Facet Models for Problem Analysis

Andreas. L. Opdahl[1] and Guttorm Sindre[2]

[1] Dept. of Information Science, University of Bergen, N-5020 Bergen, Norway
[2] Fac. of El. Eng. and Comp. Sci., Norwegian Institute of Technology, N-7034 Trondheim, Norway

Abstract. The paper points to weaknesses of modelling approaches which are *orientated* towards certain aspects of the problem analysis domain (e.g., process orientation, object orientation.) It is concluded that modelling approaches are needed that allow the modeller to 1) choose to represent a wide range of aspects of real-world phenomena depending on the problem at hand, and 2) simultaneously represent several aspects of the same real-world phenomenon whenever needed. A framework for *facet modelling* of real-world problem domains is therefore outlined. It is discussed how facet models can be defined and visualised to deal with the complexity of contemporary problem domains, and with the complexity inherent from the ambition of the facet-modelling framework itself.

1 Introduction

In problem analysis, models of the problem are established, assessed, improved and used as a starting point for IS design. Different paradigms have emerged for this purpose, corresponding to various classes of modelling approaches. Each of the approaches are *orientated* towards certain aspects of the phenomena in the real-world problem domain. Some of the most prominent types are: 1) function or process orientation (i.e., structured analysis, SA), emphasising the activities performed in the real-world system, e.g. [6, 8]; 2) information orientation (i.e., entity-relationship (ER) modelling), emphasising the information resources of the system and their relationships, e.g. [4, 24]; 3) object orientation, emphasising the objects manipulated by the system, e.g. [22, 5], and 4) subject orientation (or agent orientation), emphasising the roles, units etc. performing the activities in the system, e.g. [7].

Each of these orientations has its pros and cons. Criticism of structured analysis can be found in, e.g., [18, 3, 16]; of purely information-oriented approaches in, e.g., [10], and of object-orientation in, e.g., [15, 12, 13, 2]. The more recent, subject-oriented approaches have been less criticised in literature so far.

This paper is written from the viewpoint that clearly orientated models are not to be striven for in the early phases of problem analysis, but rather a source of problems on their own. The rest of the paper will explore this hypothesis. First, Section 2 will discuss what orientation really means, before Section 3 presents the facet-modelling framework which attempts to limit orientation problems to the extent possible. Afterwards, Section 4 will go on to outline important facet types, and Section 5 will discuss how to visualise the resulting models. Finally, section 6 will conclude the paper and suggest some paths for further work.

2 Orientation

2.1 What is It?

Four examples of possible orientations for modelling a problem domain were given above. One might think that the difference between them is that they capture different aspects of the real world, i.e., that process-orientated models capture information about processes, information-oriented ones about data structures, object-orientated ones about encapsulated data types, and so on. However, this is only half the truth. The point is rather that they capture real-world aspects *differently*. The final IS, if it is built, will have to deal with both processes and data in a manner satisfactory to human agents. Hence, all the aspects mentioned in Section 1 must be captured one way or the other at some stage of development. The difference between the modelling approaches therefore becomes one of priority:

- Some aspects are captured explicitly, others only implicitly
- Some aspects are captured earlier than others.
- Some aspects are shown in diagrams, others only textually.

Also, orientated approaches tend to promote a single aspect (e.g., functional transformations in dataflow diagrams) as fundamental to modelling. The corresponding modelling construct is therefore equipped with the most powerful abstraction mechanisms, and the other kinds of constructs are grouped around it. For instance, SA explicitly captures the processes performed, their composition to higher level processes, their sequence, and how the output of one process is passed as input to the next. This is not given as high a priority in object-oriented analysis (OOA) approaches, in which it is not easy to grasp overall dynamics without elaborate model assessment. On the other hand, OOA does easily capture that a number of processes are working on objects of the same class. This is not shown explicitly in SA and can only be established through tedious inspection of the model.

2.2 Why Avoid It?

Obviously, the priorities set by choosing one particular orientation will mean that the aspects not promoted by that orientation will be more difficult to account for during analysis. For instance,

- In SA, it is not easy to see which processes are dealing with the same objects. This makes it difficult, e.g., to ensure change-locality.
- In OOA, it is not easy to comprehend sequences of low-level processes, and since low-level processes are not aggregated to higher level processes, it is also difficult to see exactly how end-to-end services are delivered to the system's users [1]. This makes dynamic analyses more difficult.

- In both SA and OOA, the human actors in the information system and their interrelationships are usually not described in detail (e.g., persons and units, their competence and the roles they can play, the responsibility of each role etc.). This makes it difficult for future users to relate to the model and to evaluate organisational changes caused by the system.
- Most traditional modelling methods are weak on requirements traceability [9]. Since for wicked problems the solution itself is likely to later become part of new problems [19], traceability (i.e., of the requirements that led to the current state of the problem domain) should be an integrated part of the problem analysis model, just as representations of processes performed in sequence or of processes and the objects they work on.

The above points explain why it might be a good idea to reduce the importance of orientation in analysis:

Avoiding representational bias: Orientation means that some aspects of phenomena in the problem domain will be difficult to capture and/or easy to forget because the modelling constructs which represent them are less important in (or even missing from) the modelling approach used.

Avoiding perspective bias: Orientation means that the problem domain will be looked at from one particular perspective the whole time, thus hiding weaknesses that would be more apparent from other perspectives.

Avoiding communication bias: Orientation means that it will be difficult to communicate a model to people to whom the particular orientation is unnatural, although easy to others.

Avoiding interest bias: Orientation means that the problem-domain models may inherently support the participation and interests of some of the individuals and groups affected by development, but not those of others.

Another problem with orientation is that it will force you to see any phenomenon — or any aspect of a phenomenon — as represented by one particular construct of the modelling approach. However, in reality several aspects of the same phenomenon may be simultaneously important for the problem at hand. It may be important for a single modeller to be able to represent several of these aspects at the same time, and it may be important to be able to co-represent aspects emphasised by several of the individuals or groups involved or affected. Capturing several different aspects at the same time is difficult in a strictly orientated approach, where it is often necessary to select a single modelling construct to cover a phenomenon, or at least to use several independent modelling constructs for each of its aspects. A particularly interesting remark in this direction is made in [12]: With the increasing distribution and interoperability among applications, the same or overlapping information is likely to be used for many different purposes, i.e., by different applications with different perspectives. This is used as an argument against pure objects, which focus on the information itself as opposed to the different perspectives on the information.

Example 1. Consider the development of an information system for managing the production and administration equipment of a beverage production company,

which produces both beers and soft drinks. The new equipment administration system (EAS) is to keep an inventory of each of the production tools (robots, computers, etc.) owned by the company, together with information about, e.g., its current state, application, and placement.

An assembly line in the beverage company can be perceived from several perspectives, e.g., as an entity (information about the assembly line), as a relationship between two entities (the start and stop area), as a transportation or functional transformation of workpieces, as a process in which workpieces are crafted, as a store (since it introduces production delays), and as an agent transporting or processing items. Furthermore, the appropriate perspective will depend on the subjects perceiving the assembly line, e.g., inventory managers (who see it as an entity), the workers labouring along it (who see it as a process or a flow), and process managers (who see it as a store or delay). □

3 The Facet-Modelling Framework

Having discussed the weaknesses of orientation, the high-level goals for a modelling approach which attempts to avoid orientation can be identified:

- It should enable the modeller to capture any aspect of a problem-domain phenomenon, and any kind of relation between phenomena and between aspects. The choice of what to represent should be left to the modeller on basis of the problem at hand.
- If several aspects of the same phenomenon are relevant for the problem at hand, it should be possible to capture them simultaneously, instead of having to use several separate modelling constructs.

These are the two main properties of what is now to be introduced as the *facet-modelling framework*. The framework has the power to embed many of the oriented approaches. Hence it implicitly covers problem domains where these approaches are already sufficient. The facet-modelling framework is therefore not "yet another modelling approach," but an offer to generalise, integrate and extend contemporary approaches to make orientation problems less dominating in the early phases of IS development. Also, it is *not* in any fundamental sense "orientation free": Modelling is always about selecting certain aspects of the real world at the expense of ignoring others. What the facet-modelling approach contributes is a simple and intuitive, yet powerful and formalisable way of extending the set of aspects available, thus alleviating many orientation problems.

3.1 Items and Facet Types

A *facet model* represents concrete and abstract phenomena in the analysis domain as *items*. The term "item" has been chosen deliberately to avoid connotations to other analysis approaches such as ER-diagrams ("entities") or OOA ("objects"). Items do, however, share with objects the possession of an *identity*. The modeller's perception of a real-world phenomenon is represented in the facet

The real-world	The facet model
Phenomena	Items
Aspects	Facets
Relations	Referencing facets ("links")

Table 1. The relation between the real-world problem domain and basic facet-modelling concepts.

model as one or more typed *facets* belonging to the item, as shown in Table 1. This means that any number of aspects of the phenomenon that are relevant for the problem at hand can become facets of the corresponding item. Apart from its identity, an item encompasses nothing more than a set of facets.

A *facet-modelling language* is defined as a non-empty set of *facet types*. A *facet model* is correspondingly a non-empty set of *items*. All the items in a facet model have the same type, which is implicitly defined by the facet types in the language.

Hence, the facet types available to the modeller will depend on the particular facet-modelling language being applied. Examples of facet types are *methodological facets* such as the purpose of and requirements put on an item; *existential facets* such as the lifetime, physical location and movement of an item, and its material substance; *organisational facets* such as the actor of an item, the responsibilities assigned to an item, its capabilities and the roles it may assume; *informational facets* such as the material properties and data contents of an item; *structural facets* such as the parts, members, instances and refinements of an item; and finally *behavioural facets* such as the transformations, transportations and preservations effectuated by an item. This particular set of facet types is by no means the only one possible, and the authors do not attempt to argue that it is in any sense "better" than other alternatives. Hence this paper simultaneously deals with two problem-levels:

1. The idea of facet modelling in general.
2. An example of a particular facet-modelling language.

The second level serves to exemplify the first.

Example 2. Consider a facet-modelling language which integrates some of the modelling orientations commonly applied in structured analysis: dataflow diagrams, ER-diagrams, and agent models. The following facet types are needed:

- Dataflow diagram types for: external entity node, process node, store node and flow edge.
- ER-diagram facet types for: entity node, relationship node and subset edge.
- Agent model facet types for: agent node, role node, agent-role edge, and subagent edge.

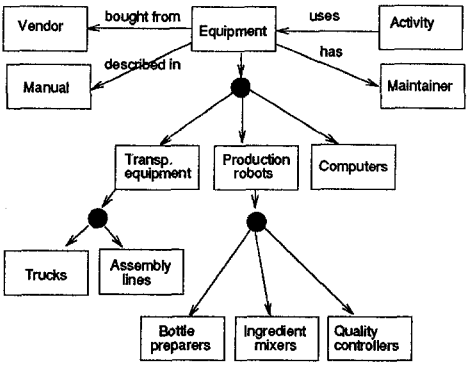

Fig. 1. Some perceived objects in the EAS.

– Inter-model facet types for: entity-flow edge (which entities move on which flows) and role-process edge (which roles perform which processes).

Hence the facet-modelling language has one facet type for each node type supported by the orientated modelling approaches it embeds, as well as additional types used for inter-model connections. An advantage with the facet based approach is that these important inter-model connections become part of the facet-modelling language, at the same level as other statements. □

3.2 Facets and Links

As already mentioned, each relevant aspect of a real-world phenomenon is represented as a facet of the corresponding item. Each item in a facet model therefore has a non-empty set of *facets*. Every facet has one and only one facet type.

Example 3. The arbitrariness of choosing which real-world aspects to represent can now be demonstrated in terms of the equipment administration system (EAS) example:

Choosing an information- or object-oriented approach, one might decide to explore what different kinds of equipment there are (i.e., build a classification hierarchy with relations.) In an object-oriented model, it would also be interesting to specify which possible operations have to be supported for each of these, as well as identifying other objects relevant for the equipment administration. A small example is shown in Figure 1, where every piece of equipment has a vendor, a manual, and a responsible maintainer within the company. It is also considered interesting to have information about what activities are using the various forms of equipment. Notice, however, that the "maintainer" and "activity" nodes represent only *information* about the phenomena, rather than the "maintainer" itself as an agent or the "activity" itself as a process.

From a process-oriented perspective, the modeller might start capturing the activities involved in equipment administration, such as equipment purchasing, registration, control, maintenance, and out-phasing. The phenomena modelled here are seen as the ongoing actions themselves and not as information about the processes, as was the case with "activity" in the object-oriented case.

From an agent-oriented perspective, one would instead start with the roles of the company, in particular those involved in equipment administration, such as equipment manager, equipment engineer, equipment assistant, operator, and secretary. These phenomena would primarily be seen as roles in the organisation, not as objects that information would have to be stored about, as was the case for "maintainer" in the object-oriented model.

The idea of a facet-modelling framework is that any orientation (function-, information-, object-, subject-, or other) may be useful, and the framework should be able to capture them explicitly whenever needed. Hence, e.g., an "equipment maintenance" phenomenon will have several facets:

- A process facet, the activity, which will have links to other activities done in connection with maintenance (at the same level of abstraction), and to activities above or below it in the decomposition hierarchy.
- An entity (or object) facet, the information about maintenance, which will have links to other information objects, again both to the same level of abstraction and to the levels below and above.
- A role facet, the maintainer, which may be linked to other roles that it communicates with and to agents able to fill this role. □

A facet is specified according to its facet type. Facet type outlines will be presented in Section 4. In general, a facet is either a single (typed) *value*, or a set or tuple of *subfacets*. Examples of value types are conventional types, such as integers and free text, as well as *references* which link facets to other items. Subfacets are themselves facets which may in turn have subfacets on their own. Links are directional in that they always go from the facet being specified to the item which is referenced. Hence the concept of "link" is not basic in the facet-modelling approach. It is just another (referencing) type of facet value. It is nevertheless a useful term when discussing and presenting facet models, and it will therefore be used in the sequel.

Example 4. In *Example 2* some of the facet types could be conveniently defined as links:

- For DFD diagrams: (external entity node, flow edge), (process node, flow edge), and (store node, flow edge).
- For ER-diagrams: (entity node, relationship node) and (entity node, subset edge).
- For agent diagrams: (agent node, subagent edge), (agent node, agent-role edge), and (role node, agent-role edge).

In addition, the inter-model facet types would probably also be represented as link facet types. □

4 Outlining Facet Types

Section 3 mentioned a large number of facet types for a particular facet-modelling language. Although it is not the purpose of this paper to define each of these proposals in detail, it is nevertheless clarifying to discuss some of them more elaborately.

4.1 Facet Type Outlines

Methodological facets such as the purpose of and the requirements placed on items might easily be specified as free text associated with the item. As the facet model becomes more detailed, it should also be possible specify requirements in 1. order predicate calculus. Furthermore, it should be possible to *link* a requirement facet to other items to represent how requirements trickle through the model. In this way, traceability is incorporated as part of the facet model itself. The structural facet types outlined below will provide such links.

Existential facets of an item are represented as simple, valued facets: Its lifetime is defined by birth and death times, while physical location and item movement are represented as a location facet. The presence of material substance of an item in the model is represented as a truth valued facet.

Organisational facets such as actors, responsibilities, capabilities and roles may again best be represented textually. Actor and role facets should of course have links to represent actor-subactor and actor-role relationships.

Informational facets such as the material properties and the data contents of an item are represented as sets of typed values.

Structural facets are represented as links to other items. Hence, the parts of an item are a set of references to items. Similarly, the members, instances and refinements of an item are also represented as links. Links can be traversed in both directions, so the inverse structural relations are also covered.

Behavioural facets are unsurprisingly the most complex facet types to define. The authors have previously considered this problem in [16, 17], and the facet-modelling framework has been designed to incorporate the ideas presented there. More specifically, a transportation (or flow) is an (instantaneous or gradual) modification of another item's position facet leaving its other facets unchanged, while a preservation (or storage) is a conservation of position. In its most primitive form, a transformation is a modification of an informational facet.

Example 5. Section 3.1 presented a simple first example which integrated some of the modelling orientations of structured analysis. It turns out that some of the facets introduced for that purpose correspond to those presented here:

- Dataflow diagrams facet types for processes, stores and flows correspond to transformation, preservation and transportation facets, as discussed in [16]. As for the external entity facet type, the actor or role facet of this section seems appropriate.

- ER-diagram facet types for entities are represented through the data contents (and possibly material properties) facets of this section. The relationship facet can be represented as a part facet comprising references to items with related entity facets.
- Agent model facet types for agents are best represented as actor facets, while roles of course remain roles. Subagents are again best represented through the part facet type. □

4.2 Facet Type Dependencies

The facets defined for an item are obviously not independent. Instead, the value of one facet — or the fact that it is specified or unspecified — has consequences for facets of the same or of other items.

Most fundamental is the lifetime facets of an item, which restrict the specification of facets such as position, as well as the item's participation in transportation, preservation, and transformation. Along the same lines, an item can only have material properties if it has material substance.

Rather than adding complexity to the framework, these and other dependency rules aid the modelling process by 1) making inconsistencies easy to detect by inspection or by formal verification; 2) making changes made to one facet immediately apparent in terms of another facet also; 3) implicitly specifying facets as consequences of other explicitly specified ones, and 4) designing versatile and friendly user interfaces.

5 Visualisation

Visual representation has many advantages over textual representation, both when it comes to comprehension (e.g., in providing at a glimpse overviews) and expressive economy [11, 23]. Therefore, many modelling languages have been designed with a particular visual representation in mind.

Large facet models will have many items, and even more interconnections among these along various dimensions. Hence they will be impossible to display nicely in two dimensions and possibly hard also in three. This is not really a new problem — all realistic models of detailed material- and information-processing systems will be too complex to visualise all at once. Therefore, abstractions and filtering techniques are needed [20, 21]. However, the inherent complexity of the facet-modelling framework makes such visualisation and filtering techniques all the more important.

No visualisation tool has yet been built for the approach presented in the previous sections. Still, it is found important to discuss visualisation here to indicate the feasibility of the facet approach for visual presentation. Actually, this section attempts to demonstrate that the facet-modelling framework may have an advantage in providing *greater flexibility for views*, since no particular view is promoted by the modelling framework as such. The ideas presented here are partly inspired by [21].

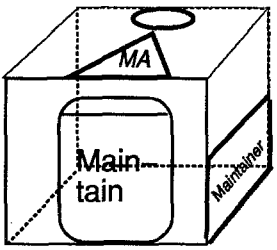

Fig. 2. An item-cube

Visualisation may be addressed according to two different principles: 1) browsing and 2) associatively selected views. In both cases, useful views will have to rely on filtering, i.e., not showing the entire model at once. The difference is that for browsing, filtering will be implicit: the nodes which are far away from the current viewpoint, will be vaguely or not at all visible. For associatively selected views, on the other hand, the user will describe through some suitable interface what should be included in the view.

Looking at browsing first, although the screen is two-dimensional, the user can easily be provided with 6 working directions: up, down, left, right, in, out, providing a three-dimensional feeling. As long as items have a fairly limited number of facets, this can be utilised to create a versatile interface for browsing. Imagine one item presented as a cube where different sides correspond to different facets. In the example in Figure 2 we see the front side being used for a process facet, the right side for an object (or entity) facet, and the top side for an agent (or role) facet. The three remaining sides are still vacant for other facets of interest. Turning the cube around, the user can focus on the facet of most interest, whose relations to other nodes will then also be shown. Moreover, zooming into a particular side, the decomposition of the corresponding facet will emerge (e.g., the decomposition of "Maintain" into lower lever activities), and zooming away from it, the composition will emerge (e.g., the higher level business function that "Maintain" is part of).

With such an approach, one could browse around more complex structures such as the one indicated in the upper part of Figure 3. However, browsing may not be suitable for all situations. For rapid exploration into a complex model, it is probably a good idea, but for detailed discussions of the model among a large group of people, it will probably be necessary to generate more persistent views. Filtering for such views may be done by

- facet (i.e., stating what facets should be included and/or excluded)
- item (i.e., stating what items should be included and/or excluded)

Of course, it should be possible to combine facet and item filtering, and also filtering statements of the inclusive (what should be *in* the view) and exclusive (what should be hidden) kind. In the example of Figure 3, a messy model —

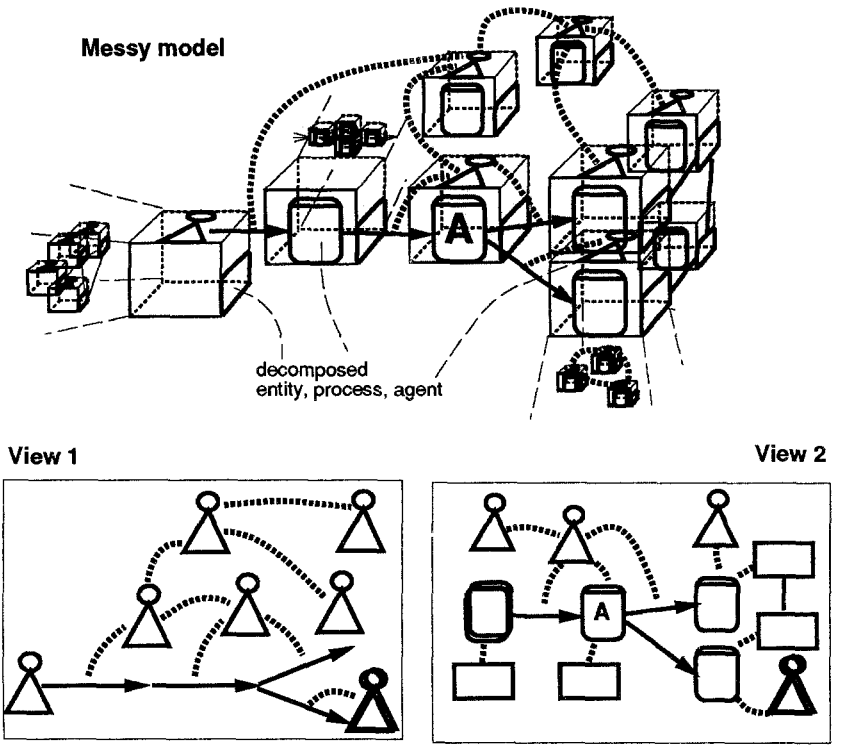

Fig. 3. Filtering

which is still only a small fraction of what would be the total facet model for a large system — has been filtered to two different views. In View 1, only flows, agents, flow-agent, and agent-agent edges are shown. Assuming that the diagram has been sensibly annotated, this could be a useful view for seeing what is moving around in the system and who is responsible for the transportation. View 2 displays only the node whose process facet is marked **A** in the messy picture, the edges directly connected to this node, and the nodes connected to these edges (i.e., filtering by item.) Both the resulting views have become so simple that they can be displayed nicely in two dimensions. For larger models, the first filtering operation might not be sufficient to yield a good looking view, so that several iterations of filtering will be required. Combining filtering and browsing might also be a good idea in many cases.

6 Conclusion

The paper has explained why clearly orientated models is not anything to strive for in the early phases of problem analysis, but rather a source of problems

on their own. To alleviate this situation, a facet-modelling framework was introduced, and a simple facet-modelling language was outlined. The paper also outlined important facet types, as well as visualisation.

Facet models represent the dynamics, processing, functions, structure and purpose of items using natural and dedicated concepts, while maintaining an internal representation of the item per se, as a sum of its defined facets. Hence, the facet approach is a departure from conventional problem analysis approaches in several ways:

- From visual models to *visualisable structures.*
  Instead of designing modelling approaches with visualisations in mind, the facet modelling approach encourages selection of important aspects of the real-world problem domain as a distinct activity from visualisation.
- From diagrams on paper to *diagramming tools.*
  A diagrammatic modelling language as of 1995 must fully utilise contemporary technology. The facet-modelling framework has been designed with powerful visualisation mechanisms in mind.
- From items as objects to *items with aspects.*
  While the object-oriented modelling approaches focus on the real-world phenomena per se, the facet-modelling approach focusses on representing important aspects of the phenomena. Furthermore, a rich set of concepts is provided for specifying relevant facets. Hence facet modelling is a departure from claims made by OOA-proponents that a single concept (the object) — or at least a small set of core concepts (e.g. objects, states, classes, inheritance, instances, operations and messages) — is sufficient to create rich and intuitive models of real-world problems.
- From description-driven to *problem domain-driven methodology.*
  Available SA and OOA approaches are description-driven in that the description techniques themselves set the agenda for which pieces of knowledge about the problem domain to collect at any given time of analysis. A facet model can be extended at any time with any type of problem-domain knowledge that may become relevant. This is in line with recent methodology frameworks, e.g., [14], which emphasise IS development as inquiry processes driven by the analysis problem itself, rather than by assumptions inherent in some fixed modelling language.
- From repositories of separate models to *a single integrated model.*
  Instead of distributing the specification of a real-world system between several (visual) models and then maintaining the references between model components, the facet-modelling approach maintains a single model only. The simplifications implied by this are obvious.

Facets can also be added for non-functional aspects of the problem domain, such as performance and reliability, and for representing developers and the development process itself as part of the model. It may be possible to regard the final implementation itself as a set of linked facets of items.

Of course, the facet-modelling framework is more ambitious than the traditional approaches. Thus, if a traditional approach is able to deliver a satisfactory

model for a certain problem, the facet framework might not be needed. However, there are several reasons to believe that there is a need for an ambitious approach covering many aspects of the problem area in one comprehensive model:

First, real-world problem domains are becoming ever more complex, partly because the easier problems are being solved and turned into shelfware, frameworks, or reusable libraries, leaving only the difficult problems for analysis.

Second, problem domain models are becoming assets in themselves, because of the drastically increased turbulence in organisations and society. The trend towards virtual organisations, organising on demand to do a particular job, means that organisations will end up being more short-lived than the ISs they use. If a company is supposed to live only for a year or two, not to speak of a month or two, it is no longer feasible to take up a traditional development project, making a detailed analysis model for business processes and the IS support needed. Instead, it will be necessary to reuse, modify and combine existing models for similar businesses, so that the IS support for the organisation can be up and running from day one, with just some customisation afterwards. When the organisation phases itself out, the models of IS support that it has generated, will be useful input to new organisations popping up. Hence, models will be a commodity in their own right, to a much larger extent than today (when many regard them mostly as a necessary step towards a running software system). Then it will also make sense to invest more resources in the construction of high quality models, and to have an ambitious approach to allow flexible customisation of these models.

References

1. S. C. Bailin. An object-oriented requirements specification method. *Communications of the ACM*, 32(5):608–623, May 1989.
2. T. Bryant and A. Evans. OO oversold. *Information and Software Technology*, 36(1):35–42, January 1994.
3. J. A. Bubenko jr. Problems and unclear issues with hierarchical business activity and data flow modelling. Technical Report 134, SYSLAB, Stockholm, June 1988.
4. P. P. S. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.
5. P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice-Hall, Englewood Cliffs, 1990.
6. T. DeMarco. *Structured Analysis and System Specification.* Yourdon Inc., New York, 1978.
7. E. Dubois et al. ALBERT: an agent-oriented language for building and eliciting requirements for real-time systems. In *Proc. of HICSS'27, vol.IV, Information Systems: Collaboration Technology, Organizational Systems and Technology*, pages 713–722. IEEE Computer Society Press, 1994.
8. C. Gane and T. Sarson. *Structured Systems Analysis: tools and techniques.* Prentice-Hall International, 1979.
9. O. C. Z. Gotel and A. C. W. Finkelstein. Modelling the contribution structure underlying requirements. In K. Pohl et al., editor, *Proc. 1st International Work-*

shop on Requirements Engineering: Foundation of Software Quality (REFSQ'94), Utrecht, June 1994.

10. J. A. Gulla, O. I. Lindland, and G. Willumsen. PPP — an integrated CASE environment. In R. Andersen, J. A. Bubenko jr., and A. Sølvberg, editors, *Advanced Information Systems Engineering, Proc. CAiSE'91, Trondheim*, pages 194–221, Heidelberg, 1991. Springer Verlag (LNCS 498).
11. D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.
12. W. Harrison and H. Ossher. Subject-oriented programming (a critique of pure objects). In A. Paepcke, editor, *OOPSLA'93 Conference Proceedings*, pages 411–428. ACM Press, 26 Sep–1 Oct 1993. (Also as ACM SIGPLAN Notices 28(10), Oct 1993).
13. G. M. Høydalsvik and G. Sindre. On the purpose of object-oriented analysis. In A. Paepcke, editor, *OOPSLA'93 Conference Proceedings*, pages 240–255. ACM Press, 26 Sep–1 Oct 1993. (Also as ACM SIGPLAN Notices 28(10), Oct 1993).
14. J. Iivari and E. Koskela. The pioco model for information systems design. *MIS Quarterly*, pages 401–419, September 1987.
15. S. McGinnes. How objective is object-oriented analysis? In *Proc. CAiSE'92: The Fourth Conference on Advanced information Systems Engineering, Manchester, UK*, Heidelberg, 1992. Springer Verlag (LNCS 593).
16. A. L. Opdahl and G. Sindre. Concepts for real-world modelling. In C. Rolland et al., editor, *Advanced Information Systems Engineering, Proc. CAiSE'93, Paris*, pages 309–327. Springer Verlag (LNCS 685), 1993.
17. A. L. Opdahl and G. Sindre. Representing real-world processes. In J. F. Nunamaker and R. H. Sprague, editors, *Proc. of the 28th Annual Hawaii International Conference on System Sciences (HICSS'28)*, volume IV, pages 821–830. IEEE CS Press, 1995.
18. C. A. Richter. An assessment of structured analysis and structured design. *SIGSOFT Software Engineering Notes*, 11(4), 1986.
19. H. Rittel. On the planning crisis: Systems analysis of the first and second generations. *Bedriftsøkonomen*, (8), 1972.
20. A. H. Seltveit. An abstraction-based rule approach to large-scale information systems development. In C. Rolland et al., editor, *Advanced Information Systems Engineering, Proc. CAiSE'93, Paris*, pages 328–351. Springer Verlag (LNCS 685), 1993.
21. A. H. Seltveit. *Complexity Reduction in Information Systems Modelling.* PhD thesis, DCST, NTH, University of Trondheim, 1994. NTH 1994:121.
22. S. Shlaer and S. J. Mellor. *Object-Oriented System Analysis: Modeling the World in Data.* Prentice-Hall, Englewood Cliffs, NJ, 1988.
23. G. Sindre. *HICONS: A General Diagrammatic Framework for Hierarchical Modelling.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Trondheim, 1990. NTH 1990:44, IDT 1990:31.
24. G. Verheijen and J. van Bekkum. NIAM: an information analysis method. In T. W. Olle et al., editor, *Information Systems Design Methodologies: A Comparative Review*, Amsterdam, 1982. North-Holland.