# Modeling the Purposes of Models

Cédric Jeanneret, Martin Glinz

University of Zurich
Binzmühlestrasse 14
CH-8050 Zurich, Switzerland
{jeanneret, glinz}@ifi.uzh.ch

Thomas Baar

Hochschule für Technik und Wirtschaft Berlin
Wilhelminenhofstraße 75A
D-12459 Berlin, Germany
thomas.baar@htw-berlin.de

**Abstract:** Today, the purpose of a model is often kept implicit. The lack of explicit statements about a model's purpose hinders both its creation and its (re)use. In this paper, we adapt two goal modeling techniques, the Goal-Question-Metric paradigm and KAOS, an intentional modeling language, so that the purpose of a model can be explicitly stated and operationalized. Using some examples, we present how these approaches can document a model's purpose so that this model can be validated, improved and used correctly.

## 1 Introduction

With the advent of Model Driven Engineering (MDE), models play a more and more important role in software engineering. Conceptually, a model is an abstract representation of an original (like a system or a problem domain) for a given purpose. One cannot build or use a model without knowing its purpose. Yet, today, the purpose of a model is often kept implicit. Thus, anybody can be mislead by a model if it is used for a task it was not intended for. Furthermore, a modeler must rely on his experience and his feelings to decide how much and which detail is worth being modeled. This may result in models at the wrong level of abstraction for its (unstated) purpose. Stating the purpose of a model explicitly is only a first step to address these issues.

Eventually, the purpose of a model can be characterized by a set of operations. There are two kinds of operations: (i) operations performed by humans to interpret (understand, analyze or use) the model and (ii) operations executed by computers to transform the model into another model (model transformations). Being able to express the purpose of a model with a set of model operations allows to measure how well a model fits its purpose. In previous work [JGB11], we have made a contribution towards measuring the confinement of a model (the extent to which it contains relevant information) given the set of formal operations to be executed on it.

Having a set of operations is not enough, though: we must ensure that these operations can be performed on the model — no matter whether these operations are performed by humans or executed by computers. For this, we have to make explicit which information the operations need from the model and we have to determine which structures a model has to conform to. In other words, we need to state which elements the metamodel must contain for enabling the operations.

Our previous work assumes that these operations and these metamodels exist. This assumption may hold in an MDE context, but not in a wider context: Often, the purpose of a model is not even stated explicitly. Thus, there is a need for (a) methods to elicit and document modeling purposes in the first place and (b) methods to operationalize these modeling purposes systematically. In goal modeling, there are many approaches for these two tasks. However, these approaches were designed for other contexts than modeling.

In this paper, we adapt two of these goal modeling approaches for systematically deriving a set of of model operations and associated metamodel elements from a qualitatively stated model purpose. First, we present Goal-Operation-Metamodel (GOM), a generalization of the Goal-Question-Metric (GQM) paradigm [Bas92]. Second, we propose to use KAOS [vL09] (a goal modeling language) as a metalanguage to create intentional metamodels.

The remainder of this paper is organized as follows. In the next section, we present the problem context of our work in more details. In Section 3, we describe the Goal-Operation-Metamodel method and we present intentional metamodeling with KAOS in Section 4. We discuss our findings in Section 5 while Section 6 discusses related work. Finally, we conclude the paper in Section 7.

## 2 Problem Context

Many modeling theories distinguish two roles in the model building process: the *modeler* and the *expert*. Modeling is a collaborative activity involving a dialog among these two roles: The modeler elicits information about the original from the expert before formalizing it, while the expert validates the content of the model as explained by the modeler. These roles and the relationships are reprented in Figure 1. Hoppenbrouwers et al. even consider the model as the minutes of this dialog [HPdW05].

While building a model may be valuable on its own, the value of modeling consists of using the model as a substitute of the original to infer some new information about it. These inferences are made by the *interpreter* – a third role related to modeling. To achieve this, the interpreter performs various *model operations* on the model, like executing queries on it, extracting views from it or transforming it to other models or artefacts.

When describing the nature of modeling, Rothenberg listed the following purposes of models [Rot89]:

> The purpose of a model may include comprehension or manipulation of its referent [the original], communication, planning, prediction, gaining experience, appreciation, etc. In some cases this purpose can be characterized by the kinds of questions that may be asked of the model. For example, prediction corresponds to asking questions of the form “What-if...?” (where the user asks what would happen if the referent began in some initial state and behaved as described by the model).

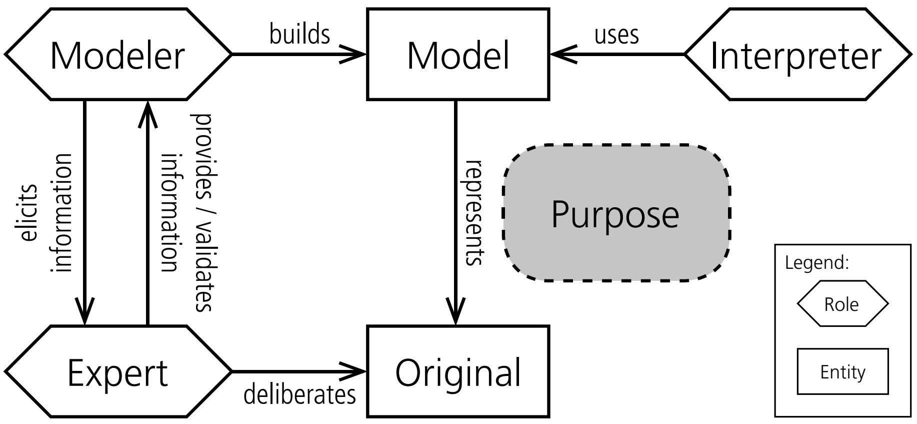


Figure 1: The roles involved in a modeling activity.

A clearly stated modeling purpose can be used as a contract between the modeler and the interpreter. Establishing contracts is costly, as they must be negotiated and edited. Nevertheless, such a contract can be useful in two ways: First, as a specification for a model's purpose, it provides a strong basis on which the model can be validated. It can also provide the modeler with some guidance for improving the model so that it reaches the right level of abstraction. Second, as a description of a model's purpose, it tells an interpreter whether the model at hand is fit for the intended use or, if several models are available, it helps him to choose which model will best fit his purpose.

In the vein of [LSS94], we consider a model as a set of statements $M$. For each modeling purpose, there is a set of relevant statements $D$. In an ideal case, the set $D$ should correspond to the set of statements in the model $M$. When the sets $M$ and $D$ differ, we can quantify the deviation of $M$ from $D$ by using measures from the Information Retrieval field: precision and recall. Precision measures the *confinement* of a model, the extent to which it contains relevant statements. Recall, on the other hand, measures the *completeness* of a model, that is, the proportion of relevant statements that has actually been modeled. By measuring the confinement and completeness of a model, a modeler can assess how adequate is its level of abstraction for its purpose. Indeed, a model at the right level of abstraction for its purpose is both confined and complete ($M = D$).

However, defining the set $D$ is challenging. In our previous work [JGB11], we made a contribution towards measuring the confinement of a model given a set of operations that characterizes its purpose. When these operations are executed on a model, they navigate through its content and gather some information by reading some of its elements. The set of elements touched by a model operation during its execution forms the *footprint* of that operation. Thus, the footprint contains all elements that affect the outcome of the operation. For a set of operations, the *global footprint* of the set of operations is the union of the footprints of each operation. This global footprint is the intersection $M \cap D$: it is the set of statements that are both present in the model and used to fulfill its purpose.

In this paper, we propose two approaches to operationalize a qualitatively stated modeling purpose into a set of model operations and their supporting metamodels. Instead of inventing new methods from scratch, we adapt two existing goal modeling techniques, GQM [Bas92] and KAOS [vL09], so that they can be used in a modeling context in addition to measurement and requirements engineering, respectively. To illustrate the use of these methods in modeling, we first present two examples.

## 2.1 Motivating Examples

In this section, we introduce two examples to motivate and illustrate our approaches to capture the purpose of a model. The first example is the *London Underground map*, used by Jeff Kramer in [Kra07] to highlight that the value of an abstraction depends on its purpose. The second example is related to Software Engineering, where an architect models her (or his) system according to the "4+1" viewpoints of Kruchten [Kru95] for making some performance analysis as described in [CM00]. We have used this example in our previous paper to explain the various usage scenarios of model footprinting [JGB11].

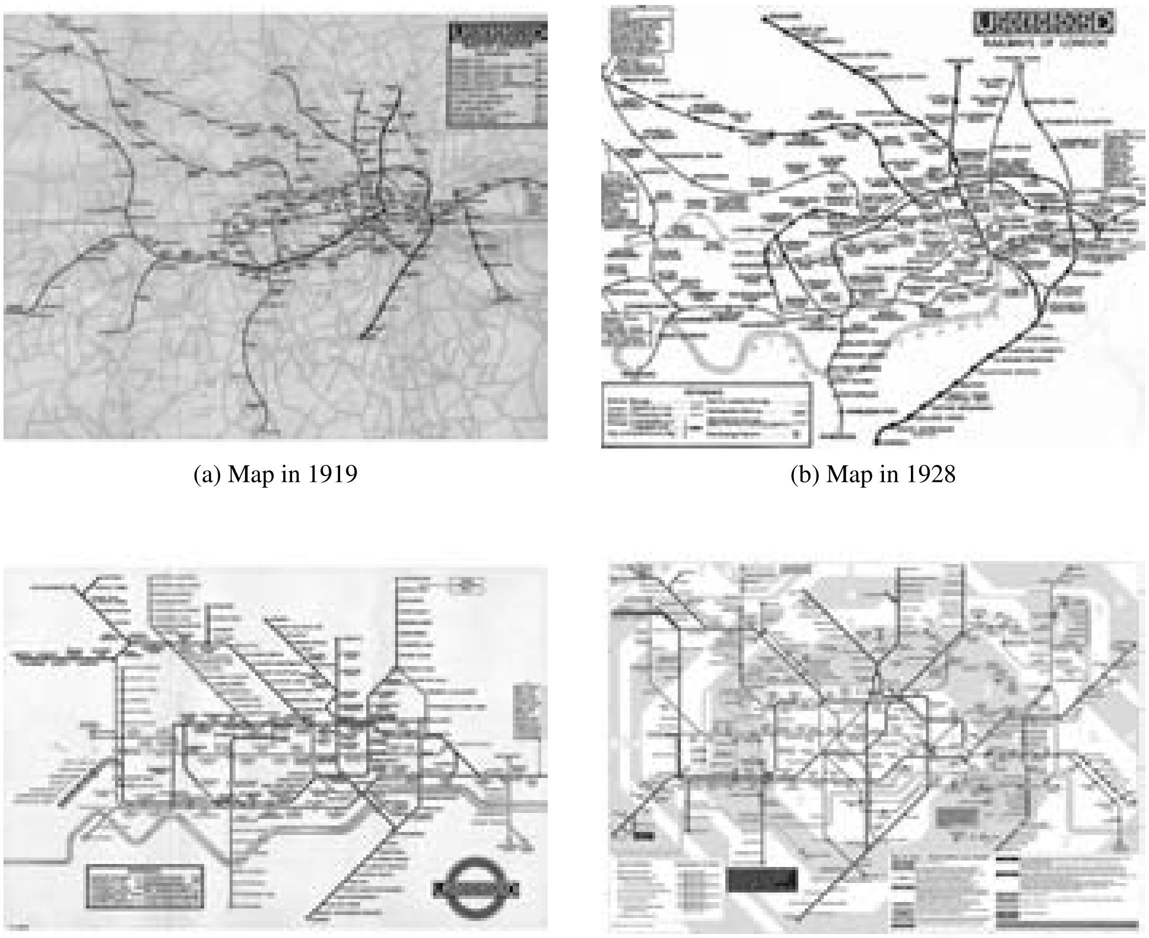
### The London Underground Map

As most major cities, London has an underground railway system. To help its users to navigate in London with it, its operator, the *Transport for London* (TfL) company provides a map of this transit system. Figure 2 shows the evolution of the map along the years. In 1919 (Figure 2a), the map was a geographical map of London with the underground lines overlaid in color. In 1928 (Figure 2b), ground features like streets were removed from the map and the outlying lines were distorted to free some space for the congested center, making it more readable. The first schematic representation of the network appeared in 1933 (Figure 2c): the precise geographic location of stations is discarded; only the topology of the network is represented. The current map (Figure 2d) contains additional information such as the accessibility of stations, the connections to other transportations systems and fare zones.

In this example, the modeler is the employee of TfL designing the map. The expert is an employee of TfL who knows the underground network well. The interpreter is a user of the map. The map is used to plan trips in London, that is, the map must help travelers to answer the following questions: how to get from A to B? How much does it cost? How long does it take? Is that route accessible for disabled people? Interestingly, in this example, the people who use the model to plan their trip also use the modeled system to actually travel in London.

### Performance Analysis on Software Architecture

For the second example, we consider an architect analyzing the performance of a piece of software. To this end, she describes its architecture using the "4+1" view model proposed



(a) Map in 1919

(b) Map in 1928

(c) Map in 1933

(d) Map in 2009

Figure 2: Maps of the London Underground.
(a), (b) and (c): ©TfL from the London Transport Museum collection
(d): ©Transport for London

by Kruchten in [Kru95]: This view model includes (1) use case and sequence diagrams for the scenario view, (2) class diagrams for the logical view, (3) component diagrams for the development view, (4) activity diagrams for the process view and (5) deployment diagrams for the physical view. Her model is first transformed into an extended queueing network model (EQNM) as explained by Cortellessa et al. in [CM00]. Performance indicators are then measured on the EQNM. The architect wants the following questions to be answered: What is the response time and throughput of her system? Where is its bottleneck?

In this example, EQNMs can be seen as the semantic domain for architecture models written in UML. There are therefore two chained interpretations: the first interpretation translates a UML model into an EQNM, while the second interpretation analyses the EQNM. In this example, we focus on the translation from UML to EQNM.

The architect plays all three roles in this example. As the architect of her software, she is the expert. As she creates the model, she is the modeler. As she uses the model for evaluating the performance of her software, she is the interpreter. However, there are two additional stakeholders involved in this example: Cortellessa and his team developed the analysis used by the architect, while Kruchten, by defining the "4+1" viewpoint, proposed a "contract" between the modeler and the interpreter.

Contrary to the previous example, the performance analysis is mostly automated. As the architect is only interested in its results, she may know little about the internals of the technique. Thus, the documentation of the analysis must state explicitly which information the analysis requires in input models.

## 3 Goal-Operation-Metamodel

GQM is a mechanism for defining and evaluating a set of operational goals using measurement [Bas92]. In the GQM paradigm, a measurement is defined on three levels:

- At the *conceptual* level, the goal of the measurement is specified in a structured manner: It specifies the purpose of the measurement, the object under study, the focus of the measurement and the viewpoint from which the measurements are taken.
- At the *operational* level, the goal is refined to a set of questions.
- At the *quantitative* level, a set of metrics is associated to each question so that it can be answered in a quantitative manner.

Our approach consists of using GQM for models other than metrics. According to Ludewig [Lud03], metrics are some kind of models. However, GQM has to be extended on its three levels to describe modeling purposes other than quantitative analysis. At the conceptual level, the goal template must support purposes like code generation[1] or documentation. At the operational level, the set of questions will be replaced by a set of (general) operations: Beside queries, one may need simulations and transformations to refine the goal stated at the conceptual level. Finally, the quantitative level becomes the *definable* level: metamodels replace metrics to support the model operations from the operational level. These operations will be run on conforming models in a similar way that questions can be answered with the value of a metric. Thus, we call this approach Goal-Operation-Metamodel (GOM).

### 3.1 GOM and the London Underground Map

Based on the GQM template described in [Bas92], we define the goal of the map as the following:

[1]Code generation, as an operation, is not supported when a model is at a conceptual level. Here, we consider code generation as the model's purpose to be described with GOM.

Analyze *the London Underground*
For the purpose of *characterization*
With respect to *reachability* and *connectivity* of its stations
From the view of *a traveler*
In the following context: *the traveler may be a disabled person*, *the tube is part of a larger transportation system*, *the map is displayed on a screen or on paper in stations*

From this goal, we derive the following questions to be answered from the model:

(a) What is the shortest path between two stations?

(b) How much does it cost to travel along a given path?

(c) How long does it take to travel along a given path?

(d) Is a given path accessible to a disabled person?

(e) When traveling along a path, at which station to leave a train?

(f) When traveling along a path, in which train (line and direction) to enter?

Table 1 lists which questions are supported by the 4 versions of the map displayed in Figure 2. All maps can be used to find the shortest path between two stations and where to step in and step off trains. However, only the 2009 version fully supports disabled people and allows for computing the cost of a trip. Since it preserves the geographic location of stations, the map of 1919 can be used to estimate the time needed for a trip (without taking transfers into account).

Table 1: Operations supported by the different versions of the map.

| Map | Path (a) | Cost (b) | Time (c) | Accessibility (d) | Step Off (e) | Step In (f) |
|---|---|---|---|---|---|---|
| 1919 | √ | – | √ | – | √ | √ |
| 1928 | √ | – | – | – | √ | √ |
| 1933 | √ | – | – | – | √ | √ |
| 2009 | √ | √ | – | √ | √ | √ |

A map conforming to the metamodel depicted in Figure 3 could be used to answer all the questions characterizing the purpose of the map. Segments, lines and stations form the topology of the network, allowing a traveler for planning (question (a)) and executing (questions (e) and (f)) a trip with the Underground. Fare zones are involved in the computation of the cost of a trip (question (b)). The accessibility of a station serves for question (d) while the distance covered by a segment is needed to answer question (c).

In this example, questions are answered “mentally” by the travelers. Still, all these questions could be formalized with queries in OCL or operations in Kermeta [MFJ05]. For example, Listing 1 presents the operation computing the cost of a trip (encoded as a sequence of segments) in Kermeta. This operation is defined for the metamodel presented in Figure 3.

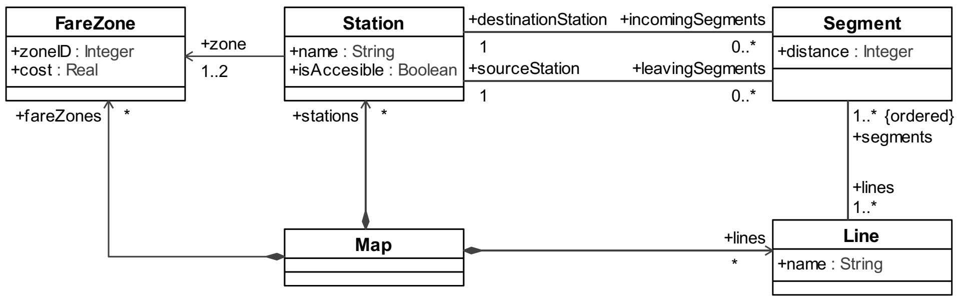


Figure 3: A metamodel for a map of the London Underground.

```
// Compute the cost of a trip
operation cost(path: Sequence<Segment>): Real is do
  result := "0".toReal
  // First, collect all traversed zones
  var traversedZones: Set<FareZone> init Set<FareZone>.new
  path.each{seg |
    var src: Set<FareZone> init seg.sourceStation.zone
    var dst: Set<FareZone> init seg.destinationStation.zone
    var inter: Set<FareZone> init src.intersection(dst)
    if not inter.isEmpty
    then
      // Both stations are in the same zone
      traversedZones.addAll(inter)
    else
      // The segment traverses a boundary
      traversedZones.addAll(src)
      traversedZones.addAll(dst)
    end }
  // Second, sum the cost of each traversed zone
  traversedZones.each{z | result := result + z.cost}
end
```

Listing 1: Operation computing the cost of a trip.

### 3.2 GOM and the Performance Analysis on Software Architecture

In this example, we only consider the immediate goal of the model, which is the generation of an EQNM, and we leave the final goal (the performance analysis) out. Still, both goals could have been captured by GOM. Slightly adapting the GQM template [Bas92], the immediate goal of the model can be stated as follows:

Analyze *the architecture of a software system*
For the purpose of *generating an EQNM*
With respect to *the scenario view* and *the physical view* as defined in [Kru95]
From the view of *the software architect*
In the following context: *the generation of an EQNM is explained in [CM00],*

*this generation is automated, the generated EQNM will be used to analyze the performance of the architecture*

[CM00] describes, formally, the various steps in the generation of the EQNM from UML models:

(1) deduce a user profile from the use case diagram,

(2) combine the sequence diagrams into a meta execution graph (meta-EG),

(3) obtain the EQNM of the hardware platform from the deployment diagram and tailor the meta-EG into an EG-instance for that platform,

(4) assign numerical parameters to the EG-instance, and

(5) assign environment based parameters to the EQNM, process the EG-instance to obtain software parameters before assigning them to the EQNM.

This chain of transformations requires information from the following UML diagrams: use case diagrams, sequence diagrams and deployment diagrams. The other diagrams of the "4+1" model — the class, the component and the activity diagrams — are not needed for this purpose.

While GOM allows to state the purpose of a model explicitly and operationalize it, goals expressed in GOM are not formal enough to be analyzed automatically, for example, to find conflicts among them. In the next section, we present how a model's purpose can be expressed in a goal-oriented modeling language.

## 4 Intentional Metamodeling

In the previous section, we presented a structured but informal way to specify a model's purpose. In this section, we introduce intentional metamodeling with KAOS, a goal modeling language designed for use in early phases of requirements engineering. A KAOS model consists of four interrelated views:

**Goal modeling** establishes the list of goals involved in the system. Refined goals and alternatives are represented in an AND/OR tree. Conflicts among goals are also represented in this diagram.

**Responsibility modeling** captures the agents to whom responsibility for (leaf) goal satisfaction is assigned.

**Object modeling** is used to represent the domain's concepts and the relationships among them.

**Operation modeling** prescribes the behaviors the agents must perform to satisfy the goals they are responsible for.

A goal can be refined into conjoined sub-goals (the goal is satisfied when all its sub-goals are satisfied) or into alternatives (the goal is satisfied when at least one of its alternatives is satisfied). Therefore, goals are represented as AND/OR trees in KAOS. In such a tree, the goals below a given goal explain how and how else the goal can be realized. On the opposite, goals higher in the hierarchy provide the rationale for a given goal, explaining why the goal is present in the model.

[vL09] provides a taxonomy of goals based on their types and their categories. There are two main types of goals: *behavioral* goals (such as Achieve, Cease, Maintain and Avoid goals) prescribe the behavior of a system, while *soft-goals* (such as Improve, Increase, Reduce, Maximize and Minimize goals) prescribe preferences among alternative systems. Similarly, there are two main categories of goals: *functional* goals (like Satisfaction [of user requests] or Information [about a system state] goals) state the intent behind a system service and *non-functional* goals (like Usability or Accuracy) state a quality or constraint on its provision or its development. This taxonomy can be helpful for eliciting and specifying goals.

Goals are refined until they are assignable to a single agent. Leaf goals are then made operational by mapping them to operations ensuring them. Operations are binary relationships over systems states. They can be derived from the formal specification of goals or built from elicited scenarios. Finally, a conceptual model gathers all concepts (including their attributes and the relationships among them) involved in the definition of goals and operations.

We use KAOS as a metametamodel and not as a metamodel as it was initially designed for: In our approach, KAOS models are metamodels. Goals depict the modeling purposes. Operations prescribe the operations that can be executed on models and the conceptual model defines the abstract syntax of the language. Thus, a metamodel written in KAOS specifies many aspects of a modeling task: it states the purpose and intended usage of models as well as their structure. In the remainder of this section, we present KAOS metamodels for our examples.

## 4.1 Intentional Metamodeling and the London Underground Map

A KAOS metamodel of the London Underground map is presented in Figure 4. The main goal of the map is to provide travelers with a means to understand how to travel from a station A to another station B successfully. To achieve this, the map must satisfy the following sub-goals: to help travelers to plan their trip, to help them to buy the right ticket for it and to help them for the navigation, that is, to prevent them from getting lost during their travel.

These goals are operationalized through the following operations performed by the traveler: find the shortest path between stations A and B, compute its cost (by summing the fares of traversed fare zones) and carry out the plan by riding on the right line and connecting on the right station. As we did in Section 3.1, we can define these operations formally and derive a metamodel to support them. For space reasons, this metamodel is not included in Figure 4, but it presented in Figure 3.

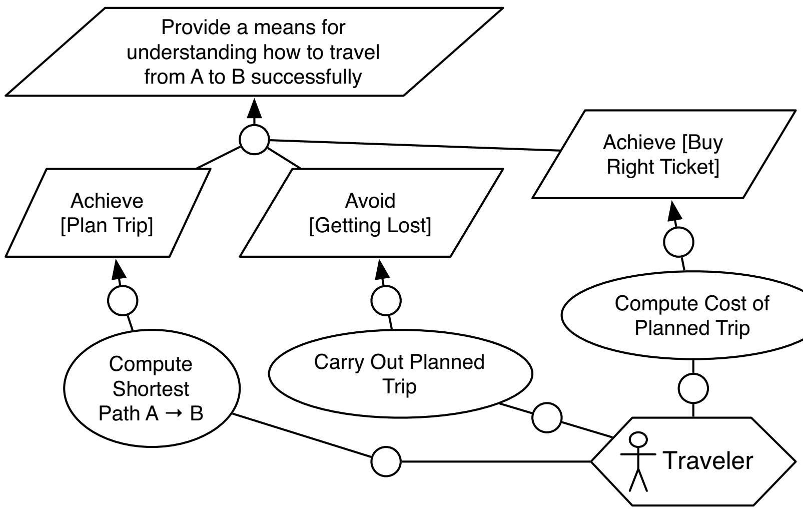

Provide a means for understanding how to travel from A to B successfully
Achieve [Buy Right Ticket]
Achieve [Plan Trip]
Avoid [Getting Lost]
Compute Cost of Planned Trip
Compute Shortest Path A → B
Carry Out Planned Trip
Traveler

Figure 4: A KAOS metamodel for the London Underground map.

### 4.2 Intentional Metamodeling and the Performance Analysis

We present an intentional metamodel of the performance analysis in Figure 5. The final goal of the architect is to analyze the performance of her architecture. This goal has been refined to three sub-goals: First, performance models are generated automatically from some UML diagrams. Then, these performance models are parametrized and solved. For space reasons, we did not further elaborate these two latter goals. We also considered UML diagrams as atoms, ignoring their internal elements such as actors, messages and nodes.

A computer is responsible for the generation of performance models. This goal is operationalized with four automated operations: generate the user profile from the use case diagram, generate the meta-EG from sequence diagrams, instantiate the meta-EG into an EG-instance with the help of the deployment diagram and generate an EQNM from the deployment diagram. These operations correspond to the first three steps described in [CM00]. The last two steps are captured in the two remaining goals, parametrize and solve the performance models.

## 5 Discussion

This paper is an initial contribution towards the modeling of models' purposes. For this, we have adapted two existing goal modeling approaches and applied them to two modeling tasks, demonstrating the feasibility of such metamodeling.

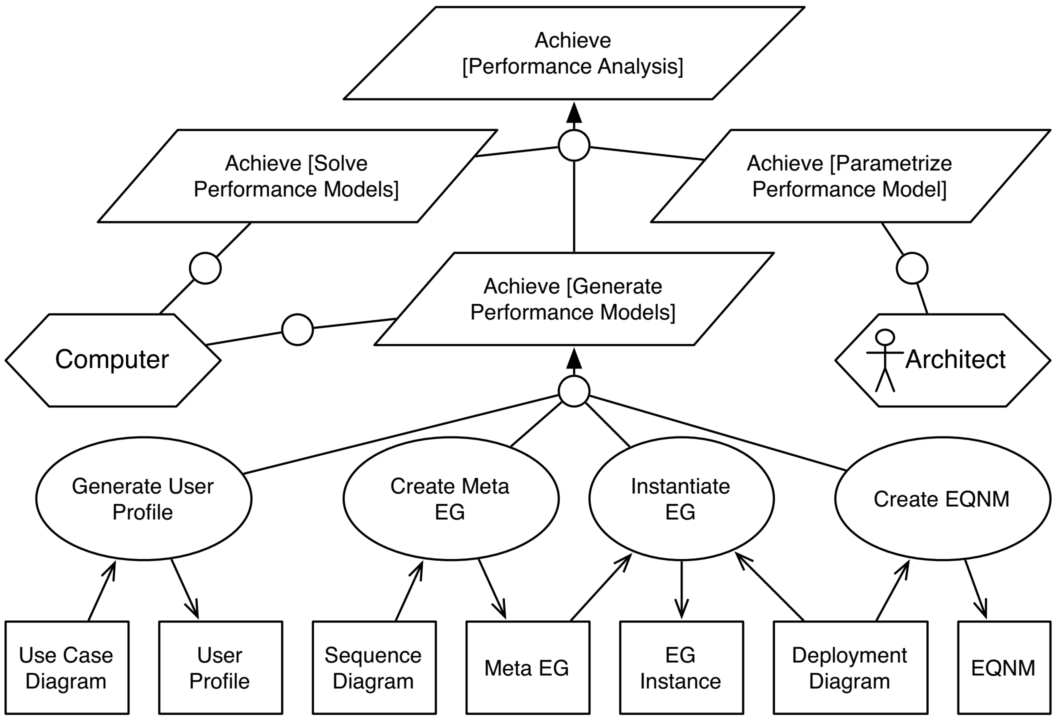


Figure 5: A KAOS metamodel for a performance analysis.

In the remainder of this section, we compare the two approaches presented in this paper, GOM and intentional metamodeling. We also discuss the benefits and difficulties related to these approaches.

### 5.1 Comparison GOM and Intentional Metamodeling

Contrary to GOM, intentional metamodeling with KAOS can capture the complete rationale behind the creation and the use of a model. As explained in Section 4, goal models are organized in AND/OR trees. By navigating the tree from an element upwards, a modeler can find the rationale explaining a given operation, meta-class or modeling purpose. Likewise, but using downward navigation, the modeler can figure out how a model purpose is realized by looking at its sub-goals or its alternatives.

In this paper, we only presented semi-formal KAOS models. However, these models can be completely formalized and thus are amenable to automated analysis, including the verification of goal refinements or the derivations of goal operationalizations [vL09]. The weaknesses of KAOS lie in the cost and difficulties of formalizing goals and operations. In comparison, GOM is a semi-formal approach. It only provides templates for stating modeling purposes and guidelines for deriving questions from this purpose. Future research should explore under which conditions a low or a high level of formality is preferred or required.

We have presented GOM and intentional metamodeling as two different approaches, because they come from different field: software measurement and early requirements engineering, respectively. Future work may integrate these two approaches, combining the ease of use of the templates and guidelines of GOM and the formality of KAOS.

### 5.2 Benefits and Limitations

Stating a model purpose and making it operational allows for measuring the fitness of a model for this purpose. A model is complete if it contains all the elements necessary to fulfill its goals. Conversely, a model element is pertinent if it contributes to the satisfaction of at least one goal. Confined models only contain pertinent elements. With a formal KAOS model, it is possible to measure these qualities objectively by establishing satisfaction arguments.

However, eliciting a model's purpose and elaborating it has a cost. The benefits must be higher than the costs if the practice is to be adopted by practitioners. Models are like systems. Making explicit requirements about models (such as stating their purpose) aims at reducing the risk of creating the wrong models. Models at the wrong level of abstraction have consequences ranging from small annoyances for their interpreters to the impossibility of fulfilling the purposes they were made for.

Furthermore, goal modeling is difficult. First, many modelers are not experienced in intentional modeling. Courses on Software Engineering or Modeling typically cover data, behavior and process modeling languages but leaves out goal modeling. Thus, (intentional) metamodelers will be rare in the near future. Second, goal models grow rapidly as goals are refined and alternatives are identified.

## 6 Related Work

In this section, we present the state of the art in metamodeling and model quality and we discuss its limitations. For van Gigch [vG91], a metamodel should cover many aspects of modeling, not only "data" metamodeling (the syntax of the language). In this vein, Kermeta proposes to metamodel the behavior of models, so that the operational semantics of models can be specified [MFJ05]. In this paper, we go one step further by metamodeling modeling agents and their goals.

In their model of modeling [MFBC10], Muller et al. place the intention of a model at the heart of their notation. They define intention as follows:

> The intention of a thing thus represents the reason why someone would be using that thing, in which context, and what are the expectations vs. that thing. It should be seen as a mixture of requirements, behavior, properties, and constraints, either satisfied or maintained by the thing.

In their notation, intentions are considered as sets and thus represented as Venn diagrams. While this notation allows to represent the intersection and the overlap among intentions, it does not allow to represent the internal content of the intention behind a model. The focus of our paper is to represent this intention, so that its modelers and its interpreters can agree and reason on it.

For Nuseibeh et al. [NKF93], a *viewpoint* consists of (1) a style (the modeling language and its notation), (2) a work plan describing the development process of the viewpoint including possible consistency check or construction operations, (3) a domain defining the area of concern with respect to the overall system, (4) a specification describing the viewpoint's domain using the viewpoint's style (in other words, the view of the system from the viewpoint) and (5) a work record keeping track of development history within the viewpoint. According to the IEEE 1471 standard, a viewpoint captures the conventions for constructing, interpreting and analyzing a particular kind of view. Thus, a viewpoint defines — among others — modeling languages, model operations that can be applied to views and stakeholders whose concerns are addressed in the views. Viewpoints define the various views (and their relationships) in a software specification or in an architecture description, thus, they provide the modeler with guidelines on what they are expected to model. However, we are not aware of guidelines to define these viewpoints, nor techniques to validate that a view actually satisfies the needs of the stakeholders using it.

In [MDN09], Mohaghegi surveyed frameworks, techniques and studies of model quality in model based software development. They identified 6 quality goals: correctness, completeness, consistency, comprehensibility, confinement and changeability. Manual reviews [LSS94] and metrics [BB06] can be used to assess and improve the confinement and completeness of models. However, these techniques are either bound to a given modeling language and a given process [BB06] or must be tailored for the modeling task at hand [LSS94]. In comparison, intentional metamodeling and GOM are not bound to any specific language or process. Because they document the purpose of models, goals expressed and operationalized in GOM or KAOS may serve as basis to derive checklists, guidelines and metrics for validating models.

In previous work [JGB11], we propose and compare two methods to compute the *footprint* of an operation – the set of all information used by the operation during its execution. *Dynamic footprinting* reveals the actual footprint of an operation by tracing its execution on the model. In contrast, *static footprinting* estimates footprints by first analyzing, statically, the definition of the operation to obtain its *static metamodel footprint*, the set of all modeling constucts (i.e., types, attributes and references) involved in this definition. The model footprint can then be estimated by selecting only those model elements that are instances of elements in the metamodel footprint. In this previous work, we assumed that the purpose of a model can be characterized by the set of operations being carried on it and that these operations were formally defined. These assumptions are reasonable in a MDE setting. Still, in this paper, we are interested in methods to specify an arbitrary model purpose and, if possible, to refine this purpose into a set of operations whose footprints can be looked at. In other words, the focus of this paper is the elicitation, documentation and operationalization of modeling purposes. The operationalization produces metamodels and operations that can be used, accessorily, as input for model footprinting.

In addition to GQM and KAOS, there are other goal oriented techniques and methods for requirements engineering, such as i* [Yu97] and Tropos [CKM02]. While we could have selected these approaches to capture and analyze the purposes of models, we chose KAOS and GQM instead for their strong focus on the operationalization of goals.

## 7 Conclusion

One cannot build a model without knowing its purpose, and one must not use a model for purposes it is not fit for. Despite its importance, the purpose of a model is often kept implicit.

In this paper, we adapted two existing goal modeling approaches — GQM [Bas92] and KAOS [vL09] — to capture the purpose of a model and operationalize it into a set of operations and a metamodel. With these elements in hands, it is possible to measure how fit a model is for the purpose. We demonstrated the feasibility of the approaches by applying them to two examples.

These early results are promising, but the benefits of such intentional metamodels remain to be established empirically (e.g., with industrial case studies). With the experience gained in modeling the purpose of models, we can elaborate the templates and adapt the guidelines offered by KAOS and GQM in more detail. In this vein, further research could define a profile for KAOS and develop specific analysis for intentional metamodels.

For the first time, goal modeling techniques were applied to modeling itself, raising many open issues: What is the source of modeling goals, that is, who are the metaexperts? Do intentional metamodels help in model management and model reuse?

## Acknowledgement

Our work is partially funded by the Swiss National Science Foundation under the project 200021_134543 / 1.

## References

[Bas92] Victor R. Basili. Software modeling and measurement: the Goal/Question/Metric paradigm. Technical Report UMIACS TR-92-96, 1992.

[BB06] Brian Berenbach and Gail Borotto. Metrics for model driven requirements development. In *28th International Conference on Software Engineering (ICSE '06)*, pages 445–451, Shanghai, China, 2006. ACM.

[CKM02] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems*, 27(6):365–389, 2002.

[CM00] Vittorio Cortellessa and Raffaela Mirandola. Deriving a queueing network based performance model from UML diagrams. In *International Workshop on Software and Performance (WOSP 00)*, pages 58–70, 2000.

[HPdW05] Stijn Hoppenbrouwers, H. A. Proper, and Th P. der Weide. A Fundamental View on the Process of Conceptual Modeling. In *Conceptual Modeling (ER 2005)*, volume 3716 of *LNCS*, pages 128–143. Springer, 2005.

[JGB11] Cédric Jeanneret, Martin Glinz, and Benoit Baudry. Estimating Footprints of Model Operations. In *33rd International Conference on Software Engineering (ICSE 2011)*, pages 601–610, Waikiki, Honolulu, HI, USA, 2011. ACM.

[Kra07] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[Kru95] Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.

[LSS94] Odd Ivar Lindland, Guttorm Sindre, and Arne Sølvberg. Understanding Quality in Conceptual Modeling. *IEEE Software*, 11(2):42–49, 1994.

[Lud03] Jochen Ludewig. Models in software engineering - an introduction. *Software and Systems Modeling*, 2(1):5–14, March 2003.

[MDN09] Parastoo Mohagheghi, Vegard Dehlen, and Tor Neple. Definitions and approaches to model quality in model-based software development: A review of literature. *Information and Software Technology*, 51(12):1646–1669, 2009.

[MFBC10] Pierre-Alain Muller, Frédéric Fondement, Benoit Baudry, and Benoît Combemale. Modeling modeling modeling. *Software and Systems Modeling*, 2010.

[MFJ05] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving Executability into Object-Oriented Meta-Languages. In *8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005)*, volume 3713 of *LNCS*, pages 264–278, 2005.

[NKF93] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. Expressing the relationships between multiple views in requirements specification. In *15th international conference on Software Engineering (ICSE '93)*, pages 187–196, Baltimore, MD, USA, 1993.

[Rot89] Jeff Rothenberg. The nature of modeling. In *Artificial intelligence, simulation & modeling*, pages 75–92. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[vG91] John P. van Gigch. *System Design Modeling and Metamodeling*. Plenum Press, New York, NY, USA, 1991.

[vL09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[Yu97] Eric Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *3rd International Symposim on Requirements Engineering (RE '97)*, pages 226–235, 1997.