Visually effective goal models using KAOS

Conference Paper · November 2007

DOI: 10.1007/978-3-540-76292-8_32 · Source: DBLP

2 authors:

Raimundas Matulevičius
University of Tartu

Patrick Heymans
University of Namur

FUNDP – PRECISE
University of Namur
Institut d'Informatique
Rue Grandgagnage, 21
B – 5000 NAMUR, Belgium

TECHNICAL REPORT

Visually Effective Goal Models using KAOS

Raimundas Matulevičius and Parick Heymans

May-July, 2007

Visually Effective Goal Models using KAOS

Raimundas Matulevičius and Patrick Heymans

PReCISE Research Center, Computer Science Department, University of Namur,
rue Grandgagnage 21,5000 Namur, Belgium
{rma, phe @ info.fundp.ac.be}

Abstract. Goal modelling languages are visual modelling languages. To communicate effectively ideas with a visual modelling language, one should follow some of basic principles. One is modularity, i.e. organising diagrams in manageable modules to avoid confusing the reader with overly complex diagrams. Another is emphasis, i.e. visually drawing the attention to the most important pieces of information. In this paper, we evaluate how the goal modelling language KAOS and its supporting tool, Objectiv*er*, help modellers respect nine visual modelling principles. From our observations, we formulate recommendations for modellers, language designers and tool developers.

1 Introduction

Goal modelling languages (GMLs) are visual modelling languages used primarily during the early stages – a.k.a. Requirements Engineering (RE) – of information system (IS) development. RE seeks to identify the requirements that the future IS has to fulfil and the constraints under which it has to operate. As of today, failing to understand the stakeholders' requirements is still the primary reason of project failure.

In this context, GMLs appear to be useful means to facilitate the identification, structuring and validation of requirements. GMLs introduce new abstractions, most notably the notion of goal, to supplement other more traditional abstractions used in data and process modelling. Goal models make the purpose and rationale of the new IS explicit, thereby preventing the development team to waste time on detailed technical descriptions if the goals are not yet clear, and allowing them to justify each detailed requirement by reference to goals.

Over time, several GMLs have been proposed, together with their supporting methods and tools. Those languages include *i** [24], TROPOS [5], NFR [6], KAOS [12, 23], GBRAM [2], and Lightswitch [21]. Various applications [1, 14] and evaluations [3, 10, 11, 15, 22] of GMLs were also reported. Following [13], we situate qualitative assessments of GMLs along three dimensions: syntax, semantics and pragmatics (see Fig. 1).

In this paper, as in [3] and [8], we examine the link between the *syntax* of GMLs – which constructs they propose and how these can be combined – and their *semantics*– how models are understood by their audience. However, whereas [3] and [8] deal with the *abstract* syntax (metamodel) of GMLs, we concentrate on their *concrete*, or surface, syntax. More precisely, we investigate the relationship between the visual display of goal models and how humans understand them.

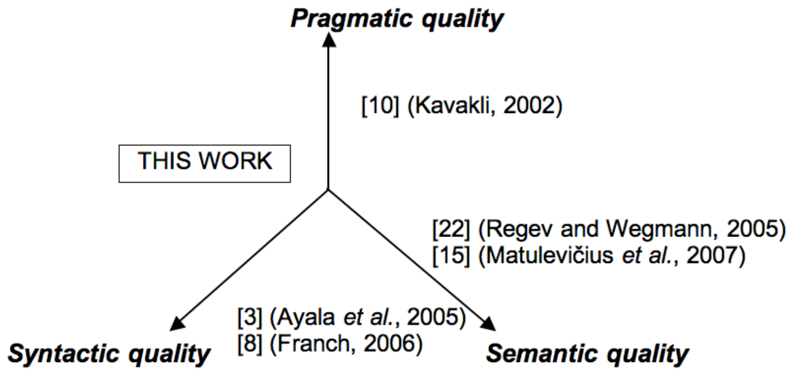


Fig. 1. Classification of GML research

Good practices for the definition and usage of visual modelling languages have been a subject of research for several decades now (see e.g. [4, 7, 19, 20]). In this work, we use a set of nine basic principles recently formulated by Moody in [17]. Those principles consolidate a large body of knowledge originating from disciplines such as human-computer interaction [19] and cognitive psychology [4]. One of these principles is modularity, i.e. organising diagrams in manageable modules to avoid confusing the reader with overly complex diagrams. Another is emphasis, i.e. visually drawing the attention to the most important pieces of information.

This paper evaluates how easy it is to follow the principles using the current version of the KAOS (Knowledge Acquisition in autOmated Specification) language [12, 23] and its associated tool, Objectiv*er* (release 2.0.0 professional edition). The resulting critical analysis of KAOS' and Objectiv*er*'s visual abilities aims to formulate recommendations (*i*) for KAOS modellers, (*ii*) for language engineers, and (*iii*) for tool developers. Although this paper is not a comparative study, it is a first step towards such a systematic comparison of different GMLs.

Section 2 describes KAOS. Section 3 recalls the nine principles [17]. Section 4 analyses how the principles are addressed in KAOS and Objectiv*er*. Section 5 formulates the recommendations. Section 6 discusses conclusions and future work.

2 KAOS

In this work we are using KAOS as defined in [12, 23]. The KAOS approach consists of a modelling *language*, a *method*, and a *software environment*. The main purpose of KAOS is to ensure that high-level goals are identified and progressively refined into precise operational statements. These are then assigned to agents of the software-to-be and its environment, both forming the so-called system-to-be. Along this process, various alternative goal assignments and refinements are considered until the most satisfactory solution is chosen.

A KAOS model consists of four kinds of diagrams: goal, object, agent and operation. KAOS constructs have a graphical (Fig. 6) and a textual syntax (Fig. 2). Selected constructs (e.g., goal, operation) can be further defined using the KAOS real-time temporal logics facilitating reasoning (Fig. 2, FormalDef). The major construct in KAOS is goal, which is a prescriptive assertion that captures an objective that the system-to-be should meet. A goal can be refined through G-refinement, which relates it to subgoals whose conjunction contributes to the satisfaction of the goal. A goal can have alternative G-refinements, which result in different software designs. Goals are refined until they are assigned to individual agents. A goal effectively assigned to a software agent is called a requirement. If a goal is operationalised and has a responsible agent, the latter performs the operations.

Goal Achieve [Meeting held]
Def Each requested meeting is eventually being held with the presence of all intended participants.
FormalDef
$\forall$m : Meeting: m.Requested
$\Rightarrow\Diamond$ m.Holds $\wedge$ ($\forall$ p: Participant): Intended (p, m) $\rightarrow$ Participates (p, m)

Fig. 2. Textual goal syntax. A goal has a name (Meeting held), a natural language definition (Def), and optional attributes like pattern (e.g. Achieve) and formal definition (FormalDef) [12]

In this work we focus on the KAOS graphical syntax. Fig. 7 represents an excerpt of the meeting scheduler model [12]. The goal Meeting held is refined into two subgoals: Participant informed and Participant info known. The latter is further refined into Participant agenda is up to date and Participant info known from agenda. The agent Scheduler is responsible for Participant informed to become true. The agent performs operations Inform about the time and Inform about the place to fulfil the requirement.

3 Principles of Effective Diagrams

The principles for the effective diagrams [17] introduce how a diagram should be prepared manipulating eight visual variables (vertical position, horizontal position, shape, colour, size, value, orientation, and texture) in order to communicate effectively *wrt* a "model of human graphical information processing, which reflects current research in human cognition and visual perception" [17]. We start with the principle of

discriminability. There are two types of discriminability. *Absolute* discriminability is reader's ability to separate diagram elements from the background (see Fig. 3). It depends on element size (a), element proximity (b), and diagram contrast (c and d). *Relative* discriminability is the reader's ability to differentiate between different element types. It relies on by the use of shapes, lines and visual variables. There are five basic geometric signs [9] – square (e.g., diamonds and rectangles are square variations), triangle, circle, cross and arrows – which are not likely to be confused. For example, KAOS uses three basic geometric signs: squares (e.g., goal and agent), circles (e.g., operation) and arrows (e.g., G-refinement, operationalisation, and assignment). Shape, colour, orientation, thickness and colour of borderline also play a very important part. For instance, in Fig. 6 constructs goal and requirement are discriminated only by borderline thickness (see also Table 2).

Modularity (or **decomposition**), defines how a diagram is organised into cognitively manageable modules, or "chunks", that would reduce diagram complexity. In order to avoid cognitive overload, diagrams should be limited to seven plus/minus two elements [17]. In our example (Fig. 6) we respect these boundaries; however, we note that goal models can quickly become complex, and the modularity is highly dependent on the modeller's skills.

Structure organises diagram elements into distinct perceptual groups. Elements in a diagram can be structured by proximity, similarity, or common region. For example, in Fig. 4 operations Inform about the time and Informed about the place are structured by *similarity* (because of labels and shape), *proximity* (because they are physically close to each other) and *common region* (because they belong to the region Participants informed).

Structuring is "an alternative and a complement to decomposition" [16]: instead of dividing a diagram into manageable modules, elements can be organised into groups.

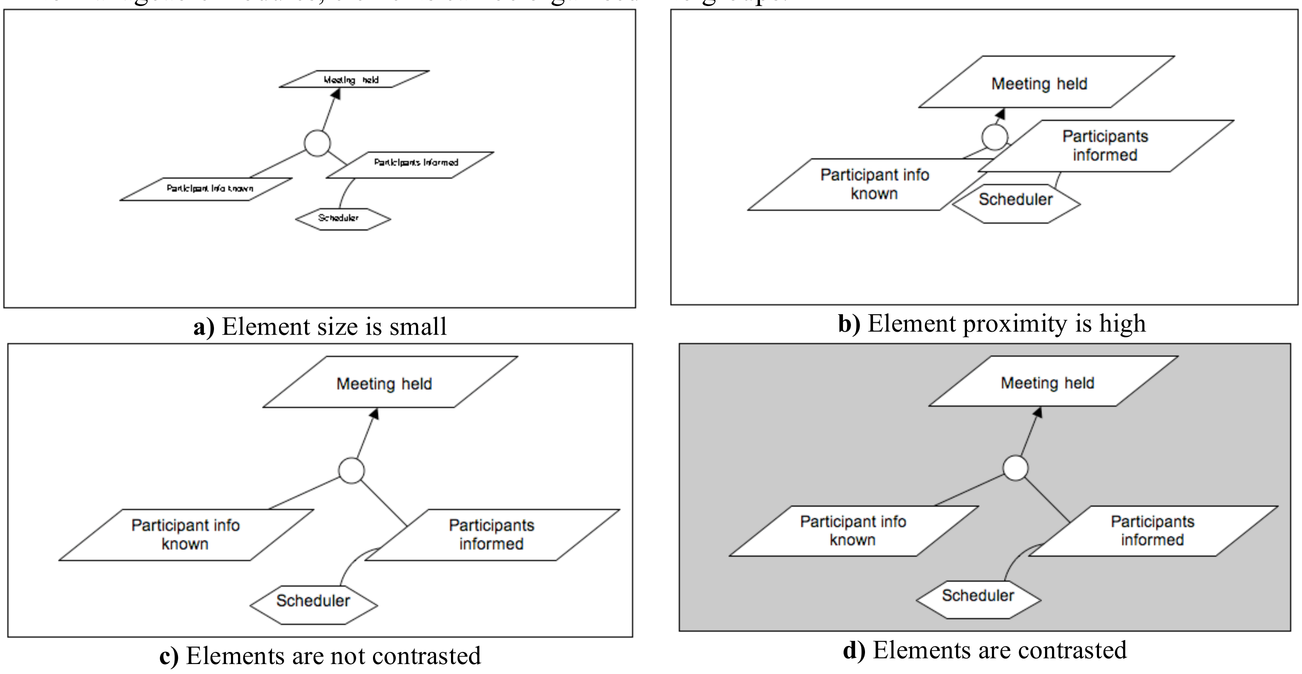


Fig. 3. Examples of absolute proximity principle

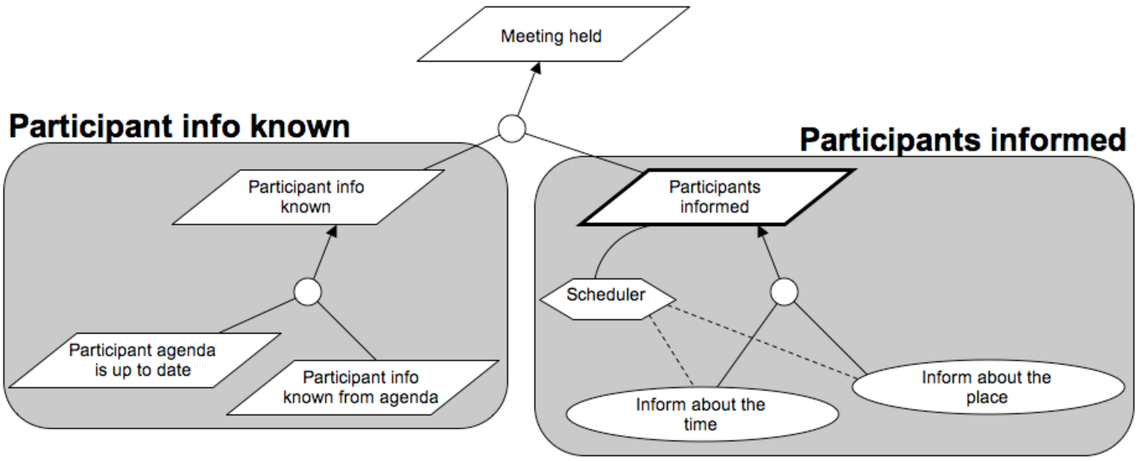
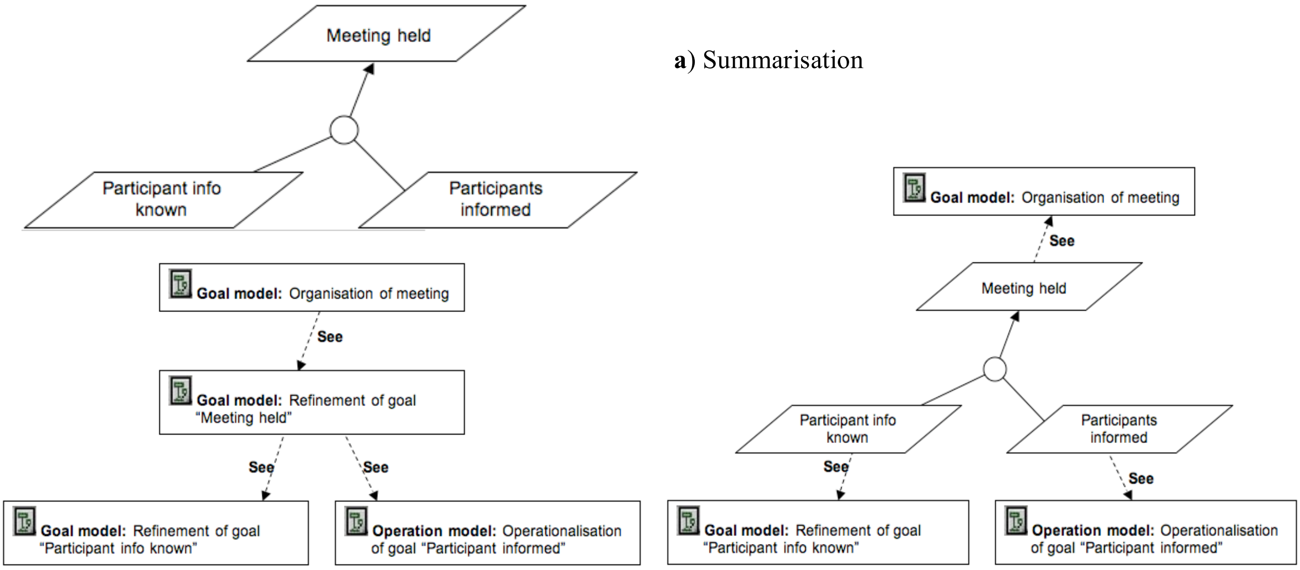


Fig. 4. Structuring the KAOS diagram

Cognitive integration deals with understanding the overall information covered by the whole set of diagrams. A model usually consists of multiple diagrams. Cognitive integration describes how different pieces of information are integrated from various diagrams. *Summarisation* is the process of creating more abstract representations of information (Fig. 5a). A *navigation map* is a representation of the entire system of diagrams and the navigation paths between them (Fig. 5b). *Signposting* includes navigation clues to show diagram transitions, in such a way providing user awareness of where they are in the system of diagrams (Fig. 5c). Other cognitive integration techniques are discussed in [16].



a) Summarisation

b) Navigational map **c)** Signposting

Fig. 5. Cognitive integration techniques supported in KAOS/Objectiv*er*

Emphasis is about drawing attention to the most important information presented in a diagram. Emphasis is made by visual variables, like shading, size, and colour of the element, font size, value and colour of the label, colour of the diagram background. Fig. 6 illustrates emphasis using font size of the label and by shading the element background (goal "Meeting held").

Perceptual directness describes the use of representations that have direct interpretation. In case of very abstract concepts, perceptually direct representations are difficult to find. Hence, arbitrary representation conventions are made, and a legend often facilitates remembering those conventions (Fig. 6).

Identification is about clear diagram labelling with title, type, and legend. *External* identification defines the correspondence between the diagram and the represented world. In Fig. 6 the diagram has a *name*: Refinement of goal "Meeting held". The diagram *type* is identified before its name, in bold **Goal model**. *Internal* identification defines the correspondence between graphical conventions and their meaning. In Fig. 6 all element types used in the diagram are indicated in its *legend*.

Visual expressiveness refers to the visual variables used to encode information. Visual variables may increase perceptual representation, accuracy and draw attention and interest. However the modellers need to be careful not to violate other principles (e.g., discrimination, emphasis, and structuring). All variables should be held constant or normalised. This helps avoiding undesirable and unintended messages of the diagram. In Fig. 6 all relative elements are normalised by size.

Graphical simplicity is about minimising the number of different conventions used. The span of absolute judgment (the ability to discriminate between perceptually distinct alternatives) is around 7 plus or minus two. In Fig. 6 we use eight KAOS constructs. In addition to graphical notations, textual information using attributes (not appearing in diagram, see Fig. 2) could be defined.

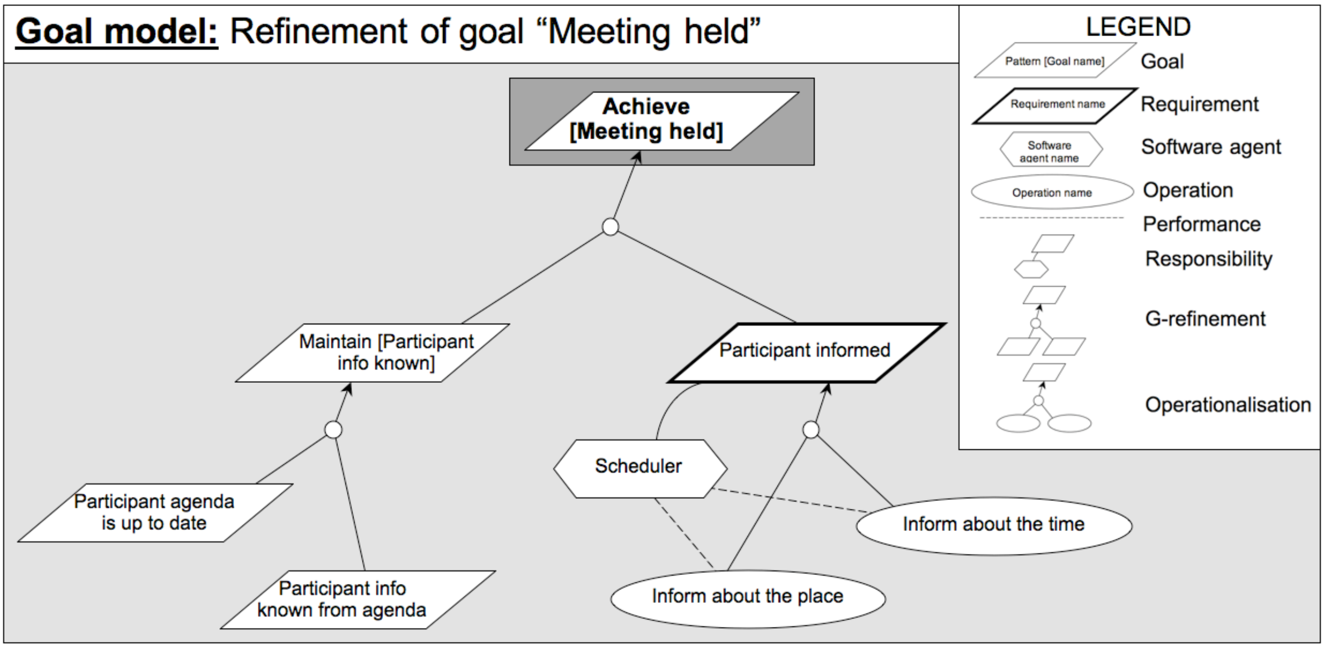
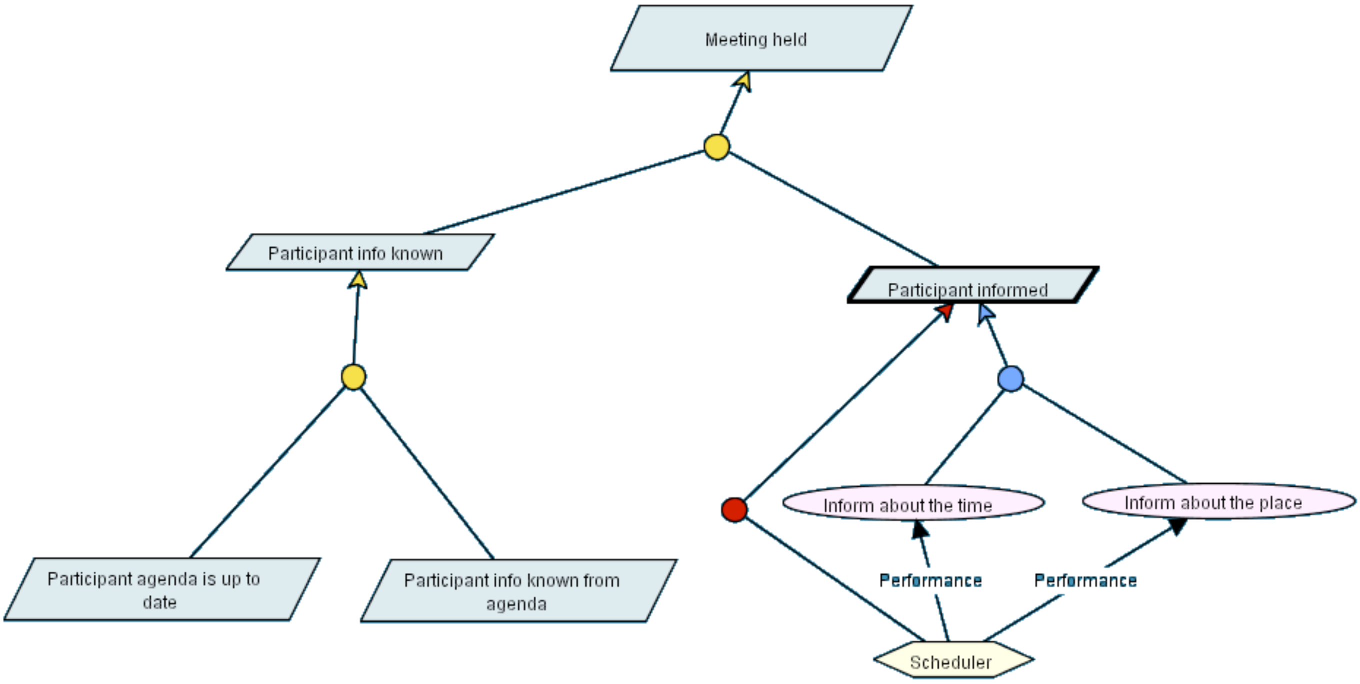


Fig. 6. KAOS diagram with drawing tools (MS Word's graphical editor)

4 KAOS/Objectiv*er* Evaluation

Following the principles for effective diagrams, in Fig. 6 we present a KAOS goal model created using *drawing tools* (MS Word's graphical editor). However, this was time consuming. Thus in this section we investigate how the principles for effective diagrams can be fulfilled with Objectiv*er*. The resulting diagram is shown in Fig. 7.



Refinement of goal "Meeting held"

Fig. 7. KAOS diagram with Objectiv*er*

KAOS includes constructs from graph and iconic classes [7], which are manipulated using visual variables (Tables 1 and 2). KAOS adopts squares, circles and arrows (Table 1). Visual variables (Table 2) are important for construct *discrimination* when modelling; but some of them (e.g. colour) play no role when a diagram is printed using a black and white printer (Fig. 7). Objectiv*er* does not provide means for contrasting elements versus background. To discriminate absolutely, the modeller can manually use element proximity and size; however, s/he has to be careful not to neglect other principles.

Table 1. Construct variation

Construct variation	KAOS/Objectiv*er*
Square	Goal Agent
Circle	Operation
Arrow	G-refinement Operationalisation Responsibility Output Input

Table 2. Relative discriminability of the KAOS/Objectiv*er* goal-related constructs

Construct	Shape	Background colour	Border line colour	Border line thickness	Orientation
Goal	Parallelogram	Light blue	Black	Thin	Right
Softgoal	Parallelogram	Light blue	Blue	Thin	Right
Requirement	Parallelogram	Light blue	Black	Thick	Right
Expectation	Parallelogram	Yellow	Black	Thick	Right
Obstacle	Parallelogram	Orange	Black	Thin	Left

To integrate *cognitively* different information from different diagrams, Objectiv*er* supports navigation map and signposting techniques (see Fig. 5 b, c). However, model *modularisation* and *structuring* depends entirely on the modeller's skills. Objectiv*er* also has no means to set boundaries for structured elements. Also there are no means to *emphasise* elements in a diagram.

When working with Objectiv*er* the diagram is *identified* by its name and type in the title bar of the active window. When printing the diagram, its name (but not type) is included at the bottom of the diagram (Fig. 7). Except for toolbars used when modelling, Objectiv*er* does not include legend on the printed diagrams.

Visual expressiveness is limited in Objectiv*er* in that a very few of visual variables (e.g., size) are used. But the tool can automatically normalise element *size*, *spacing* and *alignment*. KAOS turns out to be quite complex *wrt graphical simplicity*; its overall complexity is 18 graphical conventions. In addition to graphical constructs, modellers also have to define obligatory (e.g. Def in Fig. 2) and can define optional (e.g. FormalDef in Fig. 2) attributes.

Table 3. Analytical comparison of goal modelling languages
("–" - language does not support the property; "+" - language supports the property)

Principle	**Properties**		**KAOS/Objectiv*er***
Discriminability	**Element size**		Depends on label length
	Contrast		–
	Proximity		Manual activity
	Constructs variation	**Square**	Goal, agent, entity,
		Triangle	–
		Circle	Operation
		Cross	–
		Arrow	G-refinement, operationalisation, performance, input, output
	Visual variables	**Horizontal and vertical**	Element position depends on the modeller
		Shape	+
		Size	+ (modeller's skills)
		Colour	+
		Value	–
		Orientation	+
		Texture	–
Modularity	**Decomposition or modularisation**		Diagram division into manageable "chunks" depends on the modeller.
Emphasis	**Visual variables**	**Element shading**	–
		Element size	+ (modeller's skills)
		Element colour	–
		Text size	–
		Text value	–
		Text colour	–
Cognitive integration	**Diagrams types**		Goal model, Agent model, Operation model, Object model
	Tech-niques	**Summary**	–
		Navigational map	–
		Signposting	+
		Current context	–
Perceptual directness	**Icon representation**		Modeller must learn the icons
	Perceptual direct relationships		–
Structure	**Tech-niques**	**Proximity**	–
		Similarity	–
		Common area	–
Identification	**Diagram names**		+
	Diagram types		+/– (only when modelling)
	Labels		+
	Legend		–
Visual expressiveness	**Visual variables**	**Horizontal and vertical**	Element position depends on the modeller
		Shape	+
		Size	+/– (manually)
		Colour	+ (discrimination of elements)
		Value	–
		Orientation	+ (discrimination of elements)
		Texture	–/+ (only background grid)
	Normalisation	**Element size**	+
		Line thickness and style	–
		Label typeface, font size, and capitalisation	–
		Elements evenly spaced	+
		Alignment of elements	+ (clan layout strategy, tree layout strategy)
Graphical simplicity	**Visual categories**		18
	Other means of information		Informally and formally

5 Recommendations

Based on the analysis of KAOS we formulate recommendations (Table 4) for modellers, language engineers and tool developers.

Modeller. The only Objectiv*er* function that might help to deal with the element discriminability is proximity control (*M.1*). This is performed by organising diagrams using clan layout or tree layout strategies. Although modularisation (*M.2*) is helpful for large diagrams [16], the modeller should not overestimate it, viz. it might be inefficient for small diagrams [20]. For structuring (*M.3*), modellers have to use proximity appropriately, and structure elements based on semantics but not on syntax [20]. To ease cognitive integration, the modeller must name each diagram, as well as specify its type (*M.6*).

Objectiv*er* does not provide legends automatically, nor does it allow attaching more "direct" icons to diagram elements (*M.5*); instead modeller has to do it manually (e.g., with text-editing and drawing tools). This is a labour intensive activity, but might result in better model understanding, especially for unskilled diagram readers. *M9* stresses the "good" use of principles [20]. It means that principles should be followed with reason, e.g., sometimes well-accepted conventions in some domain or organisation might prevail although they contradict the principles.

All these guidelines might be applied separately, but we also suggest a scenario on how to use the Objectiv*er*'s functionality in order to fulfil the principles of effective diagrams [17]. The scenario is provided in appendix A.

Table 4. Recommendations for modeller, language engineer and tool developer

Modeller	Language engineer	Tool developer
M.1: Use proximity normalisation functions to discriminate elements. ***M.2***: Divide model into manageable modules. ***M.3***: Group elements in the diagram according to the semantic relevance. ***M.4***: Define visual cues that might ease information integration from different diagrams. ***M.5***: Create legends for the constructs used in the diagrams. ***M.6***: Name and specify type for each diagram in the model. ***M.7***: Learn the language and tool principles from documentations. ***M.8***: Define contextual information in the diagram. ***M.9***: Make "*good*" use of principles for effective diagrams.	***E.1***: Design a concrete syntax that would allow discriminate language constructs. ***E.2***: Define visual clues supporting different cognitive integration techniques. ***E.3***: Develop icons and relationships that would help to remember and comprehend their meaning. ***E.4***: Language constructs should be equipped with attributes for defining additional information. ***E.5***: Define simple language graphical conventions. ***E.6***: In documentation, explain each language construct (icons and relationships), visual clues, properties and graphical conventions.	**Tool should:** ***T.1***: have discrimination means. ***T.2***: provide guidelines for decomposing the model into modules. ***T.3***: have means to structure elements in a diagram. ***T.4***: provide cognitive integration techniques [16, 17]. ***T.5***: have means for emphasis of diagram elements. ***T.6***: be explained in its tutorials. ***T.7***: guide creation of the diagram using scenarios. ***T.8***: have means to define legend in a diagram. ***T.9***: include diagram name and type on the printed diagram. ***T.10***: have element normalisation means. ***T.11***: have means for controlling visual variables for every element in the diagram (model). ***T.12***: print comments, conceptual cues, textual and formal information provided in the element properties. ***T.13***: have discussion means for modellers and diagram readers.

Language engineer. These recommendations include guidelines on how to improve KAOS, or support the development of specific domain languages based on KAOS. For the visual language, an engineer should carefully choose discriminating shapes [7] and variables (*E.1*, *E.3*) as well as strive for simple graphical conventions (*E.5*). But not everything should be possible to specify graphically. Thus, language constructs have to be equipped with attributes not appearing in graphical views where textual information would be defined (*E.4*). The use of cognitive integration is much dependent on modeller. The language can support this principle (*E.2*) by suggesting different visual clues, corresponding to different integration techniques [16].

For example the concrete suggestion for E.1 is to discriminate constructs not only by colours, but also include different shapes and icons (e.g., as they are suggested in [12] or [23]). Other concrete suggestions are provided in appendixes B and C.

Tool developer. The modelling tool should provide different support for unskilled modellers (novices) and for experts. Petre notices that novices are distracted by “syntax and surface features” [20]. Thus, on the one hand, the tool features like *T.1-T3*, *T.6-T8* and *T.10* might be of great help for novices to learn both the language and the tool. On the other hand, experts tend to “handle information at a different level” [20] (e.g., *T.2-T4*) and understand importance of emphasis (*T.5*), visual variables (*T.11*) and other conceptual cues (*T.12*). Further the difference between novices and experts could be handled by the tool. For example, the tool should have an option of including a legend (*T.8*) for novices and not including it for experts.

Other concrete suggestions are provided in appendix D.

6 Discussion

In this paper we have considered some basic principles [17] for producing the effective diagrams. We have analysed how they are fulfilled for KAOS and its supporting tool Objectiv*er*. The investigation was performed by one researcher, the first author of this paper. Therefore, the research is subjective *wrt* understanding and interpreting the principles. The language and tool investigation was carried out during one week, testing them on small examples. Hence, it might be that some language or tool properties were not observed. The observations would be more accurate if the principles were applied on large-scale (industrial) models. However, our study resulted in an arguably more fine-grained analysis in comparison to [1, 3, 14].

The “goodness” of principles for effective diagrams relies on the individual skills and insight of modellers and diagram readers. For example, normalisation can affect intentional use of visual variables for emphasis, grouping and discriminability [17]. Furthermore KAOS like other GMLs deals with the abstractions, like goal, that do not carry physical representation in the real world. This makes perceptual directness difficult to achieve. The graphical conventions proposed in KAOS (as well as other GMLs) are thus arbitrary and have to be learnt before starting modelling.

The analysis indicates that KAOS can be substantially improved *wrt* most principles. The observations are in line with similar studies for other visual modelling languages (e.g. UML [18]). Thus, we have suggested recommendations for KAOS and Objectiv*er* users to produce models that communicate more effectively. Recommendations for language engineers and tool developers suggest (*i*) how to maintain and improve both KAOS and Objectiver, and (*ii*) how to devise new domain specific languages and tools based on KAOS. However, the suggested recommendations need to be validated empirically. Our future work includes analysing other GMLs, as well as investigating principles for effective diagrams in large-scale (industrial) goal models.

Acknowledgement. We wish to thank R. Darimont for the very useful comments

References

1. Al-Subaie, H. S. F., Maibaum, T. S. E.: Evaluating the Effectiveness of a Goal-oriented Requirements Engineering Method. Proc. of the 4th Int. workshop on Comparative Evaluation in Requirements Engineering (CERE’06) (2006) 8-19
2. Anton, A. I.: Goal-based Requirements Analysis. In Proceedings of the 2nd Int. Conference on Requirements Engineering (ICRE ’96), IEEE Computer Society (1996) 136-144
3. Ayala, C. P., Cares, C., Carvallo, J. P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C.: A Comparative Analysis of i*-based Agent-oriented Modelling Languages, Proc. of the 17th Int. Conf. SEKE'05 China (2005) 43-50
4. Blackwell, A., Green, T.: Notational Systems – the Cognitive Dimensions of Notations Framework, Carrol, J. M. (eds): HCI Models, Theories and Frameworks: Towards a Multidisciplinary Science, Morgan Kaufmann (2003)
5. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems, 8(3) (2004) 203-236
6. Chung, K. L., Nixon, B., Mylopoulos, J., Yu, E.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston (2000)
7. Costagliola, G., Delucia, A., Orefice, S., Polese, G.: A Classification Framework to Support the Design of Visual Languages, Journal of Visual Languages and Computing, 13 (2002) 573-600

8. Franch, X.: On the Quantitative Analysis of Agent-oriented Methods. In: Proc. of the 18th Int. Conf. on Advanced Information System Engineering (CAiSE2006) (2006) 495–509. Springer, Heidelberg (2006)
9. Frutiger, A., Bluhm, A.: Signs and Symbols: Their Design and Meaning, New York, USA, Watson-Guptill Publications (1998)
10. Kavakli, E.: Goal-oriented Requirements Engineering: a Unifying Framework, Requirements Engineering Journal, 6(4) (2002) 237–251
11. Kavakli, E., Loucopoulos, P.: Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods, Krogstie, J., Halpin, T., Siau, K., (eds.), Information Modeling Methods and Methodologies, IDEA Group Publishing (2005) 102–124
12. Letier, E.: Reasoning about Agents in Goal-Oriented Requirements Engineering. PhD thesis, Universite Catholique de Louvain (2001)
13. Lindland, O. I., Sindre, G., Sølvberg, A.: Understanding quality in conceptual modelling. IEEE Software, 11(2) (1994) 42-49
14. Matulevičius R., Heymans P., Comparison of Goal Languages: an Experiment. Proceedings of the Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2007) Springer-Verlag Berlin Heidelberg (2007) 18-32
15. Matulevičius, R., Heymans, P., Opdahl, A. L.: Comparing GRL and KAOS using the UEML Approach. Concalves, R. J., Muller, J. P., Mertins, K., Zelm, M. (eds.): Enterprise Interoperability II. New Challenges and Approaches, Springer-Verlag (2007) 77-88
16. Moody, D.: Dealing with "Map Shock": A Systematic Approach for Managing Complexity in Requirements Modelling, Proceeding of the REFSQ 2006, Luxembourg (2006)
17. Moody, D.: What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development, Appear in Proc. of the 15th Int. Conf. in Information Systems Development (ISD 2006) (2006)
18. Morris, S., Spanoudakis, G.: UML: An Evaluation of the Visual Syntax of the Language, Proc. 34th Annual Hawaii International Conference on System Sciences (HICSS-34) IEEE Computer Society (2001)
19. Narayanan, N. H., Hubscher, R.: Visual Language Theory: Towards a Human-Computer Interaction Theory, Marriott K., Meyer B. (eds.) Visual language theory, Springer-Verlag New York, Inc. (1998) 87-128
20. Petre M.: Why looking isn't Always Seeing: Readership Skills and Graphical Programming, Communications of the ACM, 38 (6) (1995) 33-44
21. Regev, G.: A Systemic Paradigm for Early IT System Requirements Based on Regulation Principles: The Lightswitch Approach. PhD thesis, Swiss Federal Institute of Technology (EPFL) (2003)
22. Regev, G., Wegmann, A.: Where do Goals Come From: the Underlying Principles of Goal-oriented Requirements Engineering. Proc. of the 13th IEEE Int. Conf. on Requirements Engineering (RE'05) (2005)
23. van Lamsweerde A.: The KAOS Meta-model: Ten Years After. Technical report, Universite Catholique de Louvain (2003)
24. Yu, E.: Towards Modeling and Reasoning Support for Early-phase Requirements Engineering. Proc. of the 3rd IEEE Int. symposium on Requirements Engineering (RE'97), IEEE Computer Society (1997) 226-235
25. A. van Lamsweerde, Elaborating security requirements by construction of intentional anti-models, in: Proceedings of the 26th International Conference on Software Engineering (ICSE '04), IEEE Computer Society, Washington, DC, USA, 2004, pp. 148–157.
26. A. van Lamsweerde, Engineering Requirements for Mission-Critical Software Systems, URL: http://ssel.vub.ac.be/francqui/, 2007.
27. Gurr, C.A., Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues. Journal of Visual Languages and Computing, 1999, 10: p. 317-342.
28. Opdahl AL, Henderson-Sellers B, (2005) A Unified Modelling Language without Referential Redundancy. Data and Knowledge Engineering (DKE). Special Issue on Quality in Conceptual Modelling 277-300.

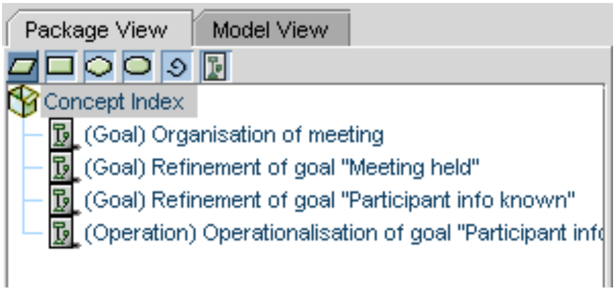
Appendix A. Use KAOS/Objectiv*er* to prepare effective goal model

(A top-down approach)

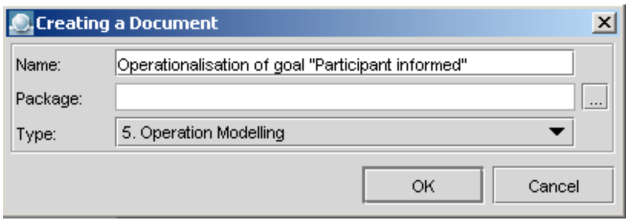
1) ***M.7*****:** Learn the language and tool principles from documentations.
(Perceptual directness, Graphical simplicity)

2) ***M.2*****:** Divide model into manageable modules.
(Modularity)

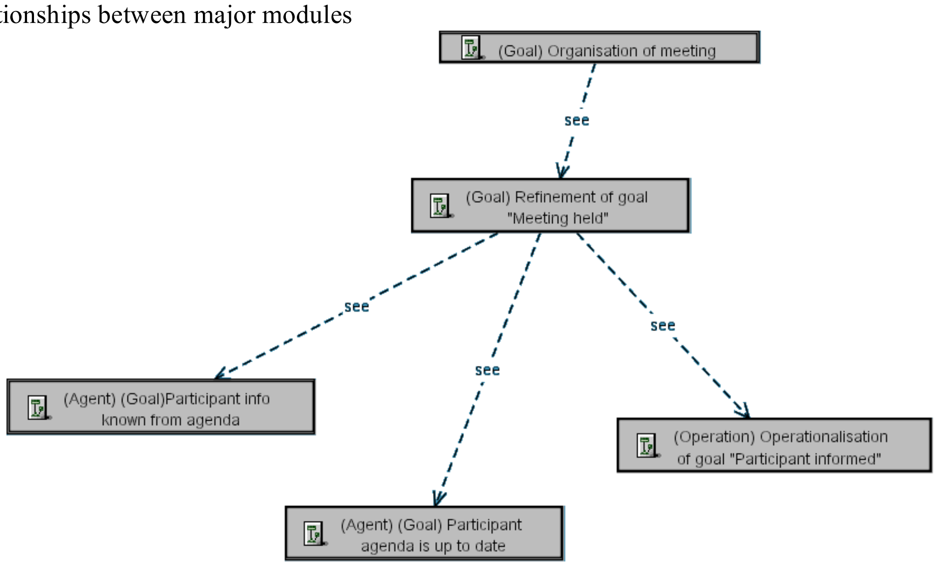
- Define the major modules the model. In Objectiv*er* the modeller can define model components by creating new diagrams.



3) ***M.6*****:** Name and specify type for each diagram in the model.
(Identification)



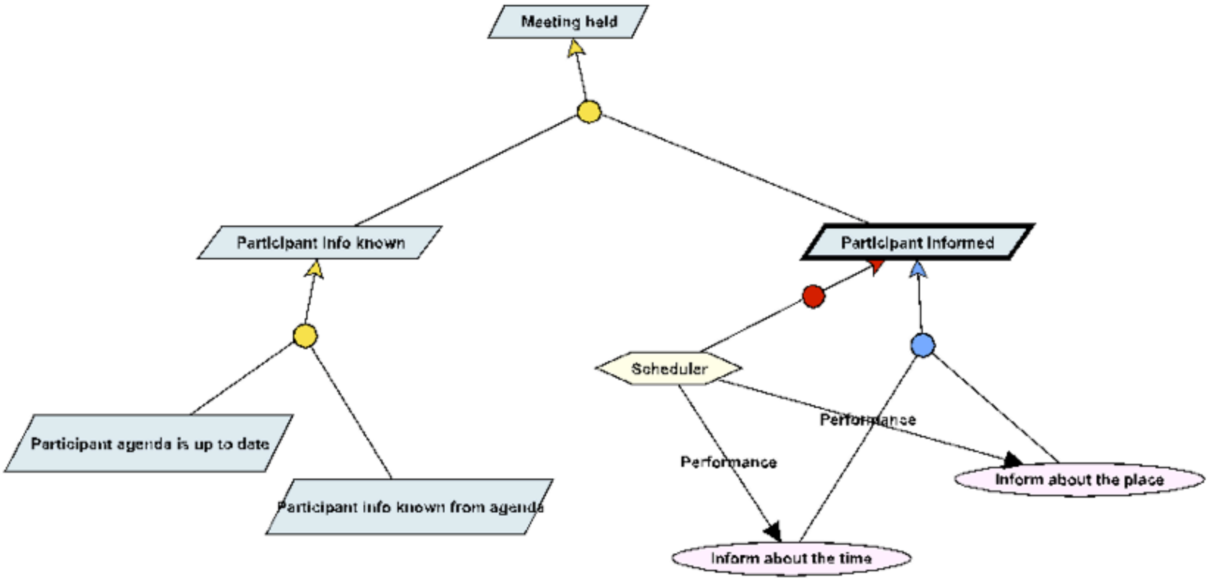
M.2**:** Divide model into manageable modules.
(Modularity, Cognitive integration)
Define relationships between major modules



Navigation map

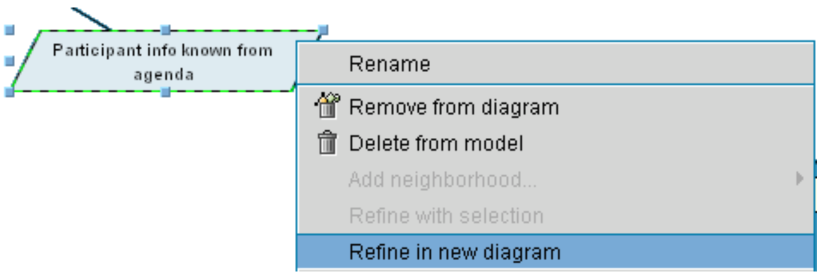
Goal diagram dependency diagram can also be prepared using the Generator functionality.

5) Define context of each diagram. If number of elements is bigger than 7+/- 2 it is advisable to reorganise diagram to smaller manageable modules (use).
(Cognitive integration)

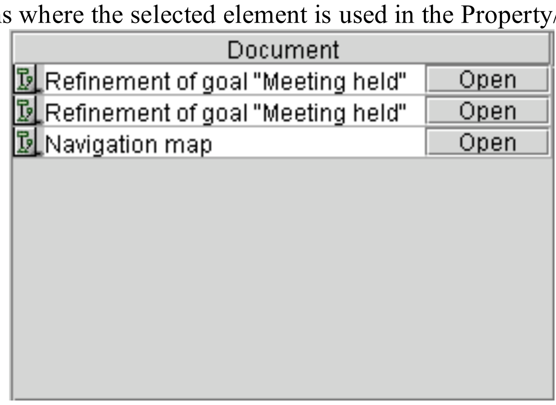


Refinement of goal "Meeting held"

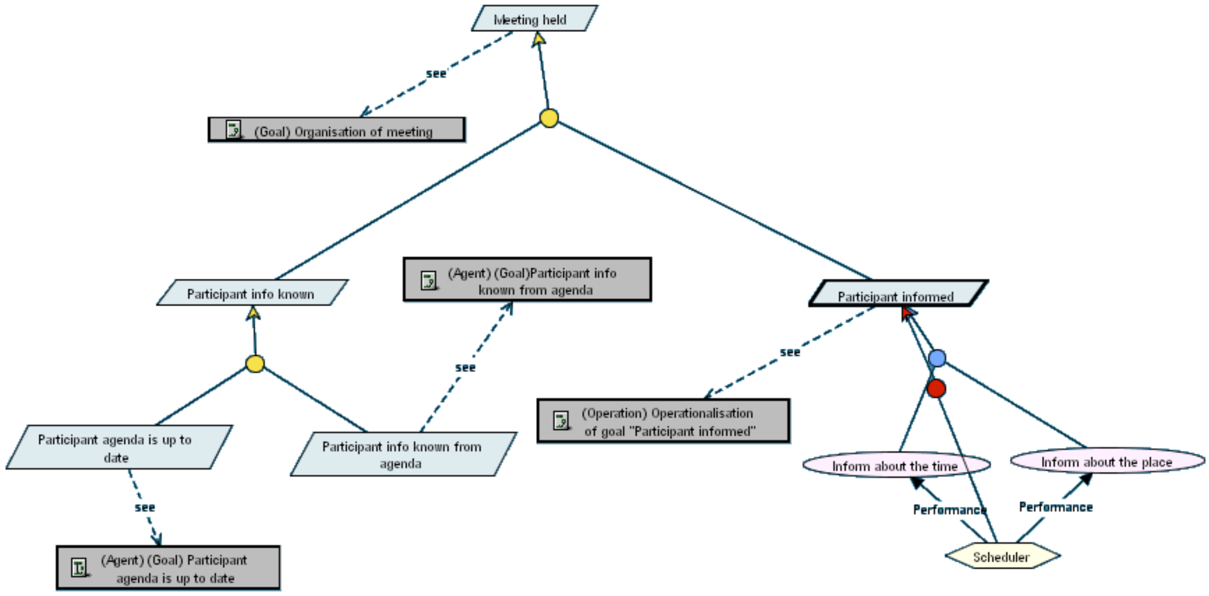
Modules (of a goal model) can be created using function *Refine in another diagram* (right mouse click on the goal concept)



The user can see the diagrams where the selected element is used in the Property/Document tab:



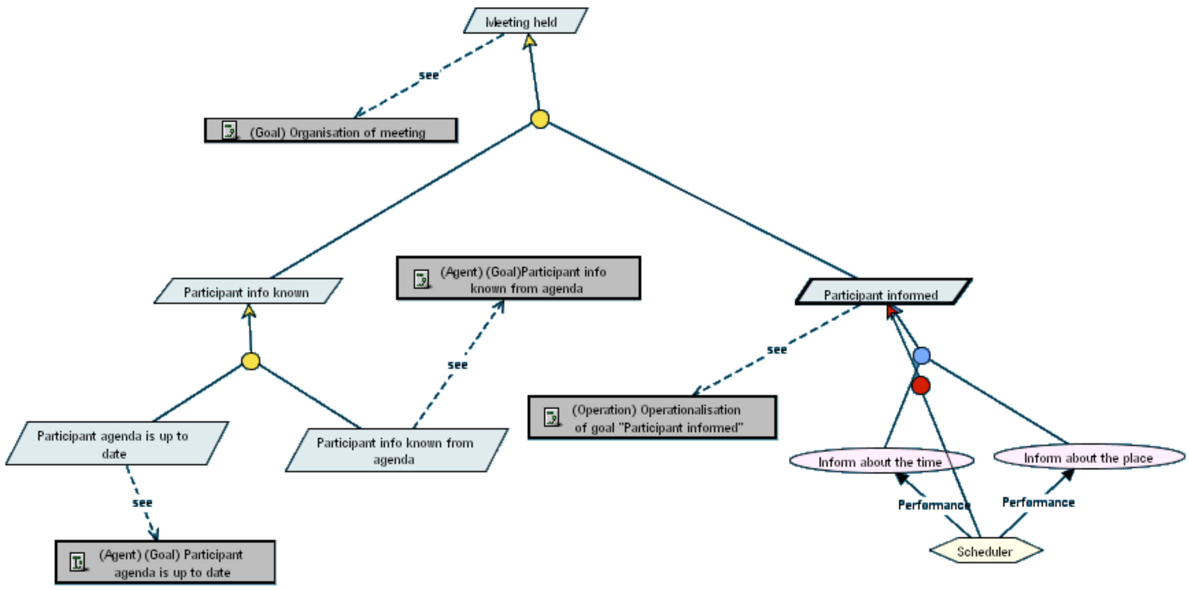
6) ***M.4*: Define visual clues that are used for information integration from different diagrams.**
(Cognitive integration)



Refinement of goal "Meeting held"

7) ***M.1*: Use proximity normalisation functions to discriminate elements.**
(Discriminability)

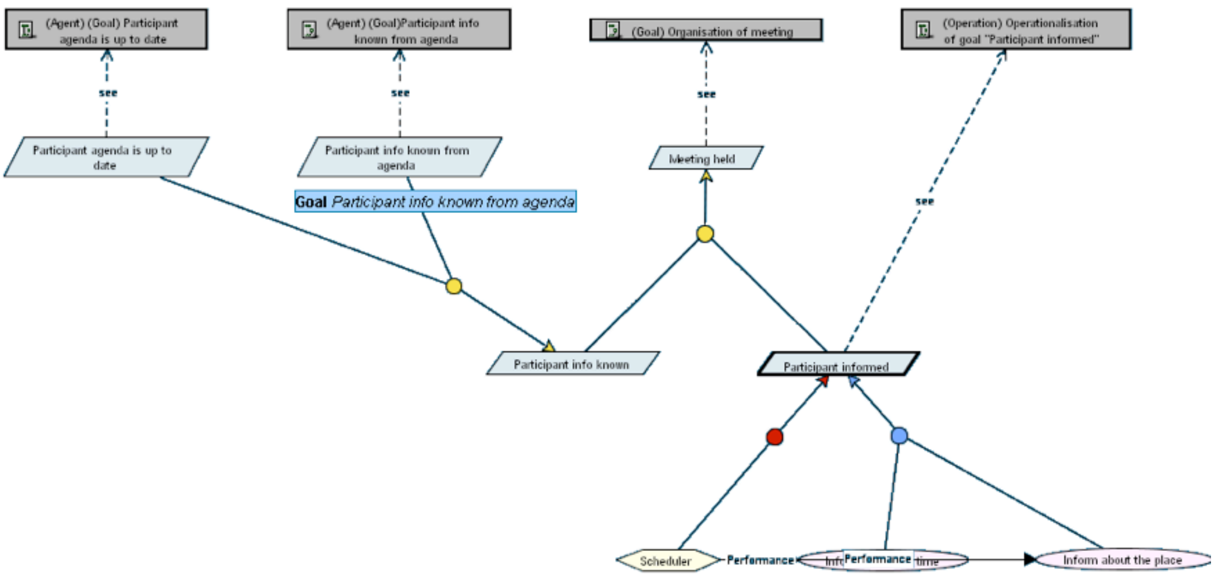
Use element normalisation functions "clean diagram layout". It will restore the size of elements.



Refinement of goal "Meeting held"

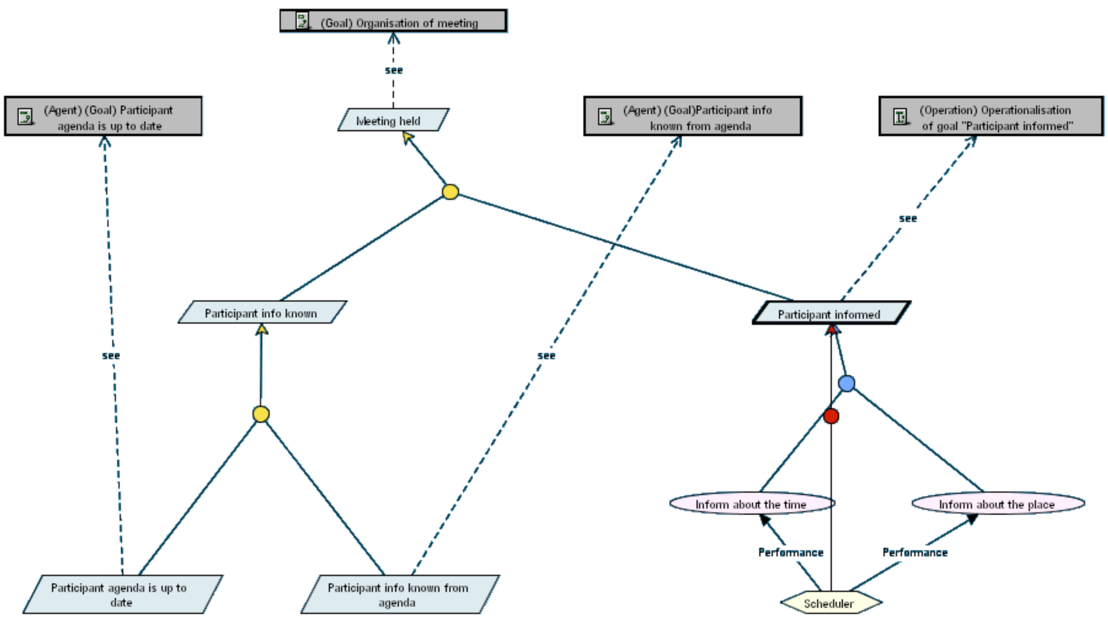
8) ***M.1*****:** Use proximity normalisation functions to discriminate elements. (Discriminability)

Clan layout



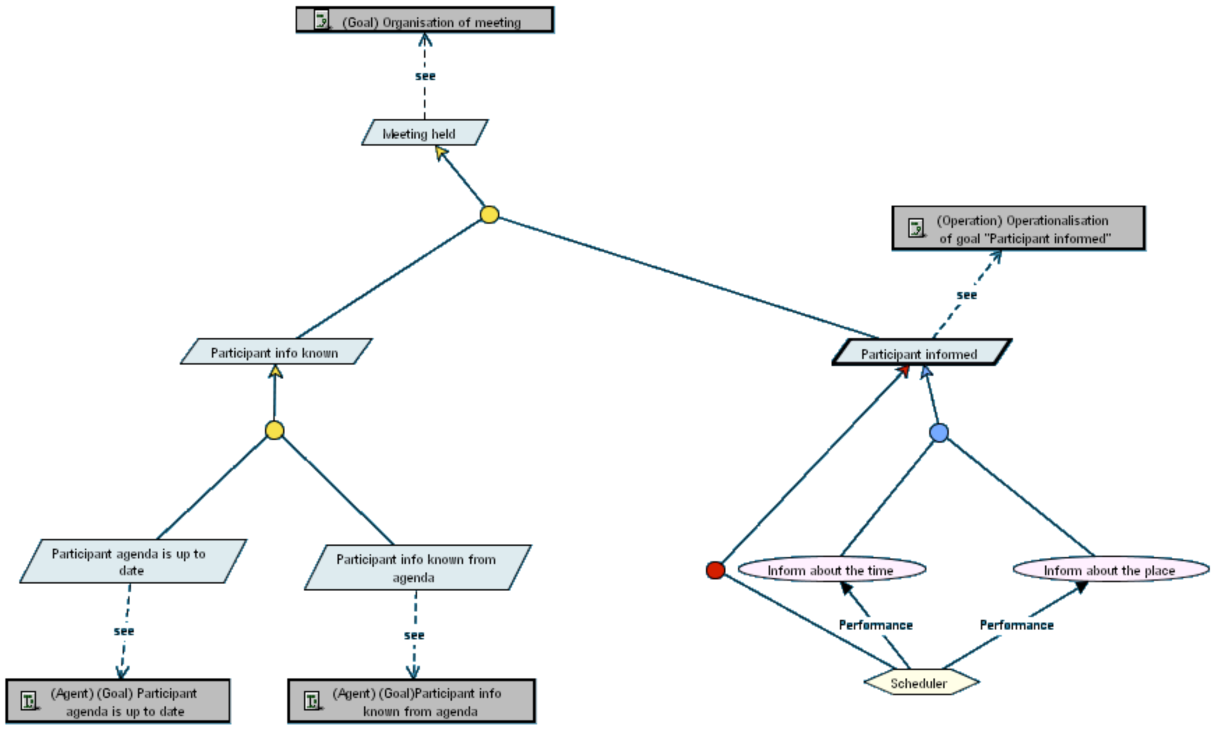
Refinement of goal "Meeting held"

Tree layout



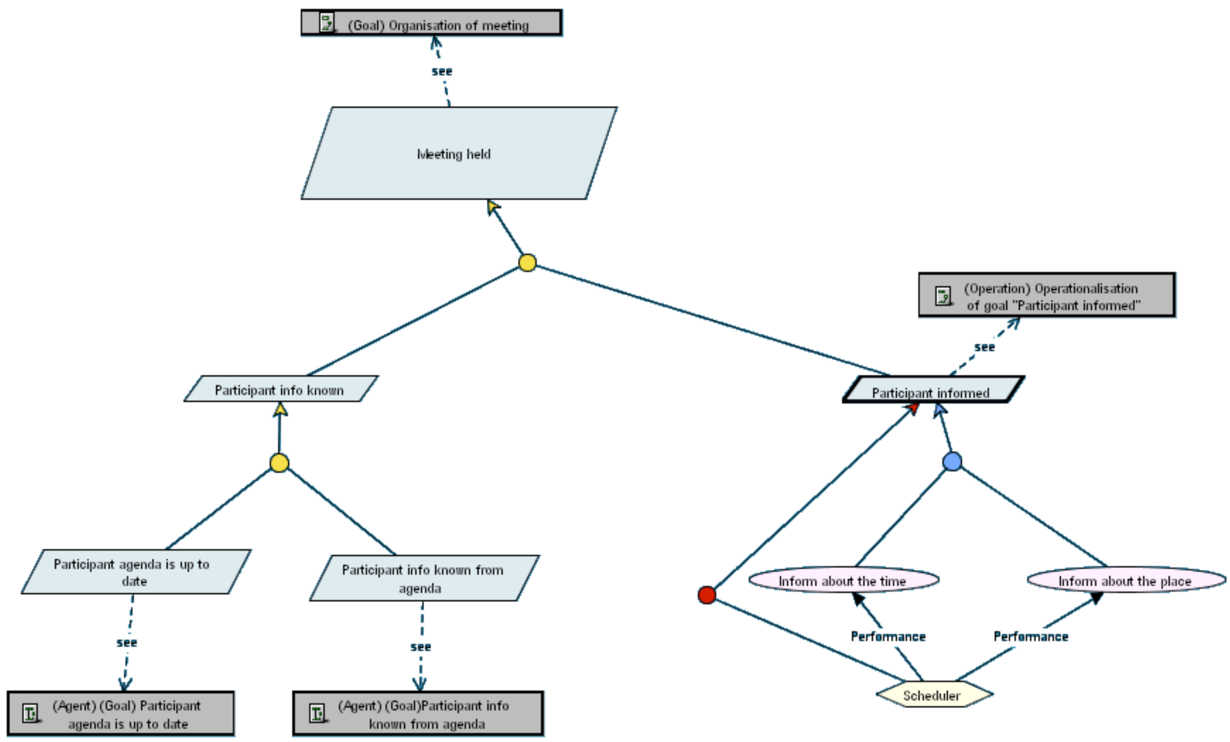
Refinement of goal "Meeting held"

9) ***M.3*****:** Group elements in the diagram according to the semantic relevance.
(Structure)



Refinement of goal "Meeting held"

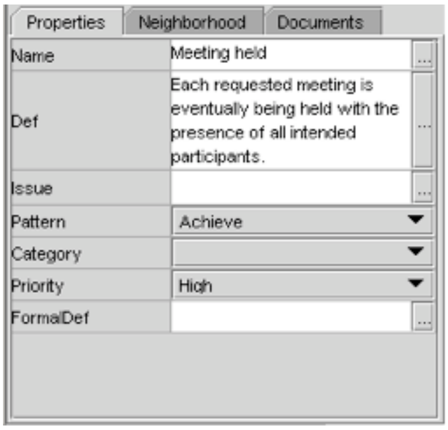
10) Emphasize element by increasing their size.
(Emphasis)



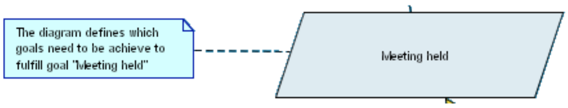
Refinement of goal "Meeting held"

11) ***M.8***: Define contextual information in the diagram.
(Graphical simplicity)
Contextual information include:

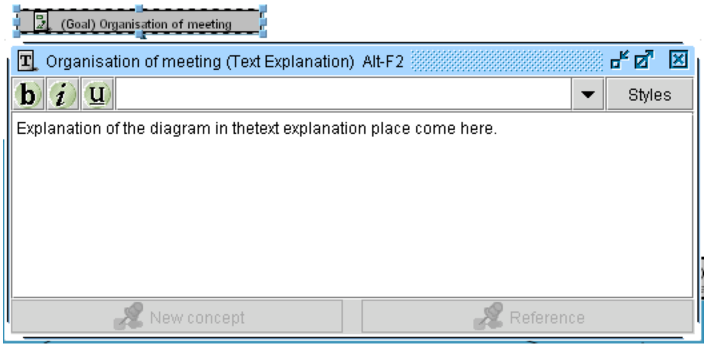
- element attributes



- notes



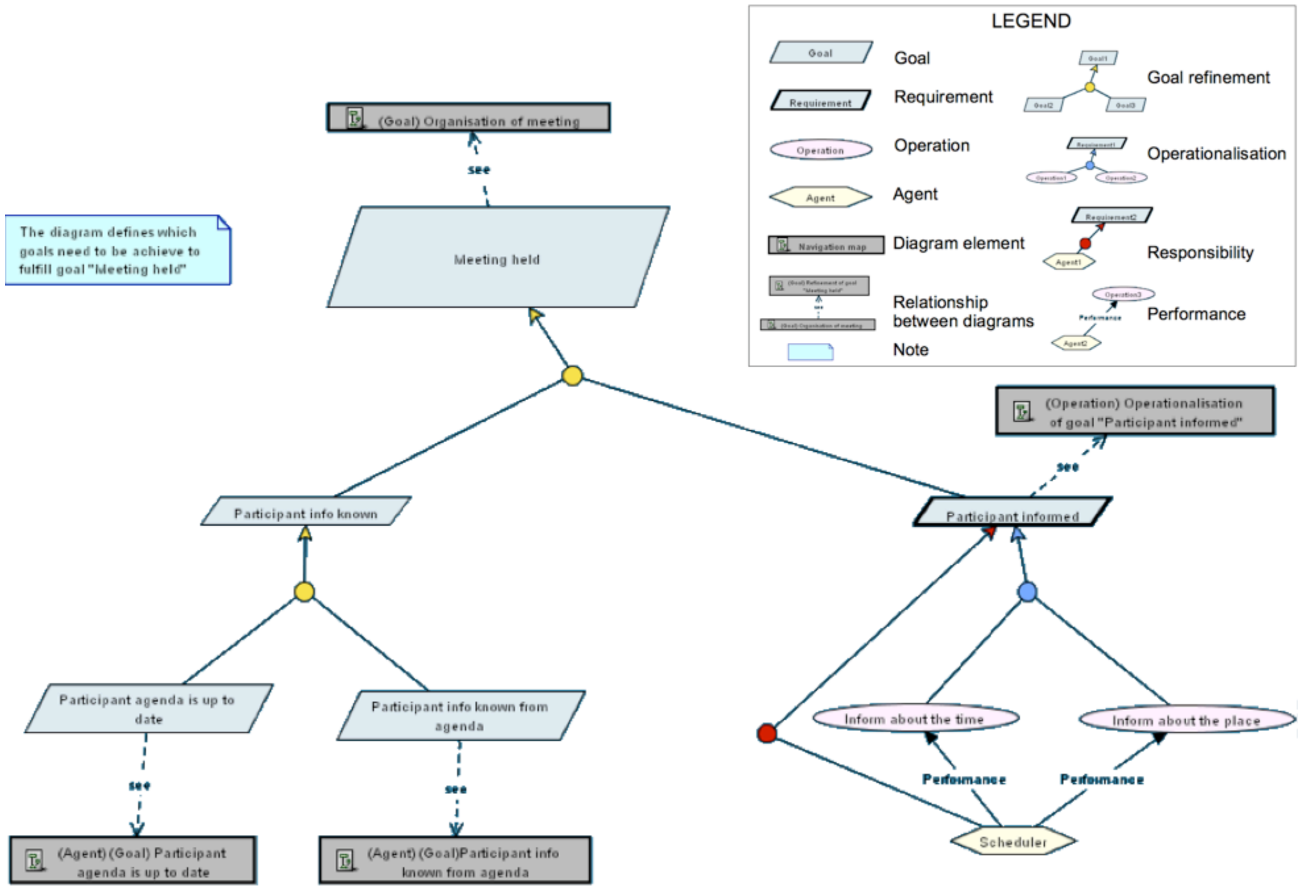
- textual explanations



- reports

12) ***M.5***: Create legends for the construct used in the diagrams
(Identification)

Since Objectiv*er* does not allow inclusion of legend to the diagram, use additional graphical editing tool.

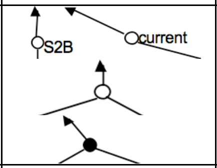
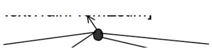
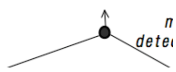
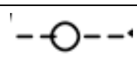
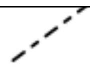


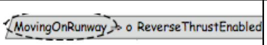
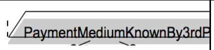
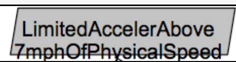
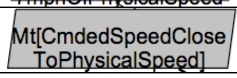
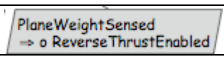
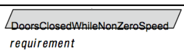
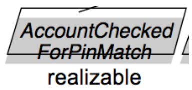
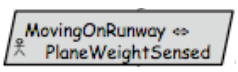
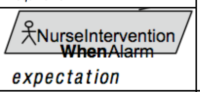
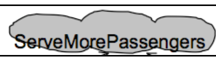
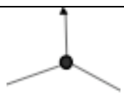
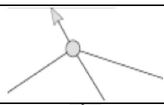
Refinement of goal "Meeting held"

Appendix B. Comparison of KAOS constructs

(Only) Some constructs	[23]	[12]	[25]	Objectiv*er* (version 2.0.0 professional edition)
Reference year	**2003**	**2001**	**2004**	**2006-2007**
Goal	Goal	ConvenientMeetingHeld	MoneyStolenFrom BankAccounts	Goal
Maintain goal	Maintain Goal	Maintain [PumpOnWhenHighWater]	-	-
Achieve goal	Achieve Goal	Achieve [PrctptsCstrRequested]		
Avoid goal	Avoid Goal	Avoid [PumpOnWhenEmpty]		
Cease goal	-	-		
Requirement	Requirement	Same as Goal, and pattern goals. Maintain [PumpOffWhenSwitchOff] Resp Pump Actuator	AccountNumber CheckedFor PinMatch CheckIteratedOnOther AccountNumbers IfNoMatch	Requirement
Expectation	Expectation			Expectation
Softgoal	SoftGoal	-	-	Softgoal
Software agent	Software Agent AgentName	Pump Actuator	-	Agent
Environment agent	Environment Agent AgentName			
Refinement	G-Refinement Complete G-Refinement Alternative G-Refinements *alt1* *alt2*	OR AND AND	*alternatives*	Complete Incomplete
Responsibility	SafeCommand Message Speed&Accel Controller	Resp Achieve [PrctptsCstrRequested] Initiator OR Resp Scheduler	-	
COMMENTS	-For "Cease goal" there is no graph. icon.	- "Cease goal" can be presented as other patterns. - No separation between software and environment agents.	- Patterns identified in the (paper) text. - Not clear if refinement is complete. - No differentiation from the goal and between requirement and expectation.	- Patterns are identified as construct attributes. - Colours. On the black-white printed it might be problematic to differentiate. - No separation between software and environment agents. - Not clear about *assignment*: during the seminar [4] it was said that this concept is taken out of the latest language version.

Models selected form the [26] (2007). Summary of the graphical constructs

(Only) Some constructs	Page 12, 15	Page 19	Page 36	Page 70	Page 79
Goal	SafeTransportation	...	-	Pump*On***When**HighWater	-
Maintain goal	-	Maintain[Safe Speed/AccelCom'ed] Mt[AccurateEstimate OfSpeed/Position]	-	-	Maintain [TrainWaiting] On (tr, b) ⇒ On (tr, b) **W** On (tr, next(b))
Achieve goal	-	Achv[ComdMsg SentInTime]	-	-	Achieve [TrainProgress] On (tr, b) ⇒ ◊ On (tr, next(b))
Avoid goal	-	-	-	-	-
Cease goal	-	-	-	-	-
Requirement	-	-	AccurateEstimate OfSpeed&Position	PumpSwitch*On* **When**HighWaterDetected	-
Expectation	-	-			-
Softgoal	-	-	-	-	-
Software agent	-	Communic Infrastruct	Speed&Accel Controller	Pump Controller	-
Environment agent	-		Tracking System	Pump Actuator	-
Refinement			-		
Responsibility	-				-
COMMENTS	- OR-refinement is identified with alternative name	- Maintain or Mt - No difference between software and environment agents	- No difference between requirement and expectation - Two different responsibility links		- Patterns identified in full word

(Only) Some constructs	Page 83	Page 85	Page 84	Page 115	Page 126
Goal					
Maintain goal	-	-	-	-	
Achieve goal	-	-	-	-	-
Avoid goal	-	-	-	-	-
Cease goal	-	-	-		-
Requirement					-
Expectation					-
Softgoal	-	-	-	-	
Software agent		-		-	-
Environment agent		-		-	-
Refinement		-		-	
Responsibility		-		-	-
COMMENTS	- Responsibility link without arrow - No differentiation between requirement and goal (expectation is differentiated with a little human).		- Goal (requirement, expectation) graphical icon the same as for object - Actor has different icon in comparison to other models - Software and environment agents are not differentiated.	- Goal, requirement, expectation are not differentiated.	- The only model where the softgoal is used. - Completely different icon for refinement

Appendix C. Recommendations for language improvement

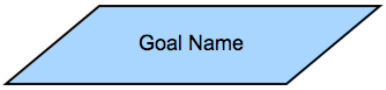
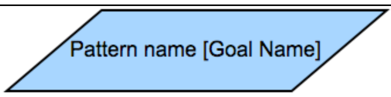
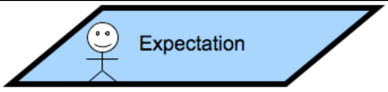
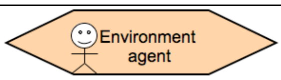
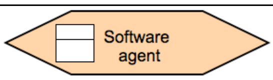
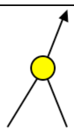
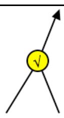
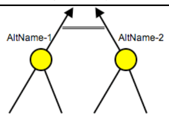
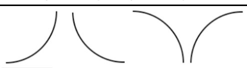
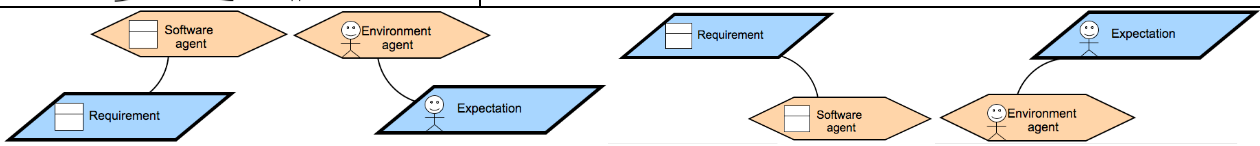
In appendix B the comparison of some constructs for KAOS is provided. Based on this comparison we will elaborate few recommendations for the concrete KAOS syntax.

E.1: Design a concrete syntax that would allow discriminate language constructs.

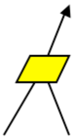
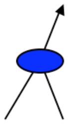
KAOS/Objectiv*er* concrete syntax is based *i*) on the colours and *ii*) on the neighbourhood concepts. In the first case the problems arise when diagram is printed using the white-black printers. In the second case the diagram requires from its reader an additional mental effort to relate neighbourhood concepts to understand the meaning of the elements. Based on these limitations we suggest:

- avoid use of similar colour to discriminate close constructs;
- use the shape or/and additional visual variables (e.g. orientation) for discriminability;
- use different colours to discriminate element groups.

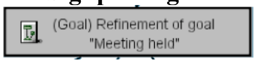
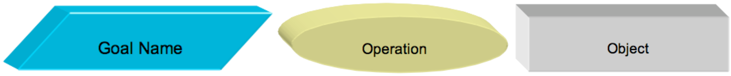
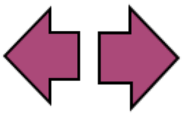
Our suggestions to discriminate elements analysed in appendix B are suggested in the table below.

Suggested construct	Description
	Goal without pattern - parellelogram, oriented to the right, light blue background, borderline thin. Label include goal name, placed in the centre.
	Goal with pattern - parellelogram, oriented to the right, light blue background, borderline thin. Label include pattern name (maintain, achieve, cease, avoid) and goal name, placed in the centre.
	Expectation - parellelogram, oriented to the right, light blue background, borderline thick. Label include goal name, placed in the centre. Before the label a small icon of human.
	Expectation - parellelogram, oriented to the right, light blue background, borderline thick. Label include goal name, placed in the centre. Before the label a small icon of “class diagram” indicating piece of software.
	Softgoal - parellelogram, oriented to the right, light blue background, borderline thin, dashed. Label include goal name, placed in the centre.
	Environment agent - hexagon, light pink background, borderline thin. Label include name of the software agent. Before the label a small icon of human.
	Software agent - hexagon, light pink background, borderline thin. Label include name of the software agent. Before the label a small icon of “class diagram” indicating piece of software.
	Goal refinement[1]
	Complete goal refinement
	Alternative goal refinement. AltName-1 – name of alternative one AltName-2 – name of alternative two
	Responsibility
	

[1] In order to differentiate between G-refinement and operationalisation on the white-black printed version of diagram, the following icons are suggested:

 - G-refinement  - Operationalisation

E.2: Define visual clues supporting different cognitive integration techniques.

	Existing functionality	Recommended improvements
Cognitive integration icons [16]	Used for **Navigation maps** and **Signposting** 	For **Summary** diagrams  They are clearly distinguished from the 2D shapes [16]. They are perceptually direct in that their 3D appearance make them look as if they are able to contain other elements [16]
		For **Zoom out** (scale up) – link to "parent" diagram  **Related diagrams** – at the same level.  **Zoom in** (scale down) – link to "child" diagram. It might also be textual definitions, diagram element names provide specific cross-references. 
		Define graphical syntax for element grouping

E.3: Develop icons and relationships that would help to remember and comprehend their meaning.

The notation should share important properties with object or relationship they represent [27]. Language notations should not be redundant, overlapping, incomplete and underdefined [27, 28]. The language should be based on the precisely defined metamodel and the metaCase tools should support definition of semantic and abstract syntax for this language.

E.4: Language constructs should be equipped with attributes for defining additional information.

Allow definition of additional attributes for the language elements.
These attributes might be used for element sorting, filtering, categorization.

E.5: Define simple language graphical conventions.

In case of very abstract concepts, perceptually direct representations are difficult to find. In addition different users might have different notation needs [20] depending on their background and the problem (and the domain) they need to solve. Also on the one hand expert users would focus on notations helping to define additional information on the model (e.g., element properties); on the other hand novice users would use the graphical elements. The metaCase tool developers should include the means to develop different notation groups for different background users. Languages/tools should be extendable with other concepts and elements to model problems of different domains.
The concrete suggestion for *E.5* includes the means which user can use to define the needed graphical elements for the modelled problem.

E.6: In documentation, explain each language construct (icons and relationships), visual clues, properties and graphical conventions

Language tutorials explain the major constructs, however they miss the detailed explanation of every single entity (element, attribute, visual clue, etc) used in the language. For example:

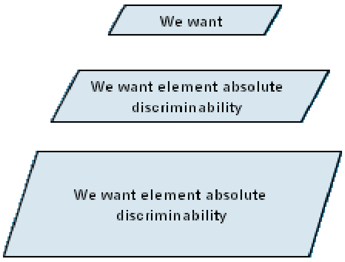
1) there is brief explanation of a *goal*. However, there is no explanation of its properties (Name, Def, Issue, Pattern, Category, Priority, FormalDef);

2) There is no description of the *required* and *domain properties* which are important attributes for operation and operationalisation;

3) There are no explanation of notations such as "Underfined relationship", "N-ary association", "Note link" and other. They are implemented and used in the tool, thus should be included in the documentation.

Appendix D. Existing and recommended functionality for the Objectiv*er*

T.1: Tool should have means for absolute discriminability.

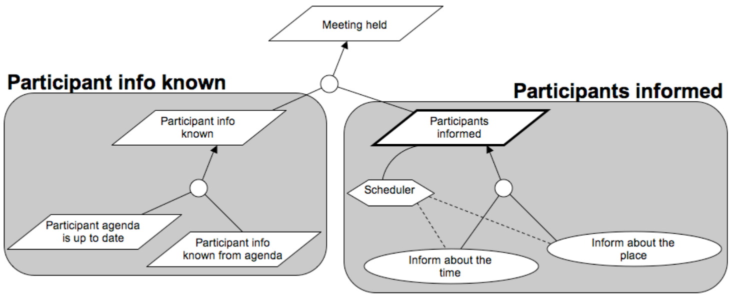
	Existing functionality	Recommended improvements
Size	- User is able to change element size manually - User can normalise (restore) size of all elements using command “Clean diagram layout”. 	- Together with the element size tool should allow control of the label size. - Tool should control size of the elements would not affect the length of the label.
Contrast	- User is able add/remove the grid to the graphical editor field.	- Tool should allow change of the background colour for graphical editor. - Tool should include printing of the background colour.
Proximity	- User is able to use different layout strategies: e.g., clan layout, tree layout. - In the properties window – user can set initial values for element spacing	- Tool should inform user about element proximity with respect to other elements. - Tool could suggest graphical patterns for the diagram layout (e.g. similar to PowerPoint patterns for slide layout).

T.2: Tool should provide guidelines for decomposing the model into modules.

	Existing functionality	Recommended improvements
Modularisation	- Users are provided with templates for the requirements specification (not tested/provided in the evaluated version of the tool) - In the goal model separate goals can be refined in different diagrams by executing function “Refine in another diagram”. - Each a detailed diagram can be opened by clicking on the diagram/element icon (Open document)	- Develop scenarios, which would guide model decomposition. Provide guidance according to the scenarios (Kaindl, 2004). If experts are using the tool, scenario guidance could be switched off. - Check number of elements in the diagram; inform the user if the number of elements in a diagram exceeds cognitive limits. - Define element filtering mechanism (e.g., filtering might be organised using information defined in the appropriate element attributes)

T.3: Tool should have means to structure elements in a diagram.

	Existing functionality	Recommended improvements
Structure	- User can manually place elements close to each other (according to proximity).	- Tool should check for possible structuring possibilities and provide the hints to the user: *i*) according to the element proximity; *ii*) according to the element similarity; *iii*) according to the common region. - Tool should have visual clues to structure elements according to the common area.



T.4: Tool should provide cognitive integration techniques [16, 17].

	Existing functionality	Recommended improvements
Cognitive integration	- Structure of the model is provided in the Explorer field. - Tool/language contain icon, which represents the diagram. This icon can be used to identify links between diagrams (signposting) or to create a global view of the model (navigation map). - Table "element–diagram" based on the indexing technique.	Based on [16, 17] additional cognitive integration techniques can be implemented: - Summarisation; - Horizontal integration (continuity between adjoining views); - Orientation (level numbering, title, locator map, and scale indicator) - Spatial contiguity (parallel views)

T.5: Tool should have means for emphasis of diagram elements.

	Existing functionality	Recommended improvements
Emphasis	Element might be manually emphasised by increasing its size.	- Tool should be equipped with means for emphasis using the graphical variables - Emphasis should be in both directions – highlighting and lowlighting.

T.6: Tool should be explained in its tutorials.
T.7: Tool should guide creation of the diagram using scenarios.

	Existing functionality	Recommended improvements
Tutorials and driving scenarios	- In the tool tutorial included language explanation on the lift example. - Explanation movies (not tested).	- Tool tutorial must be up to the current tool version. - Tool should be supported by the tool tutorial explaining the actions needed to perform with the tool. - Tool functionality should be explained in the scenarios.

T.8: Tool should have means to define legend in a diagram.

	Existing functionality	Recommended improvements
Legend	Language constructs are explained with appearing labels when modelling.	- Tool should include legend (of used constructs) on the printed diagram. - Tool should include functionality not to include legend (if experienced users are using the tool)

T.9: Tool should include diagram name and type on the printed diagram.

	Existing functionality	Recommended improvements
Name and type	- On the printed diagram at the bottom, the tool includes the diagram name.	- On the printed diagram the tool should include the diagram type. - The text-style (font, size, typeface) should editable before printing the diagram. - The position of diagram name and type should be editable by the user.

T.10: Tool should have element normalisation means.

	Existing functionality	Recommended improvements
Normalisation	- User can normalise (restore) size of all elements using command "Clean diagram layout". - User is able to use different layout strategies: e.g., clan layout, tree layout. - In the properties window – user can set initial values for element spacing.	- Tool should have means to normalise line thickness and style - Tool should have means to normalise label typeface, font size, and capitalisation.

T.11: Tool should have means for controlling visual variables for every element in the diagram, diagram background, its name, type, place in the diagram, font type, font size, orientation, and others.

T.12: Tool should print comments, conceptual cues, textual and formal information provided in the element properties.

	Existing functionality	Recommended improvements
Printing	- User can print separate diagrams - User can print the reports	- Tool should allow printing of the element properties. - Tool should allow previews and editing of model element depending on the generated previous. For instance, the modeller should be able to change placement of the elements, highlight/lowlight the elements, options for inclusion/removing the legend, etc.

T.13: Tool should have discussion means for modellers and diagram readers.

	Existing functionality	Recommended improvements
Discussion	- There is an option to work in developers' and/or reviewers' roles. Information sharing is done by exchanging the file. - Tool contains comment and review elements, where modellers can write their comments, suggestions, explanations, provide other contextual information.	- Tool should have a collaborative environment where different users would work simultaneously being in different geographical locations. - Tool should have rationale registration means (ensure the feature of model being traced back to goal/requirements source). - Tool should have the discussion means (similar to messaging programs, such as ICQ, Messenger). The discussion should be possible to rise about any element in the model. The discussion should be registered as part of the model (in order to be possible to look at it later). - The tool should ensure change propagation and approval mechanisms. - Tool should have means for the agreement about the model.