wiley & Sons.

Nicholls, J., ed., (1995) *Z Notation, Version 1.2*, The University of Oxford, September 1995.

Parnas, D.L. (1995) Teaching programming as engineering. In*: ZUM '95: The Z Formal Specification Notation* (Lecture Notes in Computer Science, Vol. 967). Ed. by J.Bowen and M.Hinchey, Springer Verlag, 1995, 471-481.

Parnas, D.L. (1995a). Language-free mathematical methods for software design. In: *ZUM '95: The Z Formal Specification Notation* (Lecture Notes in Computer Science, Vol. 967). Ed. by J.Bowen and M.Hinchey, Springer Verlag, 1995, 3-4.

Parnas, D.L., Madey, J. (1995) Functional Documents for Computer Systems, Science of Computer Programming, Volume 25, 1995, 41-61.

Proposals of the Massachusetts Hospital Life Insurance Company, to make insurance on lives, to grant annuities on lives and in trust, and endowments for children (1835), James Loring printer, Boston.

Redberg, D. (1996) *The search for the linking invariant: behavioral modeling versus modeling behavior.* In: *Object-oriented behavioral specifications*, ed. by H.Kilov and W.Harvey, Kluwer Publishers, to appear.

Swatman, P. (1994) Management of Information Systems Acquisition Projects, in *Proceedings of OzMISD'94, First Australian Conference on Modelling and Improving Systems Development*, Lilydale, Victoria, 3-4 February 1994, 115-131.

Wand, Y. (1989) A proposal for a formal model of objects, in *Object-oriented concepts, databases and applications*, edited by W. Kim and F. Lochovsky, Addison-Wesley, 537-559.

Wegner, P. (1995) Models and paradigms of interaction. *ECOOP'95 Tutorial Notes*, July 1995, Aarhus, Denmark (Brown University Department of Computer Science Report CS-95-21).

11. BIOGRAPHIES

Haim Kilov has been involved with all stages of information management system specification, design, and development. His approach to information modeling, widely used in telecommunications, financial, document management, and insurance areas, has contributed clarity and understandability to enterprise and application modeling, leading to specifications that are demonstrably better than "traditional" ones. It has been described in "Information modeling: an object-oriented approach" (Prentice-Hall, 1994). Haim Kilov is using and extending his approach in customer engagements. He is a member of and active contributor to several international standardization technical committees. He has been a speaker and a program committee member at numerous national and international conferences. His interests are in the areas of information modeling, business specifications, and formal methods.

Ian Simmonds has worked on techniques and tools for developing systems. He was a leading participant in the PACT and ATMOSPHERE ESPRIT projects and the EAST EUREKA project. In these and other projects his work focussed on the specification and systematic use of technical frameworks for CASE tools and integrated software engineering infrastructures. In particular, he was an active participant in the international (ISO) standardization of the Portable Common Tool Environment (PCTE). Since joining IBM's Insurance Research Center, he has been studying the application of object-oriented techniques to the development of business systems for the insurance industry, with an emphasis on reuse of both techniques and content. His interests include rigorous specification of business requirements, the use of these as a basis for systematic development of business systems, and the transfer of these techniques for use by IBM's customers and consultants.