Chapter 7.

A FRAMEWORK FOR UNDERSTANDING INFORMATION TECHNOLOGY RESOURCES

Information systems developers make use of information technology resources and tools. These have to be designed, and much of the research that is carried out in this area is aimed at producing better resources and tools. Resources include the languages in which knowledge may be represented (see chapter 6), ready-made modules and libraries of software that developers can make use of, and protocols for inter-program communication (which includes file protocols for various purposes). Tools include editors, compiler, linkers, debuggers, etc. usually now integrated into development environment software.

The communities that have researched and practised in this area include computer science, software engineering, systems engineering, software architecture, programming language design, KR language design, and also the data modelling, object-orientation, logic programming, knowledge based systems, and some artificial intelligence communities. There is, of course, considerable interaction and collaboration with IS developers since the latter influence the shape of tools and building blocks delivered to them (often the same person works in both areas). But whereas ISD is concerned with specific types of application, this chapter focuses on the creation and shaping of IT resources and tools that aspire to be valid regardless of type of application. Yet it must always anticipate their use by IS developers.

The central philosophical question addressed in this chapter is: on what basis can or should we understand the need for, and design of, such resources and tools? The central practical question is: what should guide such design?

7.1 INFLUENCES ON DESIGN

At least four influences on the design of such resources can be traced, especially of programming and KR languages.

1. The way machines work. Computers work by obeying instructions held in memory in sequence (some now employ several parallel sequences), sometimes jumping to a sequence beginning elsewhere in memory, and most of the instructions store bit patterns in memory cells, or retrieve from them for processing. The way computers work is reflected in assembler languages, which simply provide convenient lingual tokens to express the sequences of instructions. It is also reflected in some of what were called high-level languages in the 1960s. For example in FORTRAN IV

programs were composed of a sequence of individual instructions, with many 'GOTO' statements with numeric labels to alter the sequence. Program variables were seen as symbol level versions of memory cells. BASIC and COBOL also reflect this sequential working.

2. Convenience to programmers. But that was not very convenient to programmers. The GOTO construct, for example, was notoriously blamed for making programs difficult to understand, debug and modify, and was replaced by other control structures like REPEAT and DO. The long sequence of instructions was replaced by short syntactic units called blocks, which could be nested. Memory cells were replaced by program variables bound together in structures, and the content of these variables were expected to be processed as pieces of data rather than bit patterns, which led to the need to define their data type (integer, character, etc.). Complex mathematical formulae could be written declaratively (this latter was found in FORTRAN: its name is shortened from 'Formula Translation'). ALGOL, PASCAL, BCPL, C, C++ and JAVA are languages that reflect this motivation in their design.

3. Theories of how to represent knowledge. There are problems with such 'procedural' languages, and 'declarative' languages began to appear in response. Rather than specifying individual instructions ('what to do'), these declare 'what is so' and depend on some simple internal software engine to work out what to do. Each reflects a theory of how knowledge might be formalised and represented as a program: a knowledge representation (KR) theory. LISP reflected the theory that nested lists, with some being interpreted as (mathematical or logical) functions, can constitute a program; it was an early example of a 'functional language'. PROLOG reflects the theory that knowledge can be formalised and represented as statements in first order predicate logic, and is perhaps the best known 'logic programming' language. The Relational Data Model (RDM) as originally conceived by Codd [1970] reflected the theory that all data can be represented as points in multi-dimensional space and its processing by set-theoretic operations. The Object-Oriented (OO) model reflects the theory that knowledge can be represented as active objects that fall into predefined classes. RDM and OO are discussed later. OO is perhaps the predominant KR formalism in use today, but there is a move to a fourth approach.

4. The structure of real-life meaning. Throughout the developments above, another motivation is reflected in some languages: to recognise what is meaningful in certain application areas. Usually this is made available to the IS developers as ready-made features in a language. For example, COBOL, though designed in the 1950s, has many features that embody some of the things found meaningful in business, including records with fields, dates, currencies, etc. APL embodies some things found important in mathematical domains. During the 1970s, Stamper's [1977] LEGOL may be seen as an attempt to recognise what is meaningful in the legal domain, including agents and norms. Though not a language as conventionally understood, geographic information systems (GIS) may be seen as an attempt to recognise spatial things and operations.

Wand and Weber [1995] have attempted a comprehensive general ontology, though it might be seen as an example of reflecting a theory (the philosophical theory of Bunge) rather than as sensitive to the diversity of everyday life; it is discussed later. Many generalised versions of domain ontologies built in the 1990s using Object-Orientation may be seen as an attempt to formalise and generalise specific spheres of meaning found in applications. Design Patterns, an attempt in the field of architecture to design buildings and towns that reflect the diversity of human living [Alexander, Ishikawa, Silverstein, Jacobson, Fikidahl-King, & Angel, 1977] has been proposed as an approach to designing the building blocks required for software designers; this is discussed later.

There is some overlap between the third and fourth. The theory-based formalisms, LISP, PROLOG and OO could be seen as offering capability to handle lists, logic and active things as found in real life. But this is their hidden pitfall: making it easy and convenient to handle such things, they privilege seeing the whole of the diverse meaning of applications in those terms. This is often inappropriate and constitutes a reduction. This echoes Dooyeweerd's contention that theory as such is not incompatible with everyday life, and can even enrich it (§3.4.5); it is the theoretical attitude which causes problems, by which we try to view all the diversity of life through a single construct.

A framework for understanding KR resources must include some discussion for the link between the lingual activity of representing knowledge and how computing machines work at the bit level, and this is discussed later. Of the other three only the last will be discussed in depth here. This is so for two reasons. One is philosophical: we seek an understanding that is sensitive to the everyday, so neither theories of how knowledge may be represented nor the convenience of IS developers should circumscribe our discussion. The other is practical, as expressed in the call for 'KR to the people'.

7.1.1 'KR to the People'

Reflecting on the experience of the 1980s, the decade in which KR had become mature as a discipline and before it had been constrained to become 'knowledge management', Brachman [1990] suggested what would become likely scenarios and what issues would particularly need research. Many are still relevant today, especially one that stood out as relating KR to the wider context of IS development: 'KR to the people':

> "It is likely that by the millen{n}ium 'knowledge systems' will be a common commercial concept. This has important implications for the future of KR. Among other things, KR components will increasingly find themselves in the hands of non-experts, raising a novel set of issues" [Brachman, 1990,p.1090].

KR had become a specialist field from which 'the people' were excluded, mainly because, as mentioned above, it had become treated as a theory about how to represent knowledge. But it is 'the people' who should be the ones to use a KR language and other resources to

construct knowledge (or information) systems because it is they who are immersed in the everyday meaning of the application, including all its nuances, cultural meanings and taken-for-granted assumptions and norms. The need for 'KR to the people' is clear in knowledge management (KM), for example: much entry of complex knowledge (as opposed to simple data) is undertaken by 'the people' (e.g. managers).

But, though Brachman highlighted the issue of 'KR to the people', he made no attempt to address it. 'KR to the people' has yet to materialise. This is the main issue depicted in Vignette 1 in the Preface, where this author tried to gain 'appropriateness' by distinguishing four 'aspects of knowledge'.

7.1.2 Appropriateness

What criteria should a KR language or toolkit meet in order to facilitate 'KR to the people'? The conventionally recognised criteria that emerged in the KR community are sufficiency, efficiency and expressive power [Minsky, 1981], [Levesque and Brachman, 1985]. But these are criteria by which the various theories of KR could be evaluated, and are not sufficient for a lifeworld approach. Basden [1993] argued that another is more important: appropriateness. Appropriateness is when the mapping from the meaning of the domain to the available KR resources is natural: what is primitive to our intuition in the domain is matched by a primitive building block, leaving aggregation of building blocks to be needed only for what is complex in the domain.

In that proposal, which had its roots in the author's experience of representing knowledge relating to the laying out of electronic circuit boards and to data in general medical practice in the 1970s [Basden and Nichols, 1973], [Basden and Clark, 1979] and to the chemical industry and the surveying profession in the 1980s [Hines and Basden, 1986], [Basden and Hines, 1986], [Jones and Crates, 1985], [Brandon, Basden, Hamilton and Stockley, 1988], the author developed the notion of 'aspects of knowledge' [Basden, 1993] that had the characteristic of being irreducible to each other, long before he had encountered Dooyeweerd's notion. In retrospect, these can be aligned with some of Dooyeweerd's aspects:

- items: analytic aspect (distinct concepts)
- relationships: formative aspect (formed structure)
- values: quantitative aspect (discrete amount)
- spatial: spatial aspect (continuous extension)

with a fifth that was suggested later:

- text: lingual aspect (symbolic signification).

This concerns, not the syntax of a KR language as much as what Chomsky [1965] referred to as the deep structure of languages -- the broad kinds of meaning that its tokens have to convey or signify, regardless of syntactic form.

7.1.3 Extant KR Languages

Extant KR formalisms (KRFs) do not fulfil even that modest proposal. Table 7.1.3 indicates the degree to which procedural, functional, logic-programming and OO formalisms, COBOL and GIS, and Basden's [1993] proposal, facilitate representation of the meaning of various aspects of applications.

Table 7.1.3. Aspectual capability of extant KRLs

Aspect	Proc	F.P.	L.P.	OO	COBOL	GIS	B1993
Quant'ive	***	****	*	**	**	*	*****
Spatial						****	*****
Kinematic							
Physical							
Organic							
Psychic							
Analytic	*	*	****	**	**	*	*****
Formative	***	**	**	****	**	**	*****
Lingual	*	**	*	*	**		*****
Social							
Economic					*		
Aesthetic							
Juridical							
Ethical							
Pistic							

Several things stand out. The first is the large number of aspects for which no direct support is offered, and thus the general poverty of extant KRFs. The second is that apart from Basden's yet-to-be-implemented proposal there are no instances of full support (denoted by '*****'). For example, though most support integers and continuous numbers, only a very few languages support ratios as such (including, for example, removing common factors by Euclid's algorithm). For example, support for the important formative notion of structural relationships is inadequate by the extant KR approach (for the full richness of relationships as we experience them). (An early approach, similar to OO, Quillian's [1967] semantic nets, did have a richer notion of relationships, but sadly this was not taken up by later ones.) From this analysis, the whole picture is rather disappointing despite several decades of research and development in the shaping of technology. There still remains considerable opportunity for innovative and ground-breaking research and the proposal made in this chapter might suggest a useful approach.

Since 1993 a number of other authors have raised similar issues, including Gennart, Tu, Rothenfluh and Musen [1994], Stephens and Chen [1996], and Wand and Weber [1995]. The latter make a proposal for a KR kit which we discuss below. Greeno [1994] discussed a similar notion to appropriateness in user interface design: affordance, which originated in biology with Gibson [1977]. But, with the exception of Wand and Weber, few of these have attempted

to ground their discussions in philosophy.

The proposal outlined here is developed from a Dooyeweerdian understanding of the nature of things, meaning and norms. It defines appropriateness as offering the IS developer resources and accompanying KR formalisms that directly and naturally express aspectual meaning as we experience it in everyday life. In this way it makes a promise of 'KR to the people' that might be believable.

7.2 SEMI-MANUFACTURED PRODUCTS

The modules and other ready-made resources are designed and programmed. In ISD (chapter 6) the technical artefacts are likewise designed and programmed. So are they essentially the same? That the software engineering methods used to create general resources have proven largely inappropriate for development of IS artefacts for use in human everyday life suggests there is a fundamental difference. The kind of software entities of interest here are not shaped to any particular application, but seem more general. Does the difference lie in the degree of generality involved? Or is there some deeper difference?

The difference is not unlike that between building a house out of bricks and creating the bricks out of which the house is built, or crafting a piece of furniture by shaping blocks of timber and using nails etc., and the creation of those blocks of timber and those nails. We might build a traffic route-finder using the GD graphics library, but someone else must create the GD graphics library. What sound basis is there for differentiating between them?

7.2.1 The Notion of Semi-Manufactured Products

In his theory of entities Dooyeweerd recognised that what he called semi-manufactured products (e.g. nails, planks) differ in a fundamental way from what is made with them (such as furniture). "In modern life," said Dooyeweerd [1984,III,p.129], "materials are technically formed into semi-manufactured products, before they are again formed into utensils." The IS artefact or system may be seen as utensil; the IT resources from which it is made may be seen as semi-manufactured products.

Semi-products (or semi-manufactured products, SMPs) also do not fit the scheme neatly. Dooyeweerd discussed the example of a plank of wood that has been sawn and treated, but awaits being made into furniture or some other thing [Dooyeweerd, 1984,III,p.131-2]. He claimed these have no internal leading aspect, but an external one. Its meaning awaits fulfilment, in the construction of something else.

This is the key difference between the resources being discussed here and technical artefacts in IS. The IS artefacts have an internal leading aspect, governed by ERC, but resources have none, because their meaning awaits fulfilment.

Dooyeweerd did not finalise the idea (and there is evidence that it

was still under development [Dooyeweerd, 1984,III,p.132]). (Indeed, consideration of IT building blocks might provide fruitful ground for discussion of this issue by Dooyeweerdian scholars.) Being undeveloped, the idea cannot give detailed understanding or guidance. But it provides a useful starting point for understanding technological resources made available to IS developers.

Semi-manufactured products have the formative as their founding aspect [Dooyeweerd, 1984,III,p.131]. But whereas nails and planks are a mechanical technology (physical aspect), our building blocks are information technology (lingual aspect). They are shaped, not by the need to control and distribute physical forces, but by the need to represent fragments of reality in all its diversity.

This is presented as a general insight that might help prevent erroneous aspirations, expectations or directions in research and practice in this area. The requirements that guide creators of resources should differ from those that guide IS developers, even though both involve design and programming. Whereas IS developers bear some responsibility for repercussions on stakeholders (see 'Anticipating Use' in chapter 6), the responsibility of creators of resources do not; their responsibility is to aspectual meaning as such. All such building blocks are made use of by IS developers to construct their IT artefacts and systems. It is the joyful mission and serious responsibility of those working in this area of technology shaping to create such building blocks as make it possible for IS developers to adequately represent all the reality they wish to. As has been made clear in chapter 6, this could, in principle, involve every aspect.

7.2.2 The Creation of the Artefact

But this needs to be understood by reference to what the IS developer does when using such building blocks to construct an IT artefact (i.e. program or KBS). S/he represents meaning of the domain of application in some KR language (KRL). In this lingual process of representing that which is relevant, s/he makes use of meaningful ways of structuring (formative aspect) the concepts (analytic) that are meaningful, both of which may likewise be of any aspect, as shown in Fig. 7.2.2 (compare Table 6.5.1).

The tokens of the KR language express and implement primitive entities, properties, processes, etc. that are meaningful in given spheres (aspects). Meaning of the domain might be of any aspect. The lingual function of the tokens reaches out to all spheres of meaning of the application domain. For example, Table 7.2.2 shows the aspects in which various tokens of the C language are meaningful, along with a few standard (ANSI) functions.

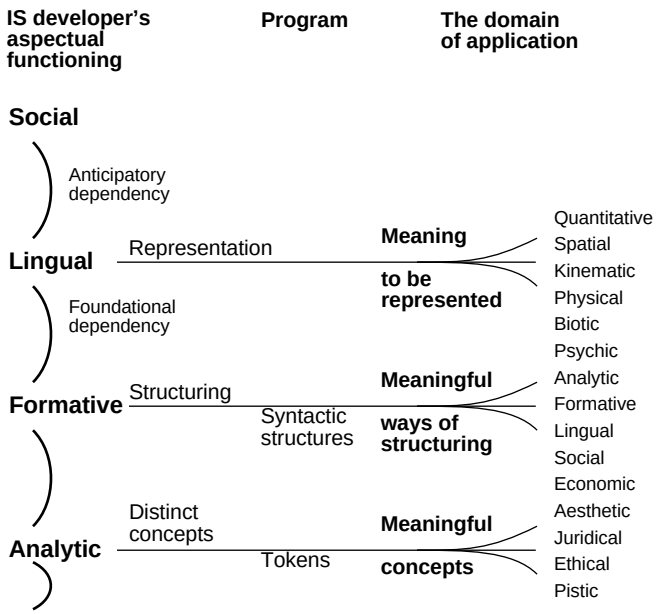

Figure. 7.2.2. The Process of Representing Meaning

Table 7.2.2. Aspects of tokens of C language

C tokens	Function in C language	Aspect
int	Amount	Quantitative
long, short	Determine how many bits to use	Psychic
char	Character	Linguaol
"..."	Text string	Lingual
>, <	Numeric comparison	Quantitative
==	Identity comparison	Analytic
=	Assignment of value to variable	Formative
++, --, +=, -=	Increment, decrement	Quantitative
&, | , >>, <<	Bitwise manipulation	Psychic
for	Apply same profedure multiple tiimes	Formative
if	Make a distinction	Analyticc
{ ... }	Block of code or structure	Formative
struct	Define a structure	Formative
a->b	Pointer	Formative
a.b	Identify part of a structure	Analytic
***var**	Define pointer	Formative
typedef	Define type of data	Analytic
ANSI function	**Function in C language**	**Aspect**
strstr()	Find substring in string	Lingual
assert()	Check program validity	Juridical
sin(), cos()	Trigonometric calculations	Spatial
qsort()	Sort a data array	Formative
remove()	Remove a file	Economic

If the IS developer is to be facilitated in representing all the diverse meaning of the domain, then the tokens and resources should reflect the types of concepts and structurings that are meaningful in every aspect. That is the central proposal of this chapter.

This proposal seems so obvious as not to require much discussion, but the ramifications of 'every aspect' are enormous. As Table 7.1.3 shows, most aspects are given no direct support. The main support is for the quantitative, psychic, analytic, formative and lingual aspects, and even in some of these support is very patchy. Though C functions for a number of other aspects are shown above, they are very few. This lack of support for most aspects might be explained by their historical focus on reflecting theories about how knowledge may be represented, rather than being orientated to the everyday lifeworld meaning of domains.

The norm of ERC, discussed in chapter 4, was that the user should be enabled and encouraged, by the represented content, to do justice to the meaning of the domain. If the IS developer -- especially 'the people' -- is to provide represented content that does this, they must find the process of, and resources for, representing knowledge easy and natural. If facilities for aspects are missing then they will find it difficult and unnatural.

7.2.3 Problems of Missing Aspects

If a sphere of meaning is missing, or made available only improperly or partially, then either the IS developers are debarred from representing such meaning, or else they must find ways to implement it in available aspects. For example, to represent quantitative meaning, tokens are needed to represent amounts, addition, subtractions, and structurings, to represent arithmetic expressions and equations, etc. -- most extant KRLs offer these. But also found in everyday quantitative life are ratios, fractions, proportions, means, standard deviations and other statistical things, approximations, and infinity -- and most of these are absent from extant KRLs.

To represent lingual meaning, the tokens and structurings needed include words, phrases, sentences, paragraphs, vocabularies, thesauri, emphasis, cross references, and the like -- but all that is usually offered is the text string and a few character-based string manipulation procedures or predicates: the necessary building blocks for the lingual aspect are mostly missing. (One exception to this is the HTML protocol, which includes tags for citing, samples, bullet lists, tables, cross references, etc. as well as all the standard 'physical' ones like italics.)

This generates a number of problems, which to this day plague IS developers.

Specialist knowledge. They need specialist knowledge of how to implement the missing aspectual things as data structures and algorithms. For example an arbitrarily complex shape (which happens to represent the boundary

fence of a piece of woodland) might be implemented as a list of coordinates that define its boundary and the type of curves or straight lines that join them (Fig. 7.2.3.1a), so they need skills in using linked-list structures. Worse, to expand the shape by a given amount (Fig. 7.2.3.1b) requires knowledge of complex trigonometry.

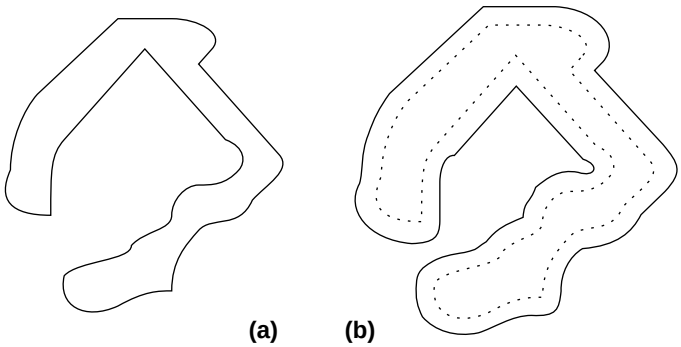

Figure 7.2.3.1. A complex shape and its expansion

Oversimplifications. Even worse, the result of such expansion might be a shape with a hole in it (Fig. 7.2.3.2), in which case a single list of coordinates is insufficient. The IS developer needs to be aware of such exceptions to the ordinary assumptions they might make. Frequently, such complexities are not discovered until long after the basic data structures have been embedded in place for some time, and it proves very costly and dangerous to change them. A very common oversimplification is relationships, which are often implemented as pointers or database keys, even though, in everyday life, relationships involve inverses, have attributes and might even themselves make relationships. Such oversimplifications might be valid at first, but can cause problems later when the system is expanded for new usage contexts.

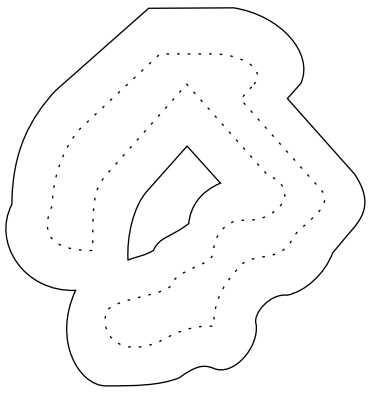

Figure 7.2.3.2. A complex shape with hole

Inner workings. Sometimes the inner workings of the resources are exposed to the danger of interference, whether

deliberate or unwitting.

Hidden side-effects. In some toolkits, the building blocks offered have hidden side-effects, which are meaningless to the pure form of the language but implemented nevertheless. For example, the ASSERT predicate in PROLOG, which, to the logical engine merely returns TRUE, but which, as a side effect, creates a new proposition or predicate. Such side-effects are dangerous, esoteric and make the IS difficult to maintain or upgrade.

Sometimes, extra building blocks are offered that implement anything other than such primitives, which can confuse 'the people' and make the IS difficult to maintain or upgrade in the future. Wand and Weber [1995] speak of 'redundant' variables.

Sometimes there is undue ambiguity about the meaning of a building block, which lead to confusion, a reduction in interoperability between systems and in misunderstandings between people, whether developers, maintainers or users. The primitives should, as far as possible, accord with informed intuition of each aspect of knowledge; this will enhance intersubjectivity between and among developers and users.

This implies that the building blocks offered should cover the entire range of aspects, and should, together, do justice to the kernel meaning of each aspect.

7.3 ASPECTUAL DESIGN OF TECHNOLOGICAL BUILDING BLOCKS

How do we do justice to each aspect? One answer is that, just as each aspect enables a fundamentally different type of being, functioning, properties, relating, rationality, normativity, etc. in the cosmos, so there should be building blocks for each aspect by which the IS developer can enable all these as represented content that is the technical artefact.

7.3.1 Philosophical Roles of Aspects to Indicate Primitives and Tokens

One proposal is that a KR toolkit should provide primitive or basic facilities that are meaningful in every sphere (aspect), which 'the people' (or other IS developers) can employ in construction of artefacts and computer systems for human life. The KR language should likewise be composed of tokens which express such basic facilities of every sphere of meaning. To do this requires reflection on the everyday experience of cosmic meaning in each aspect to identify a reasonable set of basic facilities with 'the people' would find useful and meaningful.

In each aspect, 'the people' experience entities, properties, relationships, processes and so on. The meaning of each aspect is manifested in the cosmos by means of its philosophical roles. Therefore, our proposal here is that basic facilities and tokens can be defined for each philosophical role of each aspect (refer to the relevant subsections of the section 'Philosophical Roles of Aspects' in chapter 3):

Aspect as mode of being indicates types of 'things' to make provision for (for example the sentence in the lingual aspect).
Aspect as ways of functioning suggests activity to provide as procedures (for example, Soundex searching, translation).
Aspect as basic type of property suggests attribute types to cater for (for example, emphasis, pronunciation, correctness of syntax).
Aspect as ways of relating suggests type of relationship and interacting (for example, cross reference, synonym).
Aspectual rationality indicates inferences to be built in (for example, if two sentences follow each other then it is likely they are about the same topic).
Aspectual law suggests constraints that would be meaningful (for example, vocabulary and rules of syntax).
Aspect as way of describing suggests the style of tokens of the 'language' by which this meaning might be expressed.

Thus, if we wished to create a module comprising all the useful basic facilities for an aspect, we would implement things, functionings, properties, ways of relating, inferences, constraints and UI style for that aspect. (KR 'language' is not assumed to be solely textual, and the tokens might be any input or output symbols; either as output to the user via screen, speakers etc. or as input, via mouse or keyboard gestures (for example, for the quantitative aspect, not only digits but also sliders and bars of varying length are common 'tokens'.)

7.3.2 A Practical Proposal: Aspectual Modules

Thus we are presented with a philosophical proposal: create a distinct module for each aspect. Here we present examples of what aspectual modules might look like. organised in accordance with seven roles that aspects fulfil in an application -- meaningful types of being, properties, ways of relating, actions, inferences, constraints that it would be meaningful to impose (within the software), and appropriate input and output style. All the lists give merely a few examples to illustrate or stimulate ideas, and they should be severely criticised; much more research is needed to identify proper lists. This proposal could be used in at least two ways: to suggest new lines for KR research, and, when more fully developed, to provide a yardstick by which extant KR toolkits may be evaluated.

Quantitative Aspect (discrete amount)

Things: integers, ratios, fractions, proportions, etc.; also types that anticipate later aspects such as 'real numbers' for the spatial aspect

Actions: incrementing, scaling, statistical functions, etc.

Properties: accuracy, approximation

Inferences: arithmetic

Relatings: greater and less than, sets, etc.

Constraints: e.g. a given quantity remains that quantity until changed

O: Digits, Bar length, Contour lines
I: Hit keys, Drag bar, Drag contour

Spatial Aspect (Continuous extension)

Things: space itself, shapes, lines (straight or curved), areas, regions, dimensional axes, etc.

Actions: join, split, stretch, deform, rotate, overlap, expand, etc.

Properties: size, orientation, distance, side (in, out, left, right), etc.

Inferences: those of geometry and topology

Relatings: spatial alignments and arrangements, touching, crossing, overlapping, surrounding, topology, etc.

Constraints: e.g. boundaries should not have gaps

O: Shapes, spatial arrangements
I: Drag to draw, modify

Kinematic Aspect (smooth movement)

Things: movement, path, flow, centre of rotation, etc.

Actions: start, stop, rotate, follow a path, etc.

Properties: velocity, speed, direction, divergence, curl, duration of movement

Inferences: e.g. s = v * t -- those often found in the field of mechanics

Relatings: faster/slower, forward/back, travel together, etc.

Constraints: the Hare does beat the Tortoise

O: Animation
I: Joystick/keys to give direction, speed

Physical Aspect (Energy, mass, etc.)

Things: waves, particles, forces, fields, causality, impacts; also mechanical things, chemicals, solutions, liquids, gases, crystals, materials, etc.

Actions: physical interaction, expanding a field by inverse square law, dissolving, chemical reacting, etc.

Properties: mass, energy, charge, frequency, force, field strength, Newton-power, etc.

Inferences: various energy functions, etc.

Relatings: causes, attracts/repels, etc.

Constraints: conservation of mass / energy / momentum, laws of thermodynamics, etc.

O: 3D ray-traced perspective view
I: Haptic devices

Organic (Biotic) Aspect (Integrity of organism)

Things: organism, organ, system boundary, tissue, air, food, life, population, environment, dysfunctions; checksums, etc.

Actions: regulate, grow, ingest, excrete, reproduce, repair, die, etc.

Properties: health, stamina, age, etc. (c.f. the 'stats' in role playing games)

Inferences: e.g. parent implies child

Relatings: parent/child/mate, food chains, symbiosis, system-environment, etc.

Constraints: need for sustenance and benign environment, etc.

O: Fractal 3D views
I: 'Soft' haptic device

Psychic Aspect (Sensing, feeling)

Things: signals (sounds, sights, etc.), channels, states (esp. emotional), memories, motor actions, etc.

Actions: respond, remember, forget, feel, push, etc.

Properties: colour (hue, saturation, value), pitch, volume, etc.; angry, happy, etc.

Inferences:

Relatings: e.g. stimulus-response

Constraints: sensitivity ranges of sense organs, etc.

O: Colour, sound
I: Linear sliders (e.g. HSV for colour)

Analytic Aspect (Distinction)

Things: distinct concepts, objects, labels to identify things, etc.

Actions: e.g. distinguish, deduce

Properties: truth values, difference and sameness, etc.

Inferences: those of logic, etc.

Relatings: contradiction, logical entailment, identity, etc.

Constraints: e.g. principle of non-contradiction, entity integrity (as in relational databases)

O: Icons, Menus, Tick boxes
I: Click to select

Formative Aspect (Formative power)

Things: structuring, relationships, modifications, plans, means and ends, goals, intentions, power, etc.

Actions: form, compose, relate, revise, undo, seek, effect a meaningful change (change a state), plan, etc.

Properties: feasibility, efficacy, version, strength (as of a relationship), etc.

Inferences: graph searching, synthesis activity, etc.

Relatings: means and ends, the purpose of something, sequence of operations (history), part-whole, etc.

Constraints: e.g. referential integrity

O: Box-and-arrows graph; Buttons
I: Drag boxes, arrows; Click to activate

Lingual Aspect (symbolic signification)

Things: nouns, verbs, etc.; words, sentences, etc.; bullet lists, headings, cross references, quotations, etc.; word roots, languages

Actions: write, draw, understand, send message, text search, find equivalent meaning, translate, etc.

Properties: tense, case, emphasis, cultural connotation, etc.

Inferences: those of syntax, semantics, etc.

Relatings: synonyms, antonyms, opposites, cross references, rhymes, thesaurus relationships, etc.

Constraints: spelling, grammar, pragmatic context, etc.

O: Text
I: Alpha-numeric characters (keyboard)

Social Aspect (social interaction, institutions, keeping company)

Things: person, group, role, institution, title, name, nickname, etc.

Actions: communicate, befriend, adopt a role, give respect, etc.

Properties: status, leadership, formality and informality, address (postal, phone, email), etc.

Inferences: e.g. how to address someone

Relatings: friendship, acquaintance, respect for, membership, organizational structure, hierarchies, etc.

Constraints:

O: Organisation charts, etc.
I: As analytical+lingual?

Economic Aspect (Frugality, limited resources, managing)

Things: resource, limit (complex), supplier, consumer, exchange, market, human resources, etc.

Actions: distribute resources, allocate price, etc.

Properties: limits, prices (values), etc.

Inferences: e.g. management forecasting

Relatings: supplier-consumer, relationship with resource limits, inter-currency, etc.

Constraints: e.g. no net loss of resources except via defined inputs and outputs

O: e.g. Tables of figures
I: As analytic+quantitative

Aesthetic Aspect (Harmony, enjoyment)

Things: nuances, harmonies, surprises, humour, fun, leisure, sport, etc. plus all the beings found in the various arts

Actions: harmonize e.g. music, play with, etc.

Properties: situatedness, harmony, surprisingness, paradox, interesting/boring, etc.

Inferences:

Relatings: nuance, echoing, counterpoint/complementarity, etc.

Constraints: "Less is more in art" [C.S. Lewis]

O: Decoration + accompanying music
I: As psychic with fine control
Both e.g. Colour circle device

Juridical Aspect ("to each their due")	
Things: dues, responsibilities, rights, coded laws, policies, contracts, security measures, owners, policies, (in)justice, etc.	
Actions: make contract, decide the essence of a case, judge, make retribution or recompense, etc.	**Properties:** security ratings, equity, proportionality, appropriateness, etc.
Inferences: e.g. consider evidence	**Relatings:** retribution, ownership, etc. Many cross-references between clauses
Constraints: laws of land, idea of what is due to each type of thing, ensure consistency, etc.	**O:** Text with cross references **I:** As lingual?

Ethical Aspect (Self-giving love)	
Things: attitudes, gifts, sacrifices, etc.	
Actions: give (without expectation of reward), forgive, etc.	**Properties:** generosity, etc.
Inferences:	**Relatings:** Buber's I-Thou relationship, marriage/troth, etc.
Constraints: self-giving must be genuine, not for gain, etc.	**O:** As lingual+aesthetic? **I:** As lingual+aesthetic?

Pistic Aspect (vision, faith, committing)	
Things: commitments, beliefs, trust, creeds, rituals, etc.	
Actions: make a commitment (after, maybe, weighing up the evidence), trust, worship, etc.	**Properties:** degree of certainty, trustworthiness, etc.
Inferences:	**Relatings:** committed-to, believe-in, trust, etc.
Constraints: commitments should be kept, etc.	**O:** As lingual+aesthetic? **I:** As lingual+aesthetic?

Despite the serious flaws in the lists, they exceed those supported by most extant KR approches. Furthermore, the reader is likely to agree that the things mentioned are not esoteric, but are features encountered in everyday living.

Some of the benefits that might be expected if such a proposal was actualized include the separating out of different characteristics of things (e.g. spatial things are continuous rather than objects), easing of the task of IS developers, fewer errors during development, enhanced reliability of software, enhanced ability to extend and upgrade it as requirements change, especially unforeseen ventures into

new aspects of use.

What is proposed here has not yet been implemented in full, but the author's Istar KBS software [Basden and Brown, 1996] started to be developed along these lines. One reason to believe it might be feasible (in the long term) is that there is real-life software qualified by each aspect, which offers some of the basic facilities mentioned. Table 7.3.2 shows some of the extant software that provide some of the facilities related to each aspect. It demonstrates clearly that at least some meaning in every aspect has been represented in software, and thus that it is at least possible to do this.

Table 7.3.2. Aspectual capabilities of extant software

Aspect	Example software
Quant'tive	Calculator, Statistics package
Spatial	Drawing packages, Geographic Information Systems, Computer-Aided Design
Kinematic	Animation packages, Fluid flow packages
Physical	Weather forecasting systems, Solid modelling systems
Biotic	Medical software, Genealogical software, Life games
Sensitive	Painting and photographic software
Analytical	Mind-mapping software, Deduction software
Formative	Planning software
Lingual	Word processors, KBS, Web browsers
Social	Email
Economic	ERP systems, Critical path analsys software
Aesthetic	Music composition software
Juridical	Will-writing, contract-writing software
Ethical	?
Pistic	?

7.3.3 Implementation at the Bit Level

In terms discussed in chapter 5, these basic facilities are the raw pieces of data, each being of a particular type depending on the aspect. But these are implemented in things meaningful in the psychic aspect, such as bit patterns, memory address adjacency, machine code and digital signals. So there must be a mapping between bit patterns to basic types of data, and between machine code and the valid manipulations for these types of data. But, owing to the irreducibility of the aspects, these mappings are never given a priori, but must be designed by us. In principle, aspectual basic type of data requires its own different mapping.

Table 7.3.3 shows some of the mappings for each aspect. (FPB

refers to 'fixed point binary', a quantity that increases using binary coding from 0, represented by all bits 0000..0000, to 1.0 or 100%, all bits 1111..1111, however many bits are used. Note the important word 'contiguous' in the bit column, which refers to the bit patterns running contiguously in memory, which is a phenomenon meaningful in the psychic aspect.)

Table 7.3.3. Bit pattern codings for selected aspectual building blocks

Aspect	Basic facility	Bit pattern coding
Quantitative	Integer Ratio Proportion, Probability 'Real' number	Binary, BCD Two contiguous bins FPB 00..00-11..11 FPB mantissa + Bin exponent
Spatial	2-D field (e.g. Funt) (x,y) complex number Direction, angle	Bitmap (grid of bits) Two contiguous Man+exp Circular FPB
Kinematic	Movement	Pen colours alterations Sequence of bitmaps by pointers
Physical	3-D grid Collision detection	Array of reals 'AND' bitmaps (Amiga)
Organic-Biotic	Integrity of data	Checksums
Psychic / Sensitive	Colour Picture Sound waveform	3x8 bits each FPB for RGB Grid of colour cells Contiguous array of FPBs
Analytic	Distinct concept (datum) Truth value	Allocated contiguous memory Single bit (0, 1)
Formative	Structure Relationship	Contiguous memory. Memory address (pointer), Linked list
Lingual	Text characters Emphasis (bold etc)	ISO, EBCDIC ANSI Escape sequences, or bits to flag style
Social	URL / email address	Four FPBs
Economic	Data compression	e.g. Zip coding
Aesthetic		
Juridical	Data protection	128-bit encryption
Ethical		
Pistic		

This table shows a number of things. One is that most of the bit codings are for the early aspects. Because of foundational dependency (q.v.), many types of data in the later aspects make use of types in earlier aspects. For example, economic price, double-entry book-keeping or transfer of resources usually make use of quantitative amount, analytic distinction and formative relating.

Nevertheless, there seems to be still some meaningful facility in the later aspects which cannot be seen in terms of facilities of earlier aspects, an example being data compression such as by Zip coding, which yields, not amounts or relations but merely a long bit pattern.

Notice also the empty ethical and pistic cells. This might be due to the down-playing of these aspects in modern life. It may be that if

ICT had been developed, not in the West, but in sub-Saharan Africa, where generosity is a way of life and the spiritual is not divorced from the physical, that these cells would have been full.

7.4 INTEGRATION

The distinctness of aspectual modules is guaranteed by aspectual irreducibility. But how are these to be integrated so that the IS developer can harness the philosophical roles of all aspects relevant to their needs to construct their IT artefacts or systems? This may be done by providing features that reflect the inter-aspect relationships of dependency, analogy and reaching out (§3.1.4). Some of this is found in current practice, though it is seldom recognised as such, and a Dooyeweerdian view might help clarify issues and stimulate new directions for research.

7.4.1 Foundational Inter-aspect Dependency

Inter-aspect dependency in the foundational direction (see §3.1.4) implies that a module for any aspect will require the facilities offered by a module of earlier aspects. Thus if aspect Y is later than aspect X, then a KR toolkit that has a module for Y will also usually need at least part of a module for X, incorporating X-things in its data structures (or OO classes) and calling X-procedures in its procedures. This type of inter-module link has been well-known almost since computers were invented (the subroutine concept) but perhaps in a rather arbitrary manner. The main contribution Dooyeweerd might make here is to provide strategic clarity to the designer of a suite of modules. Such clarity is particularly important to meet the challenge of complexity in OO systems.

Inter-aspect dependency implies a cosmic order among the aspects, but this is not simply a linear sequence. Rather, it forms a directed acyclic graph that, as illustrated in Fig. 7.4.1. Inter-aspect dependency can be direct or indirect. For example, in its foundational direction social interaction depends directly on lingual functioning, analytical distinguishing of who is important in a social situation, and sensitive aspect of emotion towards others. It also depends indirectly on these in that, for example, the lingual aspect itself depends on the sensitive aspect of making sounds with mouth and hearing those sounds, and the analytic aspect of distinguishing which sounds have lingual purpose. Making sounds depends in turn on physical functioning (of air-pump, tautness of vocal cords, resonant cavity of mouth, etc.). But none of these earlier aspects depend foundationally on the social (though knowledge of them does). In principle the graph is fully connected, but in practice some links are latent rather than actual; see Dooyeweerd [1984,II,p.164] for a discussion of foundational dependency.
file pix/D-DepcyDAG

Implementing this is relatively straightforward because the designer of a module will know what facilities from earlier modules will be needed. But anticipatory dependency and analogy are more challenging.

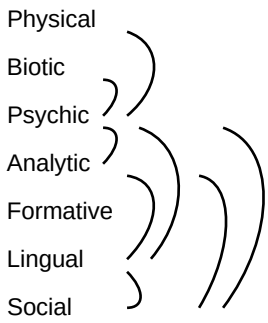

Figure 7.4.1. Directed acyclic graph of some foundational dependencies

7.4.2 Anticipatory Inter-Aspect Dependency

Dependency in the anticipatory direction looks towards what an aspect facilitates rather than what facilitates it. It is concerned with cosmic possibility, much of which has yet to be opened up. Much of this anticipatory meaning, especially in the earlier aspects, has been discovered and opened up by centuries of scientific and other endeavour, but it is likely that much has yet to be discovered and opened up in the later aspects. Anticipation is essential to Dooyeweerd's theory of progress (q.v.), and in IS it is relevant to building future-proof computer system architectures.

To Dooyeweerd, the possibility of facilitation of later meaning is within the aspect from the start. But irreducibility of aspects implies that the way in which an earlier aspect facilitates a later cannot be determined a priori, and its anticipations are opened up by creative human endeavour. This means that when we design a module for an early aspect we might find, a considerable time later, that the way we have designed and implemented it is insufficient to support the new meaning, or at least clumsy in doing so.

Compare the use of fixed-point binary numbers (FPBs) to implement probabilities (range 0--1) and angles or directions (0 -- 360 degrees), where bit pattern 111..11 implements the maximum in each case. Adding two probabilities 0.75 + 0.75 should result in either a probability of 1.0, an error flag, or both. But adding two three-quarters of 360 degrees (viz. 270 + 270) should result in 180 degrees and no error. If the quantitative module provides only one type of FPB addition-behaviour (as is usually the case) then one or other of these implementations will fail.

The designers of an aspectual module can be expected to have expertise in that aspect but perhaps not in others. The designers of the quantitative module cannot always be expected to know aforetime all the subtly different types of quantities and behaviours needed. Nor, similarly, for all other aspectual modules. This is pernicious because usually the modules that implement the earlier aspects are developed first. But unless we consider all the anticipations of later aspects during this process, then we stand the risk of overly constraining our later possibilities because of untoward assumptions made. For example, we implement the building blocks of the lingual

aspect to support lexics, syntax and semantics -- and fail to cater for the complexity of linguistic pragmatics which anticipates the social and later aspects. (The earlier example of implementing shapes in a way that precludes holes is similar, but it is within a single aspect.)

Failure to anticipate can be a particular problem with off-the-shelf modules unless one knows that its designer has an attitude of openness to later cosmic meaning.

Therefore each module should be designed in such a way that it anticipates later modules without needing to be rewritten when they arrive. It might be years before they do, by which time the module has been embedded in many applications. The challenge is how to cater for unforeseen new types of things, functions, properties, constraints, and so on, providing a clean way of enabling a module to function in a different way when unanticipated cosmic meaning presents itself. This is not the same as the challenge to the IS developer of altering a business application system when the business requirements change; that type of change usually reflects a change in the entity side. The challenge we face here is a change in (our knowledge of) the law side. It is more fundamental and requires more careful consideration.

Call-back hooks is one way of catering for unforeseen expansions, but usually they are provided grudgingly and without much careful thought. The OO community's emphasis on polymorphism might constitute a partial recognition of anticipatory dependency, but their penchant for encapsulation works against making it possible. As far as this author is aware, research into this issue of law-side anticipatory dependency is long overdue.

7.4.3 Inter-Aspect Analogy

Inter-aspect analogy, in both directions, is as difficult as anticipatory dependency to predict and cater for.

Take the case of causality, discussed in relation to KR by Nilsson [1998,p.326ff]. Causality is seen, not as physical, but as an intuitive linking of 'causes' and 'effects'. As a result of knowledge elicitation, a 'causal network' of nodes and arcs may be built, throughout which effects may be propagated from 'first causes' in order to simulate the domain or make predictions. For each aspect a slightly different 'causality' algorithm is likely to be needed. Nilsson argues that the Bayesian algorithm for accumulation of evidence is appropriate for such causal links, which makes probabilistic rather than precise calculations of effects from causes. This notion of causality has become established within the KR community so that Bayesian networks furnish the IS developer with a kind of general purpose building block for representing causality. The Bayesian algorithm is very useful because its ability to approximate can overcome many minor discrepancies between different aspectual repercussions, it can cope with non-determinate repercussions, and it is, in effect, an acknowledgement that some antecedent parameters have been omitted from the represented content. But its usefulness can mislead. Basden, Ball and Chadwick [2001] found (without any reference to

Dooyeweerd) when assessing the amount of trust one can place in Internet certificates, which involved echoes of causality in the juridical, pistic and other aspects, that the algorithms required are very different.

There are differences from anticipatory dependency, which might indicate a different approach to catering for inter-aspect analogy. Analogy lacks the necessity found in dependency, and, by its nature, analogy is often more difficult to clearly define. Perhaps extensibility features similar to those required for anticipatory dependency will be adequate, but since their exact form is likely to be different, it would be wise to keep them separate.

What is novel in this proposal is not that we can, for example, use causal networks for pistic software, but that we can guide the development of such general purpose facilities according to aspectual analogies, rather than in an ad-hoc manner.

7.4.4 Implementing Aspectual Reach-out

Aspectual 'reach-out' might be rather easier to cater for. This is not esoteric, and many examples of features that have been added to mature real-life software qualified by different aspects to make life easier for users. But these accretions have been largely ad-hoc, and Dooyeweerd's aspects could perhaps provide some systematization. For example:

- Quantitative reach-out implies a different kind of amount in each reached-to aspect. This implies a need for units associated with each aspect (feet, metres, pounds, kg, currency, etc.), the ability to convert between units at the will of the user, and to add new units when necessary.

- Analytic reach-out implies a different type of distinction in each aspect and different types of inferences. Social logic differs from physical [Winch, 1958]; see §3.1.5. This can implies different types of identification and deduction for each aspect -- as was realised quite early by for example Stamper [1977], who was struggling with how to store legal data. It may be that taking account of Dooyeweerd's suite of aspects can stimulate new directions in research, which currently are at the mercy of the accidental attempts to apply them to new types of application.

- Lingual reach-out concerns the Chomskian deep structure of languages, and can be used to test and generate proper grammar as well as sense. Dooyeweerd's aspects could point to new syntactic and semantic structures and laws to be incorporated into lingual software (such as verse).

7.4.5 Reflection

The vision of an integrated, multi-aspectual KR toolkit and language presented here is a long-term one and will require considerable research effort. Before it can be properly judged, we should attempt

to define comprehensive modules for every aspect and how to integrate them. There is evidence that this may be possible and even desirable, because features are becoming important in practical software even today, as indicated in Fig. 7.3.2, even from later aspects -- such as the juridical feature of copyright notices and authentication checks, and extant research into many of these issues continues apace, being forced upon us by our everyday experience.

So it may be that the main contribution of this proposal is not the vision of a grand multi-aspectual toolkit so much as that it provides a basis on which the diverse areas of research may be integrated as part of a wider picture.

The vision of a multi-aspectual toolkit designed in this way opens up an intriguing possibility: is it possible to insert new imaginary virtual law-spheres among the given ones, or perhaps modify the meaning of existing ones, and then create virtual worlds with these and see how well they run? This could, in principle, forge an interesting test for Dooyeweerd's suite of aspects against others.

However, there are several problems with our proposal that need to be addressed over the longer term. First, it is not always clear what shape a toolkit might take for the later aspects, such as the juridical. It might be that (as currently) it is sufficient to express all the post-lingual aspects in language. But there is reason to suspect that, as application in later aspects matures, we will find we need something that cannot be written in language but must be implemented directly at the bit level (assembly language) as a primitive symbolic signification of aspectual meaning; this is not for efficiency reasons but to do justice to its meaning. Or, alternatively, we must fall back on domain meaning being inscribed into the context of use (the user's own knowledge and lifeworld experience) rather than represented in the technical artefact as such (see 'Creating the IS' in chapter 6).

Second, if a complete set of facilities were implemented for every aspectual module would not the resulting software package be unwieldy? Though this has not been researched yet, there are two reasons for believing this might not be entirely so. One is that the aspects are readily learned, and intuitively grasped; as Winfield [2000] and Lombardi [2001] have found, it becomes second-nature to consider them in any situation. The other is that since the aspects have irreducible meaning, then clear separation should be possible between building blocks of each aspect. "A thing should be as simple as possible, but no simpler" says Budgen [2003,p.75-81], and the reality which IS developers encounter and represent is of an aspectual complexity that should not be unduly simplified, but rather supported and made explicit and understandable. Good modularity involves what they call "separation of concerns", and this is precisely what this proposal offers.

7.5 RELATING TO EXTANT DISCOURSE

The above has outlined a long-term project for a KR toolkit inspired by Dooyeweerd. Here we discuss how it might help address the

important issue of how the lay person can do their own IS development, how a Dooyeweerdian approach might be used to critique extant proposals for data models or KR languages, and its relation to Alexander's Design Patterns.

7.5.1 Dooyeweerdian Critique of Extant Data Models

It is possible to use Dooyeweerdian philosophy to analyse data models and KR approaches. There are two ways. We could set up a Dooyeweerdian proposal, as outlined above, as a yardstick against which to measure the data models; for example, we could find out in which aspects they are strong or weak. Alternatively, we can look at the problems each has experienced in everyday life and explore how Dooyeweerd might explain those problems and perhaps propose a solution. The latter will be used. The approach will be to expose root presuppositions or aspectually-inspired world-views.

Three brief analyses will be presented to illustrate the kind of approach that might be taken, rather than attempting comprehensive critiques.

7.5.1.1 The Relational Data Model

The relational data model (RDM) was defined by Codd [1970] as a reference model to structure and search data in databases. The popular version, available in current software, sees the world as tables containing records, which are composed of sets of attributes that contain values; some attributes may be keys that point to other tuples, thus forming relationships. But Codd's original data model ('pure' RDM) treated data as points in multi-dimensional spaces. Each table ('relation') is a multi-dimensional space, in which each axis of the space represents an attribute in which all its possible values are mapped onto positive integers, allowing infinitely many possible values in any attribute if desired. Each record in the table (tuple of such integers) is a point in the space. All operations are on sets of tuples and generate other sets. The latter allows operations to be chained together to provide very sophisticated overall operations.

But pure RDM gives problems. If two tuples have the same set of attributes (e.g. two men of the same name) they are the same point and so have no separate existence in the relation (cannot be stored as separate records); but in real life this restriction is onerous even if rarely encountered. Second, if we allow keys to form relationships, we introduce the problem of relational disintegrity. Third, many people misapply the relational join because they keep forgetting to make two key-attributes equal to each other. Fourth, the set of operations that immediately offer themselves do not match the operations we want in everyday life (for example, the cartesian product is almost useless while the more useful join is absent and must be assembled from basic operations). Fifth, ordering of records is not supported in pure RDM. (Practical relational models do allow duplicate records, sorting, etc. due to accretion of 'dirty' features on top of 'pure' RDM, as the language or data model was exposed to the everyday life of IS developers.)

The root of these problems becomes clear when we understand it aspectually. The analytic-formative notion of entities and relationships is reduced to the quantitative-spatial notion of points in space. Whereas the analytic aspect allows distinction of two things with same properties, the spatial does not. Treating a relationship, which is a formative thing, as quantitative forces 'the people' to handle artificial attributes, leading to the second and third problems. Ordering is a formative notion and has no meaning in the spatial aspect (though it could perhaps be argued that the quantitative aspect involves ordering). This can explain the accretion of features foreign to the original reference model.

7.5.1.2 Object- and Subject-Orientation

The object-oriented (OO) KR approach sees the world as objects that possess a number of attributes and operations, as dictated by pre-defined classes, which are very easy to define by inheriting properties from other classes. The design ethos emphasises reusability (an economic norm) by means of polymorphism and encapsulation. Much of the work of OO programming consists of defining classes of objects, in terms of their attributes and operations. For example (example from Harrison and Ossher [1993]):

Class: Tree
Attributes: Height, Weight, CellCount, LeafMass
Operations: Grow, Photosynthesize

There are many standard texts on OO, a classic being Booch [1991].

Though it has become the premier KR approach of today, it exhibits problems. While many arise from foibles of specific implementations and variations (such as whether multiple inheritance is allowed), some are more fundamental. Harrison and Ossher discuss one of these: the notion of object as of pre-defined type is not only philosophically suspect but also problematic in practice. In particular, other attributes and operations, not in the class, might be meaningful to different subjects. To a tax assessor, the tree's meaningful attributes include AssessedValue and meaningful operations include EstimateValue and ComputeTax, to a forester, these would include SalePrice, TimeToCut, ComputeProfit, while to an eagle, they would include FoodValue, ComputeFlight (the subjects do not have to be human). Such issues become important especially when developing suites of cooperating application programs.

Might this diversity of attributes be met by simply defining the class with as many attributes as possible and then filtering out those that are not needed by each program? Harrison and Ossher think not, on the philosophical grounds that "subjective perception is more than just a view filtering of some objective reality. The perception adds to and transforms the reality" [p.413].

A Dooyeweerdian analysis of OO reveals first several ways in which it follows the Dooyeweerdian view:

Classes, with their inheritable properties, may be seen as a simple attempt to implement type laws (see §3.2.5).

- The class-subclass hierarchy is commensurable with Dooyeweerd's notion of types, subtypes, etc.
- The notion of 'object' as a thing that acts and possesses properties by virtue of classes is at least commensurable with the Dooyeweerdian notion of subject as something that is active by virtue of responding to law.
- Multiple inheritance (where allowed) recognises the irreducibly distinct spheres of law and meaning.
- That objects can override what is inherited from classes reflects the plasticity of type laws.
- Polymorphism reflects all things, of whatever type, being governed by same spheres of law.

But there are also a number of differences, some of which can explain the problems and perhaps provide a way to overcome them.

- Encapsulation assumes a part-whole relationship, but sometimes the developer wishes to see or manipulate some 'hidden' property, such as efficiency. This could be allowed by restricting encapsulation to genuine part-whole relations and allowing aspects of the object that tend to become hidden, to be always visible in principle.
- The hierarchical nature of the inheritance relationship betrays a universalistic notion of the totality of things. Dooyeweerd criticises this and replaces it, not with an individualistic notion, but with enkaptic interlacements. For example:
  - Hermit crab and shell: objects linked by subject-object enkapsis
  - Data, algorithms, bit level implementation thereof: objects linked by foundational enkapsis
  - City and orchestra, football team, university: objects linked by territorial enkapsis
  - Fauna and flora living in habitat: objects linked by correlative enkapsis

  But OO can find it difficult, or at least inappropriate, to cater for these using only the part-whole and inheritance relations alone.
- The notion of object itself is problematic, because it presupposes the primacy of existence over meaning. Harrison and Ossher's notion of subject-orientation comes closer in that the nature of the object (as defined by its properties) depends on what it means to various subjects, and this is dynamic.

Harrison and Ossher use the example of a tree having many subject-relevant properties. So, coincidentally, does Dooyeweerd:

> "The tree's structure seems at first to be simple, but on deeper theoretical analysis it proves to be highly complex because this structure appears to be possible only in the universal inter-structural coherence. ... Here this natural thing proves to be included in an extremely complex interwovenness with the structures of temporal human society." [1984,III,p.833]

But Harrison and Ossher's proposal might also exhibits weaknesses, in line with the current vogue towards subjectivism. It is in danger of dissolving the object, such as the tree, into amorphous

nothingness and of giving no fundamental basis for differentiating types of object. But, as Dooyeweerd realised, everyday experience does not allow this but things present themselves to us with their own natures even while these can be of variable meaning to different subjects. His solution was type laws, which are founded on the more fundamental laws of aspects and yet are of immense plasticity, might offer the stability their proposal lacks. Even without type laws, if Dooyeweerd's notion of diverse law spheres (aspects) were to be implemented in a way that transcends all classes and objects, allowing them to respond to any such laws without requiring the developer to explicitly specify in advance which are relevant to their objects or classes, then it would be relatively straightforward to allow unforeseen properties to be added to objects in a non-cumbersome manner.

7.5.1.3 The Wand-Weber Ontology

Wand and Weber [1995] sought to define the building blocks for a truly comprehensive KR approach. They outline three models, a state-tracking model, concerned with how a computer system responds to and keeps track with the real world, a decomposition model, concerned with decomposing a system into subsystems, and a representational model, concerned with the grammatical constructs that the KRL should offer, and how they relate to the ontological constructs that should be offered, if they are to be both complete and clear. Their representational model is the most detailed and is philosophically grounded in Bunge's [1977] Ontology 1: The Furniture of the World, because, they believed, "it deals directly with concepts relevant to the information systems and computer science domains" and "Bunge's ontology is better developed and better formalized than any others we have encountered." [Wand and Weber, 1995,p.209] The grammatical constructs map one-to-one to the following ontological constructs:

- things, properties, states (stable and unstable), events (external and internal, well- and poorly-defined), transformations, histories, couplings, systems, classes and kinds,
- laws that pertain to these: state laws, (lawful) conceivable state spaces, (lawful) event spaces, lawful transformations,
- and, related to system: system compositions, system environments, system structure, subsystems, system decompositions, level structures.

This goes further towards appropriateness than other KR approaches but when they show how it may be used as a yardstick for data models by evaluating the entity-relationship data model, it becomes clear there are some deficiencies, around events, transformations and systems and, to some extent, laws. Some of them are as follows.

Event is defined as "A change of state of a thing." This may be satisfactory when considering the values held by properties, but not so appropriate when considering the creation and deletion of a thing, spatial changes or changes to the global context. Basden [1993]

argued that items and relationships cannot be reduced to values, even if they might be able to be implemented as properties. To say, in extremis, that we could treat items, relationships and properties all as the states of bits held in computer memory makes a category error, confusing the bit level and symbol level.

While the wording of the definition can be changed, it betrays a presupposition that Dooyeweerd makes visible: the reduction, encountered in the Relational Data Model, of the analytic-formative aspects to the quantitative-spatial. It may be that Dooyeweerd's notion of irreducibly different types of aspectual functioning (quantitative, spatial, analytic, formative, social, etc.) could inform attempts at a new definition of event.

Likewise, their ontology begins with 'thing', distinct. A moment's reflection on the need to deal with such things spatial extension (such as in Funt's [1980] Geometric Reasoner, and in bit-field processing) and with Umwelten, reveals that W-W's definition is too narrowly defined. It too might benefit from Dooyeweerd's insights that things are meaningful wholes, can be non-distinct in some aspects, and might include Umwelten. W-W's recognition of law is commensurable with Dooyeweerd.

Wand and Weber differentiate internal from external events on the basis of whether an event links to changes in the external world. But to 'the people' a more important distinction is between those events etc. that are meaningful to the domain of application, and those, internal to computer system, which are not meaningful. Many of what W-W call internal events, such as the calculation of secondary results and explanations, might be meaningful, whereas iteration variables, for example, might not be. Likewise a coupling is defined solely in terms of how one thing acts on another. This precludes relationships of meaning rather than action, such as ownership of a field, or which fields surround a piece of woodland. These both betray a presupposition that meaning may be seen as derivative rather than primary. Again, it is Dooyeweerd that gives us grounds for questioning it and for turning it around, to our advantage in the everyday life of the application.

W-W frequently refer to meaning, suggesting they recognise its importance in KR, but they do not incorporate it systematically into their proposal. Dooyeweerd might help them do so.

Class and kind are defined as sets of things that possess, respectively, one and more than one common properties. The only basis for differentiating types of thing is via class and kind. This precludes a subject-oriented approach as discussed above, because it presupposes some pre-identification of the properties of each type of thing for which Harrison and Ossher [1993] criticise OO. As both development and usage proceed, new properties are seen as relevant, and existing properties change, undermining W-W's notion of class or kind. Dooyeweerd's notion of types as enabling things rather than merely describing sets of attributes, as well as his treatment of typology (q.v.), might be useful in rectifying this and enriching W-W's notions.

Their notion of class or kind also means that all differences between things are to be treated as of equal stature. This is problematic because in our lifeworld, for example, birds and foxes are more alike than birds and woodlands or symphonies. Both birds and foxes are animals and are distinct entities, while woodlands are spatial extensions rather than entities and have the property of being an Umwelt, and symphonies are of human performance art. Wand and Weber's 'system environment' cannot be used to represent woodlands because it refers to that which is other than the computer system, whereas we need to represent environment-ness within our system. It is likely that both Clouser's [2005] emphasis that aspects are distinct basic types of property and Dooyeweerd's theory of entities could usefully inform W-W here.

Wand and Weber actually made use of only one part of Bunge's ontology, The Furniture of the World [1977], and only part of that, related to the notions of thing and change, and not those related to substance, form, possibility and spacetime. But Bunge's second ontology, A World of Systems [1979], which explores different systems genera -- physical, chemical, biological, social, technical -- they did not make use of. If they had, the last-mentioned problem might have been reduced.

However, we can criticise Bunge's own work. First, Bunge's presupposition of substance and form as foundational notions may be seen as rooted in Aristotle, which not everyone accepts, especially those in IS of a subjectivist or criticalist persuasion. Second, while Wand and Weber frequently mention 'meaning', Bunge does not, so we must ask how we might justifiably and systematically settle meaning into such a system of thought. Nor does he have a place for norms, echoing the Kantian Is-Ought divorce. Third, Bunge's five system genera may themselves be questioned, because the basis for accepting those five is shaky. Whereas physical, chemical, biological form a linear sequence, social and technical are placed side by side, but no justification is given for this, nor even any explanation. And where is the psychological level, which appears in most similar attempts to define levels (such as Hartmann)? The answer Bunge gives [1979,p.247] is "We might have distinguished a system genus between biosystems and sociosystems, namely psychosystems. We have refrained from doing so from fear [our emphasis] of encouraging the myth of disembodied minds." Is 'fear' a proper philosophical reason? The problem is that Bunge ultimately fails to offer a sound basis for differentiating systems genera. Fourth, Bunge focuses on the worlds of science and pays very little attention to the lifeworld. For these and other reasons, Bunge's ontology might not be the best philosophical foundation upon which to build an approach to knowledge representation.

These are the kinds of problems that Dooyeweerd predicted would always emerge from the immanence standpoint (§2.3.3): the divorcing of meaning from reality, "unmethodical treatment", and absolutization. Instead, Dooyeweerd offers an alternative rendering of these issues, based on a transcendence standpoint:

Dooyeweerd provides a philosophically integrated account of meaning in relation to being, functioning and normativity, allowing meaning and even norms to have a valid place in ontology. This would enable W-W's references to meaning to be incorporated as a systematic part of their proposal.
Aspects provide a sound basis for both differentiating and ordering systems genera. Indeed, Bunge's systems genera is a subset of Dooyeweerd's (see Table 3.1.2).
It is drawn from lifeworld and not just the worlds of science.
It does not arbitrarily separate furniture of world from world of systems; both function by same aspects, but are different functionings r.t. different aspects; result: no arbitrary division of, for example, things and relationships from justice and language.

As a result, Dooyeweerd might yield a better ontology than Bunge [1977], [1979], which, as discussed above, might clear up some of W-W's anomalies. Dooyeweerd's theory of time, which is not discussed in this work, might also be useful to W-W.

7.5.1.4 Reflection

We have used Dooyeweerd in three different ways. The critique of Codd's Relational Data Model showed how Dooyeweerd can expose aspectual reduction, of analytic-formative to quantitative-spatial, and thereby explain the problems that have arisen in practice. The critique of OO showed how Dooyeweerd can lay bare the presuppositions underlying the central notion of 'object'. While acknowledging strengths in the notion, it also exposed the nature of a fundamental weakness already detected by Harrison and Ossher [1993]. Unlike Harrison and Ossher, however, Dooyeweerd frees us from the subjectivist, anti-objectivist reaction, to acknowledge and integrate the insights of both sides. The critique of Wand and Weber's ontology showed how Dooyeweerd can be used to address specific issues and trace the root of specific problems to a variety of philosophical presuppositions. It also suggested how Dooyeweerd could replace Bunge as the philosophical foundation for Wand and Weber.

What comes through clearly from all critiques is the influence of the immanence standpoint in our data models and KR approaches. It has forced their designers towards reduction, to presupposing the self-dependence of things and/or events and away from taking meaning -- the all-important application meaning to users -- seriously as an systematically integral part of their proposals. Meaning, where it is acknowledged at all, is assumed to be reducible to epistemology. All such proposals must, of necessity, concern themselves with ontology rather only with than epistemology, and yet the immanence standpoint has driven Western thought into opposing the two and pushing ontology out of fashion in most IS communities today. But Dooyeweerd, taking the transcendence standpoint, is able bridge such gulfs, taking ontology seriously and allowing us to systematically include meaning (and also normativity) as part of that ontology. This is what our earlier proposal was able to achieve.

These three critiques have been brief and only indicative, but they have shown that further work in the directions indicated is likely to be fruitful. For our final application of Dooyeweerd we turn to what is an overall approach and ethos rather than a precisely defined data model, that based on Alexander's Design Patterns.

7.5.2 Dooyeweerd and Alexander

Information systems developers face challenges not unlike those found in architecture -- the design and putting together of a complex whole with which human beings will engage as they live, with generic ideas and components applied to the specific situations of the people for whom the product is intended. So it is no surprise to find that approaches devised in architecture are being brought into service here. Both require an interdisciplinary approach, which in turn demands a diversity of types of components that are 'reusable' and yet flexible. Design patterns is a notion borrowed from the architect, Christopher Alexander [Alexander et. al., 1977], and adapted to software by, for example, Gamma, Helm, Johnson and Vlissides [1995].

7.5.2.1 Alexander's Vision

The vision of Alexander and his team, expressed in their two books The Timeless Way of Building [1979] and A Pattern Language [1977], is to make "towns and buildings ... come alive" which will not happen "unless they are made by all the people in society, and unless these people share a common pattern language, within which to make these buildings, and unless the common pattern language is alive itself." [1977,p.x]. Design Patterns recognises the importance of designing for the everyday lifeworld of those who will use what we design.

They present a 'language' composed of 'patterns', each of which "describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." To construct the pattern language they

> "have also tried to penetrate, as deep as we are able, into the nature of things in the environment: and hope that a great part of this language, which we print here, will be a core of any sensible human pattern language, which any person constructs for himself, in his own mind. In this sense, at least a part of this language we have presented here, is the archetypal core of all possible pattern languages, which can make people feel alive and human." [ibid.,p.xvii]

His hopes for his language are expressed in interesting phraseology: "that when a person uses it, he will be so impressed by its power, and so joyful in its use, that he will understand again, what it means to have a living language of this kind." [ibid.,p.xvii].

253 patterns are defined, patterns 1-94 concern towns, 95-204 concern buildings and 205-253 concern construction -- the wider context of buildings, buildings in use, and buildings as products to develop.

7.5.2.2 Design Patterns in information systems design

In ISD, likewise, problems occur over and over again and the core solution to one problem is often applicable to others, though with some appropriate modification. So, partly in response to a rigidity that characterizes the extant KR languages, and partly because the diversity of problems is not catered for in them, Design Patterns has been used as a model to generate better KR 'languages' based on a patterns approach that could respond to the needs of software reuse and maintenance in a changing usage environment.

Gamma et. al. [1995] have detailed 23 patterns for use in object-oriented design and development, five creational patterns, seven structural patterns and eleven behavioural patterns. For example the Observer behavioural pattern defines, for a given object, which objects need to updated or informed when it changes. It is surprising how few design patterns they believe are necessary (compared with Alexander's 253). This might be because their only case study is of designing a document editor. For example, while they have a Composite pattern to generate part-whole hierarchies, they do not recognise other types of relationship. Curiously they do not treat lists as patterns but rather as 'foundation classes' that patterns may call upon. But they do invite comment and extension.

Several writers have made a critique of the patterns approach, such as Budgen [2003]. But, unfortunately -- a personal observation -- while the structural and methodological elements of design patterns that have been adopted and the software engineering community does seem to have taken seriously Alexander's desire that pattern languages should be created according to the needs of the discipline, the original vision of joy and life seems missing.

The roots of this approach are in practical and 'intuitive' notions -- which makes it important in our study as a lifeworld-oriented perspective. But it seems to lack any proper philosophical foundation or underpinning. As a result, the discipline is at the mercy of unprincipled application or development, in response to fashion or undue focus on specific classes of problem; one example is the Flyweight pattern, made necessary by a problem arising from an inappropriate implementation of text.

7.5.2.3 Dooyeweerdian analysis of Design Patterns

We can see, or perhaps feel, a degree of affinity between Alexander's Design Patterns and Dooyeweerd, in their motivation, orientation and outworking. Both thinkers "tried to penetrate ... into the nature of things" [Alexander et. al., 1977,p.xvii]. Both were courageous enough to attempt complete coverage, as Alexander's 'all possible pattern languages' and Dooyeweerd's suite of aspects. Both recognise richness. Alexander's orientation to the idea of a core of reality, a "nature of things in the environment" with a recognition that every person holds a different view, is not unlike Dooyeweerd's acknowledgement of a reality that transcends us alongside the freedom of human knowing and believing. We can see a deep similarity in their beliefs about language as something individual yet

social, and alive yet responsible.

Especially in words like ’alive’ and ’joyful’ we can immediately feel the affinity between Alexander and Dooyeweerd in their outworking. ’Alive’ is close to Dooyeweerd’s notion that all things are subjects rather than merely passive objects, and yet meaningful in every aspect (§2.4.4). ’Joyful’ is similar to the Dooyeweerdian notion of shalom as full, healthy, positive functioning in diverse aspects (§3.4.3). Each pattern is related to a ’problem’ that makes it meaningful, but also to its ’context’ within wider patterns and to ’smaller’ patterns, which are not seen as parts but as important in their own right. This echoes the Dooyeweerdian notions primacy of meaning, entitary interlacement, and enkaptic whole-whole relations.

Asking, of the 253 patterns, "What is this pattern trying to achieve? What (aspectual) normativity makes this a problem to be addressed?", to see which aspect is most meaningful in each, gives the results shown in Table 7.5.2.3.1. For this analysis, a differentiation was made in the social aspect between groups of people and cultural matters, and in the aesthetic aspect between harmony (integration, balance and coherence) on the one hand, and style (beauty, fun and rest) on the other. There is a row for patterns that could not be clearly placed, and one for multi-aspectual patterns.

Table 7.5.2.3.1. Aspectual spread of Alexander’s patterns

Aspect	**Towns**	**Buildings**	**Construction**
Quantitative		96	231
Spatial	21 37	167 195	
Kinematic	20 23 34 49 56	120 131	
Physical			211 212 213 215 218 219 225 227 234 236
Biotic	4 47 65 70 72	118 169 170 175 177	
Sensitive	54 55 60 62 71 74 82 92 93 94	97 98 105 109 114 115 119 121 125 132 137 138 142 161 163 164 173 176 180 182 190 192 196 197 199 201 202 117 203	223 230 235 237 250 233
Analytic	14 15 53 57	102 110 111	229
Formative	16 50 73 78 83	104 108 146 171	208 240
Lingual	12 18 43		
Social	2 13 27 30 31 33 36 41 44 45 48 61 67 68 75 76 77 86 89 91	95 100 101 122 123 124 127 129 139 141 143 147 148 149 151 152 179 185 186 188 193	242
Soc: Culture	6 8 40	’	253
Economic	11 19 22 32 39	103 106 145 150 153 162 178 198 200 204	206 228
Aesth: Style	28 38 46 58 59 63 69 87 88 90	112 113 128 134 135 174 191	216 217 221 224 226 232 238 239 244 245 247 249 251 252
Aesth: Hmny	3 5 9 17 29 35 42 51 52	99 107 116 126 130 133 140 156 157 158 160 166 168 181 194	214 222 241 246 248
Juridical	1 7 10 79	172 183 184	205 207 209 210 243
Ethical		136 187	
Pistic	24 66 80 81 84	154 155	
Multiple	26		220
Unplaced	25 64 85	144 159 165 189	

Such an aspectual analysis of design patterns may reveal a number of things, if we take Dooyeweerd's suite to be reasonably comprehensive and well-founded:

Aspectual spread can indicate to what extent the creators of the set of patterns are sensitive to the diversity found in the lifeworld, as opposed to focusing on certain aspects currently deemed fashionable by researchers or developers. That every aspect is represented in Alexander's set suggests he and Dooyeweerd have a similarly wide recognition of meaning.

Every thinker is part of a community but also tries to send a message. Aspectual analysis can reveal which aspects are most important to the community and to the thinker. In this case, the sensitive and aesthetic (style) aspects are to be expected in most architects, while the social and aesthetic (harmony) aspects reflect Alexander's desire that buildings should enhance community and harmonise.

A different aspectual profile would be expected in the three columns, in that Construction is a physical and technical activity, in Buildings the human user is central, and Towns concerns society and its (pistic) vision. In Alexander we find largely what we expect; if we did not, we could question their treatment of these issues.

Considering under-represented aspects can also provide insight. In Alexander, for example, why is there so little of the lingual aspect? In the three columns, respectively, it could cover public signage and communication, deliberate signification in and around the buildings, and communication during the construction process. It is only an aspectual analysis of this kind that can expose the omission of whole spheres of meaning.

The low number of unplaced and multi-aspectual patterns can indicate that the patterns are clearly thought out as to their meaning. A high number would indicate either a confused understanding, or that Dooyeweerd's suite needs modification.

If we perform a similar aspectual analysis on the patterns oriented to software in Gamma et. al. [1995], we find a very different picture; Fig. 7.5.2.3.2, where patterns are identified by page number, and are grouped into patterns that guide creation, structure and behaviour of objects in the program.

With only 23 patterns our analysis must be more cautious. Since nearly half the patterns are primarily formative, we can clearly detect a heavy emphasis on technical matters. That the juridical aspect makes a reasonable appearance suggests that Gamma et. al. recognise the need to do justice to the diversity of the world to be represented, and this is supported by examining their text.

Table 7.5.2.3.2. Gamma's Design Patterns for Software

Aspect	Creation	Structure	Behaviour
Quantitative	127		
Spatial			
Kinematic			
Physical			
Biotic			
Sensitive			
Analytic			229
Formative	107 117	151 163 185	233 273 283 305 325 331
Lingual	97		243,
Social			
Economic		'195 207	
Aesth: Style	87		
Aesth: Hmny		139	293
Juridical		175	223 315
Ethical			
Pistic			

Eight aspects are missing. But with so few patterns, we should look at groups of, rather than single, under-represented aspects. Where, for example, are the aspects which are important to the user interface: the sensitive, spatial and kinematic? Where are the human and social aspects? Examining their text we find that most of the post-social aspects concern, not human use nor the wider social context of use, but the 'objects' an classes of which the software is composed. Gamma et. al. almost wholly focus on what Alexander called construction and display little awareness of software's use and wider context. This analysis might suggest that Gamma et. al's set of patterns is in danger of directing the developer's attention away from the all-important human and social issues of use of the software in context to mere technical issues, and of failing to support the sensitive, spatial and kinematic aspects of UI.

The analysis of Alexander's patterns generated fairly supportable indications of what they deemed important, but the way aspectual analysis is used here is different. Here aspectual analysis suggests

issues that deserve further examination in the text, and it is this that exposes gaps or over-emphases in their approach to bring Design Patterns to IS.

It can also be helpful to subject single patterns, especially problematic ones, to Dooyeweerdian scrutiny. For example the Flyweight pattern [Gamma et. al., 1995,p.195] introduces clumsy complexity. It was motivated by the inefficiency of requiring each letter of text to hold the full set of attributes that text as such does (font, etc.), working to strip letters of such attributes. But it is an unsatisfactory solution because it misdiagnosed the problem, assuming that text may be seen as aggregations of letters into rows.

Dooyeweerd's treatment of wholes (see chapter 3) urges us to ask what are the meaningful wholes of this lingual thing that is text. Our answer would presumably include such things as words, sentences, headings, and would not include rows, the main meaning of which is spatial, nor include letters, since they cannot stand alone as lingual wholes. As wholes, letters are analytic rather than lingual, and so cannot be see as part of words etc. In view of this, letters should never have been treated in the way text is, with all the lingually-relevant features. The root of the problem is a fundamental misunderstanding of the nature of part-whole relations.

Sadly, such misunderstandings are very common in the OO community and indeed the whole systems community, because they have no basis for recognising true wholes nor enkaptic relations.

Thus a Dooyeweerdian analysis of meaning can serve as a useful critique of individual patterns as well as of the whole approach. The root of the problem lies not so much in the individual pattern as in the OO paradigm, which is based on an existence-oriented presupposition, the problems of which have been discussed in chapter 3 and does not provide the concepts like enkapsis with which to distinguish different types of relationship.

But Dooyeweerd can take us further than critique. If, as suggested in this chapter's main proposal, it is possible in principle to provide modules that implement the meaning of every aspect, then it should be possible to provide patterns related to every aspect and realize something of Alexander's vision in IS.

7.6 CONCLUSION

7.6.1 Overview of Framework for Understanding

This area is concerned with the design and preparation of technological resources like knowledge representation languages and basic software facilities intended to be used by IS developers -- equivalent to the design and preparation of things like bricks, planks of timber, nails, hammers, etc. to be used by builders of houses. As with the nature of computers (chapter 5), one might wonder to what extent the preparing of basic technological resources and building blocks can be viewed as everyday experience. It seems too technical a field, but a moment's reflection on the similar construction industry will affirm that there is an everyday lifeworld.

After a review of what has motivated the design of KR languages, it became clear that Brachman's notion of 'KR to the people' was a key to the everyday lifeworld of this area. This is the normative proposal that 'ordinary' people should be able to make use of these general technological resources in order to construct computer systems for themselves, without needing to call upon an expert programmer.

This led to the starting point that an important characteristic required of technological resources to achieve this is appropriateness

-- that the basic resources with which an IS might be constructed should be 'natural'. But, to understand what this means requires taking a step back to first understand the nature of these resources themselves. This is facilitated by Dooyeweerd's ontic analysis of similar resources in the construction industry:

Basic technological resources are semi-manufactured products, the leading aspect of which is not internal but external; their very nature is to reach out to the diverse spheres of meaning of the domain of appliction.

Unlike planks, bricks and nails, which are centred on the physical aspect, the operation of these resources is centred on the lingual, formative and analytic aspects which qualify knowledge elicitation and representation in ISD (chapter 6). These resources are such things as KR languages, with their tokens, inter-file protocols, subroutines, and code libraries.

Such resources have to be designed and implemented. But so do the technical artefacts of the IS developer; what is the difference? The key difference, it was suggested, is that what the IS developer designs for is concrete subject-side situations and requirements, while these basic resources embody the law side. Their generality is thus not just a matter of degree but different in kind. So the principle that should guide their design is:

For each and every aspect, a set of basic facilities and KR language tokens should be available that express its cosmic meaning, as it may be actualized in the various philosophical roles that aspects have (things, properties, relations, activity, inferences, constraints, etc.).

This is the root of appropriateness. It is based on the belief that aspects, as spheres of meaning and law, enable these very things (see §3.1.5). Thus this area makes use of Dooyeweerd's approach to ontology, as transcending us but based on meaning rather than being.

A practical proposal was made that a module that contains these things be prepared for each aspect. Evidence that this is feasible comes from the fact that software has been developed which is qualified by each aspect. The implementation of such modules at the bit level was briefly discussed. But the modules must integrate and work with each other, and be open to future requirements, especially when we begin with modules only of the earlier aspects. So:

The relations between modules reflects that between aspects: the modules are irreducible to each other, but they are to be linked by foundational dependency, anticipatory dependency, analogy and reaching-out, all of which should be taken into account in the design of modules.

Anticipatory relations are important for future-proofing the modules.

The benefits accorded by this framework are that it provides fresh impetus to consider the diversity of basic, intuitive meaning that IS

developers might encounter and not delegate the responsibility for implementing some of this to developers who might not be expert in it. Only thus will it be possible to provide the range of resources that IS developers actually need to cope with real-life complexity. This in turn will help bring 'KR to the people'.

This proposal might be seen as a grand vision for an all-encompassing KR toolkit. But, realistically, it may be more useful as a framework within which to situate extant work. For example, it could be seen as a counterfactual ideal against which to evaluate other proposals. Two examples were given by which current proposals were critiqued, that of Wand and Weber [1995], which represents one of the best-developed general KR ontologies to date, and the importation of the idea of Design Patterns from the field of architecture, which is arousing much excitement in the Object-Oriented community.

7.6.2 The Mission of Bringing Information Technologies into Being

We have been discussing how to bring small pieces of information technology into being, as building blocks and other resources for the IS developer to use. But should we bring information technologies into being? What should guide and motivate this process? A presumed inner dynamic of IT itself (technological determinism)? A social construction of technology? Market forces of the IT industry? The whim or even the 'existential joy' it brings to those who create it? And what about obsolescence?

Dooyeweerd held that humankind is called normatively to open up of the potential of all aspects as spheres of meaning and law, and this for the overall blessing of the cosmos. This idea is examined in chapter 8, where the development of technology may be seen as an 'meaning disclosure', and that of ICT as a whole as the opening up of the lingual aspect of information and representation, and perhaps also the formative aspect (technology). So the general answer to the first question is "Yes", and it should be guided by the norms of all other aspects, especially post-lingual.

But more specifically, the coming into being of any particular piece of ICT may be seen as an opening up of the aspect that is represented. The norms that should guide this progress is not that of the lingual or formative aspect themselves, but that of the represented aspect, followed closely by all other aspects, governed by the shalom principle (q.v.).

This leads us to think of obsolescence in a different way. A certain piece of technology might happen to no longer suit our current historical circumstances, but the meaning it discloses cannot be undisclosed. Therefore, in general, while we might cease to use it for a while, humanity has a mandate to protect its having-been-disclosed.

However, this disclosure of pieces of meaning is not the end of the story, because we are led outwards from the specific ones we develop to those our community develops to those society develops,

and eventually to ICT as a whole, as a global, historical, societal, religious phenomenon. This becomes the very environment in which we live and have our being, which affects our habits, aspirations and expectations, and which is itself inscribed by by these very things, our world-view. This is the topic explored in the next chapter.

References

Alexander, C. (1979). A timeless way of building. Oxford, England: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fikidahl-King, I., & Angel, S. (1977). A pattern language: Towns, buildings, construction. New York: Oxford University Press.

Basden, A. (1993). Appropriateness. In M. A. Bramer, & A. L. Macintosh (Eds.), Research and development in Expert Systems X (pp. 315-328). Cranfield, England: BHR Group.

Basden, A., Ball, E., & Chadwick, D.W. (2001). Knowledge issues raised in modelling trust in a public key infrastructure. Expert Systems, 18(5), 233-249.

Basden, A., & Brown, A. J. (1996, November). Istar - a tool for creative design of knowledge bases. Expert Systems, 13(4), 259-276.

Basden, A., & Clark, E. M. (1979). Errors in a computerized medical records system (CLINICS). Medical Informatics, 4(4), 203-208.

Basden, A., & Hines, J. G. (1986). Implications of the relation between information and knowledge in use of computers to handle corrosion knowledge. British Corrosion Journal, 21(3), 157-162.

Basden, A., & Nichols, K. G. (1973). New topological method for laying out printed circuits. Proceedings of the IEE, 120(3), 325-328.

Booch, G. (1991). Object-oriented design with applications. Redwood City, CA: Benjamin-Cummings.

Brachman, R. J. (1990). The future of knowledge representation. In American Association for Artificial Intelligence, AAAI-90: Proceedings of the Eighth National Conference on Artificial Intelligence (pp. 1082-1092). Boston: AAAI.

Brandon, P. S., Basden, A., Hamilton, I., & Stockley, J. (1988). Expert Systems: Strategic planning of construction projects. London: The Royal Institution of Chartered Surveyors.

Budgen, D. (2003). Software design (2nd ed.). Harlow, U.K.: Pearson Education/Addison-Wesley.

Bunge, M. (1977). Treatise on basic philosophy, Vol. 3: Ontology 1: The furniture of the world. Boston: Reidal.

Bunge, M. (1979). Treatise on basic philosophy, Vol. 4: Ontology 2: A world of systems. Boston: Reidal.

Chomsky, N. (1965). Aspects of the theory of syntax. Cambridge, MA: MIT Press.

Clouser, R. (2005). The myth of religious neutrality: An essay on the hidden role of religious belief in theories (2nd ed.). Notre Dame, IN: University of Notre Dame Press.

Codd, E. A. (1970). A relational model for large shared databanks. Communications of the ACM, 13(6), 377-387.

Dooyeweerd, H. (1984). A new critique of theoretical thought (Vols. 1-4). Jordan Station, Ontario, Canada: Paideia Press. (Original work published 1953-1958)

Funt, B. V. (1980). Problem-solving with diagrammatic representations. Artificial Intelligence, 13(3), 201-230.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: Elements of reusable object-oriented software. Reading, MA: Addison-Wesley.

Gennart, J. H., Tu, S. W., Rothenfluh, T. E., & Musen, M. A. (1994). Mapping domains to methods in support of reuse. International Journal of Human-Computer Studies, 41, 399-424.

Gibson, J. J. (1977). The theory of affordances. In R. Shaw, & J. Bransford (Eds.), Perceiving, acting and knowing (pp. 67-82). Hillsdale, NJ: Erlbaum.

Greeno, J. (1994). Gibson's affordances. Psychological Review, 101, 336-342.

Harrison, W. H., & Ossher, H. (1993). Subject-oriented programming: A critique of pure objects. In Andreas Paepcke (Ed.), Proceedings from Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Eighth Annual Conference, 26 September - 1 October, Washington, DC (pp 411-428). New York: ACM.

Hines, J. G., & Basden, A. (1986). Experience with the use of computers to handle corrosion knowledge. British Corrosion Journal, 21(3), 151-156.

Jones, M. J., & Crates, D. T. (1985). Expert Systems and videotext: And application in the marketing of agrochemicals. In M. A. Bramer (Ed.), Research and development in Expert Systems: Proceedings of the fourth Technical Conference of the British Computer Society Specialist Group for Expert Systems. Cambridge, England: Cambridge University Press.

Levesque, H. J., & Brachman, R. J. (1985). A fundamental tradeoff in knowledge representation and reasoning (Revised Version). In R. J. Brachman, & H. J. Levesque (Eds.), Readings in knowledge representation. Los Altos, CA: Morgan Kaufmann.

Lombardi, P. L. (2001). Responsibilities towards the coming generations: Forming a new creed. Urban Design Studies, 7, 89-102.
Minsky, M. (1981). A framework for representing knowledge. In J. Haugeland (Ed.), Mind design. Montgomery, VT: Bradford Books/MIT Press.
Nilsson, N. J. (1998). Artificial Intelligence: A new synthesis. San Francisco: Morgan Kaufmann.
Quillian, M. R. (1967). Word concepts: a theory and simulation of some basic semantic capabilities. Behavioral Science, 12, 410-430.
Stamper, R. (1977). Physical objects, human discourse and formal systems. In G. M. Nijssen (Ed.), Architectures and models in database management systems (pp. 293-311). Amsterdam: North Holland.
Stephens, L. W., & Chen, Y. F. (1996). Principles for organizing semantic relations in large knowledge bases. IEEE Transactions on Knowledge and Data Engineering, 8, 492-496.
Wand, Y., & Weber, R. (1995). On the deep structure of information systems. Information Systems Journal, 5, 203-224.
Winfield, M. (2000). Multi-aspectual knowledge elicitation. Unpublished doctoral Thesis, University of Salford, England.
Winch, P. (1958). The idea of a social science and its relation to philosophy. London: Routledge and Kegan Paul.