SINTEF

SINTEF ICT

Address: P.O.Box 124, Blindern
0314 Oslo NORWAY
Location: Forskningsveien 1
0373 Oslo
Telephone: +47 22 06 73 00
Fax: +47 22 06 73 50

Enterprise No.: NO 948 007 029 MVA

SINTEF REPORT

TITLE

Vocabulary for model-based user interface development

AUTHOR(S)

Erik G. Nilsson

CLIENT(S)

The EMERGENCY project supported by the Research Council of Norway, p.nr. 187799/S10

REPORT NO.	CLASSIFICATION	CLIENTS REF.	
SINTEF A19533	Open		
CLASS. THIS PAGE	ISBN	PROJECT NO.	NO. OF PAGES/APPENDICES
Open	978-82-14-04982-4	90B261	32
ELECTRONIC FILE CODE		PROJECT MANAGER (NAME, SIGN.) Ketil Stølen	CHECKED BY (NAME, SIGN.) Gyrd Brændeland
FILE CODE	DATE	APPROVED BY (NAME, POSITION, SIGN.)	
	2011-05-26	Bjørn Skjellaug, Forskningssjef	

ABSTRACT

The goal of the work presented in this report has been to define a precise, consistent and understandable vocabulary for our work on model-based user interface development, by finding or making our own definitions of concepts that are central in this field.

When working on the definitions, we have exploited general vocabularies for describing and categorizing computer science, software engineering, user interfaces, as well as general and specific vocabularies used in model-based user interface specification languages, and supplemented these with our own definitions where needed.

We present the concepts at seven different levels spanning from general computer science independent concepts to concepts that are specific to model-based user interface development. Definitions of concepts are based on acknowledged definitions and/or built from the more general definitions presented earlier in the vocabulary.

KEYWORDS	ENGLISH	NORWEGIAN
GROUP 1	ICT	IKT
GROUP 2	HCI	Menneske-maskin-interaksjon
SELECTED BY AUTHOR	Vocabulary	Vokabular
	Model based systems development	Modell-basert utvikling
	User interfaces	Brukergrensesnitt

TABLE OF CONTENTS

1 Introduction

In this report we try to define a precise, consistent and understandable vocabulary for model-based user interface development. The goal has been to find or make our own definitions of concepts that are central for model-based user interface development in general. It is important to have well-functioning definitions of these concepts, to be used as a basis for our work on model-based user interface development in the EMERGENCY project. A major challenge has been to come up with a vocabulary that is precise and yet understandable also for readers outside the field of model-based user interface development. In pursuing this, we look into general vocabularies for describing and categorizing computer science, software engineering, user interfaces, as well as general and specific vocabularies used in model-based user interface specification languages, and supplemented these with our own definitions where needed.

We present the concepts at seven different levels of generality. These levels span from concepts that are defined independently of the computer science field, through computer science concepts as well as general and more specialized software engineering concepts and general user interface concepts, ending up in concepts that are specific to model-based user interface development. Note that not all these levels are levels in a strict sense, as e.g. modelling concepts and general user interface concepts may be considered to be at fairly the same level.

The rest of this report consists of nine sections. In Section 2, we define the research method used when collecting and structuring the definitions in the vocabulary. In Sections 3-9 we present the vocabulary on the seven levels just mentioned. Finally, we give a brief conclusion and outline how the vocabulary will be used in our future research.

2 Research method

Defining the vocabulary has consisted of three main activities, i.e. selecting the concepts to include, finding and/or choosing definitions that are both appropriate and sound, and structuring the concepts.

When selecting the concepts to include, we have started from the most specific levels and identified a set of concepts that we have used in our previous work on model-based user interface development (Nilsson et al, 2006), and supplemented this with concepts that have emerged in our current work in the field. Based on this set of concepts, we have identified the concepts at the more general levels that were needed as basis for defining the concepts at the more specific levels. A few concepts have emerged while reading through papers and vocabularies. In these cases, the new concepts have replaced other less useful ones identified earlier.

When finding and/or choosing definitions for the concepts in the vocabulary, we have come across a number of challenges. For the most general concepts (like language and model) it is quite easy to find acknowledged definitions, but there exist a plethora of such definitions. For other concepts, especially at the general software engineering and user interface levels, it is sometimes difficult to find acknowledged definitions, and/or different definitions are diverging. For the concepts at the most specific level, the main challenge is to find acknowledged definitions. The main criterion for choosing among varying and/or diverging definitions has been to use the definitions that are most useful as (basis for defining the) user interface modelling concepts. In the cases where we have not found definitions that fit our needs, we have made our own definitions, usually by combining definitions of the more general concepts they are built from. The main sources for definitions are (sorted by priority of use): dictionaries, text books, journal articles, conference papers, project deliverables, on-line vocabularies, as well as less reliable web-based sources like Wikipedia. For a number of concepts, we have included more than one definition,

possibly with a discussion of the appropriateness of the different definitions for our needs. Unless other is noted, the last definition that is presented is the one we use.

When structuring the concepts in the vocabulary, one challenge has been to find the right level to place the concepts. In many cases, related concepts will occur at different levels, usually as more specialized concepts at the less general levels. E.g. language will be define at the basic level, while modelling language will be defined at the software engineering level and user interface modelling language will be defined at the most specific level. The main principle has been to define the concept at the most general level providing a useful definition to use either in the core vocabulary or as basis for defining concepts in the core vocabulary. It is of course a prerequisite that the concepts are used fairly often at the given level. In some cases the same concept may be used with different meanings at the different levels without being qualified (like component, type, system, platform and tool). Such concepts are defined at the most specific level providing a useful definition, possibly referring to a more general definition.

3 Basic concepts

This section contains definitions of concepts that are drawn from general language and/or other disciplines than computer science.

3.1 Language

Merriam-Webster (2011) primary definition of *language* is:

- The words, their pronunciation, and the methods of combining them used and understood by a community.

One of the secondary definitions is:

- A formal system of signs and symbols (as FORTRAN or a calculus in logic) including rules for the formation and transformation of admissible expressions.

Of these two definitions, the latter is most in line with the needs of computer science and its focus on machine execution.

The fields of semiotic (the study of signs and symbols, esp. the relations between written or spoken signs and their referents in the physical world or the world of ideas (Collins, 2003)) and linguistics (the scientific study of language (WordNet, 2011)) both have *syntax*, *semantics* and *pragmatics* as central concepts.

3.2 Syntax

Merriam-Webster (2011) primary definition of *syntax* is:

- The way in which linguistic elements (as words) are put together to form constituents (as phrases or clauses).

3.3 Semantics

The American Heritage Dictionary of the English Language (2009) defines *semantics* as:

- The meaning or the interpretation of a word, sentence, or other language form.

This definition fits well with the definition of syntax in the previous section.

3.4 Pragmatics

One of the Merriam-Webster (2011) definitions of *pragmatics* is:

- A branch of semiotics that deals with the relation between signs or linguistic expressions and their users.

Collins English Dictionary (2003) defines pragmatics as:

- The study of those aspects of language that cannot be considered in isolation from its use.

Although the first definition probably is most in line with the definitions for syntax and semantics in the two preceding sections, we find the latter to be most useful, as it is more open and focus on how a language is used.

3.5 Method

The American Heritage Dictionary of the English Language (2009) defines *method* as:

- A means or manner of procedure, especially a regular and systematic way of accomplishing something.

WordNet (2011) adds the notion of steps in a method:

- A way of doing something, especially a systematic way; implies an orderly logical arrangement (usually in steps).

As a basis for further definitions, we combine these two definitions into this one:

- A means or manner of procedure, especially a regular and systematic way of accomplishing something using an orderly logical arrangement (usually in steps).

3.6 Methodology

The American Heritage Dictionary of the English Language (2009) defines *methodology* as:

- A body of practices, procedures, and rules used by those who work in a discipline or engage in an inquiry; a set of working methods.

3.7 Model

In the context of philosophy/logic, the American Heritage Dictionary of the English Language (2009) defines *model* as:

- A simplified representation or description of a system or complex entity, esp. one designed to facilitate calculations and predictions.

Although this definition is from a different domain than software engineering, it is a good basis for more specific definitions. The way model is defined above, it denotes an artefact that is the result of doing a *modelling task*. Model may also be used as a verb.

3.8 Specification

One of the American Heritage Dictionary of the English Language (2009) definitions of *specification* is:

- A detailed, exact statement of particulars, especially a statement prescribing materials, dimensions, and quality of work for something to be built, installed, or manufactured.

Comparing the definitions of model and specification together, especially in a software engineering context, it is evident that models may be used as part of a specification, and that specifications may include parts (e.g. requirements) that are not possible or convenient to express using models. Furthermore, models may be used in many other contexts than as a part of a specification, e.g. to express aspects of a glossary, which is an application of models that we use in this report.

3.9 Domain

WordNet (2011) defines *domain* as:

- The content of a particular field of knowledge.

A common synonym to domain, especially when used in more formal settings, is *universe of discourse.*

3.10 Type

Collins English Dictionary (2003) defines *type* as:

- A kind, class, or category, the constituents of which share similar characteristics.

It should be noted that type is used both qualified and unqualified in a number of ways in software engineering, like data type in a programming language and an abstraction of a class in abstract data type sense. In our further use of type we apply it in the more general sense defined by the Collins English Dictionary.

3.11 Transformation

In the context of linguistics, the American Heritage Dictionary of the English Language (2009) defines *transformation* as:

- A rule that systematically converts one syntactic form or form of a sentence into another.

3.12 Modality

In the context of physiology, the American Heritage Dictionary of the English Language (2009) defines *modality* as:

- Any of the various types of sensation, such as vision or hearing.

3.13 Relations between concepts

In Fig. 1 through Fig. 3, we have shown how the basic concepts defined in this section relate to each other. The models are expressed using UML class diagrams (Pilone, 2005).

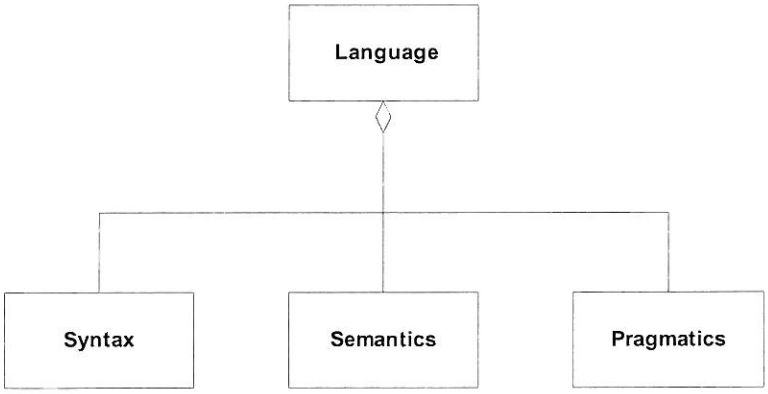

Fig. 1 - Language, syntax, semantics and pragmatics

Fig. 2 - Methodology and method

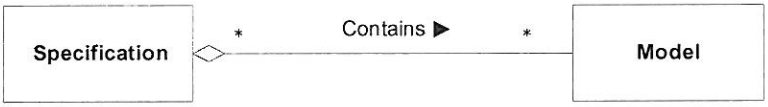

Fig. 3 - Model and specification

4 Computer science concepts

In this section the concepts are drawn from computer science in general, typically concepts that are more general than software engineering concepts.

4.1 System

Collins English Dictionary (2003) defines *system* as:

- A group or combination of interrelated, interdependent, or interacting elements forming a collective entity; a methodical or coordinated assemblage of parts, facts, concepts, etc.

Even though this definition covers system as used in computer science, we apply the definition given in the Random House Dictionary (2011) in the context of computers:

- A working combination of hardware, software, and data communications devices.

4.2 Platform and Implementation platform

Used in the context of computer science, platform is sometimes qualified as *computer platform* or *software engineering platform.*

In this context, The American Heritage Dictionary of the English Language (2009) defines *platform* as:

- The basic technology of a computer system's hardware and software that defines how a computer is operated and determines what other kinds of software can be used.

WordNet (2011) defines *platform* as:

- The combination of a particular computer and a particular operating system.

Our need for the platform concept is mainly for describing targets for transformations from abstract specifications/models all the way to implementations running on a platform. Adding our focus on user interface development, we refine the definitions above into our own definition of an *implementation platform*:

- The combination of an operating system and a windowing system.

Operating and windowing systems are usually pairs, but there are operating systems (e.g. UNIX-based ones) where there are different windowing systems running on the same operating system.

4.3 Component

WordNet (2011) defines *component* as

- An artifact that is one of the individual parts of which a composite entity is made up; especially a part that can be separated from or attached to a system.

Even though this definition covers components as used in computer science, we apply the more precise definition given in UML (OMG, 2008):

- A modular part of a system, that encapsulates its content and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces.

4.4 Systems development

Based on the definition of system in Section 4.1, we define *systems development* as:

- Development of computer-based systems.

The concept is related to software engineering, but systems development may involve handling organizational aspects and well as choices regarding the hardware aspects of the given platform, and is thus a slightly wider concept.

4.5 Relations between concepts

In Fig. 1, we have shown how the computer science concepts defined in this section relate to each other and to some of the basic concepts.

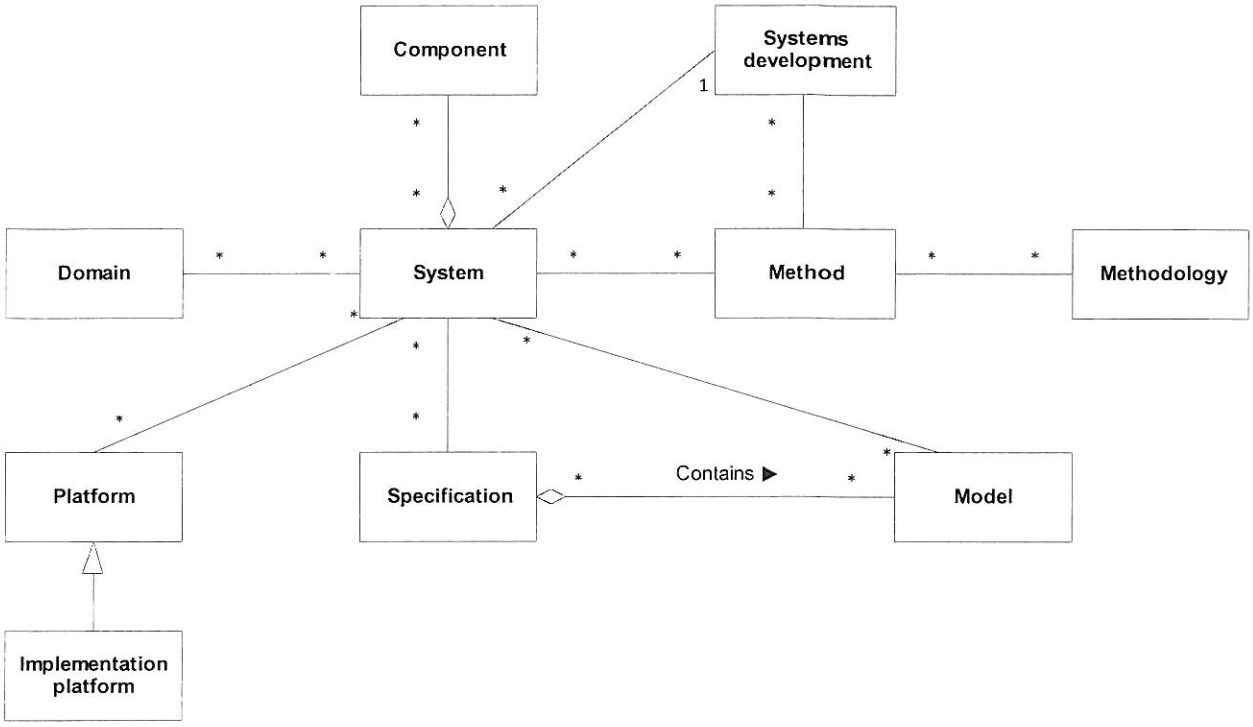

Fig. 4 - System, platform, component and various basic concepts

5 Software engineering concepts

In this section the concepts are drawn from software engineering in general, and are thus not user interface specific.

5.1 Programming language

The American Heritage Dictionary of the English Language (2009) defines *programming language* as:

- An artificial language used to write instructions that can be translated into machine language and then executed by a computer.

The American Heritage Dictionary of the English Language (2009) defines *syntax* in the context of computer science to be:

- The rules governing the formation of statements in a programming language.

This definition is quite close to a definition applying the linguistic definition of syntax given above on a programming language, which could be something like "the way in which the elements of a programming language are put together to form statements in this language".

Applying the same approach for *programming language semantics* gives this definition:

- The meaning or the interpretation of statements in a programming language.

Using this approach for programming language pragmatics would lead to a definition like "the relation between the elements of a programming language and their users". To make the concept a bit more precise, we give this definition of *programming language pragmatics:*

- A method or other guidance on how to use a programming language.

5.2 Modelling language

Based on the definitions of language and model above, we define *modelling language* as:

- A language to express models.

In a similar manner, we define *modelling language syntax* to be:

- The way in which the elements of a modelling language are put together to form legal models in this language.

In software engineering, it is quite common to make a distinction between *abstract* and *concrete* syntax of a language. The abstract syntax is a definition of the element in the language and their structure, usually expressed in a meta model. The concrete syntax is the actual textual, graphical or other representations of constituents of the language. In the field of programming languages, this distinction is primarily used to make a distinction between the internal representation of a program in the compiler (abstract syntax) and the syntax used by the programmer (concrete syntax) (The Free On-line Dictionary of Computing, 2010). In modelling languages, the distinction may be visible for the modeller, as there may be one graphical syntax used in diagrams, and one or more textual syntax used for representation and exchange of the models by computers.

In the same way as modelling language syntax, we define *modelling language semantics* to be:

- The meaning or the interpretation of models expressed in a modelling language.

Finally, we define *modelling language pragmatics* to be:

- A method or other guidance on how to use a modelling language.

5.3 Specification language

Based on the definitions of language and specification above, we define *specification language* as:

- A language to express specifications.

Specification languages vary more than modelling languages, i.e. when making a specification, a number of specification languages may be used, varying from natural language, through using modelling languages to express models that are part of a specification, to formal specification languages.

For specifications expressed using natural language or a modelling language, syntax, semantics and pragmatics are defined above. For specifications express using other means, we define *specification language syntax* to be:

- The way in which the elements of a specification language are put together to form legal specifications in this language.

In the same way as specification language syntax, we define *specification language semantics* to be:

- The meaning or the interpretation of specifications expressed in a specification language.

Finally, we define *specification language pragmatics* to be:

- A method or other guidance on how to use a specification language.

5.4 Tool

Collins English Dictionary (2003) defines *tool* as:

- Anything used as a means of performing an operation or achieving an end.

Even though this definition covers tool as the concept is used in software engineering, we apply the definition given in The American Heritage Dictionary of the English Language (2009) in the context of computer science:

- An application program, often one that creates, manipulates, modifies, or analyzes other programs.

5.5 Programming

WordNet (2011) defines *programming* in the context of computer science as:

- Creating a sequence of instructions to enable the computer to do something.

The Wikipedia (2011) article describing computer programming includes the presence of source code.

5.6 Model-based systems development

Our traditional notion of model-based systems development seems to be more or less a Norwegian speciality. The concepts that are mostly used for this are *model-driven engineering (MDE),* of which OMG's *model-driven architecture (MDA)* is an example.

ERCIM Working Group Software Evolution (2008) defines *MDE* as:

- A software engineering approach that promotes the use of models and transformations as primary artifacts throughout the software development process. Its goal is to tackle the problem of developing, maintaining and evolving complex software systems by raising the level of abstraction from source code to models. As such, model-driven engineering promises reuse at the domain level, increasing the overall software quality.

5.7 Property

Wiktionary (2011) defines *property* as:

- An attribute or abstract quality associated with an individual, object or concept.

Although quite general, this definition fits fairly well to the software engineering field. Generally, property is often used as synonym to attribute, but in software engineering there is a distinction in the way that an attribute is more representational oriented (attribute of class or data base record), while property also may be used more abstractly. Specifically in some software engineering tools, property is used for denoting predefined attributes of components in the application programming interface (API) of the platform or the tool.

5.8 Model transformation

Based on various definitions above, we define model *transformation* as:

- A rule that systematically converts a model to another model in the same or a different modelling language

5.9 Relations between concepts

In Fig. 5 through Fig. 7, we have shown how the software engineering concepts defined in this section relate to each other and some of the basic and computer science concepts.

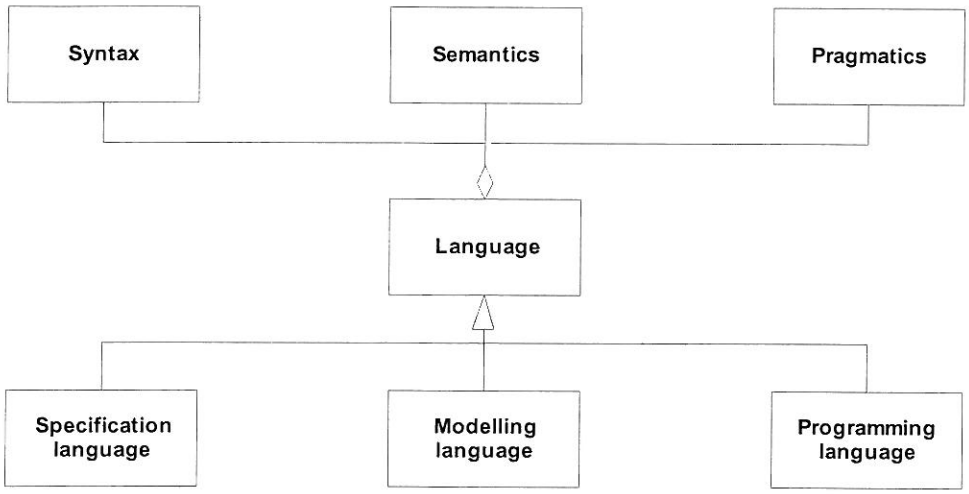

Fig. 5 - Languages

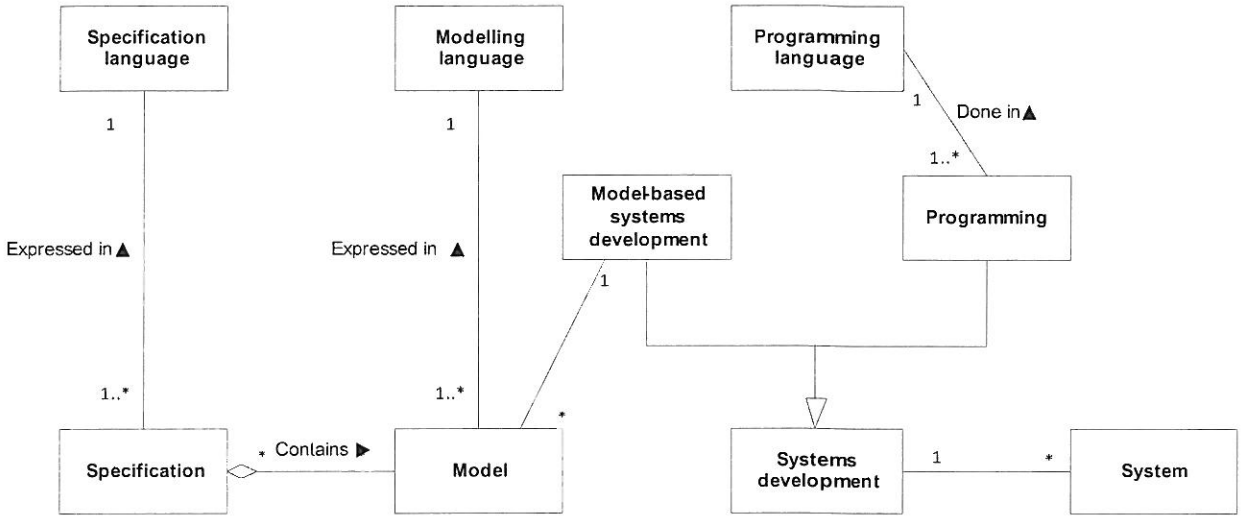

Fig. 6 - Languages and systems development

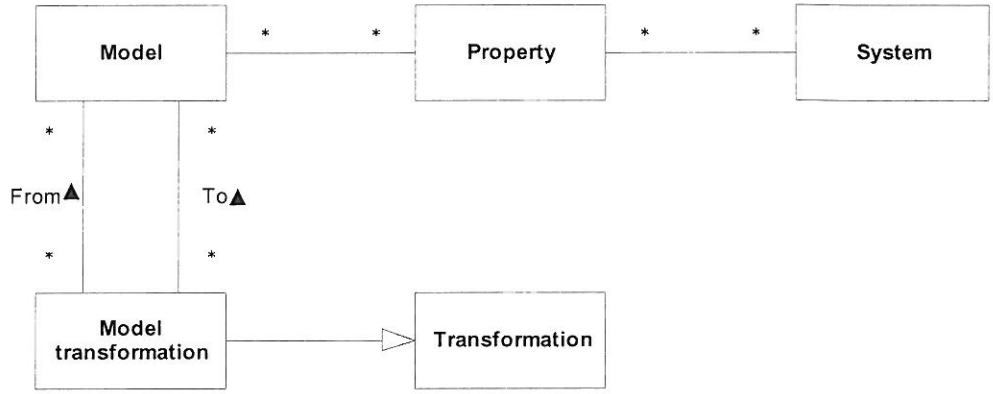

Fig. 7 - Property and model transformation

6 Software engineering tool concepts

In this section the focus is on concepts describing tools used in software engineering.

6.1 Systems development tool

The term *Systems development tool* does not seem to be widely used. In Wikipedia (2011), *Software development tool(s)* redirects to Programming tool, described as:

- A **programming tool** or **software development tool** is a program or application that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object.

The Free On-line Dictionary of Computing (2010) defines *software engineering environment* as:

- A set of management and technical tools to support software development, usually integrated in a coherent framework.

The Wikipedia description is too restricted for our needs. Although the Free On-line Dictionary of Computing is more general than the Wikipedia description, it only partly covers our notion of a systems development tool, i.e. it fits with IDE well, but not a modelling tool. We define *systems development tool* to be:

- Any tool that support systems development, including modelling tools, screen painters, integrated development environments (IDEs), as well as more restricted programming tools.

6.2 Modelling tool

By *modelling tool* we mean:

- A systems development tool supporting development/description/editing of models in a given modelling language. An example of such a tool is the parts of IBM's Rational Software Developer that supports drawing and documenting UML models.

6.3 Screen painter

Wikipedia (2011) describes *graphical user interface builder* as:

- A **graphical user interface builder** (or **GUI builder**), also known as **GUI designer**, is a software development tool that simplifies the creation of GUIs by allowing the designer to arrange widgets using a drag-and-drop WYSIWYG editor. Without a GUI builder, a GUI must be built by manually specifying each widget's parameters in code, with no visual feedback until the program is run.

This description includes the integration of drawing user interfaces and programming their behavior. This is on the one hand wise, as a pure drawing tool is of little practical use other than for paper prototyping, but on the other hand, including too much programming functionality makes such tools quite similar to IDEs. Thus, we make a more restricted definition of *screen painter* to be:

- A systems development tool supporting development of user interfaces through "painting" them on the screen.

6.4 Integrated development environment

Wikipedia (2011) describes *IDE* as:

- An **integrated development environment (IDE)** also known as *integrated design environment* or *integrated debugging environment* is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, a compiler and/or an interpreter, build automation tools and a debugger. Sometimes a version control system and various tools are integrated to simplify the construction of a GUI.

As can be seen by comparing this description with the description of GUI builder, an IDE focuses more on the programming process.

6.5 Transformation tool

Based on various definitions above, we define *transformation tool* as:

- A tool that transforms one software engineering artefact to another software engineering artefact, being of the same or a different type.

In a model-based software engineering setting, the artefacts are usually a combination of models and programs. This definition also covers tools like compilers.

6.6 Code generator

Based on various definitions above, we define *code generator* as:

- A transformation tool that transforms a model to source code or a representation that may be interpreted.

6.7 Relations between concepts

In Fig. 8, we have shown how the software engineering tool concepts defined in this section relate to each other and some of the basic, computer science and software engineering concepts.

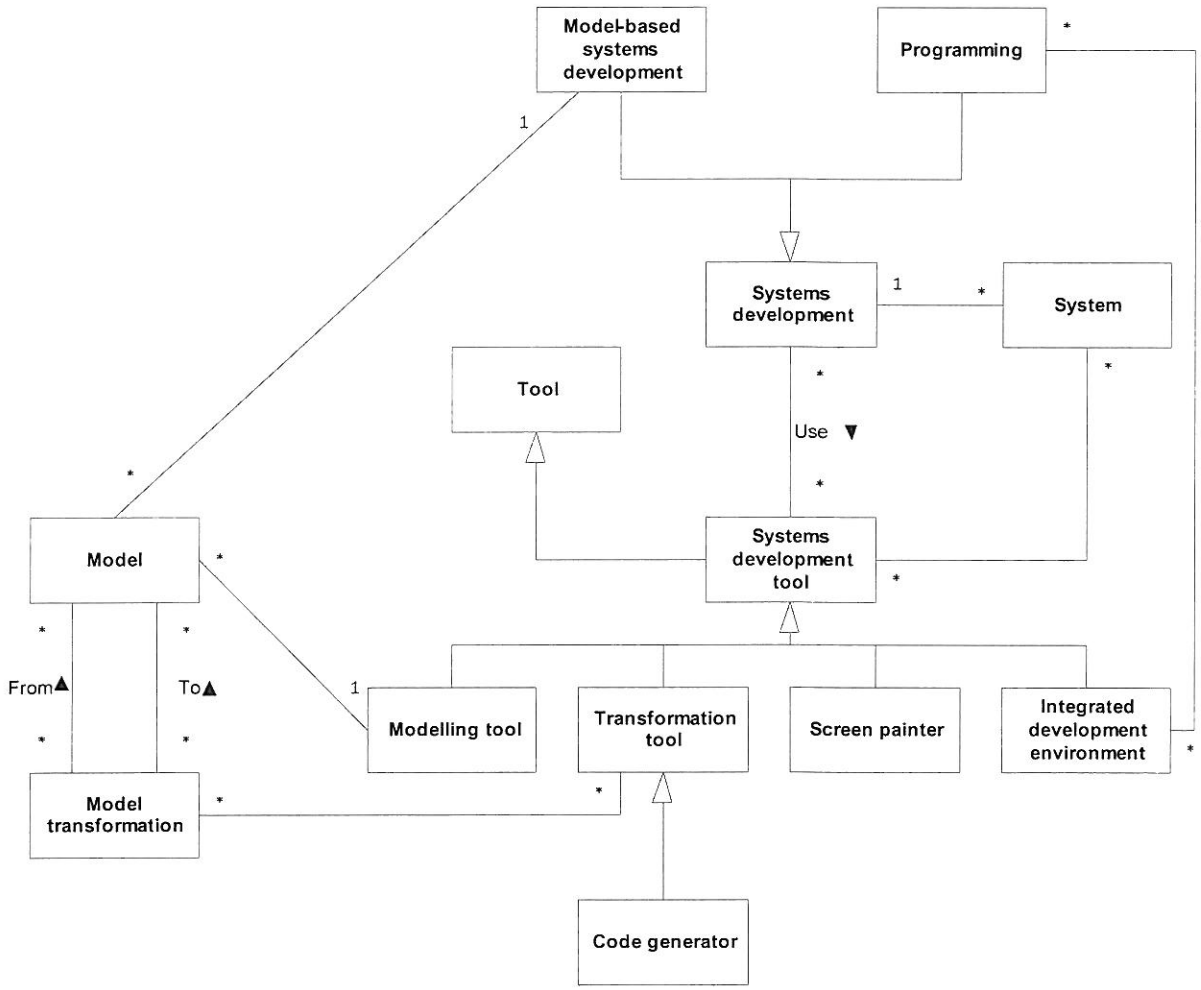

Fig. 8 - Software engineering tools

7 Modelling concepts

In this section the focus is on concepts describing modelling in software engineering, and thus the concepts are not user interface specific.

7.1 Meta model

ERCIM Working Group Software Evolution (2008) defines *meta model* as (referring to the MOF standard):

- According to the Meta-Object Facility (MOF) standard, a *metamodel* is a model that defines the language for expressing a model.

Although this definition of meta model is not used in the current version of the MOF standard (OMG, 2006), we find the definition useful. In the current version, a meta model is defined as "a model used to model modeling itself". This definition is supported in the Wikipedia (2011) entry on metamodelling in the context of software engineering, which describes *metamodelling* loosely, yet in a useful way:

- *Metamodeling* is the construction of a collection of *concepts* (things, terms, etc.) within a certain domain. A model is an abstraction of phenomena in the real world; a metamodel is yet another abstraction, highlighting properties of the model itself. A model conforms to its metamodel in the way that a computer program conforms to the grammar of the programming language in which it is written.

For our further definitions, it is useful to view a meta model both as something defining a language for expressing a model and as a model of a model, thus applying both these definitions. Furthermore, a meta model may also denote the schema of a repository for storing models.

7.2 Task model

Allen (1997) defines a *task model* as:

- A *task model* is a description of a task as well as strategies for completing that task.

Paternò (1999) adds the aspect of using an application for completing tasks, and defines *task models* as:

- Task models describe how activities can be performed to reach the users' goals when interacting with the application considered.

Task models are usually considered being a result of task analysis and/or task modelling. Paternò connects task analysis with the requirement or analysis phase of systems development, stating that the purpose of task analysis is to identify what the relevant tasks are, while he connects task modelling mainly to the design phase, stating that the purpose of task modelling is to build a model which describes precisely the relationship among the various tasks identified. These relationships include temporal and semantic ones.

Paternò's definition and considerations fit fairly well with our view on task models, but as we think that task models also should cover descriptions of tasks performed without ICT support, we use this simpler and wider definition:

- A model of tasks performed by users.

7.3 Domain model

Based on the definitions of domain and model above, we define *domain model* as:

- A model expressing relevant aspects of a given domain.

The term *conceptual model* is sometimes used as a synonym to domain model, but in our view conceptual model is a wider term. Also, the term *concept model* is sometimes used as a synonym to domain model, e.g. in the CAMELEON reference framework (Calvary et al, 2002, Calvary et al, 2003). In this framework, "domain models" is used as a common term for task and concept models, a use that is consistent with our definition.

7.4 Relations between concepts

In Fig. 9, we have shown how the modelling concepts defined in this section relate to each other and some of the basic concepts.

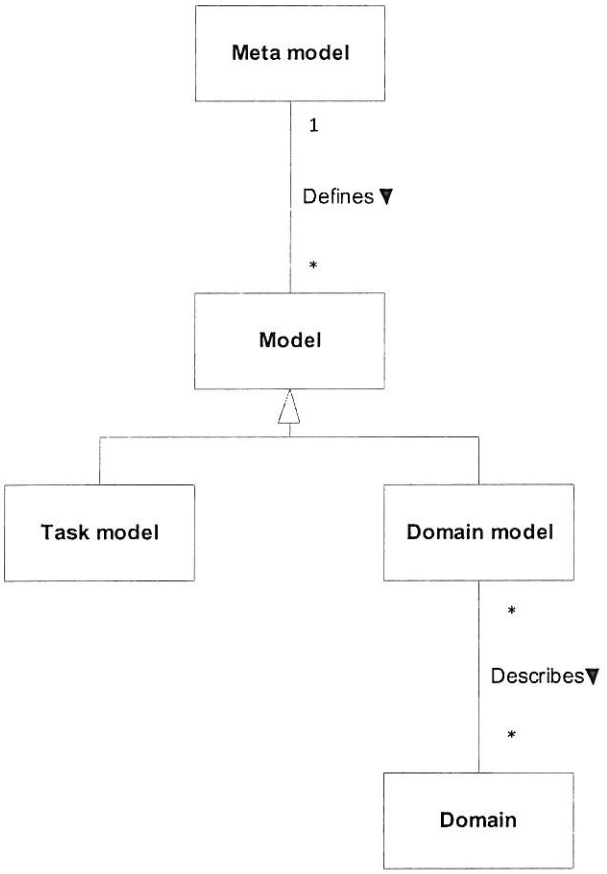

Fig. 9 - Models

8 General user interface concepts

In this section we define some basic user interface concepts. These concepts characterize technology and context for running user interfaces, and are thus independent of how user interfaces are developed.

8.1 User interface and Graphical user interface

The Free On-line Dictionary of Computing (2010) defines *user interface* as:

- The aspects of a computer system or program which can be seen (or heard or otherwise perceived) by the human user, and the commands and mechanisms the user uses to control its operation and input data.

WordNet (2011) defines *graphical user interface* as:

- A user interface based on graphics (icons and pictures and menus) instead of text; uses a mouse as well as a keyboard as an input device.

8.2 User interface type

We define *user interface type* as:

- A paradigm characterizing the look and feel of a large set of applications and services.

Examples: Character-based UIs (also called command-line interfaces), Graphical UIs (GUI), Web UIs (WUI).

This definition is supported by the Wikipedia (2011) article on user interface.

8.3 User interface style

We define *user interface style* as:

- A way of presenting information and offering interaction mechanisms to the user exploiting a uniform visual presentation and interaction techniques. User interface styles are usually mixed on implementation platforms, and usually also in applications, but less often in individual windows and dialogs.

Examples: map-based, forms-based, text-based.

The notion of user interface style is used in a similar way in the Interaction-Design.org Encyclopedia (2011) article on interaction styles, using command entry, form fill-in, menu selection and direct manipulation as examples, thus having styles that are a bit more general than our.

8.4 User interface modality

Based on the definitions of modality and user interface above, we define *user interface modality* as:

- Interaction mechanisms exploiting a limited set of modalities.

Examples: pen-based, speech-based, gesture-based.

8.5 Multi modal user interface

In the CAMELEON glossary (2003), *multi-modal user interface* is defines as:

- A user interface that supports multi-modality.

Furthermore, *multi-modality (of a user interface)* is defined as:

- Capacity of a system to support multi-modal interaction, i.e., the user is provided with more than one modality (simultaneously or not) to observe the system state and/or can use more than one modality (simultaneously or not) to communicate information to the system.

8.6 Target

The CAMELEON reference framework (Calvary. 2002) defines a *target* to be:

- A "triple of the form "e, p, u" where e is an element of the environments set considered for the interactive system, p is an element of the platforms set considered for the interactive system, u is an element of the users set for the interactive system.

With regards to our needs, this definition covers too much. Thus, we want to restrict the definition to aspects of the user interface and the platform on which it runs:

- An arbitrary combination of platform, type, style and modality used in a running user interface.

8.7 User interface component

Based on the definitions of component and user interface modality above, we define *user interface component* as:

- A component representing user interface functionality.

Compared to the way component is usually used in computer science, user interface components tend to be rather "small" components, like labels, buttons and various containers. User interface components representing more complex compositions are more seldom, and are mostly used for standardized dialog boxes like a file open dialog. User interface components are often also denoted widget, gadget or user interface control.

8.8 Abstract user interface component

Based on the definitions above, we define *abstract user interface component* as:

- A user interface component that is described independently of the target.

8.9 Concrete user interface component

Based on the definitions above, we define *concrete user interface component* as:

- A user interface component that is target specific.

8.10 Program-based user interface development

Based on the definitions above, we define *program-based user interface development* as:

- Development of the user interface part of an application using programming as a central means.

This definition implies that such development typically utilizes integrated development environments. It does not exclude the use of screen painters, as these usually must be supplemented by programming, certainly to specify behaviour, but also to some extent to specify or change the presentation.

8.11 Model-based user interface development

Based on the definitions above, we define *model-based user interface development* as:

- Model-based systems development focusing on development of the user interface part of an application.

8.12 Relations between concepts

In Fig. 10 through Fig. 12, we have shown how the general user interface concepts defined in this section relate to each other and some of the concepts defined in the previous sections.

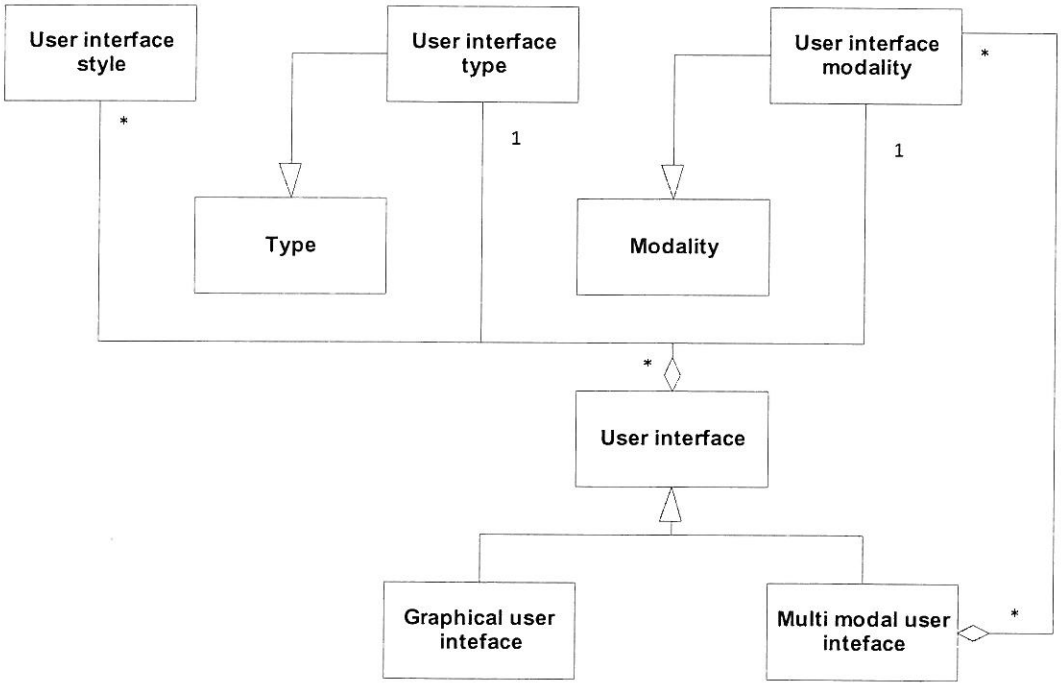

Fig. 10 - User interface, style, type and modality

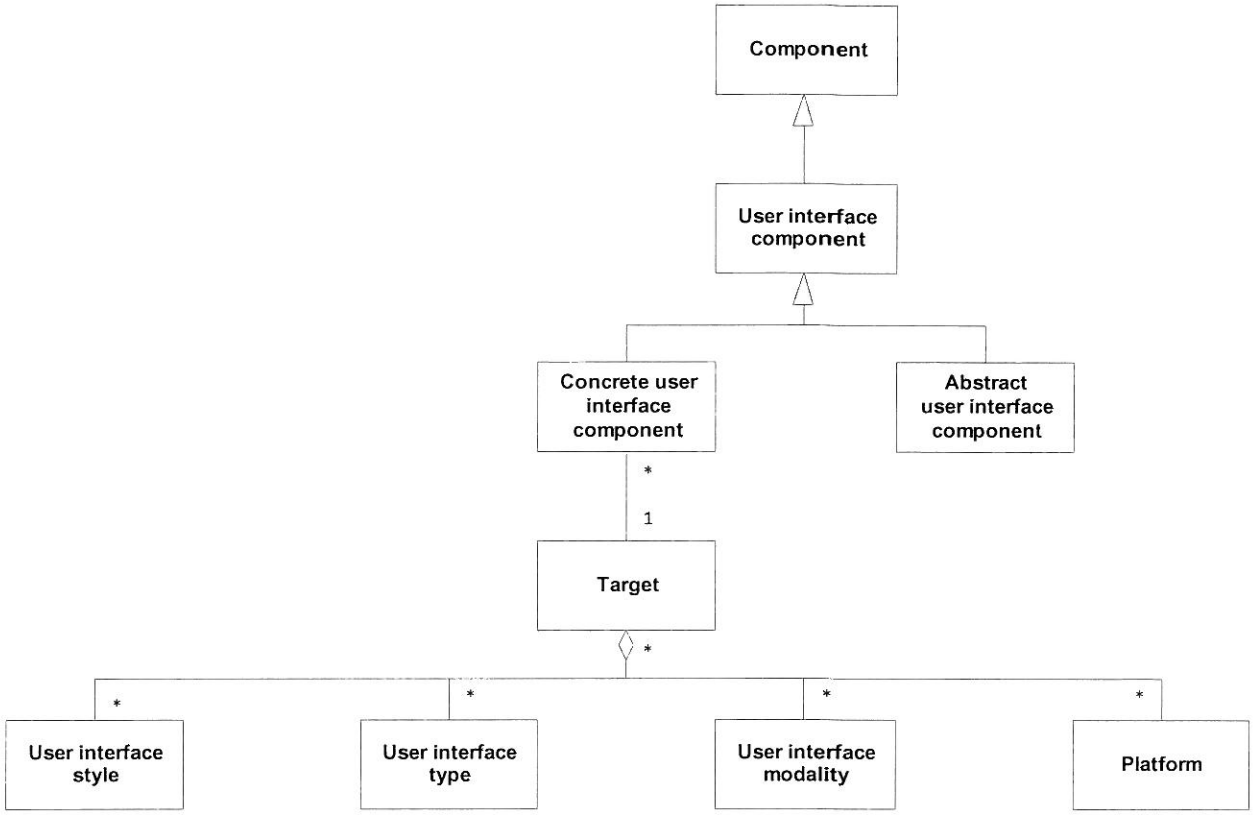

Fig. 11 - User interface component and target

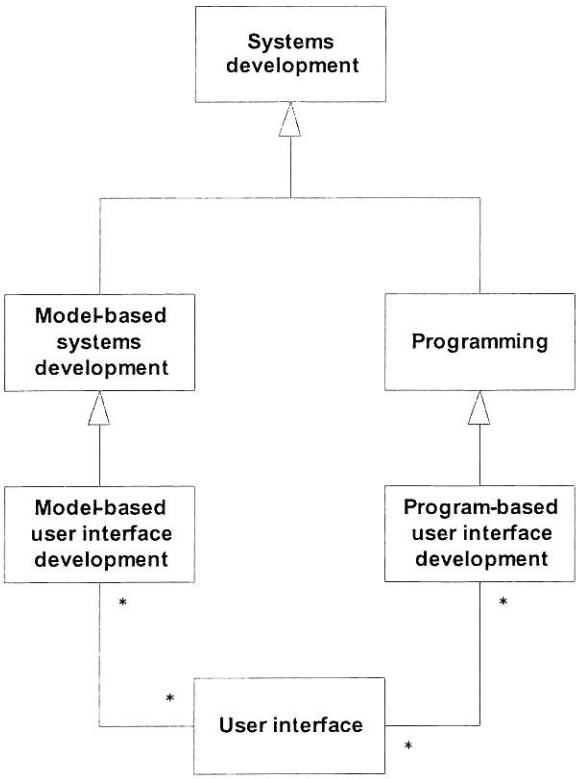

Fig. 12 - User interface development

9 Model-based user interface development concepts

In this section we define central concepts for model-based development of user interfaces.

9.1 Interactor

In the CAMELEON glossary (2003) two definitions of *interactor* are given:

- An abstraction of a software component that allows users to manipulate and/or observe domain concepts and functions.
- A computational abstraction that allows the rendering and manipulation of entities (domain concepts and/or tasks) that require input and output resources.

The first definition is quite close to our definition of abstract user interface component above. The second definition is wider, and in line with Trætteberg's (2002) definition of an *interactor*:

- A generic mediator of information, between the system and user, in both directions, or the generic component from which such a mediator is built.

The second CAMELEON definition as well as Trætteberg's definition allow "larger" interactors that are compositions of other interactors, and thus we find it more flexible. Modelling languages following the first CAMELEON definition usually need special modelling concepts for handling containment.

9.2 Abstract interactor

Based on the definitions of interactor above, we define *abstract interactor* as:

- An interactor that is defined independently of a target.

9.3 Concrete interactor

Based on the definitions of interactor above, we define *concrete interactor* as:

- A target-specific interactor.

9.4 User interface model

Based on the definitions of user interface and model above, we define *user interface model* as:

- A model expressing one or more aspects of a user interface.

User interface model should be considered an abstract concept, i.e. it is a common term for a set of models that describe different aspects of user interfaces. This means that there will never be any "instances" of user interface models. Examples of types of user interface models are *presentation models, dialog models* and *user model* (see CAMELEON glossary (2003), Nilsson (2001), Szekely (1996), Trætteberg (2002) and Wikipedia (2011) for more comprehensive lists). It should also be mentioned that the term user interface model is often also used to denote models that do not express aspects of a user interface directly, but that rather are used as basis for making other user interface models. Examples of such models are *task models* and *domain models.*

9.5 Presentation model

Calvary et al (2003) define *presentation model* as:

- A presentation model specifies presentation aspects of the resulting UI in abstract terms and how to invoke them.

A presentation model is usually manifested as instance hierarchies of some type of building blocks. The concepts used for these building blocks vary between modelling languages, but usually involves some sort of abstract interactors as well as different types of abstract containers.

9.6 Dialog model

Paternò (1999) defines *dialog model* as:

- An abstract description of the actions, and their possible temporal relationships, that users and systems can perform at the user interface level during an interactive session.

Such a description often contains references to different (parts of) presentation models and how the user may navigate (in an abstract sense) between them.

9.7 User interface specification

Based on the definitions of user interface and specification above, we define *user interface specification* as:

- A specification of one ore more aspects of a user interface.

Unlike user interface model, user interface specification should not be considered an abstract concept. As with any specification in software engineering, specific user interface models may be an important part of a user interface specification.

9.8 User interface modelling language

Based on the definitions of user interface and modelling language above, we define *user interface modelling language* as:

- A language to express user interface models

Syntax, semantics and pragmatics of such a language may in the same way be defined as specializations of the corresponding definitions in the section on software engineering concepts above.

9.9 User interface specification language

Based on the definitions of user interface and specification language above, we define *user interface specification language* as:

- A language to express user interface specifications

Syntax, semantics and pragmatics of such a language may in the same way be defined as specializations of the corresponding definitions in the section on software engineering concepts above.

9.10 Abstract user interface

Calvary et al (2003) define *abstract user interface* as:

- A canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent from the concrete interactors available on the targets.

They also describe an abstract user interface to be a collection of user interface models. Szekely (1996) use the concept *abstract user interface specification* quite similarly as Calvary et al use abstract user interface. We find that omitting specification makes the concept more flexible. We prefer to define *abstract user interface* as:

- A collection of user interface models describing the user interface part of an application using abstract interactors.

9.11 Concrete user interface

Calvary et al (2003) describes a concrete user interface as being dependent on target-specific concrete interactors. Szekely (1996) use the concept *concrete user interface specification* quite similarly as Calvary et al use concrete user interface. We find that omitting specification makes the concept more flexible. We prefer to define *concrete user interface* as:

- A collection of user interface models describing the user interface part of an application using concrete interactors.

9.12 Final user interface

Calvary et al (2003) describes a *final user interface* as being generated from a concrete user interface, and expressed in source code. In our view, there should not be a need for source code, as it should be possible to interpret a final user interface. A more general definition of *final user interface* is:

- The resulting user interface from using model-based user interface development, i.e. the user interface intended for end users.

9.13 Relations between concepts

In Fig. 13 through Fig. 17, we have shown how the model-based user interface development concepts defined in this section relate to each other and some of the concepts defined in the previous sections.

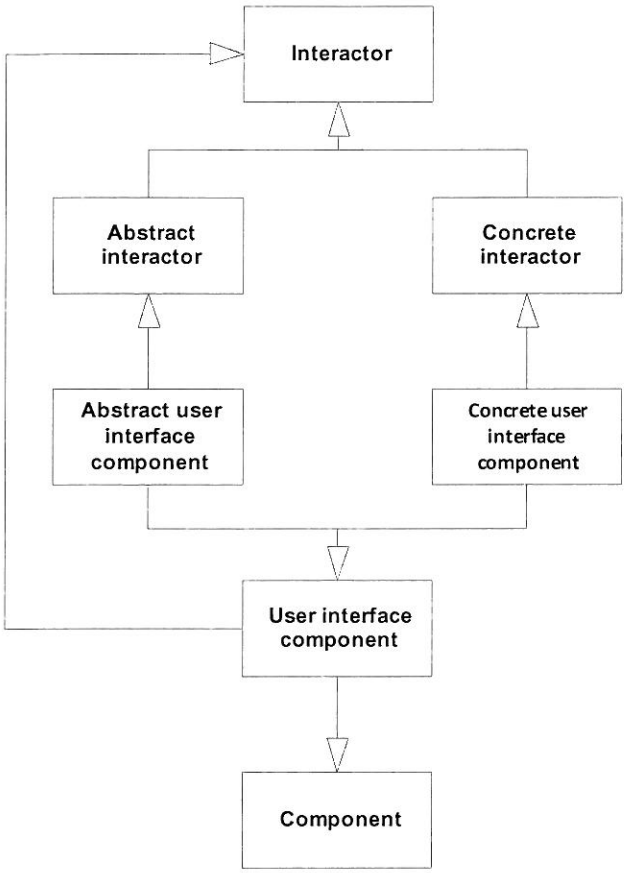

Fig. 13 - Interactor

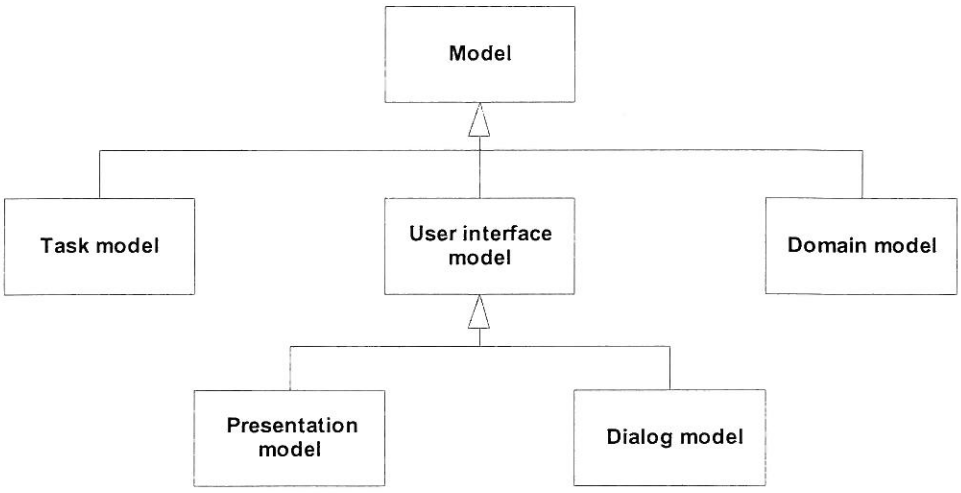

Fig. 14 - User interface models

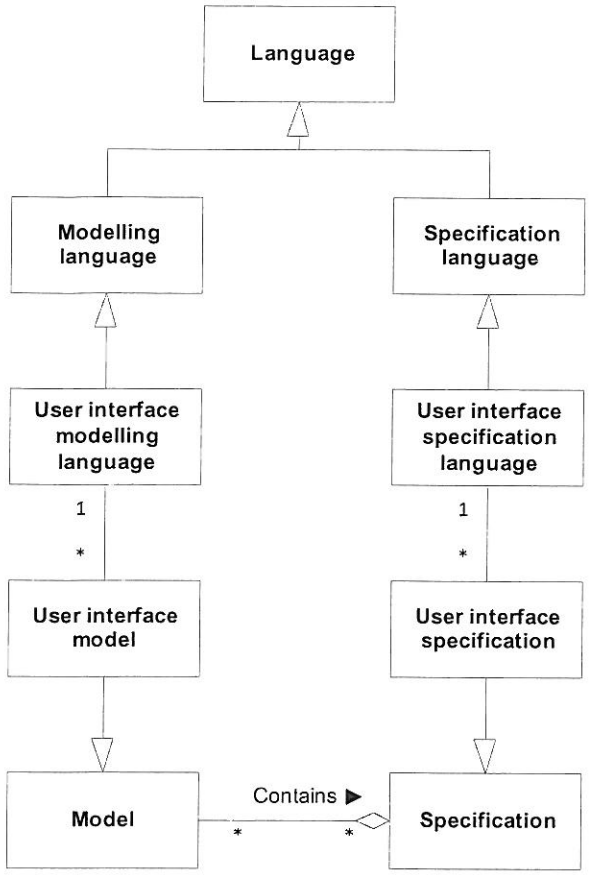

Fig. 15 - User interface model and specification languages

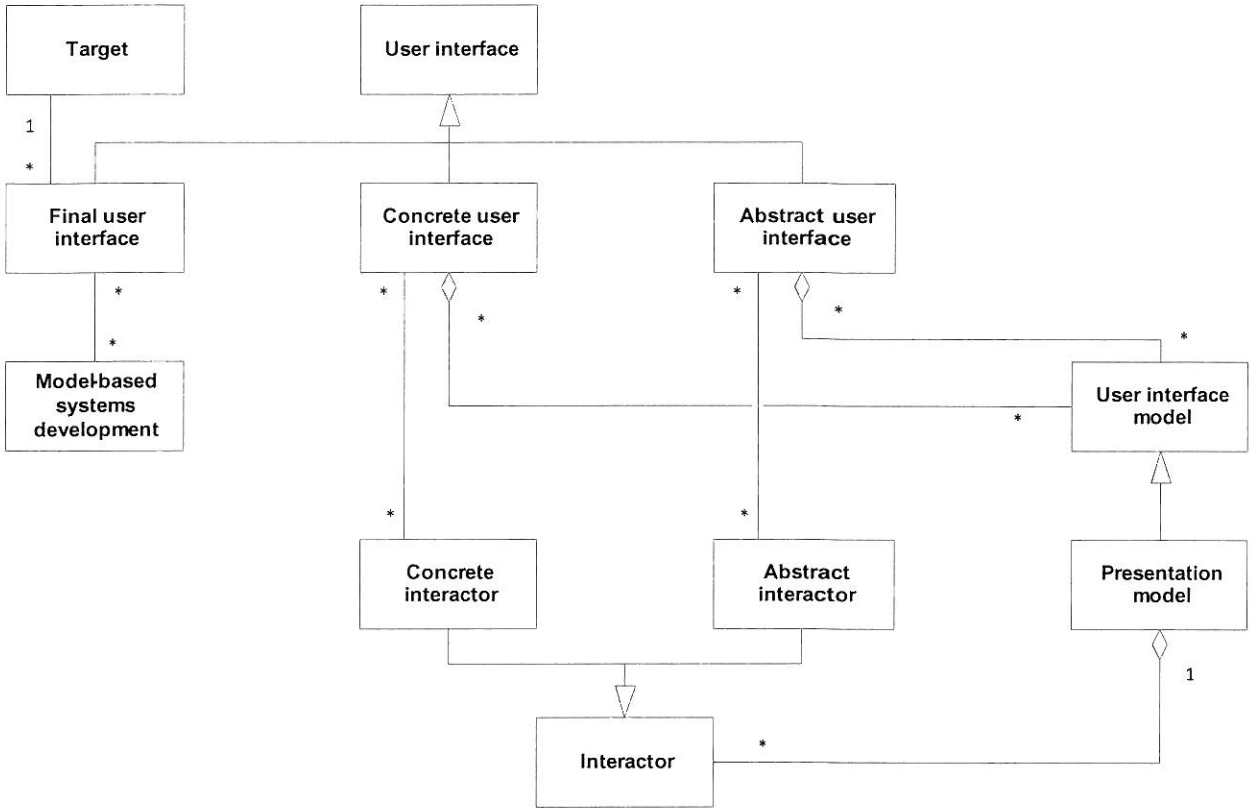

Fig. 16 – Abstract, concrete and final user interfaces

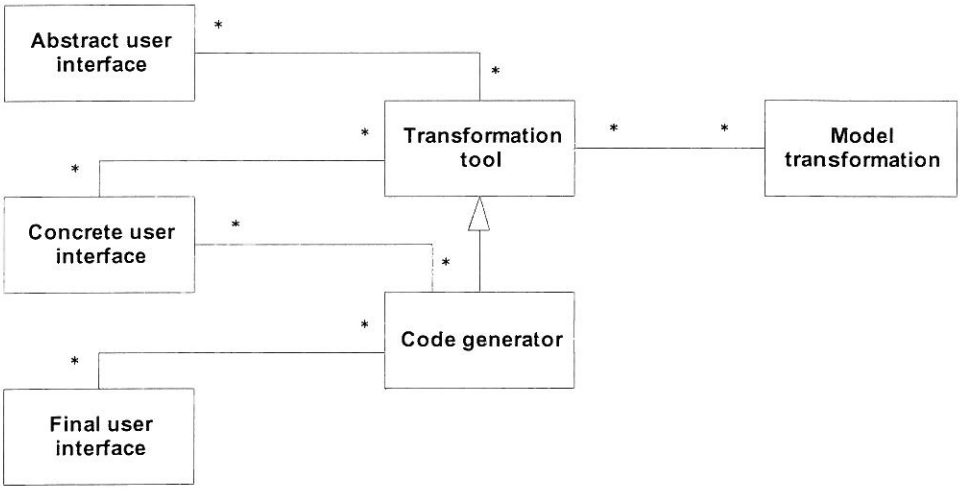

Fig. 17 – Transformations

10 Conclusions and future research

In this report, we have presented a vocabulary for model-based user interface development, where the concepts are built from more general concepts, partly being independent of computer science and partly collected from different fields within computer science, software engineering and user interfaces. The vocabulary defines model-based user interface development concepts in a general way, independently of application domains.

In our work on efficient development of highly flexible user interfaces supporting emergency response, using model-based user interface development approach is one of several possible approaches. The vocabulary will be an important basis for assessing how well existing approaches for model-based user interface development approaches are able to handle the special challenges in the emergency response domain (Nilsson & Stølen, 2010), including the applicability of further developing our prior work on model-based user interface development (Nilsson et al. 2006), as well as assessing whether deficiencies in these approaches may be overcome using different development approaches, like domain specific models.

Acknowledgements

The work on which this report is based is supported by the EMERGENCY project (187799/S10), funded by the Norwegian Research Council and the following project partners: Locus AS, The Directorate for Civil Protection and Emergency Planning, Geodata AS, Norwegian Red Cross, and Oslo Police District.

Alphabetic list of concepts in the vocabulary

References

Allen, Robert B. (1997): *Mental Models and User Models;* In Helander et al (ed.): Handbook of Human-Computer Interaction, second edition, North-Holland, 1997, p. 49-63

The American Heritage Dictionary of the English Language; Fourth Edition; 2009. Houghton Mifflin Company. Accessed through http://www.thefreedictionary.com/ (accessed February 18, 2011)

Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., Vanderdonckt, J. (2002): *The CAMELEON Reference Framework*; Deliverable 1.1, CAMELEON Project; Available at: http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf (accessed Feb. 9, 2011)

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. (2003). *A Unifying Reference Framework for Multi-Target User Interfaces;* Interacting with Computer 15,3 (2003) 289–308

Collins English Dictionary, Complete and Unabridged; HarperCollins Publishers. 2003. Accessed through http://www.thefreedictionary.com/ (accessed February 18, 2011)

CAMELEON glossary (2003); Deliverable 1.1 Companion, CAMELEON Project; Available at: http://giove.isti.cnr.it/projects/cameleon/glossary.html (accessed March 4, 2011)

ERCIM Working Group Software Evolution (2008): *Terminology*; Available at http://wiki.ercim.eu/wg/SoftwareEvolution/index.php/Terminology (accessed Feb. 18, 2011)

The Free On-line Dictionary of Computing (2010); Available at http://foldoc.org/ (accessed Feb. 18, 2011)

Interaction-Design.org Encyclopedia (2011); Available at http://www.interaction-design.org/encyclopedia/ (accessed Feb. 18, 2011)

Merriam-Webster Dictionary (2011); Available at http://www.merriam-webster.com/ (accessed February 18, 2011)

Nilsson, E. G. (2001): *Modelling user interfaces – challenges, requirements and solutions;* Proceedings of Yggdrasil 2001 – Norwegian Computer Society's annual conference on user interface design and user documentation.

Nilsson, E. G., Floch, J., Hallsteinsen, S. & Stav, E. (2006): *Model-based User Interface Adaptation;* Elsevier Computers & Graphics, 30 (5) 2006, p. 692-701

Nilsson, E.G. and Stølen, K. (2010): *Ad Hoc Networks and Mobile Devices in Emergency Response – a Perfect Match?* Proceedings of Second International Conference on Ad Hoc Networks, Victoria, British Columbia, Canada

OMG (2006): *MOF Core specification;* Available at http://www.omg.org/spec/MOF/2.0/ (accessed February 18, 2011)

OMG (2008): *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2;* Available at http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/ (accessed February 18, 2011)

Oviatt, S. (2002): *Multimodal interfaces;* In Jacko, J. & Sears, A. (Eds.) The Human-Computer Interaction Handbook. Lawrence Erlbaum, 2002, p. 286-304

Paternò, F. (1999): *Model-based Design and Evaluation of Interactive Applications;* Springer, 1999

Pilone, D. (2005): *UML 2.0 in a Nutshell;* O'Reilly, 2005

Random House Dictionary; Random House, 2011; Accessed at http://dictionary.reference.com/ (accessed February 18, 2011)

Szekely, P. (1996): *Retrospective and Challenges for Model-Based Interface Development;* In Computer-Aided Design of User Interfaces – Proceedings of CADUI '96

Trætteberg, H. (2002): *Model-based User Interface Design;* PhD thesis, NTNU; ISBN 82-471-5459-5

Wikipedia, The Free Encyclopedia (2011); Available at http://en.wikipedia.org/wiki/Main_Page (accessed February 18, 2011)

Wiktionary, the free dictionary (2011); Available at http://en.wiktionary.org/wiki/Wiktionary:Main_Page (accessed Feb. 18, 2011)

WordNet, a Lexical Database for English (2011); Available at http://wordnet.princeton.edu/ (accessed February 18, 2011)