Towards a Theory on the Role of Ontologies in Software Engineering Problem Solving

Conclusions from a Theoretical Model of Methodological Works

José M. Cañete* and Francisco J. Galán

Faculty of Computer Science,
University of Sevilla (Spain).

Abstract. We present and validate a theoretical model of methodological works in Software Engineering that, without claiming for completeness, allows us to investigate the role of ontologies in the problem solving process related with the development of software. Our main conclusion is the potential of ontologies as resources for an individual to *think* during problem solving. We argument that suitable ontologies can support solving strategies as well as motivate their invention. We also conclude the importance of accompany an ontology with knowledge that guides the engineer in reasoning with its concepts.

The model regards a methodological work as an heterogeneous theory about a class of problems and about a number of conceptual elements. Some of the elements are ontologies, which play the role of identifying and relating aspects of the knowledge about the class of problems, making up novel perspectives on the problems that may promote solving strategies.

For illustration purposes, we take Jackson's "Problem Frames" as a case study. We analyse this work through the former model, identifying the ontologies, guides, and promoted strategies. Then we propose an alternative ontology, based on that used in the KAOS approach; we reformulate some parts of Jackson's work through this ontology and propose a strategy as well as some guides.

Keywords: Ontologies, Methodologies, Modelling, Problem Solving, Cognitive Science.

1 Introduction

Modelling languages have been used for years in Software Engineering, and they are currently broadly extended. Textbooks and papers are plenty of modelling-

* Corresponding author. Contact at `jmcv@us.es` or at José Miguel Cañete Valdeón, ETS de Ingeniería Informática, Avda Reina Mercedes, S/N, 41012, Sevilla, Spain.
This work has been partially funded by the research project TIC 2003-02737-C02-01.

related concepts as "model-driven engineering", "Model-Driven Architecture", and "Unified Modeling Language". In a previous paper we investigated the uses of these languages in the context of several software development methodologies, finding that the most popular use is that of *description* (Cañete et al., [1]). Frequently described subjects are the system-to-be and its environment. However, we also discovered that the models created with some languages are used for *reasoning* about some aspect of the development *problem*, allowing to obtain useful conclusions that, in some cases, could even motivate some design decision[1]. This fact leads to an interesting question: *what is the relation between modelling languages and human reasoning during problem solving?* This paper aims to contribute to answer this question. The followed approach is the study of ontologies in methodological works. We base this decision on three arguments.

First, the semantic conceptualization that is the basis of any modelling language can be regarded an ontology. Second, ontologies allow to acquire, organize, represent, and deal with *knowledge*. These activities are important for anybody that is solving a problem. Besides, in the case of solving a software development problem, it is necessary to have general knowledge about aspects of the class which the problem belongs to. And it is also useful to have some general knowledge about heuristics and other kinds of well-founded guides that suggest how to address the problems in the class. In conclusion, knowledge is important in problem solving, and ontologies are good instruments for managing knowledge. Therefore, a software engineer is likely to use several kinds of ontologies while she reasons in the resolution of a problem. The third argument to base our approach is that methodological works can be regarded as sources for the previously cited general knowledge (we will prove this later in this paper). In conclusion: ontologies and methodological knowledge are software engineer's tools in reasoning during problem solving. Their study seems a promising starting point for answering the question that we have formulated at the beginning of this section.

To this aim, we propose and validate a theoretical model of methodological works and we use it as an instrument for investigation. We obtain a number of predictions from the theoretical model, including the claim that ontologies may promote reasoning strategies for problem solving. The methodological work "Problem Frames" (Jackson, [11]) is used throughout this paper for illustration purposes.

The rest of the paper is organized as follows. Section 2 contains some background terms from Philosophy of Science that are necessary for the remaining parts. Section 3 describes and validates the theoretical model. In Section 4 we reason with the model and obtain a number of predictions. Section 5 summarizes the conclusions and exposes our current works. We close in Appendix A with an example of reasoning with ontologies.

[1] Note that we are not referring to languages intended to *describe* the reasoning process; this aspect constitutes an interesting research area in which important contributions have been made, particularly those by Potts and Bruns ([17]), and by Ramesh and Dhar ([18]).

2 Scientific Theories, Models, Hypotheses, and Ontologies

In this section we review some terms that we use in the rest of this paper, from the perspective of Philosophy of Science. The central concept is that of "*scientific theory*". There are several philosophical approaches to what a theory is. A broadly accepted approach is the so-called "semantic view". It considers that a theory can be defined by a class of structures that provide an interpretation for it (a *semantics*); these structures are called *theoretical models* or, simply, *models*. Models can be defined in a variety of languages, none of which is the basic or unique expression of the theory. Some contributions to the semantic view are those by Suppes ([21, 22]), Suppe ([20]), van Fraasen ([7]), and Giere ([8]).

In particular, Giere's approach ([8]) understands a theory as comprising two elements: (1) a family of interrelated *theoretical models*, and (2) various *theoretical hypotheses* that claim the *similarity* among models in the family and parts of the real world, in indicated respects and to some specified degrees of accuracy. Giere's theoretical models are conceptual, idealised systems (e.g. those discussed in mechanics texts) that jointly provide the semantics of a theory. Hypotheses are true if the models do fit the world in the indicated respects and degrees, and they are false otherwise. Theoretical models of the same family are related between them by similarity relations ("resemblance" –Giere, [8], p. 86); in some cases, they may constitute different approximations to a real world situation.

Morgan and Morrison ([14]) argue that scientific models are instruments for investigation, and they point out several functions of models as instruments. One of these functions is to aid in theory construction. The theoretical model that we have proposed (Section 3) is intended to investigate the relationship between ontologies and reasoning during the resolution of problems; therefore, the model contributes to the development of a theory about such relationship.

Giere ([9]) argues that theoretical models can be used for making *predictions* about the reality that they represent. If the model is proven to fit the world in certain respects and to some specified degrees of accuracy, then the predictions made from the model are also true in the world. Predictions, in turn, allow to *learn* with the model, another of the characteristics pointed out by Morgan and Morrison ([14]). Section 4 describes some predictions obtained from our proposed model of methodological works.

Ontology is a branch of Philosophy concerned with the study of what exists. In Computer Science, ontologies are of great interest for knowledge acquisition and representation, and recently also for Semantic Web. A popular ontology definition is that by Gruber ([10]): an ontology provides "*an explicit specification of a conceptualization*". Mylopoulos ([15]) emphasizes the role of ontologies in acquiring the right concepts to model a world for which one would like to do computations or knowledge management operations. Jurisica, Mylopoulos, and Yu ([12]) classify ontologies for knowledge representation into four broad categories: static, dynamic, intentional, and social.

3 A Theoretical Model of Methodological Works

In this section we present and validate a theoretical model that fits a class of methodological works. In spite of its simplicity, the model has been an adequate instrument for investigation of the role of ontologies in problem solving, allowing us to obtain a number of conclusions in the form of predictions, which are exposed in the next section. The reality to be modelled is constituted by the methodological works in Software Engineering. A methodological work is that aimed to be applied by a practising engineer to any problem in a class, with the hope of contributing to its resolution. They form a conceptual reality, and we can find descriptions of them in research papers and textbooks.

3.1 Description of the Model

The first component of our model is a study of the class of problems that are intended to be solved. The analysis covers different aspects of the problems, which probably constitute novel approaches to the study of the class. Several concepts are defined, and the study of some of the identified aspects is presented through these concepts. Sets of interrelated concepts are grouped in ontologies.

The model also incorporates a number of guides, which are suggestions for the practising engineer of activities to do. Some of these guides are specific for several of the former ontologies, suggesting how to use them to achieve some purpose which, in turn, contributes to the resolution of the problems in the class. A special type of guides is constituted by logical schemes that, when they are instantiated by the practising engineer on a concrete problem, result in conclusions that contribute to the problem resolution (e.g. to conclude to make some design decision). The ontologies with guides may have a textual or graphical syntax associated to their concepts, although it is not strictly necessary.

The concepts introduced to study the class of problems can have properties on their own, from an abstract point of view. A last component of the model is constituted by these properties, together with the properties derivable from the former guides.

The above components are related by a constraint: the concepts and guides must actually contribute to the resolution of the class of problems. If this constrains holds and the former studies are correct, the methodological work is considered to be valid.

3.2 Validation of the Theoretical Model

Giere ([9]) proposes a program to validate theoretical hypotheses, and, hence, theoretical models. The program is based on making predictions from the theoretical model. If such predictions do not agree with experimental data, then the model does not fit the world and the hypothesis is false. Otherwise, the hypothesis is considered to be true if there are no alternative models that explain the same predictions.

Seven predictions from our model are presented in the next section where we also reason the soundness of each one, thus contributing to the validation of the

Table 1. Some of the problem aspects studied in "Problem Frames"

Aspect	Description
$\mathcal{A}_1$	There exist classes of typical software problems. Some of these classes have typical decompositions in terms of others.
$\mathcal{A}_2$	The physical, spatial extension of software problems.
$\mathcal{A}_3$	The extension of software problems from the viewpoint of the customer's needs.
$\mathcal{A}_4$	The different roles played by the physical elements of a software problem.
$\mathcal{A}_5$	The variability in each class of softw. problems from a physical perspective.
$\mathcal{A}_6$	The diversity in the domain nature and its impact in each class of software problems.
$\mathcal{A}_7$	The logical correction of each class of software problems.
$\mathcal{A}_8$	The impact of the failure of a reliable domain in a software problem.

Table 2. Some of the concepts defined in "Problem Frames" for studying each problem aspect. We have put together each group of related concepts in an ontology

Aspect	Concepts	Ontology
$\mathcal{A}_1$	Problem Frame, Information Display Frame, ...	$\mathcal{O}_1$
$\mathcal{A}_2$	Domain, Interface, Phenomenon, Description, ...	$\mathcal{O}_2$
$\mathcal{A}_3$	Requirement, Customer's Authority, Customer's Responsibility	$\mathcal{O}_3$
$\mathcal{A}_4$	Operator, Machine, Display, Real World, Workpieces, ...	$\mathcal{O}_4$
$\mathcal{A}_5$	Variant, Description Variant, Operator Variant, ...	$\mathcal{O}_5$
$\mathcal{A}_6$	Flavour, Static Flavour, Dynamic Flavour, ...	$\mathcal{O}_6$
$\mathcal{A}_7, \mathcal{A}_8, ...$	Concern, Frame Concern ($\mathcal{A}_7$), Reliability Concern ($\mathcal{A}_8$), ...	$\mathcal{O}_7$

whole model. We have not found an alternative theoretical model that explains all the predictions.

Besides, our model agrees with Wieringa's account on design-related research (Wieringa, [26]). He reasons that, during any design process, both the problem properties and the solution properties must be studied. According to the author, this is also applicable to the design of methods.

3.3 Example: Modelling the "Problem Frames" Methodological Work

The preface of "Problem Frames" (Jackson, [11]) states on page xii: "*The central idea of this book is to use problem frames in problem analysis and structure*". A "software problem" is a general and incomplete specification of the responsibilities of a software system in the context of a composite system[2] in which it is immersed. Analysis is the problem of identifying the concerns and difficulties of a software problem. Structure is the problem of designing a correct decomposition of a software problem into subproblems, which ideally contributes to an easier

[2] A composite system includes people, hardware, software, and lexical entities.

Table 3. The fist column indicates some guides included in "Problem Frames". The second column specifies the ontologies that are directly involved in each guide. The page and chapter references are relative to (Jackson, [11])

Guides	**Ontologies**
$\mathcal{G}_1$ (*heuristic*): identify ancillary problems. This guide is based on the knowledge that in most software problems there are ancillary subproblems surrounding the core (p. 293).	$\mathcal{O}_1$
$\mathcal{G}_2$ (*heuristic*): identify and address the concerns of the frames that have already been identified for a problem. This guide is based on the knowledge that each problem frame has a number of typical concerns (chapter 9).	$\mathcal{O}_1 + \mathcal{O}_7$
$\mathcal{G}_3$ (*heuristic*): study the software problem beyond the software system interface. This guide is based on the knowledge that the software problem is immersed in and interacts with a composite system (pp. 7–10).	$\mathcal{O}_2$
$\mathcal{G}_4$ (*heuristic*): expand your study of the composite system to the extent that the customer's responsibilities are covered, without trespassing the customer's authority. This guide is based on the knowledge that the software problem requirements must not be too small in relation to the customer's responsibilities, and that the customer's authority limits the scope of what the software system may legitimately be designed to do (pp. 29–33).	$\mathcal{O}_2 + \mathcal{O}_3$
$\mathcal{G}_5$ (*heuristic*): a valid way to address the failure detection in the reliability concern of a problem is to insert an information subproblem to audit failures. This guide is based on knowledge about the reliability concern (pp. 248–257).	$\mathcal{O}_1$ + Reliability Concern ($\mathcal{O}_7$)

development of the software. Therefore, we can summarize the class of problems which the methodological work is intended for as: "*how to analyse and structure software problems?*".

The approach includes a vast study of numerous aspects of the cited class of problems. Table 1 shows a possible relation of some of these aspects; other classifications can also be valid. Several concepts defined in the method for the study of each aspect have been collected in Table 2, where we have also proposed a possible grouping of the concepts in different ontologies. The main concept is "Problem Frame", which is a synonym for a known class of software problems. The only concepts in Table 2 that have an associated syntax are those of ontology $\mathcal{O}_2$ together with the concept "Requirement" (in $\mathcal{O}_3$).

As we will prove in Section 4.1, some ontological concepts, in addition to be useful for the study of the class of problems, are also intended to contribute to the engineer's reasoning in solving any particular problem of the class. Such ontologies are associated with guides. In "Problem Frames" we can find guides to be used with several ontologies, including $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{O}_3$, $\mathcal{O}_4$, and $\mathcal{O}_7$; Table 3 shows some of them. Some guides as $\mathcal{G}_2$ require concepts from several ontologies.

Note that the ontologies without an associated syntax may also have associated guides (e.g. $\mathcal{O}_1$).

One of the guides intended to help in locating and bounding software problems is related with the customer's authority and responsibility (pp. 31–33). We have identified it as $\mathcal{G}_4$ in Table 3. The "customer" is a notional person representing all the people who are entitled to contribute to the requirement in a software problem (Jackson, [11], p. 363). The guide is intended to be used with the so-called "context diagrams", which are elaborated from the concepts of ontology $\mathcal{O}_2$. It suggests that the domains that must be considered in analysing a software problem must be limited by the customer's authority, while covering the customer's responsibility. But the ontology $\mathcal{O}_2$ does not make explicit the concept of "Domain Responsibility", so we find that, in practice, the guide is difficult to be used with $\mathcal{O}_2$. We will return to this topic in Section 4.5.

"Problem Frames" includes some properties of the ontological concepts from an abstract point of view (third component of the model). For example, those concepts with an associated syntax (e.g. those in $\mathcal{O}_2$) have a set of abstract properties that allow to combine instances of them, forming different graphical models (diagrams). A sample property is that two instances of "Domain" cannot be directly associated but they need an instance of "Interface".

4 Predictions from the Theoretical Model

In this section we present seven predictions inferred from the theoretical model previously introduced. We argument the validity of each obtained prediction, thus contributing to the validation of the whole theoretical model (Section 3.2).

4.1 Ontologies may Promote Strategies

Research from Cognitive Psychology shows that individuals develop and use *strategies* to solve problems, not necessarily in a conscious manner (Schaeken et al., [19]; Van der Henst et al., [23]). We propose the following working definition: a strategy is a particular reasoning approach towards the resolution of a problem in a certain class of related problems. We are interested in those strategies that can be applied not only to a particular problem instance in the class, but to all of them or, at least, to a number of them.

In his famous problem-solving method from 1945, George Polya emphasizes the importance of considering different aspects of the problem and combining them to form novel perspectives, which may lead to a solution strategy: "*Consider your problem from various sides. Emphasize different parts, examine different details, examine the same details repeatedly but in different ways, combine the details differently, approach them from different sides. Try to see some new meaning in each detail, some new interpretation of the whole.*[...]" (Polya, [16], p. 34). Our theoretical model of methodological works agrees with this principle: the concepts in the ontologies consider different aspects of the problems and they allow to study these aspects together. Therefore, ontologies that capture aspects

Table 4. The two main strategies in "Problem Frames". Note that $\mathcal{S}_2$ is a sub-strategy for realizing $\mathcal{S}_1$. Page numbers refer to (Jackson, [11])

Ontology	Strategy
$\mathcal{O}_1$	$\mathcal{S}_1$: Analyse a software problem by reducing it to a combination of known class of problems. Design a structure for a software problem by composing known classes of problems (pp. 59–61).
$\mathcal{O}_2$	$\mathcal{S}_2$: Ground the analysis and structure of software problems in observable, physical phenomena: this will help to check whether we are really satisfying the requirements or not (pp. 22–23).

of the problem may inspire the emerging of new strategies to address the problem. The hypothesis that considering novel aspects and combining them can lead to new ideas is consistent with the creativity theory by the psychologist Boden ([2, 3]), who defends the association of concepts as a valid process of emerging of new ideas. Ward et al. ([24]) refer to this process as "conceptual combination".

According to our initial definition, a strategy may contribute only to some respect of the overall resolution of the problem. Therefore, a number of strategies may be necessary to constitute a complete method for a class of problems; this is one of the reasons why the theoretical model allows several ontologies (another reason will be explained in Section 4.5). For example, sometimes a strategy is needed to carry out a higher-level strategy. This happens in Jackson's Problem Frames (see the example below).

From the preceding discussion we can conclude that it is possible to design ontologies that motivate the invention of strategies that contribute to the resolution of some problem. At the moment we do not have a theory that fully characterizes the class of ontologies that promote strategies. However, in this section we have proven that ontologies that explore different perspectives of problems are good candidates for promoting useful strategies.

Example: Strategies in "Problem Frames". Table 4 show the two main strategies proposed by the "Problem Frames" approach. The observation that there exist classes of typical software problems (aspect $\mathcal{A}_1$ in Table 1) has motivated strategy $\mathcal{S}_1$. This strategy is quite general, and it needs at least another one to be realized; $\mathcal{S}_2$ proposes a possible way of achieving $\mathcal{S}_1$. It has been motivated by the observation that each class of software problems has a defined spatial structure (aspect $\mathcal{A}_2$).

4.2 Guides Suggest How to Carry Out Strategies Promoted by Ontologies

The former prediction has proven that ontologies included in the theoretical model of Section 3.1 may inspire strategies. The theoretical model also includes guides related to the use of the ontologies. In this section we will prove that these guides are the methodologist's suggestions for carrying out the corresponding strategies promoted by the ontologies.

Consider an ontology without associated guides. According to the problem-solving approach and creativity theory exposed in the previous section, the practising engineer could still invent her own strategy while experimenting with an ontology on a concrete problem, even in the absence of guides. However, in the general case, it is not possible to prove the contribution of such an ontology to the resolution of the class of problems, implying that the methodological work could not be validated. As validation of the method is one requirement of our model, the role of ontologies without guides is restricted to the study of some aspect of the class of problems.

4.3 Ontologies in Methods are Reasoning Instruments

Ontologies are intended to be used by the practising engineer, together with guides that help her to carry out the related strategies. To applying an strategy means that the practising engineer must *reason* with the concepts in the corresponding ontology. Guides help her in this reasoning to a certain extent, specially those that we have called "logical schemes" in Section 3.1.

4.4 Strategies Promoted by Ontologies Apply Knowledge to the Resolution of Problems

The ontologies of the theoretical model have two kinds of associated knowledge. On the one hand, a portion of the study on the class of problems: the one related to the problem aspects that the ontological concepts represent. On the other hand, the abstract properties of the concepts. Therefore, any strategy promoted by some ontology is based on and applies the knowledge associated to its ontology.

4.5 A Method may Have Alternative Strategies

We have reasoned in Section 4.1 that our model allows that several strategies coexist, each one contributing to solve some respect of the whole problem. However, the model does not impede that two strategies contribute to the resolution of the *same* respect. Each one could apply a different portion of the method knowledge, and each one could be driven by different sets of guides. If, in dealing with a concrete problem, strategy A has failed in solving a subproblem or it is not applicable to the concrete case, the practising engineer could try strategy B for solving the same subproblem. Below we present an example of an alternative strategy to $\mathcal{S}_2$.

Example: Introduction of A New Strategy in "Problem Frames". The different roles introduced by Jackson (aspect $\mathcal{A}_4$, ontology $\mathcal{O}_4$) denote different *responsibilities* of the domains with respect to the composite system. The fulfillment of the responsibilities causes the appearance of an *emergent behavior* (Wieringa, [25]), which may or may not be what the customer expects. The study of responsibilities in composite systems has a long tradition in Software Engineering (Feather, [5]). A related concept is that of *Goal*, which has been proven to be a useful resource for this kind of analysis, particularly in the KAOS

Agent an active system component (or "processor") which may have choice of behaviour to ensure the goals it is assigned to (Feather, [5]).

And/Or Goal Reduction a mechanism for goal refinement: g is a reduction of G iff achieving goal g possibly with other subgoals is among the alternative ways of achieving goal G (Dardenne *et al*, [4]).

Goal an objective to be achieved by the system under consideration (Letier and van Lamsweerde, [13]). "System" refers to the composite system consisting of the software-to-be together with its environment (Fickas and Helm, [6]).

Goal Pattern classification based on the temporal behaviour required by the goal. It can be *achieve*, *cease*, *maintain*, and *avoid* (Dardenne *et al*, [4]).

Responsibility Assignment assigning responsibility to an agent means that this agent must restrict its behaviour so as to ensure the goal (Dardenne *et al*, [4]).

Fig. 1. Some concepts of the KAOS ontology ($\mathcal{O}_G$)

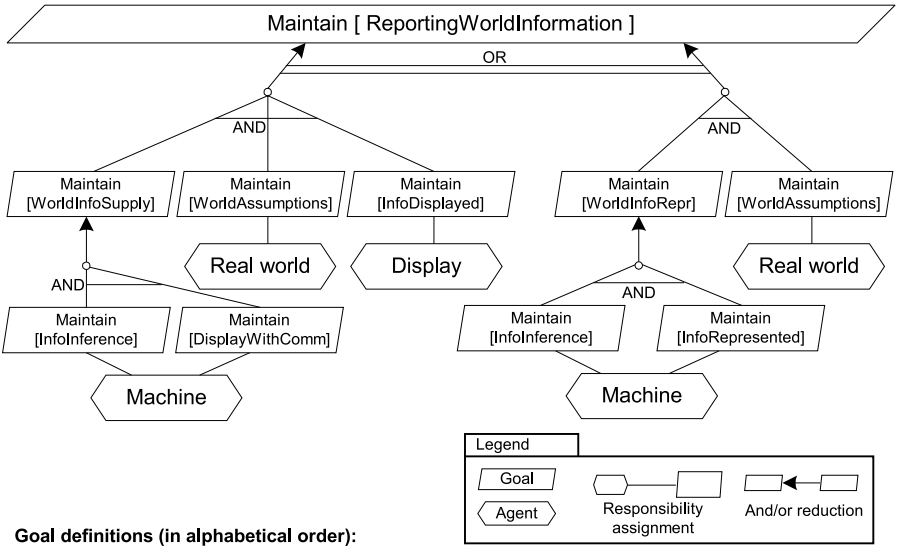
approach (Dardenne, Fickas, and Lamsweerde, [4]). This has encouraged us to borrow the KAOS ontology from (Letier and Lamsweerde, [13]), and to use it to study the class of problems addressed in (Jackson, [11]). Figure 1 defines some of the concepts in the cited ontology, which we will denote as $\mathcal{O}_G$.

This study revealed that each problem frame has a defined structure in terms of goals. For example, Figure 2 shows the goal structure of the "Information Display Frame". The figure describes two alternative ways of achieving the goal Maintain[ReportingWorldInformation], which is the higher-level goal of the frame. The left goal tree requires the collaboration of three agents: Machine, Real world, and Display. The right goal tree does not require the Display, because the data about the real world are represented as pure information (in a lexical domain).

The fact that each problem frame has a corresponding goal structure, motivated us for defining the following strategy, which contributes to achieve the general strategy $\mathcal{S}_1$:

> $\mathcal{S}_G$: in problem analysis, identify problem frames by looking for the goals and responsibilities in the problem; in problem structure, design subproblems by thinking about goals that must be satisfied by lower-level goals, which will be ultimately realized by agents. For both purposes, use the knowledge of the goal structure of problem frames.

In order to realize this strategy, we have proposed a number of guides related with $\mathcal{O}_1$ and $\mathcal{O}_G$. Table 5 shows some of these guides. Figure 3 in Appendix A shows an example of reasoning with $\mathcal{O}_G$ in analysing and structuring the "package router control problem" (p. 270 of Jackson, [11]). Note also that $\mathcal{O}_G$ allows a more easy application of guide $\mathcal{G}_4$ (Section 3.3.), as it makes explicit the responsibilities assigned to each Domain (Agent).



Goal definitions (in alphabetical order):

Maintain[DisplayWithComm] = Inferred information is translated to display commands.
Maintain[InfoDisplayed] = Display what is indicated through commands.
Maintain[InfoInference] = Information of interest about the world is inferred from observed data.
Maintain[InfoRepresented] = Inferred information is represented in a lexical domain.
Maintain[ReportingWorldInformation] = Certain information of interest about the world is continually needed at the required place in the required form.
Maintain[WorldAssumptions] = Assumptions about the world used on inference are never violated.
Maintain[WorldInfoRepr] = Information of interest about the world in represented in a lexical domain.
Maintain[WorldInfoSuppy] = Information of interest about the world is supplied to Display in the form of displaying commands.

Fig. 2. Goal structure for the "Information Display Frame". The figure includes the legend for the some concepts in $\mathcal{O}_G$

4.6 Methods Can be Regarded as Scientific Theories

If we recall the description of the theoretical model (Section 3.1), it contains a study of the problems in a class. Such class is a reality, so the study can be considered as a theory that claims that the obtained results fit such reality.

Other elements in the model are ontologies and guides. They may exist previously in another context, or they may be invented by the methodologist when she developed the method. In any case, they constitute conceptual realities. The model contains a study of the properties of these elements; as before, this study can be regarded as a theory about a reality. It is necessary for proving the correctness of the methodological work (Section 3.1).

In conclusion, the model can be interpreted as consisting of two theories: one referred to the class of problems, and the other one referred to conceptual elements (ontologies and guides).

Table 5. Some proposed guides for reasoning with $O_1 + O_G$ to realize $\mathcal{S}_G$

Guides	Ontologies
$\mathcal{G}_6$ (*logical scheme*): we have realized that some goal g, which appears as assigned to the software system in the initial statement of some problem p, corresponds to the higher-level goal in the goal structure of a certain problem frame F. Therefore, it is unlikely that the software system alone could operationalise g on its own. Therefore, let us assume that our problem p fits frame F. Applying the goal structure of F to p, we discover the remaining agents and their responsibilities in terms of sub-goals. We verify our initial assumption by checking that the assigned goals make sense in the context of p.	$\mathcal{O}_1 + \mathcal{O}_G$
$\mathcal{G}_7$ (*logical scheme*): We are in doubt about if a certain problem frame F fits our problem p. Let us assume it fits. Applying the goal structure of F, we obtain the relation of agents that should participate and their responsibilities in terms of operationalisation of goals. *If* we find that (1) either some goal demands more than its agent in p is able to do, or (2) several goals demand too little from their associated agents in p, wasting their capabilities, *then* we can conclude that the initial assumption is probably false.	$\mathcal{O}_1 + \mathcal{O}_G$

4.7 The Study of Problems is Central to the Design of a Method

According to the model prediction of Section 4.1, the study of the class of problems under different concepts may motivate the invention of resolution strategies, and therefore it is central to the design of methods. The study is also necessary to prove that the methodological work is valid; this is described in the constraint stated in the theoretical model (Section 3.1). This reason has been also pointed out by Wieringa ([26]).

5 Conclusions and Current Work

This paper has presented a contribution to the relation between ontologies that constitute modelling languages and an individual's reasoning process during problem solving in Software Engineering. Our research method has been to make predictions from a theoretical model of methodological works. This approach has led to a number of interesting conclusions (Section 4), including the property that ontologies may inspire solving strategies, and hence they are essential instruments for reasoning during problem solving. In particular, this property establishes that it is possible to design modelling languages that help the engineer to reason in the problem solving process of software development. However, we do not have a complete theory that characterizes the whole class of ontologies that motivate reasoning strategies. Our current work is to advance in this research area.

Acknowledgments. We want to thank the anonymous reviewers for their work in reading the first version of this paper and for their useful comments. We also want to thank María del Carmen Serrano Jiménez for the English grammatical revision of this work.

References

[1] Cañete, J.M., Galán, F.J., Toro, M. (2004). Some Problems of Current Modelling Languages that Obstruct to Obtain Models as Instruments. *Proceedings of the IX Spanish Conference on Software Engineering and Databases (JISBD'2004).*

[2] Boden, M. (1990) *The Creative Mind: Myths & Mechanisms*. Basic Books.

[3] Boden, M. (1994). What is creativity? In M. A. Boden (Ed.), *Dimensions of creativity* (pp. 75-117). The MIT Press.

[4] Dardenne, A., van Lamsweerde, A., Fickas, S. (1993). Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20, 3–50.

[5] Feather, M. (1987). Language Support for the Specification and Development of Composite Systems. *ACM Transactions on Programming Languages and Systems*, 9:2, 198–234.

[6] Fickas, S., Helm, R. (1992). Knowledge Representation and Reasoning in the Design of Composite Systems. *IEEE Transactions on Software Engineering*, 18:6, 470–482.

[7] Fraasen, B. van (1980). *The Scientific Image*. Clarendon Press.

[8] Giere, R. (1988). *Explaining Science.A Cognitive Approach.*University of Chicago Press.

[9] Giere, R. (1997). *Understanding Scientific Reasoning*. Fourth Edition. Harcourt Brace College Publishers.

[10] Gruber, T.R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5, 199–220.

[11] Jackson, M. (2001). *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley.

[12] Jurisica, I., Mylopoulos, J. Yu, E. (2004). Ontologies for Knowledge Management: An Information Systems Perspective. *Knowledge and Information Systems*, 6, 380–401.

[13] Letier, E., van Lamsweerde, A. (2002). Agent-Based Tactics for Goal-Oriented Requirements Elaboration. *Proceedings of the 24th International Conference on Software Engineering* (ICSE'2002). ACM Press.

[14] Morgan, M. and Morrison, M. (1999). Models as mediating instruments. *Models as Mediators. Perspectives on Natural and Social Science*, pp. 10–37. Cambridge University Press.

[15] Mylopoulos, J. (1998) Information Modeling in the Time of the Revolution. *Information Systems 23 (3–4), 127–155.*

[16] Polya, G. (1945) *How to Solve It. A New Aspect of Mathematical Method*. Princeton University Press.

[17] Potts, C. and Bruns, G. (1988). Recording the Reasons for Design Decisions. *Proceedings of the 10th International Conference on Software Engineering*, pp. 418–427.

[18] Ramesh, B. and Dhar, V. (1992) Supporting Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Transactions on Software Engineering*, 18(6), 498–510.

[19] Schaeken, W., De Vooght, G., Vandierendonck, A., d'Ydewalle, G. (2000). *Deductive reasoning and strategies*. Lawrence Erlbaum Associates.
[20] Suppe, F. (1977). *The Structure of Scientific Theories*. University of Illinois Press.
[21] Suppes, P. (1961). A Comparison of the Meanaing and Use of Models in the Mathematical and Empirical Sciences. *The Concept and Role of the Model in Mathematics and Natural and Social Sciences*, pp. 163–167. Reidel.
[22] Suppes, P. (1967). What is a Scientific Theory? *Philosophy of Science Today*, pp. 55–67. Basic Books.
[23] Van der Henst, J.B., Yang, Y., Johnson-Laird, P.N. (2002). Strategies in sentential reasoning. *Cognitive Science*, 26, 425–468.
[24] Ward, T. B., Finke, R. A., Smith, S. M. (1995). *Creativity and the mind: Discovering the genius within*. Plenum Press.
[25] Wieringa, R. (2003). *Design Methods for Reactive Systems: Yourdon, Statemate and the UML*. Morgan Kaufmann.
[26] Wieringa, R. (2004). Requirements engineering research is the study of design. Internal report. Department of Computer Science, University of Twente, the Netherlands.

A An Example of Reasoning with Ontologies

We show a simple example of reasoning with ontologies $\mathcal{O}_1$ and $\mathcal{O}_G$. The aim is to analyse and structure the "package router control problem", which also solved in pp. 270–291 of (Jackson, [11]) with strategies $\mathcal{S}_1$ and $\mathcal{S}_2$. The following problem statement has been extracted from page 270 of the same reference:

A package router is a large mechanical device used by postal and delivery organisations to sort packages into bins according to their destinations. The packages carry bar-coded labels. They move along a conveyor to a reading station where their package-ids and destinations are read. They then slide by gravity down pipes fitted with sensors at top and bottom. The pipes are connected by two-position switches that the computer can flip (where no package is present between the incoming and outgoing pipes). At the leaves of the tree of pipes are destination bins, corresponding to the bar-coded destinations. A package cannot overtake another either in a pipe or a switch. Also, the pipes are bent near the sensors so that the sensors are guaranteed to detect each package separately. However, packages slide at unpredictable speeds, and may get too close together to allow a switch to be set correctly. A misrouted package may be routed to any bin, an appropriate message being displayed. There are control buttons by which an operator can command the controlling computer to stop and start the conveyor.

The problem is to build the controlling computer to obey the operator's commands, to route packages to their destination bins by setting the switches appropriately, and to report misrouted packages.

Thinking in terms of $\mathcal{O}_G$, we can identify three goals from the problem statement, which appear as assigned to the software system (box 1 in Figure 3). Reasoning with the knowledge from guide $\mathcal{G}_6$, we conclude that the goal Report misrouted packages may be the high-level goal of an Information Display Frame. Hence, it is unlikely that the Machine could operationalise this goal only by itself. Box 2 in Figure 3 shows the identified frame, which is an instance of the Information Display Frame concept in $\mathcal{O}_1$. Next, according to the suggestion of

$\mathcal{G}_2$, we realize that the Reliability Concern is important for this problem: the assumptions about the packages may likely fail, and this would bring undesirable consequences for the composite system (we cannot trust in agent Router & packages to satisfy its goal). Following guide $\mathcal{G}_5$, we introduce a new Information Display frame that audits these possible failures (box 3 in Figure 3).

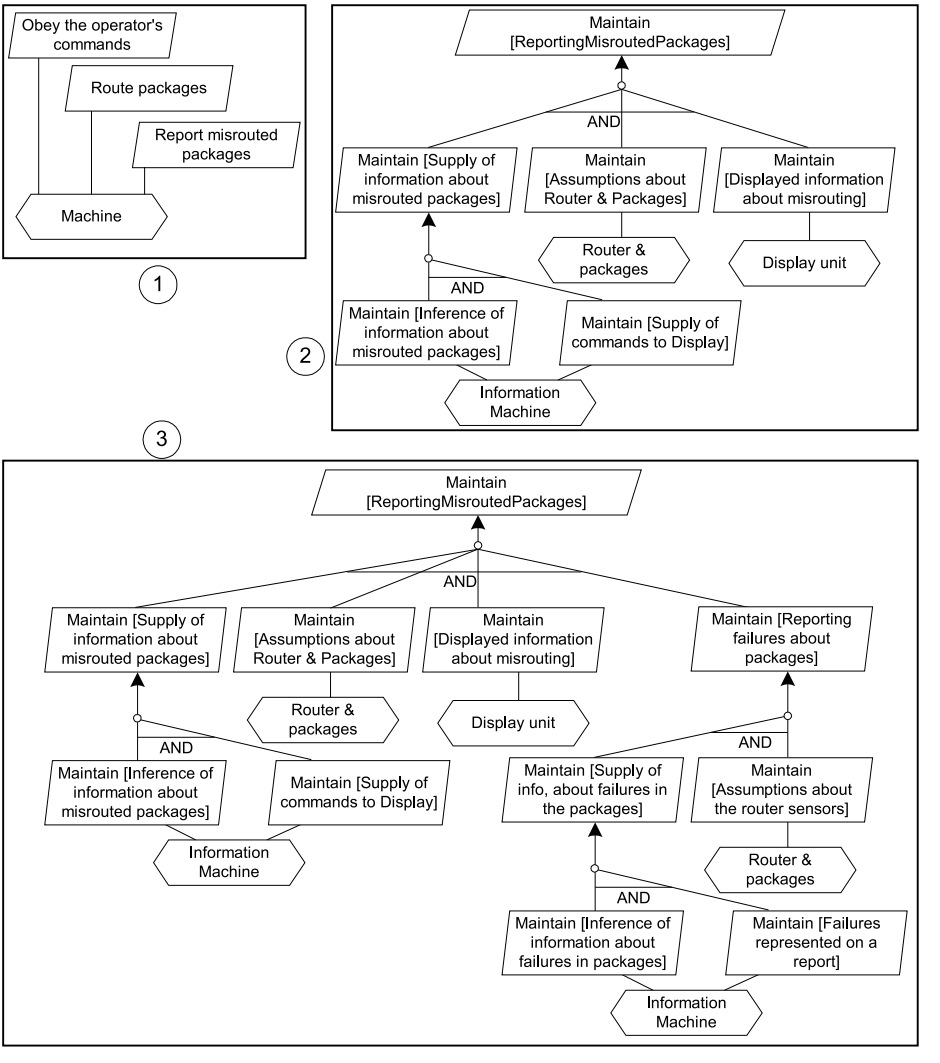


Fig. 3. A simple example of reasoning with $\mathcal{O}_1$ and $\mathcal{O}_G$