FACET MODELLING: AN APPROACH TO FLEXIBLE AND INTEGRATED CONCEPTUAL MODELLING†

ANDREAS L. OPDAL1 and GUTFORM SINDRE2

1Department of Information Science, University of Bergen, Norway
2Department of Computer and Information Science, Norwegian University of Science and Technology, Norway.

(Received 27 July 1995; in final revised form 15 July 1997)

Abstract — The paper points to weaknesses of conceptual modelling languages that are oriented towards certain aspects of the problem domain of information systems (IS) development, e.g., activities, information resources, objects, or actors. It is concluded that modelling languages are needed that allow modellers to 1) freely choose to represent a wide and extensible range of aspects of problem domain phenomena contingent on the problems at hand; 2) simultaneously co-represent several aspects of the same problem domain phenomenon whenever needed; 3) reflect semantic relations between these aspects in the problem domain models; and 4) extend the set of kinds of aspects that can be represented and visualised throughout problem analysis as understanding of the problem domain and the problems at hand increases. An approach called facet modelling of real-world problem domains is therefore outlined to deal with the complexity of contemporary analysis problems. It is shown how facet models can be defined and visualised, before facet modelling is discussed in relation to other recent ideas and techniques in the IS development field. Case studies are currently in progress to evaluate various implications of the facet modelling approach empirically. Copyright ©1997 Elsevier Science Ltd

Key words: Perspective Integration, Conceptual Modelling, Problem Analysis, Facet Modelling

1. INTRODUCTION

During problem analysis, conceptual models of the problem domain of information systems (IS) development are established, continuously assessed and improved, and then used as a starting point for IS design. Numerous modelling languages have emerged for this purpose. Each of them are oriented towards certain aspects of the phenomena in the real-world problem domain. Some of the most prominent orientations are: 1) transformation orientation (i.e., structured analysis, SA), emphasising the activities performed in the problem domain (e.g., [16, 23, 59, 26]); 2) information orientation (i.e., entity-relationship (E-R) modelling), emphasising information types and relationships between them (e.g., [11, 60, 64]); 3) object orientation, emphasising interacting problem domain phenomena (e.g., [57, 12]), and 4) subject orientation, emphasising the actors, agents, roles, organisational units, etc. which perform activities (e.g., [18, 36]).

Each of these orientations has its pros and cons. Criticism of structured analysis can be found in e.g., [53, 9, 48], of purely information-oriented languages in e.g., [26], and of object-orientation in e.g., [38, 28, 29, 8]. The more recent and less common subject-oriented languages have been less criticised in the literature so far.

This paper is written from the viewpoint that strongly oriented models are not to be striven for early in problem analysis, and that strong orientation may instead cause modelling-related and methodological problems during problem analysis because of

†Recommended by Juhani Iivari
- limited freedom of *choosing* to represent different perspectives on the problem domain contingent on the problem domain and the problems at hand;

- reduced possibility of *co-representing* multiple perspectives on the problem domain simultaneously;

- weak or no support for *integrating* these perspectives by representing and managing semantic relations between them in the problem domain model;

- no way of *extending* the modelling language used to represent and visualise new perspectives which are discovered to be relevant during problem analysis.

The importance of perspectives in IS development has been emphasised by Dahlbom & Mathiassen [14] who note that “our world is shaped by our experience of it. We see different things, have different perspectives, structure the world differently, depending on interests, background, education and culture”. Also, Nygaard [44] has pointed out that program development always involves several individuals and groups of individuals with divergent knowledge, backgrounds, interests, motives, values, beliefs, and hence different perspectives. The rest of the paper will explore this assumption by presenting a framework for *facet-modelling* which increases freedom of choice with respect to perspectives and simultaneously supports co-representation, semantic integration and extension of perspectives. Several case studies using the facet-modelling approach are in progress, and the paper reflects early results from these studies.

First, Section 2 will elaborate further on what orientation really is, before Section 3 presents the facet-modelling approach itself and Section 4 briefly discusses how to visualise the resulting models. Section 5 points out that the idea of representing a problem domain from several perspectives simultaneously is not new in itself, and proceeds to compare facet modelling with several related ideas and techniques, such as multi-perspective modelling, viewpoints, hyperstructures, metamethodologies, object-oriented analysis, multiple inheritance and instantiation, as well as subject-oriented programming. Section 6 finally concludes the paper and suggests paths for further work.

2. ORIENTATION

2.1. Orientation: What is It?

The introduction presented four examples of possible orientations for representing a problem domain. One might think that the difference between them is that they capture different aspects of the problem domain, i.e., that transformation-oriented models describe when and where information and materials are manipulated, that information-oriented ones describe information elements and restrictions on them, that object-oriented ones describe how problem domain phenomena interact, that subject-oriented ones describe human actors, agents and roles, and so on. However, the main point is rather that they capture similar aspects of problem domain phenomena differently: The final IS, if it is built, will have to reflect the problem domain appropriately with regard to both manipulations of information and materials, interrelated information elements, interacting phenomena, as well as human actors, agents, roles, etc.

A related point is stressed by Ivvari & Koskela [33] and Ivvari [31, 32] in their work on the PIICO model of an information and software system product. The PIICO model comprises a comprehensive, multi-levelled set of structural, functional and behavioural aspects which can be identified in any information and software system regardless of how it has been developed. Zachman [70] and Sowa & Zachman [61] also present a framework for and taxonomy of information

---

1. The phrase “problems at hand” is meant to reflect that IS development may be initiated in response to several problems or opportunities. Even in situations where there is seemingly a single, clear motivation for developing a new system, this motivation will be perceived differently by different stakeholders, so that it is more appropriate to speak of “problems” than of a single “problem”. 

systems modelling. Like the present work, both contributions emphasise that there are numerous perspectives on problem domains and information systems, and that all these perspectives must be captured in one way or the other at some stage during IS development for them to be reflected in the final system. On the other hand, both PIOCO and the Sowa & Zachman framework are restricted to a limited set of pre-defined perspectives, whereas the facet modelling approach to be introduced in this paper allows introduction of new perspectives when needed.

The above discussion indicates that the main difference between the orientations of Section 1 is not which aspects they capture and represent. Instead, the difference is one of focus, representation, dedication, visualisation and sequence, in that an oriented modelling language typically prescribes that:

- some aspects are promoted as fundamental to modelling, whereas other aspects are covered mainly to set the context of the promoted ones (focus);
- some aspects are represented explicitly, others only implicitly (representation);
- some aspects are covered by dedicated modelling constructs, whereas others are less accurately covered by general ones (dedication);
- some aspects are visualised in diagrams, others only recorded textually (visualisation);
- some aspects are captured before others (sequence).

Example 1 In dataflow diagrams (DFDs), the aspect represented by “processes” (i.e., functional transformations) tends to be promoted as fundamental to modelling. Processes are provided with the only powerful abstraction mechanism in the language (process decomposition), and the modelling constructs representing other aspects, such as storage, flow of data and external entities, are grouped around them. Also, processes are usually represented with larger icons than the other constructs. Both processes, stores, flows and external entities are represented explicitly and by dedicated modelling constructs in DFDs. Other aspects, such as what kind of data that are transformed and stored, are only treated implicitly and more generally in terms of the data catalogue descriptions for stores and flows. This also means that transformations, stores and flow of data are all visualised in diagrams, whereas information about the kinds of data that are manipulated is only recorded textually. Finally, processes, stores, flows and external entities tend to be captured before the corresponding parts of the data catalogue are written.

2.2 Orientation: Why to Avoid It?

From the previous section, it is clear that choosing a particular orientation will give priority to certain aspects at the expense of others. As a consequence, less emphasised aspects will be more difficult to account for during problem analysis. They will be more weakly reflected in the requirements specification and hence probably in the final information system.

Example 2 For instance:

- In DFDs, it is not easy to see which transformations that are dealing with the same objects. This makes it difficult, e.g., to ensure change-locality. On the other hand, object-oriented analysis (OOA) models explicitly represent which activities that manipulate objects of the same class. This is not shown explicitly in a DFD and can only be established through tedious model inspection.

- In OOA models, it is not easy to identify sequences of low-level transformations. Since low-level transformations are not aggregated into higher-level transformations, it is not easy to grasp overall information flow without elaborate model assessment. As a consequence, it is also difficult to see exactly how end-to-end services are delivered to the system’s users [3].
Both DFDs and OOA models lack dedicated modelling constructs for representation of actors, agents and organisational units, as well as their competences, responsibilities, privileges, roles, etc. As a consequence, the humans in the problem domain and the relations between them are usually not described in detail. If they are described at all, it is usually by using less accurate modelling constructs. This makes it difficult, e.g., for future users to relate the conceptual model to their own working experiences and to comprehend organisational changes caused by the system.

The points listed in Section 2.1 explain why it might be a good idea not to restrict analysis to a single (or to a small set of) orientation(s):

Orientation is a matter of prioritising different kinds of information about the problem domain, because some aspects of phenomena 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 language used (foci). As a consequence, the problem domain will be looked at from a single or a few perspective(s) the whole time, thereby hiding weaknesses that would have been apparent from other perspectives. A related awkwardness with oriented modelling languages is that they force the modeller to see any phenomenon — or any aspect of a phenomenon — as represented by a single construct. However, in reality several aspects of the same phenomenon may be equally important for the problems at hand at the same time. This may be due to multiple users or modellers with different perspectives, or to a single user or modeller who perceives several perspectives as relevant simultaneously. Capturing several different aspects at the same time is difficult in a strictly oriented language, 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. In addition, and as already mentioned, the set of available modelling constructs has been determined in advance by the modelling language orientation.

Orientation means that some aspects of phenomena in the problem domain will be captured in more general terms because the modelling language lacks accurate, dedicated concepts for representing them (dedication). Furthermore, it is usually not possible to know in advance which kinds of aspects that will turn out to be relevant for problem analysis so that an appropriate language can be chosen at the outset. Wand & Weber [65] note that the “ease of mapping from the modelled domain into the modelling constructs and the absence of ambiguity in the resulting model depend on how close the modelling primitives are to the elementary objects in the modelled domain”, and they also point out that a single (oriented) modelling language might fit one part of the modelling needs much better than the other. To compensate for this, several languages may be used, but in that case the process of translation between them may lead to loss of information and inaccurate mapping. Wand & Weber also state that “there are reasons to suspect that such a mapping is forced implicitly even when only one modelling language is employed, and that this mapping might lead to loss of relevant aspects”. [65]

Orientation means that some aspects are not represented explicitly, and if they are represented they may not be visualised in models (representation and visualisation). Ivari & Koskela [33] point out that distinct aspects of an IS reflect different choice and quality criteria, and that restricting the kinds of aspects that can be represented therefore tends to limit the kinds of criteria that are taken into account when developing a new information system. Along similar lines, orientation implies capturing some aspects before others (sequence). Sowa & Zachman point out that [61]: “Order implies priorities. It creates a bias toward one aspect at the expense of others”. [...] “The sequence determines the value system being applied in making the design tradeoff decisions [...].”

Finally, oriented conceptual models will inherently encourage the participation and support the interests of some of the individuals and groups affected by a new IS, at the expense of the participation and interests of others. According to Nygaard & Srgaard [45], a systems analyst “exercises perspective power and establishes a perspective monopoly by insisting upon, and achieving, the exclusive use (in the development process) of facts, experience, concepts, techniques and tools which are meaningful within the framework of a system perspective.” As a result, other stakeholders — which may not be confident with or even capable of understanding the analysts perspective — become completely dominated. As observed in case studies by Gjersvik [24], employees at unequal levels in an organisation have different abilities and preferences when it comes to understanding
models. Managers have the highest preference for overall, abstract models, whereas shop-floor workers tend to prefer more concrete representations of the problem domain. Also, orientation means that communication will be easy for stakeholders for whom the particular orientation is natural, and more difficult for others.

**Example 3** E.g., using aspects known from conventional oriented languages, a (particular) conveyor belt in a beverage company, BevCo, can be perceived as a piece of information (similar to an entity instance of conventional E-R modelling); as a transporter of workpieces; as a pre- server or blocker (since transportation takes time); as an actor capable of and/or responsible for transporting workpieces; and as a factor which has consequences for product quality. As already mentioned, the main reason for this is that the same problem-domain phenomenon is perceived differently from different perspectives. Which of the above aspects that is considered to be most appropriate will depend on who the observer is, e.g., inventory managers (who see it as a piece of information), the workers labouring along it (who see it as a transporter), process managers (who see it as a delayer), and quality controllers (who see it as a quality factor), as well as on the observer's intentions. Other potentially relevant aspects include the conveyor belt perceived as an expense, as a tax-deductible investment, as a hazard to the shop-floor workers health, as the source of a requirement on a future computerised IS, or as the fulfillment of a requirement on the current production plant. The situation is further complicated because the same observer may perceive different aspects of the same phenomenon to be important in different situations and at different times. E.g., although the workers at BevCo in most contexts perceive the conveyor belt as a transporter, with properties such as transport speed, they may sometimes instead regard the belt as a health hazard, with ergonomic measures among the relevant properties. One reason for this is that the observers’ intentions are continuously changing.

The above example has primarily been chosen as a simple and straightforward way of explaining the idea of facet modelling without requiring a detailed understanding of any less common modelling constructs or modelling languages. As a consequence, a few well-known and well-understood aspects and perspectives have been emphasised, typically the ones covered by conventional oriented modelling languages, such as E-R modelling and physical DFDs. This emphasis will also be reflected in the subsequent discussion, but it should not be taken as an indication that the facet-modelling approach is inherently limited to these conventional aspects. Indeed, the potential scope of facet modelling extends broadly across different perspectives [44], modelling levels [31, 70, 61], modelling worlds [34, 41] and IS development phases. The particular set of aspects used in the examples of this paper is very conventional and by no means the only one possible.

**Example 4** The managers of BevCo suspect that current equipment maintenance procedures both impede productivity and raise production costs: Although the utilisation of many kinds of production equipment is low, management has observed that production delays nevertheless occur because of misplaced or malfunctioning equipment. Also, maintenance costs have exceeded expectations several years in a row. Managers suspect that reporting routines for equipment use are insufficient, and employees take liberties in relocating equipment on the fly without the managers’ permission or knowledge whenever there is a local need. Hence, a new equipment administration system (EAS) is being planned to keep track of the company’s production equipment and how it is used, so that decisions about equipment purchase, allocation and maintenance can be made based on up-to-date records and histories of equipment allocation and use. The new EAS is to keep an inventory of each of the production tools (robots, controllers, sensors, etc.) in the company’s possession, together with information about, e.g., their current statuses and allocations. The choices of aspects that eventually become reflected in the problem domain model may be essential to which requirements that will be uncovered, to which groups of stakeholders that will have influence on the development process, and hence to how well the final EAS will support BevCo and its employees.
3. THE FACET-MODELLING APPROACH

3.1. Goals

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

**Perspective freedom:** It should enable the modeller to choose to capture any kind of aspect of a problem-domain phenomenon, and any kind of dependency between aspects. To the extent possible, the choice of what to represent and when to represent it should be left to the modeller contingent on the problem domain and the problems at hand.

**Perspective co-representation:** Whenever several aspects of the same phenomenon are relevant to describe the problem domain, it should be possible to capture them simultaneously as well as structurally and semantically close to one another in the model instead of having to use several isolated modelling constructs and several isolated partial models.

**Perspective integration:** If several aspects of the same phenomenon are semantically related, this should be reflected in the problem domain model. E.g., the transporter and delayer aspects of the conveyor belt at BevCo are probably related semantically because they receive their inputs from and deliver their outputs to the same locations, as will be demonstrated in Example 10. The modelling language must support representation and management of such relations.

**Perspective extensibility:** As new kinds of aspects are recognised as relevant to the problems at hand during analysis, it should be possible to extend the modelling language to account for them. Also, it should be possible to visualise perspectives freely based on the already recognised kinds of aspects that are represented in a model.

These are the four main characteristics of what will now be introduced as the **facet-modelling approach**. The approach has the power to embed many conventional oriented languages. Hence it implicitly covers problem domains where these languages are already sufficient. Even more importantly, the approach also makes it possible to include within the same conceptual model other aspects which are less often taken into account today. Indeed, the facet-modelling approach encourages introduction of new kinds of aspects as problem analysis proceeds. Therefore, facet-modelling is not "yet another modelling language". Instead, it encourages generalisation, integration and extension of existing languages, and thereby makes orientation obstacles less dominating in problem analysis. However, it is not in any fundamental sense "orientation free": Problem analyses always emphasise certain aspects of the problem domain at the expense of others. What the facet-modelling approach contributes is a simple and intuitive, yet powerful, formalisable and extensible way of alleviating many methodological and modelling-related obstacles of orientation. Also, facet modelling is not "against method". Creating a representation of a problem domain must always be based on some fundamental guidelines for how to proceed through problem analysis. However, the ideal sequence of steps to be taken will often depend on discoveries made as analysis proceeds. Facet modelling therefore provides an approach to problem analysis which places as few implicit restrictions as possible on the sequence in which the different aspects of problem domain phenomena are captured, and on which aspects that are captured.

3.2. Basic Assumptions

A real-world problem domain is a collection of **phenomena**, which may be either concrete or abstract: Concrete phenomena are **material things**, e.g., "conveyor belt #5" and "production plant B" at BevCo, whereas abstract phenomena correspond to **social constructs**, i.e., "socio-cultural formations" and "institutions" [4], "social concepts" and "social institutions" [54]. Examples of abstract phenomena are BevCo’s "overall mission" and its "procedure for soda production", both of which reflect more than just the physical media on which they may be imprinted. Because the phenomenon concept encompasses both material things and social constructs, it is robust with
respect to conflicting metaphysical positions such as realism and anti-realism [2]. A further distinction is made between phenomena which represent individual things or constructs, and those which represent sets or categories of other phenomena. E.g., at BevCo, the concept of “conveyour belt” is a \textit{categorical phenomenon}, while “conveyour belt #5, the one between the filling and capping stations in the soda-production line” is an \textit{individual phenomenon}. Phenomena are \textit{dumb}, and hence, events are not phenomena\footnote{Instead, events would reflect changes pertaining to phenomena, and they will therefore typically show up in the definition of the semantics of a particular facet-modeling language.}.

Phenomena possess \textit{properties}. Examples of properties of “conveyour belt #5” are its current allocation, its type and time since last maintenance, its transportation speed, its ergonomic measures, and its reliability. Phenomena and properties are similar to “things” and “properties” in the Bunge-Wand-Weber model (e.g., [66]).

As pointed out in Section 2.2, users and modellers always perceive the phenomena and properties in the problem domain from a \textit{perspective}, and Example 3 listed several relevant perspective holders in the conveyour belt example, such as inventory managers, shop-floor workers, process managers and quality controllers. Due to their perspectives, a user or modeller does not perceive the problem domain as a collection of phenomena per se, but instead perceives particular perspective-determined \textit{aspects} of the phenomena. E.g., at BevCo, the inventory managers might perceive “conveyour belt #5” as a piece of information, the shop-floor workers as a transporter of workpieces, whereas the process managers might perceive it as a delay of workpieces. This is also the case for abstract phenomena, e.g., the “procedure for soda production” at BevCo might be perceived as a means by the process managers, as a constraint by the shop-floor workers, and as a process by the quality controllers. Only in rare cases would someone perceive a conveyour belt in its brute materiality according to modern physics; as a purposeless, and hence meaningless and structureless, spatial extent which is mostly empty and which interacts with its environment according to the rather strange laws of quantum mechanics. Nor would anyone intuitively perceive the soda production procedure at BevCo according to Berger & Luckmann [4]: as a social institution upheld by a continuing dialectical process of externalisation, objectivation and internalisation, or of habitualisation, institutionalisation and reification. Nevertheless, scientists argue that these two descriptions are closer to the ontological essence of concrete and abstract phenomena, respectively, than are the much more intuitive aspects listed earlier. Indeed, the only two modes of thinking in which the two perceptions described above appear appropriate at all, are as part of scientific and philosophical though, two perspectives within which materiality and constructedness might indeed be relevant aspects of phenomena.

Which aspect that is perceived from a particular perspective is related to which subset of the properties of a phenomenon that is perceived as relevant from that perspective. Takagaki & Wand [62] note that “[w]hile the properties that a thing possesses are part of the nature of the thing, the properties that are observed in a certain context depend on the purpose of modelling the thing”. It is added that not only the intention and purpose of modelling the thing has this effect: Which properties that are perceived also depends in a more general sense on the users’ and modellers’ perspectives. E.g., at BevCo, the inventory managers would be concerned with informational conveyour-belt properties such as its current allocation, type and time since last maintenance, whereas the shop-floor workers would be concerned with transportation issues such as transportation speed, as well as ergonomic measures. Furthermore, in the Bunge-Wand-Weber model, “classes” reflect sets of “things” that possess a particular set of properties (Weber & Zhang [67]). A phenomenon therefore belongs to different “classes” when perceived from different perspectives, and each “class” corresponds to a particular aspect of the phenomenon. Aspects are therefore similar to “classified things” in the BWW model.

The question of whether classes are defined universally (as implied by the BWW model) or in relation to perspectives is left open here. For our purposes, it suffices to assume that both alternatives are possible, and that perspective holders may structure the world partly by socially constructed and partially by genetically innate — and therefore widely shared — ideas. Checkland [10] notes that “it seems clear that the supposedly “innate” ideas may have two sources. They may indeed be part of the genetic inheritance of mankind, truly innate; or they may be built up as
a result of our experience of the world”. Because the aspect concept encompasses both universal and socially defined classifications, it is robust with respect to conflicting metaphysical positions such as realism, conceptualism and nominalism [2].

Some properties of a phenomenon may be possessed by several aspects of that phenomenon. E.g., in the above example, both the inventory managers and the shop-floor workers might perceive the “reliability” of “conveyor belt #5” as a relevant property, and this property might therefore be possessed both by its informational and by its delayer aspect. Such common properties are the sources of semantical relations between aspects and thereby between perspectives as mentioned in Section 3.1. Some properties also correspond to dependencies between aspects of two or more phenomena. These properties are akin to “mutual properties” of the Bunge-Wand-Weber model [67], and they are the source of relationships between phenomena.

As is implied by that above discussion, a perspective is assumed to comprise as set of aspects of a set of phenomena, along with their corresponding subsets of properties.

3.3. Items, Facets and Facet Models

A facet model comprises descriptions of all the phenomena in the problem domain that are perceived by a user or modeller as relevant to the problems at hand. Each such phenomenon is represented in the facet model as an item. The term “item” has been chosen to avoid connotations with basic modelling constructs used in other modelling languages, such as “entities” and “entity types” of E-R models, and “objects” and “classes” of OOA models. Items do, however, share with objects the possession of an identity.

A facet model comprises descriptions of all the aspects of phenomena in the problem domain that are perceived by a user or modeller as relevant to the problems at hand. Each such aspect is represented in the facet model as a facet. Facets are of items. Every facet is of a single item. Each item must have at least one facet, and it may have several. An item represents a phenomenon and its facets represent aspects of that phenomenon, as is shown in Table 1. Hence more than one aspect of the same phenomenon can be represented simultaneously in the facet model, resulting in an item with one facet for each relevant aspect of the phenomenon.

Apart from its identity, an item is nothing but facets, and all facets are of items. Hence an item and its facets are fundamentally inseparable. The relation between them is analogous to those between a substance and its form, a jar and its shape, or a cube and its sides — one can not be without the other. As a consequence, an item is always created simultaneously with its initial facet(s) and destroyed along with its last one(s). In particular, the analogy of a cube and its sides is useful, and a (hyper-)cube with sides representing facets is an appealing metaphor of and visualisation technique for items.

<table>
<thead>
<tr>
<th>The problem domain</th>
<th>The facet model</th>
</tr>
</thead>
<tbody>
<tr>
<td>phenomena</td>
<td>items</td>
</tr>
<tr>
<td>aspects</td>
<td>composite facets</td>
</tr>
<tr>
<td>properties</td>
<td>primitive facets</td>
</tr>
<tr>
<td>dependencies</td>
<td>links</td>
</tr>
<tr>
<td>perspectives</td>
<td>views</td>
</tr>
</tbody>
</table>

Table 1: The Relation Between the Real-World Problem Domain and Some Fundamental Facet-Modelling Concepts. The various concepts of this table are explained throughout Sections 3 and 4.

Example 5 Figure 1 depicts a cube visualisation of the Conveyor belt #5 item from a facet model of BevCo. The front corresponds to an information facet, the right side to a transporter facet, and the top to an actor facet. Subsequent sections will discuss the types, structure and content of facets.
A facet model is defined as a non-empty set of facets. By definition facets are of items, so a facet model may alternatively be regarded as a non-empty set of items. The former definition has been chosen as primary, since it defines facet models in terms of their epistemology rather than through ontology, as will be discussed in Section 3.10. Also, it is more consistent with the definition of facet languages that will be given in the next section. Figure 2 depicts the relationships between items, facets and facet models using the Enhanced-ER (EER) model [19].

3.4. Instance Facets, Type Facets and Metatypes

Facets are either instances or they define the types of other facets. Type facets define the structure and content of instance facets, just as the types of conventional programming languages define the structure and content of variables. An instance facet replicates the structure of its type facet and instantiates the content types defined within that structure. All facets are typed, and it follows that even type facets must be instances of metatypes. A facet-modelling language, or facet language for short, is therefore preliminarily defined as a set of facet types and metatypes. A facet model of a particular facet language is a set of instance and type facets which are instances of the facet types and metatypes in the language (possibly indirectly).

Example 6 At BevCo, the production workers see conveyor belt #5 as a transporter, similar to a "flow" in a physical DFD. Hence the Conveyor belt #5 item has a transporter instance facet. The type of the transporter instance facet is defined by the particular facet language used. On the other hand, the inventory managers see conveyor belt #5 as a piece of information, similar to an "entity instance" in an E-R model. Therefore, the Conveyor belt #5 item also has an information instance facet.

However, while workers might feel most comfortable with describing their workplace in terms of concrete, individual phenomena, inventory managers might be more concerned with characteristics of all the conveyor belts at BevCo in general, and in particular with which kinds of information

---

1For the time being, the terms “facet types” and “metatypes” are used to define facet languages instead of “type facets” and “metatype facets”, as will be described in Section 3.6. Until then, consider “facet type” and “type facet” as different terms which denote the same concept. Less importantly, “metatype” and “metatype facet” also denote the same concept.
that are needed about them. Therefore, a Conveyour belt item could also be included in the facet model to represent those aspects and properties of conveyour belts in general that are perceived as relevant by the stakeholders at BevCo. For instance, it would have an information type facet to represent the kinds of information about conveyour belts that are needed for equipment maintenance. The information type facet is similar to an “entity type” in an E-R model. It is an instance of an information metatype which is defined at the language level, and at the same time it defines the type of the information instance facet of the Conveyour belt #5 item. In this way, the information instance facet is an indirect instance of the information metatype, as depicted in Figure 3a. Of course, coloring or some other visual cue would be used to distinguish between instance facets, type facets and metatypes when the resulting model is visualised.

The Conveyour belt item could also have had a transporter type facet to represent the general characteristics of conveyour-belt transporters at BevCo. In that case, the facet language would have had to provide a transporter metatype. In turn, the transporter instance facet of the Conveyour belt #5 item would be an instance of the Conveyour belt item’s transporter type facet.

In the above example, conveyour belt #5 is an individual phenomenon whereas the conveyour belt phenomenon was categorical, and they were represented by an individual and a categorical item, respectively. However, the facet modelling approach does not treat individual and categorical items any differently from one another, and no relationship is defined between categorical and individual items to reflect which individual problem domain phenomena that belong to which categorical ones. All instantiations are between facets and not between items. However, whether a phenomenon is categorical or individual may be regarded as one of its relevant aspects (or properties), and as such it can be represented as a facet of the corresponding item. This kind of facet can also be used to reflect which individual problem domain phenomena that belong to
which categorical ones. In this way, distinctions between categorical and individual items, and management of relationships between them, is relegated to the language definition level. Although this makes facet language definition more complex, it increases the flexibility of the approach as more practical experience is gathered from applying it to real problems.

Accordingly, the restrictions on facet instantiation have deliberately been kept to a minimum, and, in principle, the same item may comprise both facets which define types of other facets and some which do not. E.g., a Soda production log item with an information preserver (or store) instance facet, might also have an information type facet to represent the type of content of the store.

Although the approach is flexible, a particular facet language may restrict the range of possible instantiations, e.g., to facets of different items, and the restrictions may also distinguish between facets of categorical and individual items. These restrictions may be specified in terms of the facets mentioned above which reflect how individual items are grouped by categorical ones. In particular, they may specify which of the type facets of the categorical item that define mandatory facet instances of the individual items, and which that are optional. Also, they may specify whether an individual item may have instance facets of other types than those specified by the categorical item. Relegating facet instantiation rules to the language level in this manner keeps the facet modelling approach flexible, and broadens the range of problem domains, perspectives and problems at hand it can potentially cover.

Another reason for keeping the instantiation rules flexible at this stage is that instantiation means different things contingent on the problem domain and the problems at hand. For instance, a model of actor categories and the communication structure between them could either be instantiated *populatively* into a model of individual actors and the communication structure between them, or *temporarily* into a scenario describing a particular sequence of communicative exchanges between actor categories. Defining instantiation at the facet language level makes it possible to tailor it to specific needs, and to use several types of instantiation within the same model. Of course, the increased flexibility of the approach carries with it a danger of defining facet languages which are incomprehensible. It is therefore likely that more restrictions will be built into the approach later.

Figure 4 depicts the relationships between instance facets, type facets, metatypes and facet languages. For the time being, metatypes are not facets, but this will change in Section 3.6. Also note that facets type and metatypes may be “defined-by” several languages, which would correspond to sharing and reuse of facet types and metatypes as will be discussed in Section 3.6.

![Fig. 4: The Relationships Between Instance Facets, Type Facets and Metatypes.](image-url)
3.5. Composite and Primitive Facets and Subfacets

Facets are intended to represent numerous and highly diverse kinds of aspects, ranging from the informal to the formal, from the soft to the hard, from the general to the special, from the whole to its parts, etc. A powerful and flexible mechanism is therefore needed to accurately specify the structure and content types of a particular kind of facets, just as the type declarations of conventional programming languages do for a particular kind of variables. As already mentioned, the structure and content types of instance and type facets are defined by type facets and metatypes, respectively. Since an instance facet replicates the structure of its type facet and instantiates the content types defined within that structure, it follows that the structure and content of type facets and metatypes must be similar both to one another and to instance facets. This typing system is inspired by Telos [42, 40, 34], which supports an arbitrary number of instantiation levels. Currently, however, the facet modelling approach is restricted to four levels; a single metatypetype (which types the metatypes), the metatypes, the type facets, and the instance facets. This restriction has been introduced to limit the complexity of the approach as more practical experience is gathered, and it may be removed later. Also, Telos provides specialisation in addition to instantiation, which might sometimes make facet models more intuitive. E.g., in Example 6, the facet language might define a transporter type facet, which could be specialised into a conveyor belt transporter type facet of the categorical Conveyor belt item. It could then be specialised into a conveyor belt transporter instance facet of the Conveyor belt #5 item. Introducing specialisation is another future possibility. For simplicity, the following discussion will only discuss the relation between type and instances facets. Similar relations exist between the other levels in the instantiation hierarchy.

A type facet is either primitive or composite. Since instance facets replicate the structure of type facets, an instance facet is either primitive or composite also. A primitive type facet defines a primitive instance facet whose value is of a primitive type, such as an integer, boolean, enumeration or free text. In particular, free-text facets are useful for representing less-formal aspects such as the goals and purposes of phenomena early in problem analysis. Primitive instance facets represent properties, and are similar to what is commonly called “attribute values” in many conventional conceptual modelling languages. Accordingly, primitive type facets are similar to “attributes.” A composite type facet defines a composite instance facet whose value is a tuple, set, list or other collection of other instance facets, called instance subfacets. Type subfacets of the composite type facet recursively define the types of each instance subfacet. Hence, a composite instance facet replicates the structure of its composite type facet, whereas each primitive instance facet instantiates the primitive content types defined within that structure, as was explained in Section 3.4.

Although items have facets and facets have subfacets that are themselves facets, a fundamental distinction remains between the two. Items represent problem domain phenomena per se, whereas facets represent aspects of those phenomena as they are perceived by a user or modeller. What the subfacet construct provides is a way of representing that an aspect of some phenomenon can be perceived at different levels of granularity. At the most detailed level of perception, it is perceived as a composition of properties which are represented as primitive subfacets, as is depicted in Table 1. Another way of clarifying this relation, is to regard items as representing the material things and social constructs (i.e., concrete and abstract phenomena) that the problem domain model is about, while instance facets represent facts about these things and constructs. Facet types and metatypes accordingly classify the relevant types of facts.

Figure 5 depicts the relationships between composite and primitive facets, primitive types and subfacets. Note that this distinction is orthogonal to the distinction made in Figure 4 between instance facets, type facets and metatypes. It would have been possible make the facet-subfacet specialisation disjunctive by introducing “top-level facet” as an explicit concept. In the next section, Figure 7 will extend Figure 4 with metametatypes.
Example 7 A very simple transformer type facet could be defined as composite and including

- a primitive subfacet specifying that transformer instance facets have a name;
- a composite subfacet specifying that the instances have input ports; and
- a composite subfacet specifying that the instances have output ports,

where the transformer instance subfacets’ ports indicate where they receive inputs from and deliver outputs to. These two latter composite subfacets are in turn sets of individual (input and output) port facets, as is shown in Figure 6a. In this and some subsequent figures, facets are shown as rectangles and facet-subfacet relations are indicated by arrows. Double-headed arrows indicate sets. Figure 6b visualises a transformer facet using a “process”-symbol from DFD modelling. □
3.6. Facet Languages and Initial Items

In Section 3.4 and Figure 3a, a facet language was preliminarily defined as a set of facet types and metatypes. An alternative way of defining a facet language is to introduce an initial item with exactly these types and metatypes as facets, as is shown in Figure 3b. Indeed, even the metatetatype can be a facet of this item\footnote{Since it is likely that facet instantiation will eventually be restricted to facets of different items, this would be an exception to that rule.}, and its facets become “part-of” all facet models which “is-of” the language it defines. One appealing consequence is that the facet models become more self-contained because they now define an essential part of their own language.

The initial item can be regarded as a convenient and simplifying “trick”, or it can be interpreted as a representation of the facet language itself regarded as an abstract phenomenon. However, when it is regarded as a phenomenon, the modelling language is different from the other phenomena represented in the facet model because it is part of the development world, as opposed to the subject and usage worlds of IS development [34, 41]. Should facet modelling be applied to the development world in addition to these worlds, the initial item would nicely provide one of the necessary bridges between them.

Example 8 In the case of problem analysis at BevCo, the managers want a view of the organisation with emphasis on information about the organisation’s equipment, and how that equipment manipulates information and materials. From their perspective, an appropriate (but trivial) facet language would integrate two of the modelling orientations commonly applied in structured analysis: E-R models, which comprise entity types and relationship types between them, and physical DFDs, which consist of processes, stores, flows and external entities. As was indicated in Example 6, for this particular problem, entity types and relationship types correspond to type facets, and they are useful for describing aspects of phenomenon categories rather than aspects of individual phenomena. On the other hand, DFD concepts correspond to instance facets which are useful at the individual level, i.e., the language represents particular processes, stores, flows and external entities. However, since there are many different kinds and interpretations of DFDs and their relationships to E-R models, such facets might have been defined and used differently for other purposes. The Soda production log, which was an individual item with an information type facet inspired by entity types, is a case in point.

In the BevCo example, the initial item would have metatype facets for information type facets (inspired by entity types) and relationship type facets, as well as type facets for transformer instance facets (inspired by processes and already discussed in Example 7), preserver instance facets (inspired by stores), transporter instance facets (inspired by flows) and external entity instance facets. However, the shop-floor workers might find it hard to recognise themselves in the resulting models, as actors are only coarsely represented in DFDs as external entities. To cover their perspectives also, additional type facets would be provided for actor instance facets and role instance facets\footnote{It is likely that the role instance facets and external entity instance facets would have been merged by this language.}, e.g., inspired by the extended actor modelling language [36] of the PPP environment for IS development [26].

The initial item for the BevCo case would have type and metatype facets for each of the above instance and type facets. In addition, Example 10, which considers the process managers’ view of the production process, will use type facets inspired by timed Petri-net models (e.g., [1]) to represent production delays. Example 11 of Section 3.8 will demonstrate how the facet language also specifies link types between the various type and metatype facets.

The example must not be taken to suggest that new type and metatype facets must be defined from scratch for every new analysis problem. Instead, comprehensive libraries of predefined facets should eventually become available so that an appropriate initial item can be assembled quickly and easily for the problem domain and the problems at hand. In addition, completely new type and metatype facets can be defined and added to the initial item by instantiation of an existing metatype facet or of the metatetatype facet. In this way, the facet modelling approach becomes extensible, and existing and new type and metatype facets can be integrated semantically through
subfacet sharing, which will be introduced in the next section. As a result, the facet language will grow as the result of continuous language extensions and of practical application, evaluation and improvement, providing a larger library of type and metatype facets to reuse from later.

To summarise, some benefits of introducing initial items are that:

- Facet model portability — and thereby reuseability — is increased because every model includes an essential part of its own facet language definition through the initial item. This is important, because different development projects would tend to use more or less different facet languages, which would otherwise have made reuse more difficult.

- Adding new type or metatype facets to the Initial item is a simple and intuitive way to support extension of facet languages. Managing a facet language, and extending it in particular, thereby becomes similar to modifying items, and it can be handled by many of the same language and user-interface mechanisms.

- As will be explained in Section 3.7, defining shared subfacets between the type and metatype facets of the initial item is a simple and intuitive way to represent and manage semantical relations between facets. In this manner, semantical relations can be managed at the language definition level using constructs and mechanisms that are already part of the approach.

- Detection and resolution of inconsistencies between type and metatype facets reused from different languages become special cases of detection and resolution of inconsistencies between facets of the same item.

Figure 7 is a modification and an extension of Figure 4 to account for metametatypes and the initial item. In particular, both metatypes and the metametatype, have now become facets. Again, this distinction is orthogonal to the distinction made in Figure 5 between composite and primitive facets and subfacets. Therefore, there are composite and primitive facets and subfacets both at the metametatype, metatype, type and instance levels. If, at a later stage, additional restrictions on facet instantiations are built into the approach, some of them may be representable by decomposing the "has/is-of" relationship between items and facets to show, e.g., that the initial item may not have instance facets.

---

![Diagram](image-url)  
**Fig. 7**: Modification and Extension of Figure 4 to Account for the Metametatype Facet and the Initial Item. Again, this distinction is orthogonal to the distinction made in Figure 5 between composite and primitive facets and subfacets.
3.7. Shared Subfacets

Previous sections have established a framework of initial, categorical and individual items with typed facets. However, no way of establishing relationships between facets has been presented so far. As mentioned in the basic assumptions of Section 3.2, some properties of a phenomenon may be possessed by several aspects of that phenomenon or of different phenomena. Such common properties were the sources of semantical relations between aspects of the same phenomenon, and of dependencies between aspects of different phenomena, respectively. The two types of common properties are represented by two types of common subfacets in facet models. Whereas this section discusses shared subfacets of facets of the same item, the next section will discuss links between facets of different items.

A shared subfacet is common to several composite facets of the same item at the same time. As a consequence, changing the value of the subfacet in the context of one of its owner facets would implicitly and instantaneously appear as a change of the subfacet in the context of its other owner facets also. Subfacets which are not shared are called private subfacets, or just subfacets for short. Shared subfacets are always common to facets of the same item. Figure 8 depicts the relationships between private subfacets, shared subfacets and links (which will be introduced in the next section), and between composite and primitive facets.

Example 9 Figure 9 depicts simple subfacet sharing between the transformer type facet of Example 7 and the relationship type facet mentioned in Example 8. The two facets share a name subfacet, meaning that whenever an item has both a transformer and a relationship type facet, those two facets will share their name subfacet. Since instance facets replicate the structures of their type facets, this also implies that name subfacets will be shared between transformer and relationship instance facets. Note that subfacet sharing does not imply that an item, e.g., with a transformer facet, must also have a relationship facet, or vice versa.

In Figure 9, the relationship type facet is composite and includes

• a primitive shared subfacet specifying that relationship instance facets have names.

Accordingly, the transformer type facet would still be composite and including

• a primitive shared subfacet specifying that transformer instance facets have names.
As a result, changing the transformer instance facet’s name would implicitly and instantaneously appear also as a change of the relationship instance facet’s name. The dot-framed box of Figure 9 indicates that the subfacet is shared through arrows from the transformer type name and the relationship type name boxes. Hence the single, shared subfacet is drawn as three distinct boxes in the figure. The advantage of this notation is that the shared subfacet can have different labels in the context of different owner facets.

Note that the facet modelling approach does not specify that all facets of the same item should have the same name, but only offers this as a possibility through the shared subfacet mechanism. The decision of which subfacets to share is left to the designers, extenders and maintainers of the facet modelling language used.

In addition to making changes made to a subfacet immediately apparent in the context of all its owner facets, shared subfacets provide a powerful tool for avoiding inconsistencies between facets. Specifically, two or more primitive instance facets are inconsistent if they simultaneously represent the same property by more than one value. Shared subfacets make it possible to avoid this kind of inconsistency completely, by making the relationship between properties and primitive facets one-to-one when designing a facet language. However, this is only an option that should be used to the extent that it is considered favourable by the language designers, extenders and maintainers.

**Example 10** At BevCo, the capsulier robot at the end of conveyer belt #5 is to be upgraded. From the perspective of BevCo’s process managers, this modification implies deleting the conveyer instance facet of the Old capsulier item, and creating a New capsulier item with an instance facet that represents the new robot as a delay. Figure 10 shows that this results in an inconsistent facet model: Whereas process managers perceive the production process as a timed Petri-net model of production delays, shop-floor workers see it as a system of transformers and conveyers. These two perspectives are represented in Figure 10a as linked timed Petri net-inspired instance facets and linked DFD-inspired instance facets, respectively. Modifying this representation only from the process managers’ perspective results in an inconsistent model, as is shown in Figure 10b. The reason is that the transporter instance facet of the Conveyer belt #5 item remains linked on the output side to the transformer instance facet of the Old capsulier item, whereas its delay instance facet becomes linked on the output side to a facet of the New capsulier item.

The underlying cause of this inconsistency is that the transporter and delay aspects of the conveyer belt phenomenon take their inputs from and deliver their outputs to the same places. This means that they are semantically related, and therefore the possibility of inconsistency can be removed by subfacet sharing between transporter and delayer facets: The transporter type facet can be defined as a triple of an input port, a name and an output port, while the delayer type facet comprises a set of input ports, a name, an average delay time and a set of output ports. According to these definitions, the output ports of the two type facets are semantically related because they reflect the same physical location to which the conveyer belt delivers its outputs, and the above inconsistency was made possible because this was not accounted for in their definitions. The dot-
Fig. 10: Upgrading the Capsuler at BevCo: Whereas process managers perceive production as a timed petri-net model of production delays, shop-floor workers see it as a system of transformers and transporters. Replacing the capsuling robot only from the process managers' perspective results in an inconsistent model.

Fig. 11: Subfacet Sharing Between the Transportation and Delayer Instance Facets of Figure 10 on to Avoid Inconsistencies.

framed box of Figure 11 shows how the set of output ports of the Conveyor belt #5 item's delayer instance facet is defined as identical to the output port of the transporter instance facet by subfacet sharing†. Hence modifying the former would implicitly modify the latter and vice versa, thereby avoiding conflict. Figure 11 also depicts subfacet sharing on the input side.

As indicated by the above examples, well-managed subfacet sharing is a powerful tool for language definers and modellers because it avoids many inconsistencies by encouraging clean management of semantical relations between aspects. However, at the same time its use is potentially dangerous when carried too far, as it may lead to incomprehensible language semantics, and unexpected and unwanted cascading of updates between the perspectives of different users and modellers. In particular, subfacet sharing has consequences for how type and metatype facets must be defined. Without further experiments with formal type and metatype facets and larger examples, however, it is difficult to assess how strictly such restrictions should be imposed, and the issue remains an interesting topic for further work.

Subfacet sharing obviously can not resolve all inconsistency problems when analysing a complex problem domain. As pointed out by Pohl [52], the degree of agreement reached on a specification among stakeholders is one of three dimensions along which the requirements engineering process should proceed. Therefore, the facet modelling approach should eventually support:

†Of course, this would implicitly limit the number of input and output ports of the delayer to one.
Facet Modelling

- representation, management and analysis of inconsistencies between instance facets, and
- identification and resolution of those inconsistencies as analysis progresses.

As for representation, management and analysis, it is presently possible to represent inconsistent instance facets through multiple instances of the same type facet for an item. Management and analysis problems are currently being addressed within viewpoints research (e.g., [43, 21]), and hopefully, results from that area will also be useful in relation to facet modelling.

As for identification and resolution of inconsistencies, facet models nicely and explicitly capture that different instance facets represent distinct aspects of (or facts about) the same phenomenon when seen from different perspectives. Therefore, the approach localises facts to items which represent the phenomena they are about, and thereby makes a large class of inconsistencies much easier to detect. Again, results from viewpoints research may also be useful (e.g., [20, 30]). It is repeated that not all inconsistencies and conflicts are easily identifiable and resolvable in this way: Facet modelling only simplifies resolution of the ones that were already resolvable in the first place.

3.8. Links

Shared subfacets are always common to facets of the same item. It is also possible to conceive a subfacet which is common to two or more facets of different items. This class of common subfacets are called link subfacets, or links for short. Links play an important role in facet models because they represent the dependencies between aspects of distinct problem domain phenomena that were mentioned in Section 3.2. As such, they represent properties that are mutual to two or more aspects of different phenomena, and they are similar to “mutual properties” of the Bunge-Wand-Weber model [67]. This is indicated in Table 1. Hence, although the link is essential to the facet-modelling approach, it is not a basic construct, just a special kind of (common) subfacet. Note that links may have subfacets on their own, as may all common subfacets.

In a facet model, the facets of distinct items are related through link subfacets, and only through link subfacets. As a consequence, items in a facet model are related through their links, and only through their links. Since the link subfacet mechanism does not impose any order on the facets it is common to, links are always bi-directional. However, direction may be imposed between linked facets by the semantics of a particular facet language. Link instances represent a particular and important kind of facts about the problem domain, specifically those facts which are about more than one phenomenon.

Example 11 In order to integrate the DFD-inspired type facets of the initial item from Example 8, their definitions should comprise link types between the DFD-inspired type facets to specify, e.g., that transporter instance facets are linked to other DFD-inspired instance facets both at the “input” and “output” sides. Similarly, the ER-inspired metatype facets would have link metatypes between them to specify that relationship type facets connect information type facets.

Figure 8 depicted the relationships between private subfacets, shared subfacets and links, and between composite and primitive facets. The two distinctions are orthogonal both to one another, and to the distinction made in Figure 7 between metatatype, metatype, type and instance facets. Hence every facet can be described along three independent dimensions:

- as the single metatatype, a metatype, a type or an instance facet;
- as a primitive or a composite facet; and
- as a “top-level facet”, a private subfacet, a shared subfacet or a link.

Also note how the cardinalities of the “has/is-of” relationship between items and facets have changed in Figure 8 in comparison to Figure 5, since a link “is-of” several items. At the cost of making Figure 8 considerably more complex, it is possible to show how the one-to-many relationship from facet to item only applies to links, and that the relationships from all the other facet subtypes to links are still one-to-one.

1With the single exception that the metatatype facet probably cannot have any meaningful link.
Example 12 The transformer type facet of Example 7 comprised two composite subfacets specifying that transformer type and instance facets have input and output ports, while Example 10 defined a transporter type facet as a triple of an input port, a name, and an output port. As indicated by the doubly dot-framed box of Figure 12, the input ports of the transformer type facet are now linked to the output ports of the transporter type facet, and vice versa. The example also illustrates directed links, since the two links may be interpreted semantically to have direction from the transporter to the transformer and from the transformer to the transporter, respectively.

In Figure 12, the transporter and transformer type facets did not share subfacets on the input nor on the output side. As a consequence, inconsistencies such as those described in Example 10 could potentially arise. Also, transporter instance facets could only be linked to transformer instance facets, and not to other transporter instance facets, and accordingly for transformers. Figure 13 depicts a more flexible arrangement which integrates the type facet definitions for transporters, transformers and delays from Figures 11 and 12. Here, the input port subfacets, as well as the output port subfacets, are shared, and the single shared input port subfacet is linked to the single shared output port subfacet.
3.9. Type and Metatype Facet Outlines

As already established, the type and metatype facets that are available to the modeller will depend on the facet language used and how it is extended throughout problem analysis. Rather than defining a particular facet language in detail, the aim of this paper is therefore to establish and present the underlying idea of facet modelling. To indicate its versatility, it is nevertheless useful to outline some of the possible categories of type and metatype facets. The following list is based on [49]. [47] outlines a facet language reflecting the framework of Zachman [70] and Sowa & Zachman [61].

Informational type and metatype facets would define instance facets for representing the material properties and the data content of an item. Since one major effect of behavioural instance facets is to modify the informational instance facets, they will be important when formally defining the dynamics of a facet language.

Existential type and metatype facets would define instance facets representing an item’s duration, its physical location and movements, as well as whether its represents a concrete or an abstract problem-domain phenomenon.

Behavioural type and metatype facets would define instance facets specifying the dynamic behaviour of items. Not surprisingly, they will have complex structures compared to other categories of type and metatype facets. The authors have previously discussed behavioural descriptions of concrete problem domains [48, 30], and a first attempt to embed behavioural type and metatype facets within the facet-modelling approach has been presented in [51]. In general, behavioural instance facets must be composite and they must contain links to the type facets whose instances they manipulate, typically to informational and existential ones. More specifically, e.g., a transporter instance facet defines a manipulation of some items’ location instance facets, whereas a transformer instance facet might define a manipulation of some other items’ material-property and data-content instance facets. A preserver instance facet, on the other hand, expresses a restriction on manipulating any of some items’ instance facets for some duration.

Organisational type and metatype facets would define instance facets to represent aspects such as actors, agents, roles, organisational units, responsibilities, and capabilities.

Finally, teleological type and metatype facets would define instance facets to represent the purpose of, goals of and the requirements placed on items. Furthermore, these instance facets should be linked to one another to support traceability (e.g., [25]), by representing and maintaining relationships from existing requirements, through other requirements, and back to their originate statements of problems or opportunities (pre-requirements traceability). By embedding requirements traces within the facet modelling language, traceability can be provided as an inherent part of a facet model.

This concludes our brief outline of possible categories of type and metatype facets. It can be concluded that the essential semantics of a facet language reside in the definitions of its type and metatype facets, and that there are many possible choices both of types and metatypes and of semantics. The authors emphasise that the examples outlined here are not in any sense promoted as “better” than other alternatives.

Figure 14 depicts how the set of type and metatype facets actually used during problem analysis may not be a subset of the set of available predefined type and metatype facets due to extensions made during analysis. In the extreme case, it is even possible for a facet model to be created from user-defined types and metatypes only. Finally, the horizontal line of Figure 14 indicates the distinction between formally defined and informal type and metatype facets.

3.10. Ontology and Epistemology

The distinction between facets and items reflects the classical philosophical distinction between ontology and epistemology [2]: Whereas ontology is the field of metaphysics dealing with the nature of existence, epistemology is concerned the study of the nature of knowledge and how knowledge is justified. It follows that items form the ontology of facet modelling because they represent the phenomena that exist in the problem domain and are perceived as relevant to the problems at hand [47]. Facets, accordingly, form the epistemology of the approach, as they represent the known
facts about each of those phenomena (apart from their existence and relevance to the problem at hand) and about dependencies between them.

In the terminology of Searle [54], concrete phenomena correspond to material things which are **ontologically objective**, whereas abstract phenomena correspond to social constructs which are **ontologically subjective**. Facets of the corresponding items in turn represent the facts that are known about those things and constructs. Section 3.9 pointed out that the essential semantic content of a facet language resided in its type and metatype facets. In a similar vein, it can now be concluded that the essential semantic content of a facet model resides in the facets of its items, underlining the choice made in Section 3.3 to define a facet language as a set of facets rather than of items. Apart from its facets, an item is nothing more than an identity.

Searle also distinguishes between facts that are **epistemically objective** and **epistemically subjective**. From this point of view, inconsistent instance facets may be introduced into a facet model for two major reasons:

- incorrect representation of epistemically objective facts; and
- correct representation of epistemically subjective facts.

**Example 13** The shift from conventional oriented modelling to facet modelling is depicted in Figure 15, both parts of which depict two human subjects perceiving a problem domain phenomenon, as usual conveyour belt #5 at BevCo. Because of their different perspectives, the subjects perceive the phenomenon differently, and consequently disagree on how to represent it. In Figure 15a, a conventional oriented language — in this case DFD — forces the subjects to create two unrelated representations of the phenomenon, a transformer and a preserver. The resulting redundancy may imply inconsistencies which are hard to detect, and even if they are detected, managing them safely will require specific naming conventions or extensions to the oriented modelling languages used.

However, in this example, the subjects would probably easily have managed to agree on the actual existence of conveyour belt #5 within BevCo. The problem of redundant representation resulted from the incapacity of the oriented modelling language to capture this agreement. A fundamental idea behind facet modelling is therefore the distinction between agreeing on the existence of a problem-domain phenomenon and representing how distinct subjects perceive it. Figure 15b demonstrates how the agreed-upon conveyour belt is represented as an item, to which each subject may then add the facet(s) they find most appropriate. In short, the introduction of a single item captures the ontological agreement of existence, whereas the introduction of multiple facets captures the epistemological disagreement of perception. As a consequence, the two aspects are simultaneously represented close to one another in a single facet model.

[Diagram of types and metatypes]
Section 5.4 will discuss differences between facet modelling and the modelling languages commonly associated with object-oriented analysis.

4. VISUALISATION

Visual representation has many advantages over textual representation, both when it comes to comprehension (e.g., in providing at-a-glance overviews) and expressive economy [27, 58]. Therefore, many modelling languages have been designed with particular two-dimensional visual representations in mind. In order to fully utilise contemporary IS development tools, facet models have instead been designed as flexible, visualisable structures that can be visualised in numerous ways when needed, depending on the problems at hand.

Large facet models will contain many items, even more facets, and still more links between them. As a consequence, they quickly become impossible to display nicely in the two or three dimensions provided by a computer display. This is not really a unique concern in facet modelling — all comprehensive models of complex problem domains quickly become too complex to visualise all at once, and visualisation and filtering techniques are needed [55, 56]. However, the inherent complexity of the facet-modelling approach makes such visualisation and filtering techniques all the more important. Although no visualisation tool has yet been built for the facet-modelling approach, it is important to discuss visualisation here to demonstrate the feasibility of practical facet modelling. In fact, it is suggested that facet modelling may have an advantage in providing greater flexibility for views, since no particular view is promoted by the approach as such. The ideas presented here are partly inspired by [56].

Visualisation may be addressed according to two different principles: 1) browsing and 2) associatively selected views. In both cases, visualisation relies on filtering, i.e., on not showing the entire model at once. Whereas view-selection makes filtering explicit, browsing makes filtering implicit by the introduction of a user-changeable “current point of view”. Model elements which are structurally “far away” from the current point of view, will be vaguely or not at all visible. For associatively selected views, on the other hand, the user or modeller will describe through some suitable interface what should be included in and what should be excluded from the view.

Considering browsing first, although the screen is two-dimensional, the user or modeller can be provided with six virtual working directions: up, down, left, right, in, out, giving a three-dimensional feeling. As long as items have a limited number of facets, this can be utilised to create a versatile interface for browsing. The cube metaphor mentioned in Section 3.3 and used throughout the paper is a good starting point for browsing facet models. Figure 1a showed different facets as sides of a cube which represented an item. The three remaining sides were vacant for other facets of interest. A point-and-click interface can be provided to rotate the cube in three dimensions so that additional facets can be depicted. Figure 16 shows the result of rotating the cube of Figure 1a (a), so that the delay facet of Figure 10 becomes visible at the expense of the actor facet. Of course, this does not restrict the number of facets of each item to six.
metaphor can easily be regarded as a three dimensional visualisation of an n-dimensional item cube with any number of facets.

A simple and raw browsing interface for experienced facet modellers can be built around item cubes in a virtual three-dimensional space on-screen. An example of three-dimensional visualisation has already been given in Figure 10. Considering the facet depicted on the front side of an item cube to be “current” by convention, the modeller can turn his or her attention to new facets of interest by rotating each of the cubes in this figure. When an item cube is rotated, and a new facet becomes “current”, other item cubes may be automatically rotated by the modelling tool to reflect this new focus of attention. Also, other items may appear or disappear as a result of the rotation to indicate which other items that are linked to the new “current” facet through some facet. Zooming into a particular side of a cube, a decomposition of the corresponding facet will become visible, and zooming away from it its composition will become visible. Since the cube visualisation is virtual, it is not even restricted to six sides. The resulting item cubes do not follow the expected behaviour of physical cubes, but they may nevertheless be useful for experienced modellers as a way of rapidly exploring and maneuvering within facet models. At the same time, the principle of showing nodes that are far away from the current point of view vaguely or not at all means that the number of links that have to be updated remains manageable (although large).

However, browsing structures like the ones in Figure 10 is probably not suitable for novice modellers when the facet models become non-trivial, and the resulting visualisations quickly become more confusing than clarifying. The alternative is associatively selected views of the facet model. Usually, views will be two-dimensional, but even three-dimensional views would constitute a simplification compared to browsing. Filtering of views can be done by

- stating which *type and metatype facets* that should be included in and/or excluded from the view and/or
- stating which *items* that should be included in and/or excluded from the view.

Of course, it should be possible to combine type and metatype facet and item filtering, as well as to combine *inclusive* (what should be included in the view) and *exclusive* (what should be excluded from the view) filtering. In the example of Figure 10, it is easy to see how the models can be filtered into a view which only depicts transformers and transporters, or into a view which only depicts delayers. Although it is not evident from Figure 10, an even more interesting use of filters is to extract views which comprise facets and links between facets which are not conventionally depicted together. In the BevCo example, a view might be extracted which showed actor and transformer instance facets, as well as information type facets and link types depicting responsibility relations.

The filtering idea is particularly attractive in connection with automatic diagram layout techniques [63]. For larger models, it may be difficult to find the right view in only a single filtering step. Iterative filtering should therefore be supported. Combining filtering and browsing might
also be a good idea in many cases, e.g., browsing to a particular point of attention to select a particular view from which browsing may later be continued.

5. FACET MODELLING IN CONTEXT

Facet modelling, as described here and in [49, 51, 47], is not an isolated idea. Several recent results in the IS field and neighbouring fields have been motivated by many of the same problems as facet modelling and, as a consequence, exhibit similarities. These developments complement both one another and the facet modelling approach. Together, they support the motivation behind facet modelling by demonstrating that similar problems have called for similar solutions in slightly different, but related, areas. To place some of the implications of the facet modelling approach in perspective, the most important of them will be presented briefly in this section.

5.1. Multi-Perspective Methodologies

The idea that a problem domain should be represented from several perspectives simultaneously during problem analysis is old in the IS development field, and is reflected in the framework presented by Olle et al. [46] as well as in numerous existing IS development methodologies, e.g., Yourdon Inc.'s SA methodology [69], as well as UML [7] and Fusion [13]. These multi-perspective methodologies provide multiple partial modelling languages of which more than one is commonly to be used in the same IS development phases, for the same modelling worlds [34, 41] and at similar modelling levels [31, 70, 61]. Each modelling language defines a set of modelling constructs (e.g., transformations, classes or states) which typify model elements of partial models. Cross-model references across partial model boundaries and languages are provided to identify elements that represent the same problem domain phenomenon.

<table>
<thead>
<tr>
<th>The problem domain</th>
<th>Multi-perspective modelling</th>
<th>Facet modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>phenomena</td>
<td>cross-model references</td>
<td>items</td>
</tr>
<tr>
<td>aspects</td>
<td>model elements</td>
<td>instance and type facets</td>
</tr>
<tr>
<td>kinds of aspects</td>
<td>modelling constructs</td>
<td>type and metatype facets</td>
</tr>
<tr>
<td>perspectives</td>
<td>partial models</td>
<td>views</td>
</tr>
</tbody>
</table>

Table 2: The Relation Between Multi-Perspective Modelling and Facet-Modelling Constructs.

As is shown in Table 2, the partial modelling language constructs of multi-perspective modelling languages play the part of type and metatype facets, while partial model elements correspond to instance and type facets, and cross-model references are akin to items. The partial models themselves are analogous to the views discussed in Section 4.

The motivation behind multi-perspective methodologies is similar to that behind facet modelling. In particular, the two share a concern with expressiveness and with the importance of co-representing multiple perspectives on the problem domain. However, the two differ in emphasis. Where multi-perspective methodologies approaches the problem pragmatically and bottom-up by choosing a battery of otherwise unrelated existing modelling languages and establishing cross-model references between them, facet modelling is an attempt to attack the multi-perspective problem conceptually and top-down. As such, the approach offers several potential improvements through the provision of a semantically accurate modelling concept (the item) to replace cross-model references; the ability to visualise new perspectives (views) at any time by filtering of the facet model; the ability to extend the space of possible perspectives (views) at any time by introduction of new type and metatype facets; as well as the item-cube metaphor and the corresponding visualisation. In addition, facet modelling offers a fundamental conception of the nature of multi-perspective models through the dichotomies of phenomena and aspects, existence and perception, as well as ontology and epistemology.
5.2 Viewpoints

Viewpoints research [37, 35, 43] regards perspectives (or views) on an artifact or on a system as partial descriptions which differ because project participants (or agents) have different roles and responsibilities assigned to them [21]. A viewpoint commonly refers to a view together with the agent that holds it. A fundamental idea behind viewpoints research is that viewpoints should be managed by the method users as first class objects together with, and at the same level as, other development products. In particular, explicit relations must be maintained between views, between views and agents, and between agents.

Whereas facet modelling is concerned with the nature of how different views are related to one another through the underlying real-world phenomena, viewpoints research focuses on the representation and management of such views during practical group work, as well as an integration and resolution of views. Hence, their emphases appear complementary: Facet modelling may provide a conceptual and representational framework for viewpoint-oriented development, and viewpoints research may contribute solutions to many of the potential management and consistency problems caused by richness of the facet modelling approach.

5.3 Metamethodologies

It has become widely accepted that no single IS development methodology or associated modelling language is likely ever to become sufficient for all analysis problems. In particular, this applies to problem analysis and requirements elicitation methods, as has been pointed out by Davis [15]. Flaatten [22] notes that “Since many problems may need solutions in a given project, the ideal methodology should be able to mix and match techniques to develop the various parts of a single system”. One consequence has been the introduction of metamethodologies for IS development which continually assess the current state of development and use it for planning and re-planning subsequent development activities. Examples of metamethodologies are Iivari & Koskela [33] and Iivari’s [31, 32] PI COCO model and Boehm’s spiral model [6]. Both models of the IS development process advocate a pragmatic and flexible approach to IS development aimed at minimising risk.

Facet modelling and metamethodologies are similar in that they are both motivated by the need for flexibility with respect to how problem analysis should be carried out for a particular problem domain. However, their scopes are different. Whereas facet modelling focuses on flexible and extensible modelling, metamethodologies also cover development methodology and possibly tool issues. Hence facet models could provide flexible support for a metamethodology, while the metamethodology in turn could provide methodology support for facet modelling. In particular, a suitable metamethodology would guide:

- the selection among available type and metatype facets,
- the extension of the facet language during problem analysis, as well as
- the facet modelling activity itself.

5.4 Object-Oriented Analysis

One of the most dominant recent trends in the IS field has been the appearance of a number of methodologies and modelling languages for object-oriented problem analysis and systems design. Of particular current interest are the UML [7] and Fusion [13] approaches. Like facet modelling, OOA attempts to represent concrete and abstract problem domain phenomena, as well as phenomenon categories, directly in the problem domain model as objects and classes. Also, most mainstream OOA methodologies support multi-perspective conceptual models (see, e.g., [68]). The partial models are related through cross-model references. In this respect, object-oriented analysis approaches are similar to the multi-perspective methodologies discussed in Section 5.1.

Figure 17 depicts the difference between facet-modelling and OOA at a deeper conceptual level. In Figure 17a, two perceiving subjects are encouraged to reach agreement about the existence of a problem domain phenomenon through an object-oriented representation. However,
the resulting object or class is not well-suited to represent the subjects’ individual perceptions of the phenomenon, except through low-level constructs such as operations, contracts, etc. In Figure 17b, a facet model item has taken the place of the object, and its facets capture the individual perceptions. Comparing Figures 15 and 17, facet modelling can therefore be seen as a synthesis of conventional object-oriented modelling and object-oriented modelling. However, it is also possible to regard object-orientation as another conventional orientation alongside, e.g., E-R modelling and DFDs.

5.5. Multiple Inheritance and Multiple Instantiation

Multiple-inheritance is known from both extended E-R modelling and object-oriented modelling. Using object-oriented terminology, multiple inheritance means that an object class can inherit the characteristics of several other classes in a generalisation hierarchy, in particular their attribute types and methods. If each of these general classes are thought to correspond to a type or metatype facet, the multiply inherited class will in effect simultaneously have an instance or type facet for each of them. Some object-oriented representation languages, e.g., Telos [42, 40, 34], short-cuts multiple inheritance further by multiple instantiation. In Telos, an already instantiated object can be re-instantiated with a new class. In effect, the object acquires the attributes and operations of the new class and simultaneously retains its old one(s).

Multiple instantiation is similar to the facet modelling idea in that it emphasises the need to represent a problem domain phenomenon in several ways — i.e., as belonging to several different classes — simultaneously. However, their emphasis differ, as multiple instantiation is a technique for providing this possibility within object-oriented representations, whereas facet modelling is a way of conceiving multiple perspectives on a problem domain. As such, multiple inheritance and instantiation would be useful both in a formalisation and a representation framework for facet modelling and facet models, and as mentioned in Section 3.5, the typing system of facet modelling is inspired by that of Telos. Also, multiple inheritance and instantiation might be useful techniques for the subsequent design and implementation of information systems that were initially specified in terms of facet models.

5.6. Subject-Oriented Programming

Harrison & Ossher [28] criticise the use of object-oriented programming (OOP) for the development of evolving systems of cooperating applications: As applications become increasingly distributed and interoperable, Harrison & Ossher claim that the same information will more and more often have to be shared between several cooperating applications. Furthermore, these applications will more and more often have been developed for different purposes by different people and will therefore reflect different perspectives on the problem domain. As a result, they also see the shared information differently. According to Harrison & Ossher, this creates maintenance problems
with object-oriented programs, because pure object-orientation provides no support for representing and maintaining multiple usage perspectives on the information. An alternative programming orientation coined subject-oriented programming is introduced to alleviate this.

Although Harrison and Ossher’s paper addresses a problem which is quite distinct from problem analysis, their concern mirrors one of the main ideas behind facet modelling: that of co-representing multiple perspectives on a problem domain. The two ideas can therefore be regarded as isomorphic as both are motivated by a need to equip something with several perspectives on a problem domain. In subject-oriented programming, that something is an already existing suite of software systems. In facet modelling, it is a conceptual model. In both cases, the problem is resolved by a paradigmatic shift from focussing exclusively on problem-domain phenomena per se, towards also representing human subjects’ perceptions of them. Subject-oriented programming may be a natural implementation technique for information systems that have been specified in terms of facet models.

6. CONCLUSION

The paper has explained why clearly oriented models are not anything to strive for early in problem analysis, but rather a source of obstacles on their own. To alleviate this situation, the facet-modelling approach was introduced and a simple facet language was outlined. The paper also discussed visualisation techniques, before facet modelling was placed in a context of recent related ideas and techniques.

Work is currently in progress at the University of Bergen on developing a framework for facet language definition and on defining an initial facet language. Early ideas based on the framework of Zachman [70] and Sowa & Zachman [61] have been presented in [47]. In addition, several case studies are currently in progress in order to evaluate various implications of the facet modelling idea empirically. Early experiences from these studies have influenced the present paper.

6.1. The Facet-Modelling Shift

The facet-modelling approach is a departure from conventional oriented problem analysis languages in several ways:

- From visual models to visualisable structures.
  Instead of designing modelling languages with predetermined two-dimensional visualisations in mind, the facet-modelling approach encourages the definition of languages as richer and more complex structures which can be visualised in a number of ways. Furthermore, the set of available modes of visualisation is not predetermined by the language definitions, as new kinds of views can be created by filtering when needed. This reflects the shift from pen and paper to computer-supported tools as the most important support technology for model creation and maintenance.

- From aspects as model elements to facets of items.
  Whereas conventional oriented modelling languages focus on aspects of phenomena in the problem domain, the facet-modelling approach represents aspects as facets of items representing their underlying phenomena. This extended view has several advantages which were summarised in Section 5.1, such as semantically accurate items replacing cross-model references, introduction of new views at any time, extension of the space of possible views at any time, the item-cube metaphor and visualisation, as well as the conceptual dichotomies of phenomena and aspects, existence and perception, and ontology and epistemology.

- From items as objects to items with facets.
  Whereas object-oriented modelling languages focus on problem domain phenomena per se, the facet-modelling approach focuses on representing important aspects of the phenomena, as was depicted in Figure 17. Hence facet modelling is a departure from claims made by OOA proponents that a single construct (the object) — or at least a small set of constructs (e.g,
objects, states, classes, inheritance, instances, operations and messages [5]) — is sufficient to create rich and intuitive models of real-world problem domains.

- From prescriptive to problem-driven methodology.
  Most contemporary SA and OOA methodologies are prescriptive in that the modelling languages used set the agenda for which pieces of knowledge about the problem domain to collect during analysis. In contrast, a facet model can be extended at any time with any type of problem-domain knowledge that is perceived as relevant. This is in line with recent metamodeling approaches (e.g., Boehm [6] and Livari [33]) which emphasise problem analysis as an inquiry process driven by the problem domain and the problems at hand, rather than by assumptions inherent in some already selected modelling language. Hence, facet modelling is not “against method”, but critical of modelling methods whose activities are biased towards collecting information needed from the point of view of a particular modelling orientation, rather than from the characteristics of the particular problem domain and the problems at hand.

- From solution-orientation to client learning.
  As observed in [17], the objective of problem analysis is not always a solution in the form of a new IS or a re-engineered business process. The objective may also be to increase the client’s understanding of the problem domain, i.e., learning. Facet models, which allow integrated and simultaneous representation of several alternative perspectives on the problem domain, are consistent with this idea. The explicit representation of ontological agreement of existence through items is helpful in detecting apparent or real conflicts of perspective.

- From repositories of separate models to a single integrated model.
  Instead of distributing the representation of a problem domain between several partial models of different languages, and then maintaining references between their model elements, the facet-modelling approach provides a framework for integration of several partial modelling languages. An industrial seminar arranged by the European Commission [39] to identify research needs within requirements engineering concluded that: “There are a number of specification notation techniques available, each tailored to suit specific system aspects, though none adequate for dealing with all the required aspects”. […] “New specification techniques were not deemed relevant. […] but new methods and tools should be developed for integrating a minimum set of existing notations to: […] support translations […] produce partial and different views […] merge different notations that document the different aspects”. This industrial emphasis placed on integrative capacity alone might justify exploring the facet modelling approach further.

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 even be possible to regard the final implementation as a set of linked facets of items. Mylopoulos’ view of IS development as an activity dealing with four worlds — the problem world, the usage world, the system world and the development world [34, 41] — is likely to be used as a tool to manage the complexity of the resulting facet models and languages in further work on the approach.

6.2 Towards Core Metamodels of Problem Domains

An ambitious attempt to solve the “contingency problem” discussed in Section 5.3 is to integrate a large (ideally exhaustive) set of modelling orientations in a single, integrated and formalised core metamodel. In contrast, facet modelling is based on the assumption that the set of kinds of knowledge about a problem domain that may be relevant for problem analysis is potentially unlimited, and cannot in general be known in advance. It follows that the assumptions behind facet modelling are inherently in contrast to the core metamodel ambition: With information systems pervading almost all aspects of humans’ daily lives, it seems improbable that a single metamodel can be built, much less formalised, which covers all problem domains in all problem
situations at all times. Instead, facet modelling therefore provides languages that can be extended to meet the problems at hand as knowledge about the problem domain increases.

Scaled down to realistic proportions, however, the core metamodel ambition is nevertheless worth pursuing. Less ambitious core metamodels could be built for a selection of common aspects, for specific problem domains, and for specific classes of information systems. A scaled-down metamodel would comprise an integrated structure of both informal and formally defined constructs usable for representing a wide range of common problem domain aspects. Furthermore, the metamodel should be extensible to less common aspects when needed, and extensions should be reusable so that the core metamodel is allowed to grow with time. It appears that the facet modelling approach is therefore well matched with the requirements for a modelling language which could represent and operationalise a core metamodel. It is also likely that facet languages will evolve in the direction of core metamodels through continual extensions as new type and metatype facets are being defined for particular analysis problems. The shared subfacet mechanism discussed in Section 3.7 may be used to build new type and metatype facets “on top of” existing ones through subfacet sharing at the type and metatype levels. As a consequence, the newly introduced types will also be integrated with the existing ones.

In addition, facet modelling offers a representation framework which can potentially be used to integrate several core metamodels which cover different kinds of aspects, distinct problem domains, or different classes of information systems. However, such core metamodel integration presupposes the resolution of a host of problems other than representation and is left for further work.

6.3. Further Work

Facet-modelling approach was first introduced in [49] and [51] is still a relatively new idea. Therefore, most of the work remains. Further research will be centered around three main issues:

- Definition of a facet language kernel, preferably with a core of formal definitions.
- Evaluation of the facet-modelling approach through larger examples and realistic case studies.
- Implementation of a facet-modelling tool based on this kernel, and preferably supporting type and metatype facet extensions.

At the University of Bergen, work is underway on defining facet languages and facet modelling methods, and on evaluating the approach through practical case studies. The relation between facet modelling and work on managing, analysing and resolving inconsistencies in viewpoints research must be explored further.

The authors believe that the complexity of contemporary problem domains will continue to increase in years to come. One reason is that the easier analysis problems are gradually being solved and turned into textbook knowledge, shelfware, frameworks, or reusable libraries, leaving only the difficult problems left for analysis work. Another reason is that automated information processing is becoming an increasingly important part of most enterprises. As such it is becoming more tightly coupled with larger parts of the enterprise than before. As a consequence, the authors believe that the call for ambitious problem analysis languages will grow stronger in years to come.

As pointed out by Nygaard & Sgaard [45], “Computers require unambiguous, formal descriptions of what they are to do, and they do it whether it makes sense or not. Human beings use ambiguous languages where connotations are just as important as the “direct meaning”, and they consider — in most cases — whether things make sense or not. [...] Research in language design, artificial intelligence, and other disciplines of informatics will help to narrow this gap, but will never bridge it”. Although it does not solve all problems, facet modelling is likely to represent an improvement on current modelling approaches when it comes to managing multiple perspectives during problem domain analysis.

Acknowledgements — The authors are indebted to the anonymous referees who have carefully reviewed the paper. In particular, Professor Juhani Iivari has provided numerous useful comments and suggested many important improvements to several earlier versions.
REFERENCES


