Capturing and Structuring the Meaning of Communication in the Building and Construction Industry

WIM BAKKEREN[1]
PETER WILLEMS[2]

ABSTRACT

Integration of the computer applications used in the building industry requires information systems that support the communication between these applications. Currently this communication is realised via human interpretation and understanding. An important question in this context is: "what makes communication meaningful?". The meaning of communication has two aspects: (1) the intention: the general idea behind the communication, and (2) the extension: the set of things to which the communication applies. This paper describes these aspects of meaning and mechanisms used by human beings to define meaning. To enable information systems to support communication the intention and the extension must be represented in a computer interpretable form. The representations should be manageable, reusable and extendable. This requires structuring of the representations, which can be achieved by modular modelling and layering. This paper describes these stucturing mechanisms.

Key Words
meaning of communication; representation of meaning; structuring representations; modularity; layering

INTRODUCTION

Building projects are carried out by participants from different disciplines. These participants are usually not employed by the same company. Consequently the excecution of a building project involves many different companies, usually in a new combination. The realisation of good communication in such projects is a difficult task to achieve. Therefore the communication makes use of standardised classification systems, regulations, and codes.

Nowadays most participants in a building project perform their activities aided by computers. In contrast, the communication between particpants is still carried out with conventional media (eg, drawings and specifications). The

1 *Assistant Researcher, Delft University of Technology, Department of Civil Engineering, PO Box 49, 2600 AA Delft, The Netherlands.*
2 *TNO Building and Construction Research, Department of Computer Integrated Construction, Delft University of Technology.*

process of interpreting and understanding the exchanged information is carried out by human beings. This is time consuming and error prone.

To solve the problem described above the exchange of information has to be supported by information systems too. This can only be achieved when the computer aided communication, just like conventional communication, is carried out in a standardised way, eg, according to the standard for the exchange of product model data (STEP). This standard is based on the application of product models, which are collections of all relevant data concerning the product at hand.

The development of standards like STEP is necessary for a better usage of information systems in manufacturing processes. Currently the research community is not only trying to answer the question "which information is exchanged in manufacturing processes?" But also "how to structure the parts of standards for information exchange to keep them manageable, reusable, and extendable?". To answer this second question is, for instance, one of the main tasks of the ESPRIT project PISA. This paper, which tries to contribute to the answer consists of two parts. Firstly, it deals with human communication in general. To support communication with information systems a clear understanding of it is nessecary. Secondly, it elaborates on the principles that information technology (IT) and computer science offer us to implement the required aspect of communication support. The first part of the paper presents ideas from philosophy, linguistics, artificial intelligence, and database engineering. The second part presents the results of research carried out at the department of computer integrated construction (CIC) Of TNO Building and Construction Research in cooperation with the Department of Civil Engineering of the Delft University of Technology.

THE MEANING OF COMMUNICATION

A product model enables information systems to support communication. To achieve this the answer to the question: "what makes communication meaningful to the participants in a manufacturing process?: Has to be known.

An Inquiry into Meaning

The most elementary way of communication between human beings is speech. This communication only makes sense when the spoken words have meaning for those who communicate. The meaning of a word has two aspects (Mattessich, 1978 and Sowa, 1984). The first aspect is the extension of the word, which is the set of all things to which the word applies. The extension of the compound noun "office building", for example, is the set of all existing office buildings. The second aspect is the intention of the word, which is the general idea behind it. This aspect corresponds to the dictionary definition of a word and is formed by perception of the things that belong to the extension.

The intention of the compound noun "office building", for example, is something as "a building in which business affairs are carried on" (derived from Webster's New Twentieth Century Dictionary).

Communication makes use of symbols, eg, spoken or written words, gestures, and drawings. The intention of a symbol is also called the concept and the extension is also called the referent. The things that belong to an extension are called elements of that extension. This paper presents symbols in italic face between double quotation marks ("symbol"), concepts with small capitals and single quotation marks ("concept"), extensions between single quotation marks ("extension"), and elements in italic face between single quotation marks ("element").

The relation between a symbol and the things to which it applies is an indirect relation established via the concept (Ogden, 1969 and Sowa, 1984). Perception maps an observed thing to a person's concept and expression, eg, speech, maps the concept to the symbol. This idea is illustrated with the meaning triangle, as shown in Figure 1 (Ogden, 1969). In the opposite direction the relation is indirect too: when a person hears or sees a symbol interpretation maps the symbol to the person's concept and understanding maps the concept to the referent.

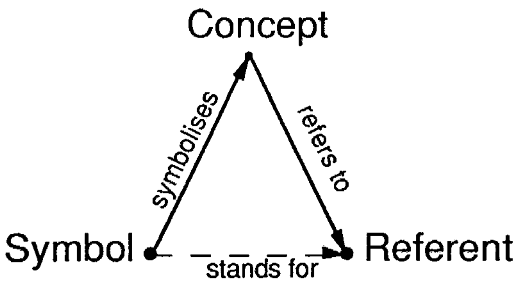


Figure 1. The Meaning Triangle

Things are often called entities or objects. In the context of IT both "entity" and "object" can be defined as "anything about which information can be acquired or used" (derived from (Fulton, 1992a)). It is important to notice that "entity" and "object" (as used in this paper) apply to real-world things and not to entities in EXPRESS schemata, entities in entity-relation diagrams or to objects in object-oriented programming languages. According to ontology an entity has properties (Mattessich, 1978), which together determine to which extension the entity belongs, ie, the properties characterise an entity. The intention implies these properties. An entity that matches the description given by the intention is an instance of that intention. The set of all instances of an intention is called the denotation of that intention (Sowa, 1984).

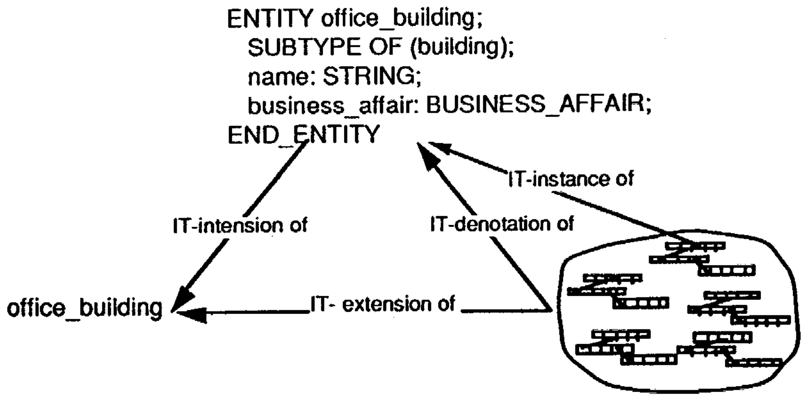
This section starts with the question "what makes communication meaningful?". As described above, the intention and the extension together make communication meaningful. However, intentionally correct communication is possibly false. The sentence "this paper is written on a MS-DOS computer" is meaningful but false (this paper is written on a Macintosh). The sentence "this section deals with the meaning of symbols", on the other hand, is not only intentionally correct but also has extensional meaning. The sentence applies to a situation in the real world: the sentence is true.

The Representation of Intentional and Extensional Meaning

To enable information systems to support communication the intention and the extension have to be made explicit. In other words, the real-world entities to which the communication applies and the general idea behind it have to be represented in a computer interpretable way. The intention can be represented with a conceptual schema, which represents the definition of properties that must be true for all instances of the intention. The instances, which together form the extension, are represented with data that have to conform to the conceptual schema. The conceptual schemata together with the data enable the information system to provide information to users of the system. This information is, as in human communication, provided with symbols, eg, tables, graphs, or text shown on the display or printed on paper.

The notion of a conceptual schema and corresponding data can also be found in the product modelling approach of STEP. Figure 2 shows an EXPRESS entity that represents the concept "office building". The data corresponding to this EXPRESS entity represent existing or possible office buildings. The EXPRESS entity and the stored data enable the exchange of information between computer applications with, for example, a STEP physical file. The information provided by the physical file is structured according to the EXPRESS entity and applies to the data stored in the database. For the interpretation of the physical file, the EXPRESS entity is needed.

The distinction between a conceptual schema and data that represent (existing or possible) real-world instances is the basis for the ANSI/X3/SPARC Reference Model for DBMS Standardization (Burns, 1985). This reference model consists of two dimensions of which the intention/extension dimension is one. The conceptual schema and the data each belong to a separate layer of this dimension, which is also part of the information resource dictionary system (IRDS) framework (irds). A layer on the intention/extension dimension contains the (IT-)instances of the layer directly above and at the same time these (IT-)instances are the (IT-)intentions for the layer directly beneath. The lowest layer, called the application layer in the IRDS framework, contains real-world entity representations, the so called application data. The layer directly above this layer, the IRD layer, contains conceptual schemata, ie, the intentions for the application layer. A conceptual schema itself is considered to be an IT-instance of a dictionary schema, which is a schema that describes other schemata (a meta-schema). The layer that contains the dictionary schema is called the IRD definition layer.



```
ENTITY office_building;
  SUBTYPE OF (building);
  name: STRING;
  business_affair: BUSINESS_AFFAIR;
END_ENTITY
```

Figure 2. The Meaning Triangle Applied to the STEP Product Modelling Approach

When the intention/extension dimension is applied to the STEP product modelling approach the IRD definition layer contains a meta-schema that specifies the properties of EXPRESS schemata. This meta-schema is, however, missing in the current situation. The semantic unification meta-model (SUMM) (Fulton, 1992a) is an effort to provide a model type that is powerful enough to model other models. Currently the SUMM has no modelling language attached to it (Fulton, 1992b) but a SUMM language would enable the use of a SUMM meta-schema. SUMM schemata themselves should be instances of a meta-meta-schema. This meta-meta-schema can be a SUMM schema too, because the SUMM is powerful enough to model other models. The meta-meta-schema is on the fourth intention/extension layer, called the IRD definition schema layer (see Figure 3).

IRD Definition Schema layer	meta-meta-models e.g. SUMM schema of SUMM schemata
IRD Definition layer	meta-models or model types e.g. SUMM schema of EXPRESS schemata
IRD layer	conceptual models e.g. EXPRESS schema of building data
Application layer	product data e.g. data of existing or possible building

Figure 3. Four intention/extension layers as defined by the IRDS applied to the product modelling approach of STEP

Human Structuring of Intentional Meaning

A concept is a general idea that human beings share about real-world entities and implies the properties of these entities. Such general ideas derived from the perception of real-world entities are abstractions of reality. The formation of a concept can be carried out by so called abstraction mechanisms, which enable the definition of a concept in terms of other concepts. This section describes two abstraction mechanisms. The first mechanism is the generalisation/specialisation mechanism, which enables the usage of general concepts in the definition of specialised concepts. The second mechanism is the aggregation/decomposition mechanism, which enables the usage of simple concepts in the definition of complex concepts.

Generalisation and Specialisation

A concept defines its denotation by describing the properties that must be true for all its instances. When some instances have properties in common that distinguish them from the other entities in the denotation then these entities form a subset of the original denotation. The concept that defines the subset is a specialisation or subtype of the original concept and the original concept is a generalisation or supertype of this subtype.

Specialisation of a concept results in the reduction of the concept denotation, which is the result of adding the differentiating properties to the concept or of constraining the possible values a property can have. The specialised concept has a smaller amount of instances than its supertype but it describes its instances more accurately (Sowa, 1984).

The G/S-mechanism can be used to define a concept. This is the Aristotelian way and uses a genus and differentiae (Sowa, 1984). The genus is the supertype of the concept that is defined and the differentiate are the properties that distinguish the new concept from other subtypes of the genus.

Aggregation and Decomposition

According to the ontology entities have properties (Mattessich, 1978). A concept defines these properties, ie, a concept defines a collection of properties that characterise the instances of the concept. Therefore, a concept is considered as an aggregation of property definitions. The concept 'BOOK', for example, is an aggregate of properties of books such as 'TITLE', YEAR PUBLISHED', 'NUMBER OF PAGES', 'AUTHOR', etc. A property that is part of one concept can be an aggregate itself. The property 'AUTHOR' for instance is an aggregate of the properties 'NAME', 'GENDER', 'BIRTH DATE', etc.

Aggregating properties and concepts results in a collection of related concepts, called a schema. A schema describes the properties of the instances of the concept and the relations that usually occur between these instances and instances of other concepts. This way of concept definition corresponds to Wittgenstein's method (Wittgenstein, 1963). A schema defines a concept by describing the family resemblance of the instances of the concept.

A special form of aggregation is association. Association is collecting instances of the same concept into a group that is considered as an instance of another concept. The concept "fleet" for instance refers to collections of instances of the concept "ship".

When the entities in an aggregation have mutual relations the aggregation is called a system (Mattessich, 1978), a composition of entities. A concept that refers to systems consists of concepts that refer to the parts of the systems. The opposite of aggregation, association and composition is decomposition, which can be used to model complex systems. Using decomposition, the concept about systems is divided in concepts about the parts. This can be continued until the desired level of detail is reached.

REPRESENTATION SYSTEMS

For information systems to be able to support the communication in manufacturing processes computer interpretable representations of the intention and the extension are needed. Representation systems provide the functionality to realise this. They provide functionality at a conceptual level, ie, a level concerned with human perception and thought, and at an implementation level, ie, a level concerned with the storage of data in a computer. The first part of this section describes currently available mechanisms to structure representations. The second part deals with structuring mechanisms that can help to build manageable representations for the complex problems the product modelling community is facing today.

Available Functionality

Representation systems provide several mechanisms to structure representations. The mechanisms are: (1) instantiation, (2) grouping, (3) sharing, and (4) inheritance.

Instantiation

Probably the most important mechanism available in all representation systems is instantiation. Through instantiation two levels of representation can be distinguished: (1) the level of stored data, the instance level, and (2) the level of the description how the data should be stored, the template level. These levels have a relative nature: a template level may be an instance level with respect to a meta-template level, ie, a level where the template level is described.

Obviously, the instantiation mechanism realises the distinction between representations of the intention and of the extension. Instantiation has, however, not only conceptual but also implementation aspects. It is the process to store a unit of data, called a class instance, according to a particular template, called a class template. A class instance has a unique address, which can be used to refer to, and stores an indicator to the class template it is an instance from. Further it allocates storage slots and possibly occupies them. The contents of a storage slot may change during the class instance's existence. The changes should always be in agreement with the corresponding class template. A class instance usually does not change its type (the class template correspondence) or increase or decrease its number of storage slots.

Grouping

The second structuring mechanism is grouping. Grouping assembles a set of simple constructs into a complex construct. This section describes four kinds of grouping: (1) binary-string/bit grouping, (2) attribute/binary-string grouping, (3) class/attribute grouping, and (4) schema/class grouping.

The most basic form of grouping is binary-string/bit grouping. This mechanism groups binary digits (bits: ones and zeroes) into a binary string of a certain length. Such a binary string is an instance of a particular binary-string template and represents a value (eg, C, 4, 3.234, or TRUE). The following four binary-string templates, or basic types, are distinguished: (1) character template, (2) integer number template, (3) real number template, and (4) Boolean template.

The second kind of grouping is attribute/binary-string grouping. This mechanism groups a number of binary-string instances into an attribute instance. The number of binary-string instances an attribute instance may contain is defined through its minimum and maximum cardinality at template level. Several forms of attribute/binary-string grouping can be distinguished, eg, array, list, bag, set, queue, stack, tree, binary tree, and circular list. These forms can be pre-defined as library templates instead of being part of the representation system. Attribute/binary-string grouping facilitates conceptual association.

The third kind of grouping is class/attribute grouping, which is closely related to conceptual aggregation. It enables the collection, into one template, of representations of those properties that characterise certain entities in the Universe of Discourse (UoD). The mechanism groups several attribute templates into one class template. The corresponding class instances are collections of attribute instances (see Figure 4).

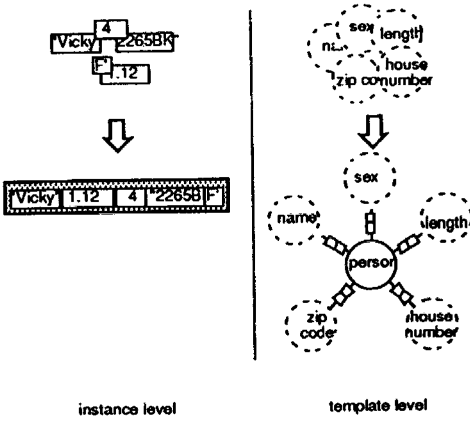


Figure 4. Class/attribute grouping. The left part shows the grouping at instance level. The right part shows the corresponding grouping at template level.

The fourth kind of grouping is schema/class grouping. At the instance level this mechanism groups class instances into a schema instance. This grouping corresponds to a grouping of class templates into a schema template at the template level. The mechanism is closely related to Wittgenstein's definition of concepts with schemata.

Sharing

Sharing is another mechanism available in most representation systems.

The mechanism is an important technique to reduce redundant data and enables relations between instances if sharing is also conceptually appropriate. This paper deals with two kinds of sharing: attribute sharing and class sharing. At instance level the objective is value sharing while at template level sharing supports inheritance (without redefinition).

Attribute templates that must be shared between several class templates can be grouped in a separate class template to refer to. This form of sharing implements sub-type relations between classes.

Attribute instances that must be shared between several class instances can be grouped in a separate class instance to refer to. The formation of this separate class instance is described at the template level (see Figure 5).

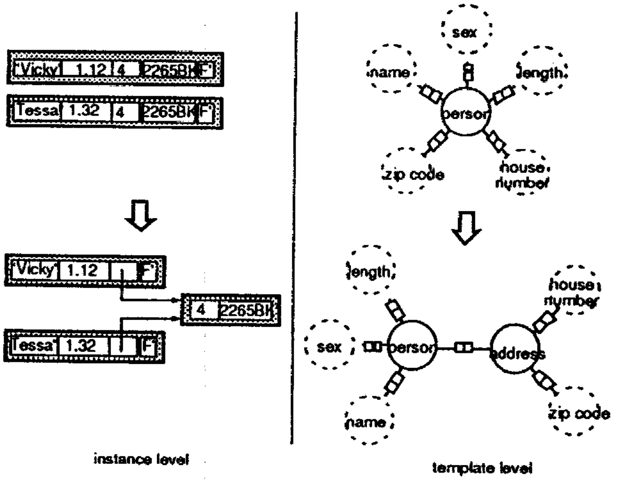


Figure 5. Sharing of Attribute Instances

Class templates that must be shared between several schema templates can be grouped in a separate schema template to refer to. This form of sharing implements sub-type relations between schema templates.

Class instances that must be shared between several schema instances can be grouped in a separate schema instance to refer to. This form of sharing implements library instances. There is a strong resemblance with the sharing of attribute instances as described above.

Inheritance

Another mechanism is inheritance. Inheritance can be based on the data sharing principle described above as long as the inherited data can be used without modification. If modification is necessary, part of the shared data must be copied and modified. This section describes attribute inheritance, which can be applied at template level as well as instance level.

Attribute-template inheritance is based on the sub-type relationship between class templates. The descendent class template inherits all attribute templates from its parent class templates and may introduce changes to these attributes. Changes should impose more constraints on the value domain or cardinality defined by the attribute template.

Attribute-instance inheritance is based on the sharing of attribute instances. A descendent class instance inherits its attribute instances from a parent class instance. It may substitute the inherited value by explicitly specifying its private value.

Sharing and inheritance at the template level facilitate conceptual specialisation. Most representation systems do, however, not support definition of concepts with the genus-and-differentia mechanism. Also generalisation is usually not fully supported.

Extended Functionality

The mechanisms described above are available in most representation systems. They do, however, not suffice for the formation of manageable representations. Therefore a need for mechanisms that fulfil the requirements of the product modelling community exists. These mechanisms can be divided into two groups: mechanisms for modularity and mechanisms for layering. Modularity enables the division of complex representations into smaller, independent parts. Layering enables the set up of a framework in which the modular parts can be placed.

Modularity

Within the scope of single schema templates traditional modelling techniques have sufficient capabilities to describe a UoD adequately. Most methods do, however, not support a schema-template hierarchy and inter-schema template referencing. How to manage a set of distinct schema templates to behave as a single schema template is best understood by reasoning the other way round: how to break down a monolithic schema template into a set of module schema-templates. On the instance level the General AEC Reference Model (GARM) (Gielingh, 1988) facilitates the decomposition of schema instances into a set of module schema-instances. To illustrate this mechanism this section uses the problem of parts and assemblies at the instance level. Thereupon the applicability of the mechanism at the template level is analysed.

In a monolithic model parts are linked directly to each other to create an assembly (see Figure 6). Besides the impossibility to break down this structure into manageable pieces, this type of modelling offers no facilities to communicate with the schema instance's environment.

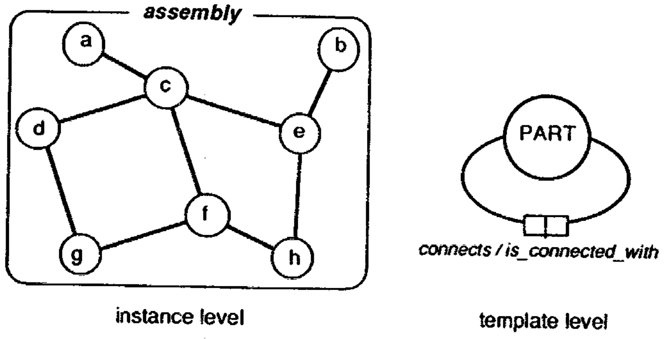


Figure 6. Monolithic modelling. The left part shows an example of parts [circular objects a..h] directly linked to compose an assembly [blended rectangular]. The right part shows the corresponding schema template.

A primary requirement to break down a structure into smaller segments is a mechanism to attach or detach the connections between parts. This can be achieved by the definition of explicit connections that consist of two components called ends, which each represents one side of the connection (see Figure 7). A proper connection consists of two mating ends.

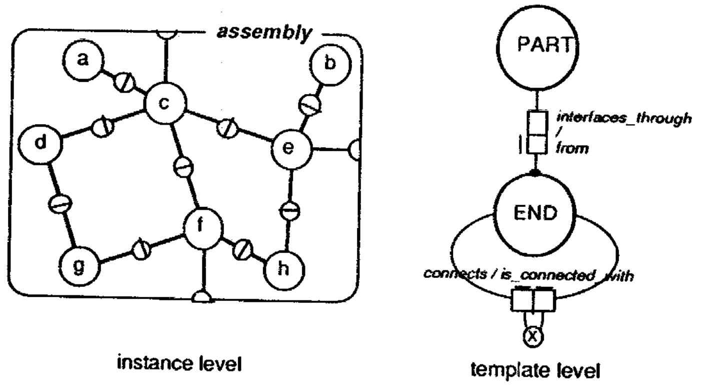


Figure 7. Modular Modelling

An assembly as described above can be broken down into sub-assemblies. The mated ends offer possible intersection points. To recover the original assembly structure information is needed concerning which-end-mates-which-end. This information can be collected at the assembly level in a template assembly-end, which collects the dangling ends (see Figure 8). Two assembly-ends can compose an assembly connection.

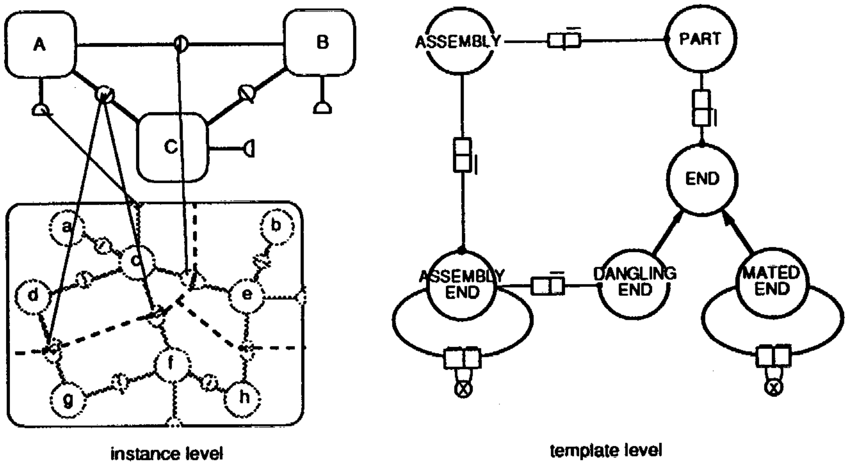


Figure 8. Inter-assembly Connectivity

The result resembles a one layer decomposition. The GARM extends this recursively into a multi-layer decomposition. This step is postponed here. First the question is answered if this mechanism, meant to be used on the instance level, is also applicable to the template level.

When applied to the template level part maps to class-template and assembly maps to SCHEMA-TEMPLATE (see Figure 9). MATED-ENDs represent relations between CLASS-TEMPLATEs. To understand a particular end the relation must be directed. A SOURCE-END represents an attribute template referencing another CLASS-TEMPLATE, and a TARGET-END may be viewed as an address to reference the attached CLASS-TEMPLATE. DANGLING-ENDs represent the schema template interface. A DANGLING-SOURCE-END represents an external reference (or private interface map (Kirkley, 1991)) and a DANGLING-TARGET-END represents a global address (or public interface map (Kirkley, 1991)) for the attached CLASS-TEMPLATE.

The GARM modular network ideas can also be applied to the generalisation/specialisation dimension. Consequently, the CLASS-TEMPLATE-ENDs can be applied in various ways, either as class template relations or as sub-typing relations. It seems obvious to make a clear distinction between these two applications. Moreover, there are other conditions that justify explicit grouping of CLASS-TEMPLATE-ENDs, eg, the dimension concept as mentioned in the ISO-STEP framework document

(Kirkley, 1991). The visibility of a particular class template in a certain dimension depends on the fact if an explicit target-end has been defined for that dimension.

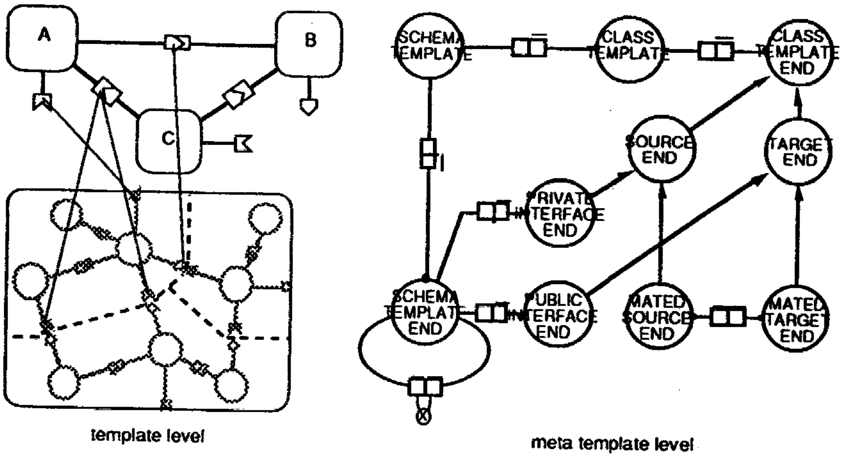


Figure 9. Inter-schema Template Connectivity

As the previous paragraphs show, the mechanism of the GARM can also be applied at the template level. As said above, the GARM applies the mechanism recursively, which results into a decomposition tree. Ignoring the particular semantics of the GARM functional-unit and technical-solution concepts, the so called Hamburger diagram is an example of modularity in the decomposition dimension (see Figure 10). The decomposition tree can now be assembled from separate modules each spanning one decomposition level. Modules can be plugged in or out.

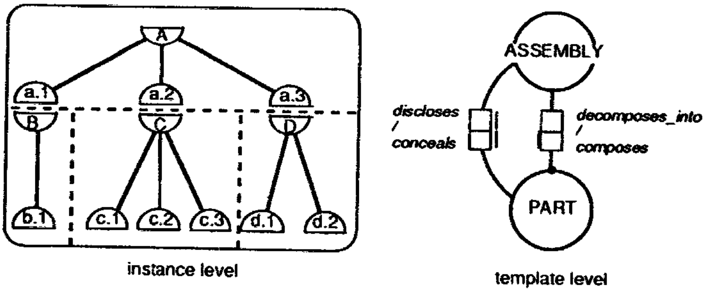


Figure 10. Hamburger diagram. An example of modular decomposition. Parts are symbolised by semi circles with a flat bottom side and assemblies by semi circles with a flat top side.

The recursive GARM approach can be upgraded to the schema-template level. Now, the decomposition of a high level class template can be elaborated in a separate schema template and afterwards be attached to the high level class template. This technique is used in a model for the exchange of road design information: the Road Model Kernel (Willems, 1990).

Layering

Modularity is a necessity for structuring schema templates in a framework, however, modularity in itself is not sufficient. It will not prevent a tremendous number of schema templates entangled in a huge spaghetti-like structure. Layering defines a structure of place holders or slots that can be filled by one or more schema templates.

An important design decision applies to the question if a layer may host more than one schema template, and if so, how those schema templates should interrelate. Obviously, a one-to-one correspondence of layer and schema template is unambiguous and simple. However, this may result in large schema templates. Schema-template decomposition, as described in the previous section, could solve that.

A single layering system will not be feasible. Therefore a framework is needed with several layering dimensions. Such a framework can be represented as a multi-dimensional table. Each table cell represents a slot, possibly hosting a schema template (see Figure 11).

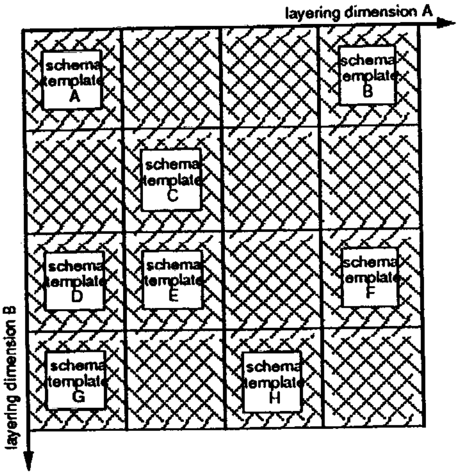


Figure 11. Multiple Layering System

Schema templates must not refer to each other directly. This would contradict the assumed plug-in architecture characteristic for layering. Therefore, schema templates must refer to each other indirectly via a neutral intermediary, which is part of the layering system: the interfaces between the layers. This principle harmonises quite well with the specification versus implementation aspects. The framework itself contains the specifications, while an individual schema template composes the implementation for a specific slot. Figure 12 shows a framework including layer interfaces. The inter-schema template relations are cut into two trajects: from source schema template to interface and from interface to target schema template.

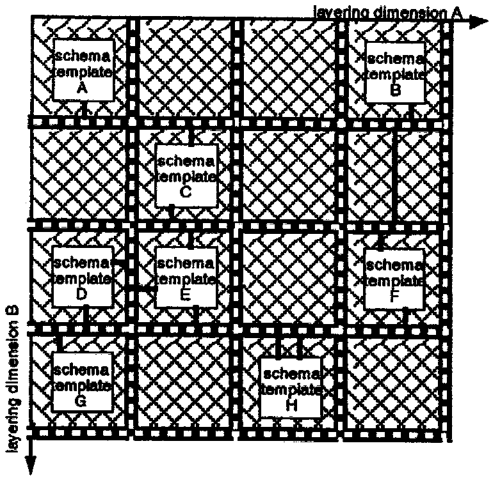


Figure 12. Layer Interfaces

CONCLUDING REMARKS

This paper presents results from research carried out at the department of Computer Integrated Construction (CIC) of TNO Building and Construction Research. The research is carried out in the context of several projects, like ESPRIT projects, projects for the Dutch Ministry of Transport, Public Work and Water Management, and a CIC project in which several PhD students of the Delft University of Technology participate. The ongoing development of a computer tool that is suited to be used in these different projects forms an important part of the activities at the CIC department of TNO. The ideas presented here a part of a contribution to the ESPRIT project PISA. Several aspects of the ideas are also used in the development of the computer tool mentioned above. Furthermore are they used in the different PhD projects mentioned above.

References

Burns, T, Fong, E, Jefferson, D, Knox, R, Mark, L, Reedy, C, Reich, L, Roussopoulos, N and Truszkowski, W (1985), "Reference Model for DBMS Standardization," Tech. Rep., NBSIR 85-3173, May.

Fulton, J A, Zimmerman, J, Eirich, P, Tyler, J, Burkhart, R, Lake, G F, Law, M H, Menzel, C, Speyer, B, Stump, R and Williams, A (1992a), "The Semantic Unification Meta-Model: Technical Approach," Tech. Rep., ISO TC184/SC4/WG3 N175, Released, October.

Fulton, J A (1992b), "Enterprise Integration using the Semantic Unification Meta-Model." in Enterprise Integration Modeling, Petrie, C.J., The MIT Press, pp 278 - 289.

Gielingh, W F (1988), "General AEC Reference Model," Tech. Rep., P.O. Box 46 2600 AA Delft, the Netherlands, BI-88-150, October.

Information Resource Dictionary System (IRDS) Framework. International Standard, June 1990.

Kirkley, J R and Seitz, B K (1991), "STEP Framework, Concepts and Principles," Tech. Rep., March, draft document.

Mattessich, R (1978), Instrumental Reasoning and Systems Methodology: An Epistemology of the Applied and Social Siences. Vol 15, Theory and Decision Library. P.O. Box 17, Dordrecht, Holland: D. Riedel Publishing Company.

Ogden, C K and Richards, I A (1969), The Meaning of Meaning: A Study of The Influence of Language upon Thought and of The Science of Symbolism., International Library of Psychology, Philosophy & Scientific method. Broadway House 68-74 Carter Lane London: Routledge & Kegan Paul Ltd.

Sowa, J F (1984), Conceptual Structures: Information Processing in Mind and Machine., The systems programming series. Addison-Wesley Publishing Company.

Willems, P H (1990), "The Road Model Kernel, Version 0.2," Tech. Rep., TNO report, B-89-831, March.

Wittgenstein, L (1963), Philosophical Investigations. Basil Blackwell.

c. 1993, Management of Information Technology for Construction, K. Mathur et al (Eds), World Scientific Publishing Co., Singapore.