

Modeling of information systems advanced course

summary

NTNU TDT4252 Spring 2008

Alexander Nossum
alexanno@stud.ntnu.no

May 20, 2008

Abstract

This paper is an attempt to summarize the topics covered in a course given at NTNU Trondheim. The summary is intended to be as brief as possible, but I recognize that topics I have a personal interest in have been promoted. In general the course is an advancement of the course; “TDT4250 Model-driven development of information systems”. As such it is more focused on bleeding edge technologies and research topics and has, in general, a more abstract view on conceptual modeling and modeling languages.

The paper is written by a student during the course and comes with *absolutely no warranties!* I am glad if it can be of any help, but do not guarantee for the correctness nor the scope.

Contents

1	Everything is hierarchical	3
1.1	Classification	3
1.2	Aggregation	4
1.3	Generalization	4
1.4	Association	4
2	Modeling perspectives	5
2.1	Structural	5
2.2	Functional	5
2.3	Behavioral	5
2.4	Rule-oriented	6
2.5	Object-oriented	6

2.6	Social-communication	6
2.7	Actor/Role perspective	6
3	Goal oriented modeling	7
3.1	Rule vs. goals - historical overview	7
3.2	The notion of rules	8
3.3	Introducing actors	8
3.4	Why do the why	10
3.5	Involving other techniques	10
4	Modeling security requirements	11
4.1	Dividing security	12
4.2	Expressing security	12
5	Value modeling	14
6	Process modeling	14
7	Ontologies	14
7.1	Bunge-Wand-Weber	15
7.2	Evaluating language quality	15
7.3	Destroying the silos	17
8	SEQUAL Quality framework	18
8.1	Quality of models	19
8.1.1	Physical quality	19
8.1.2	Empirical quality	20
8.1.3	Syntactic quality	20
8.1.4	Semantic and perceived semantic quality	20
8.1.5	Pragmatic quality	21
8.1.6	Social quality	21
8.1.7	Organizational quality	22
8.2	Quality of languages	22
9	Participatory modeling	23
10	The future	24
10.1	Enabling the user	24
10.2	Richer than reality	25
10.3	The downside	25

1 Everything is hierarchical

Humans tend to be geared towards hierarchical structuring, especially when it comes to large and complex structures. Thus it is natural to use hierarchical-like structures in conceptual modeling. The definition of an hierarchy is to strict (Krogstie and Sølvsberg 2000) and is loosend up for the purpose of conceptual modeling. However, the main concept in an hierarchy is the relationships between objects. In this section we will discuss the relationships to describe a hierarchy and their application in conceptual modeling.

1.1 Classification

Classification is a relation describing the relationship *across meta-levels*. Typically instances are looked upon as higher order object-type by using the *is_instance_of* relation. For example:

Alexander Nossun *is_instance_of* Author

Classification is different from the other relationships as it relates *across meta-levels*, and can thus be considered to be orthogonal to the other types of relationships.

In cognitive psychology it is found that humans classifies as a way of remembering. The dominant ways humans classifies are by: attribute-theory, prototype-theory and exemplar-theory. Classification can be seen as an application of this theories.

In conceptual modeling there are a set of dominant classification levels that are used, they are:

- Instance

Alexander Nossun as Author of this paper

- Model (Type-level)

Author, Paper (Paper has one or more authors)

- Meta model

Entity class, relationship class

- Meta-meta model

Node, edge

1.2 Aggregation

Aggregation is a way of composing objects. It uses the *is-part-of* relation. It is a fairly intuitive relationship, but nevertheless an important one. Example can be found in figure 1.

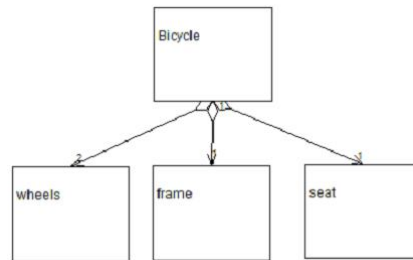


Figure 1: Aggregation relationship

1.3 Generalization

Generalization uses the *is-a* relation. It is motivated by the needs of generalizing concepts that in turns can be specialized by other. It is one of the main concepts of object-oriented programming implemented as inheritance. An example of generalization is depicted in figure 2.

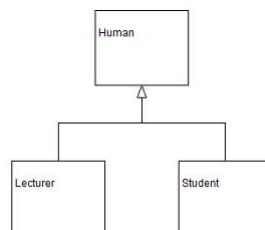


Figure 2: Generalization relationship

1.4 Association

Association uses the *is-a_member* relation and thus relates objects into a membership. For example *men* and *woman* are *member* of the *sex-group*, but not a specialization of it, as for the generalization relationship. The main difference between association and generalization is that an associated object does not (necessarily) have similar attributes, but is just associated with the membership.

2 Modeling perspectives

In modeling the central task is to conceptualize ones perception of a reality. This can be done in many perspectives, each having their own area of focus.

However, there are no perspective that is “the best” in all cases. A need to combine several perspectives are often rising. In this section we will cover the seven main perspectives in conceptual modeling, their area of focus and examplar languages that have this perspective.

2.1 Structural

Structural perspective is a *static description* of the structure in a domain. The main concept is *entity* (object, phenomena, concept, thing, referent...). Example of structural languages are; *Data-modeling* (ER), Semantic nets, conceptual graphs and Ontology languages (OWL). Data-modeling probably being the most common structural language used. In meta-modeling of languages, a structural language is (often) used.

2.2 Functional

Often called *process-oriented*, and has thus nothing to do with functional programming. Functional perspective describes a *dynamic process* and its behavior. The main concept is *process* (function, transformation, activity, action, task...). A well known and heavily used functional language is *DFD*. In addition, there are a lot of extensions and variants such as *EEML*, *BPMN*, *UML Activity Diagram* etc. The perspective is heavily used to capture business processes and system processes and in general workflow modeling.

2.3 Behavioral

Behavioral perspective focus on capturing the *dynamics of a system*. Main concepts are *states* and *transitions* between states. Often behavioral perspective is called state-diagrams, state-modeling...

Examples of languages are *State Transition Diagrams* (STD/STM), State-charts and Petri-nets. The perspective is frequently used in real-time systems and telecommunication systems.

Although it is a clear distinction among behavioral and functional perspectives, they are *often combined* in modeling languages. Languages such as UML Activity Diagram, EEML and BPMN have mechanisms for combining both these perspectives and can thus model both the functional perspective and the control flows.

2.4 Rule-oriented

Rule-oriented perspective focus on describing the connection of *goals/means*. The main concept is *Rule* (goal, constraint, logical rule...). With the definition of a rule being; “something that influences the actions of a set of actors”.

The standard form of a rule is on the form; **IF condition THEN action**.

Examples of rule-oriented systems and languages are; *Rule hierarchies* (I*, EEML - see Goal modeling section 3), *Tempora* and traditional *Expert systems*.

2.5 Object-oriented

The manifesto for object-oriented perspective is to describe the world as *autonomous communicating objects*. Thus the main concept is *object* which is seen as:

An entity with a unique identity with a local state only accessible by sending messages to the interface of the object

Examples of object-oriented modeling languages are *UML (all)*, *OOA etc...*. The perspective has mainly come from the *design and programming of object-oriented systems and programming languages*. The acceptance and use of the object-oriented perspective is heavily growing and can (almost?) be considered the de-facto standard of modern software engineering. In particular UML and extensions of UML is the most common languages to use in modeling.

2.6 Social-communication

Social-communication perspective describes the (social) *communication structure*, and are often called *language action perspective*. It is based on philosophy of (natural?) language. The perspective captures different types of speech acts and put them together into conversations. Typically this perspective is found in the field of CSCW¹ and Groupware.

The main concepts in the perspective are: *Speech acts* and *Conversations*.

In addition to this there are several different speech acts in the perspective, however we will not go into detail on this here.

2.7 Actor/Role perspective

The actor and role perspective focus on the description of *organizational or system structure*. It is influenced by work in AI² with their intelligent agents, object-oriented programming, modeling of organizational structures and group dynamics.

¹Computer Supported Cooperative Work

²Artificial Intelligence

Main concepts are *actors* and *roles*. The definition of these concepts are slightly different. Where actor is:

A phenomena that influence the history of another actor

and role is:

the behavior that is expected by an actor (by other actors) when filling the role

Languages such as I* and EEML focus on this perspective and is discussed further under goal modeling in section 3.

3 Goal oriented modeling

When performing an analysis of a software system, either to-be or as-is, it is very useful to know the motivation behind the system - in a sense, *why*. Traditionally developers tend to be more geared towards an exploration of *how*, and thus a more technical, and directly, solution oriented, approach. However, when the system is deployed or used, the system often fails at doing what it actually should do, thus not-fulfilling the *why*. We will discuss some techniques aimed at capturing the *why* and the *who*, in the context of goal/rule-modeling in concert with actor/role-modeling.

- Related Languages:

I* (SD/SR), EEML, UML Use Case diagram

3.1 Rule vs. goals - historical overview

Rules in software are nothing new, in AI most of the software relies on rule-machines. The problem with such an approach is that all rules are considered to be in a rule-space and can “fire-at-will”. In combination to this structure there is often a large number of rules. Combined this makes it almost impossible to understand or get any real semantics out of the structure for humans. Keeping in mind that modeling is largely driven by its ability for communication and human understanding, the traditional approach is insufficient and there is a need for introducing *hierarchies* with *dependencies* as a structure.

Goal modeling is all about this. Interpreting rules as goals and incorporating goals with interdependencies - thus creating a hierarchy with dependencies.

As mentioned above, there is also a need to capture the *who*. This is solved by introducing *actors*, or roles, as a part of the goal-model, thus enabling a relationship between a goal and an actor (person, system, etc.).

3.2 The notion of rules

In informatics and mathematics rules are considered to be very strict and concerning only *true* or *false* as a result - referred to as being of a *crisp* nature. This has influenced the area of requirement engineering thus often focusing only on functional, easy to test (true/false), requirements. However, quality requirements often termed non-functional requirements are also, if not more, important to the requirement analysis. Quality requirements are often of a more *fuzzy* kind, enabling the result to be either true, false or *somewhere in between*.

In a goal-modeling task all relevant goals should be taken account for, including fuzzy rules. Fuzzy rules exists, not only in pure requirements but also as *tacit norms, obligations etc.* in an organization. These rules are especially interesting to capture since they often are not externalized and thus play an important role for the goal model as a whole. Such rules are often termed *deontic rules*, from ethical philosophy. Deontic rules are not strict requirements in the sense that their absence will produce a direct failure, however it will probably *reduce the obtained quality*.

In the context of goal modeling deontic rules are mapped to a *softgoal* concept. An illustrative example of a goal model can be found in figure 3. As we can see in the model, the *recommend* relationship and the “Give interest to ...” are in combination a softgoal, and thus models a deontic rule.

3.3 Introducing actors

Goals in every form, either soft or crisp rules, are always influenced or dependant in some way to an actor. It is important to capture these dependencies in combination with the goals.

Strategic Dependency Models (SD) are a branch of the I^* modeling language and are geared towards connecting goals to actors. The idea of SD is to give a *high-level overview* of the *external* dependencies among actors. In figure 4 we see that the actors have dependencies among each other, but there are no notion of dependencies or goals that affect the actor internally, such as “private” goals.

Strategic Rational Models (SR) is an extension of the SD approach and is primarily focused towards the *internal* dependencies for the actors. In addition the internal dependencies are given labels to provide a *hierarchy* for the *internal goals*. Figure 5 depicts an SR model. As we can see the actors are *blown up* and the internal goals are represented.

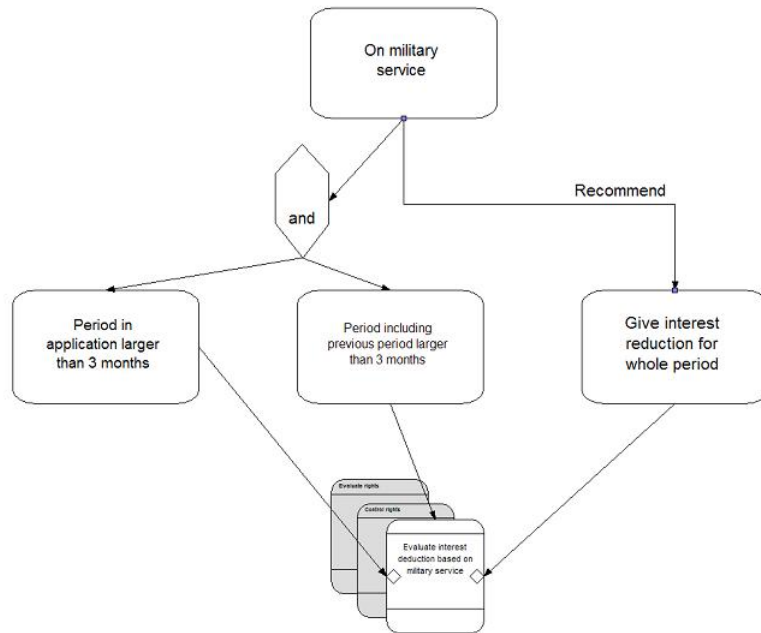


Figure 3: Example of a goal model including deontic rules and mapping to a process.

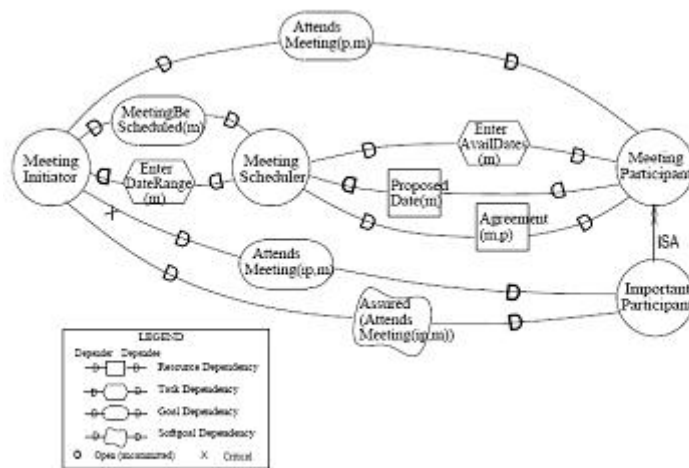


Figure 4: Strategic Dependency Model of a meeting scheduler system

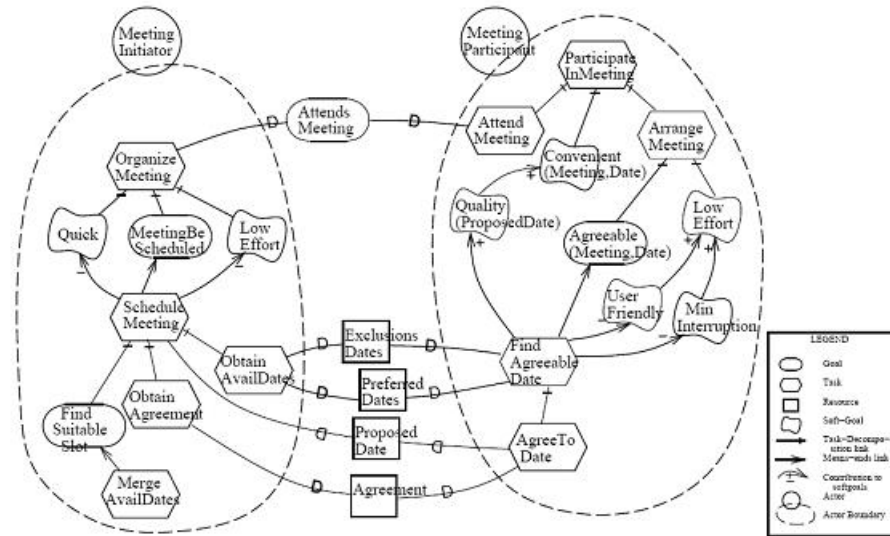


Figure 5: Strategic Rational Model of a meeting scheduler system

3.4 Why do the why

As mentioned in beginning of this section there are several aspects that often are disregarded when modeling a situation (as-is or to-be), especially when it comes to requirement analysis. One of the most common disregards is towards the goals, including softgoals and thus the quality aspects.

These aspects are very important, especially in an organizational setting. Often an organization has a set of business goals and rules, externalized and tacit, these in turn can come from other authoring organizations, such as a state providing laws and regulations. In addition there are always ethical issues to consider.

Goal modeling is a technique to properly capture these goals, combining both externalized and tacit rules and norms provided either internally or by external actors. The motivation for this is to capture and externalize these aspects in an attempt to communicate the overall context, or “the whole picture”, in a good way. Ideally the appropriateness should be good both for human- as well as for technical actors.

3.5 Involving other techniques

Goal modeling have its limitation, one especial limitation is the capturing of scenario modeling, as found in UML use case diagrams. Maiden et al. (2004) presents a case from the requirement phase of a large air traffic control system. The sys-

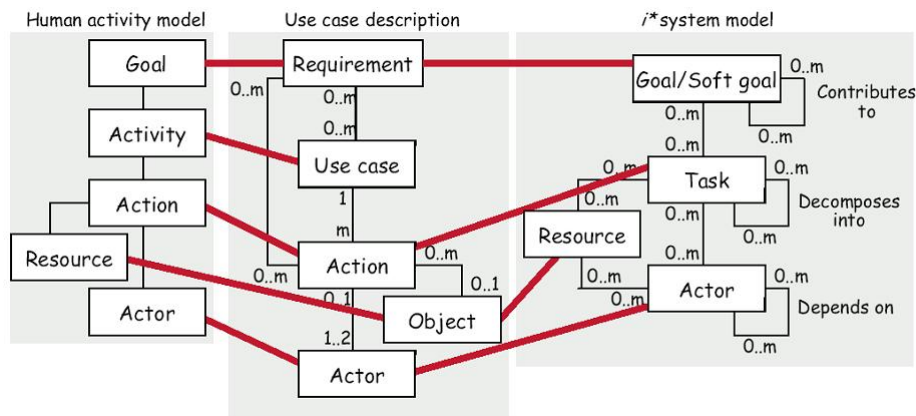


Figure 6: Mapping between different languages on meta-level used in Maiden et al. (2004)

tem involves a large number of people and systems, thus being a *socio-technical system*. Modeling the requirements for a new system would benefit from modeling in different perspectives to fully capture all relevant aspects, especially the social perspective such as user interaction.

In the case they combine human activity modeling, I* goal modeling and UML use case diagrams. One important issue to address is the need to *coordinate* and *synchronize* the different models, this was done by *mapping concepts* in the languages on a *meta-level*. The mapping is shown in fig 6. In addition to this the modeling process included regular steps for synchronizing the models.

Experiences from this show that it is possible to use different modeling techniques and it is possible to gain a lot of benefits from this. However, the process is not light-weight. It is fairly complex in terms of coordination and requires a lot of time (the case in Maiden et al. (2004) used 9 months).

4 Modeling security requirements

Security requirements have come increasingly important as software has become the main drivers for many critical tasks in the society as well as for the way of living. However, security in IT is difficult. It is considered to be impossible to get 100% security, this does not mean we should not try. Traditionally security requirements are handled poorly in the requirement phase. They are often handled late in the process, thus being constrained by other non-security requirements, or, in worst case, after the system is finished. This approach is very costly and potentially very risky.

4.1 Dividing security

There are many different types of security requirements. In the list we present some main types that are common for security. It is useful to know the different types as it will provide a mechanism for preventing overlooking any possible threats, improve understanding of prioritization and in general get an idea of what lies in the phrase security. We will not go into the details on the different types in this paper as this can be found in other sources (Firesmith 2003, Sindre and Opdahl 2005, 2007).

- Identification, Authentication, Authorization, Non-repudiation
- Immunity, Integrity, Intrusion detection, Privacy
- Physical protection, survivability
- Security auditing, System maintenance security

4.2 Expressing security

In an attempt to capture security requirements Sindre and Opdahl (2005) have proposed a modeling language called *misuse cases*. The language is based on UML Use Case diagrams, but with an *inverted view*. The user is replaced with a *misuser* and actions are replaced with *threats*. An example of a misuse case is depicted in figure 7. This technique is aimed at getting an overview of the possible threats, connect the threats to the functionality, brainstorming and identification of threats and to prioritize. In addition to the graphical model one can also supply with a textual description, as with regular UML Use Cases. Textual descriptions may vary in complexity and be lightweight or more detailed.

The technique is fairly new and has some weaknesses. An important issue to consider in misuse cases, and for security elicitation in general is the *analysis-paralysis* phenomena. There are (almost) an infinite number of possible threats that is more or less relevant for the case, capturing all is neither possible nor feasible, one therefore needs to have mechanisms to know/decide *when to stop*.

In addition to the misuse case approach, there have been performed analysis on using inverted notations on a more general level, capturing dependability threats (Sindre and Opdahl 2007). We will not go into further details on the research on this but show an example of inverted notation for Sølvsbergs Referent Model (Sindre and Opdahl 2007) in figure 8.

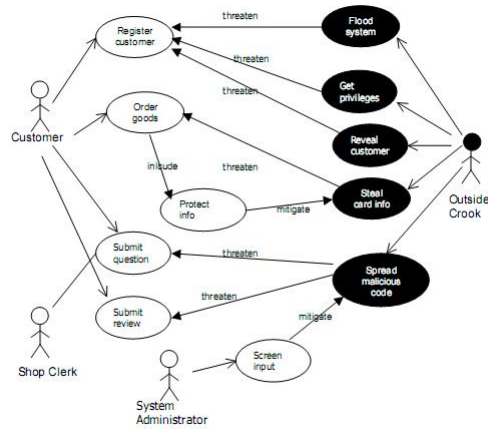


Figure 7: Security requirements using misuse cases in combination with UML Use Cases

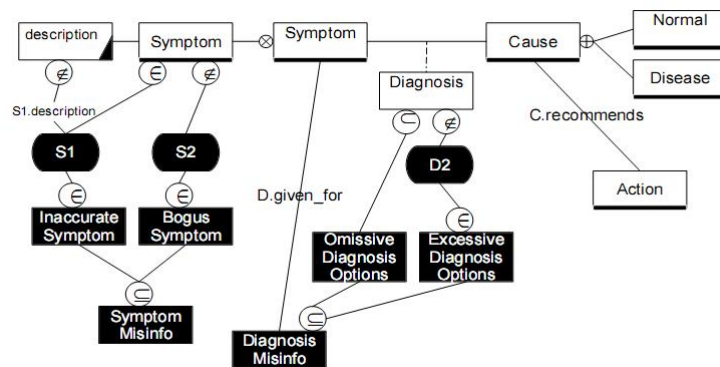


Figure 8: Dependability threats for a medical advice using inverted notation

5 Value modeling

When developing an information system the motivation is often (always) motivated by some sort of added value to the stakeholders. This is in particular true when it comes to enterprise settings. In almost every process several actors are involved in producing value among each other. This collaboration of value production is called *value constellations*.

In an attempt to properly understand and analyze the value constellation it is proposed a language called *e³-value*. The motivation behind the language is to get a precise and shared understanding of the value constellation, to check against business rules, be able to perform feasibility analysis and as a starting point for information systems development.

There's a wide range of possibilities that could be included in the model. However to keep pragmatic and empirical qualities the scope is limited to a very high level of abstraction, only focusing on; *economic value*, and the *flow* of these. Thus *abstracting away* many details, specifically business processes, coordination issues and information systems. However gaining a focus on what the enterprise can *offer* and what it expect in *return*. The resulting perspective is of a *transaction economics perspective*.

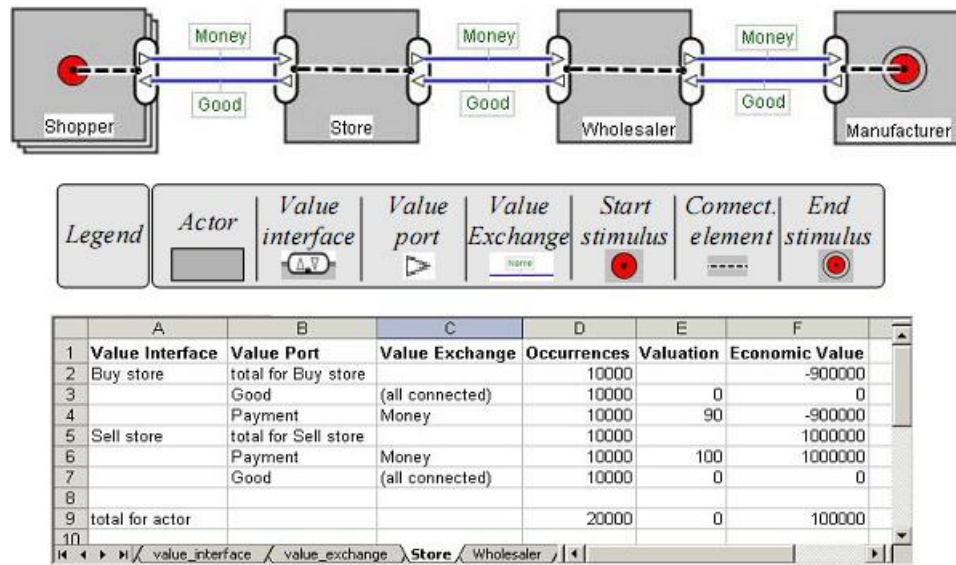
In figure 9 a “hello world” in e³-value is presented. The figure provides an overview of the essentials of the language. However there are many concepts that are not included in this model. The language supports some abstraction mechanisms, especially is the mechanism allowing *composition*, thus enabling for example partnership as an actor composition. Additionally the language support *dependency paths*, allowing a more strict flow of the value, this makes the model more “computable”.

6 Process modeling

Process modeling is one of the most accepted and used forms for modeling, probably next to modeling as a design mechanism in software development. We will not cover any specifics on process modeling in this paper as it is covered by Nossum (2007), which is a brief summary of a course focusing more on process modeling and business processes in general.

7 Ontologies

Ontologies are a way of representing concepts used to describe something. In conceptual modeling, *meta-models* are a looked upon as an ontology for a language. Meta-models, are, as mentioned before, *models of modeling languages*. We will in

Figure 9: “Hello world” in e³-value

this section focus on ontologies in a more philosophical sense, briefly introducing Bunge-Wand-Webers ontology, how this can be used in an analytical evaluation and lastly, UEMML, an approach for enabling *model interoperability*.

7.1 Bunge-Wand-Weber

Bunge-Wand-Weber (BWW) have described an ontology to capture the overall concepts of the world. The ontology is therefore very general and on a very high level of abstraction. Figure 10 shows a fragment of the ontology which is the most relevant to conceptual modeling and thus modeling languages.

7.2 Evaluating language quality

One approach that can be used to evaluate the quality of a language is an *analytical* evaluation. The idea is to perform a systematic comparison of the meta-model of the language to an accepted general ontology, such as the BWW-ontology. The general idea is to evaluate the *expressive power* of the language. It is important to know that this, analytical comparative approach relies fully on the assumption that the ontology is *sensible*.

Opdahl and Henderson-Sellers (2002) performs an analytical evaluation of UML1.3 vs. the BWW ontology. In their approach they investigate different mappings between the two meta-models. Below is the possible mapping-types

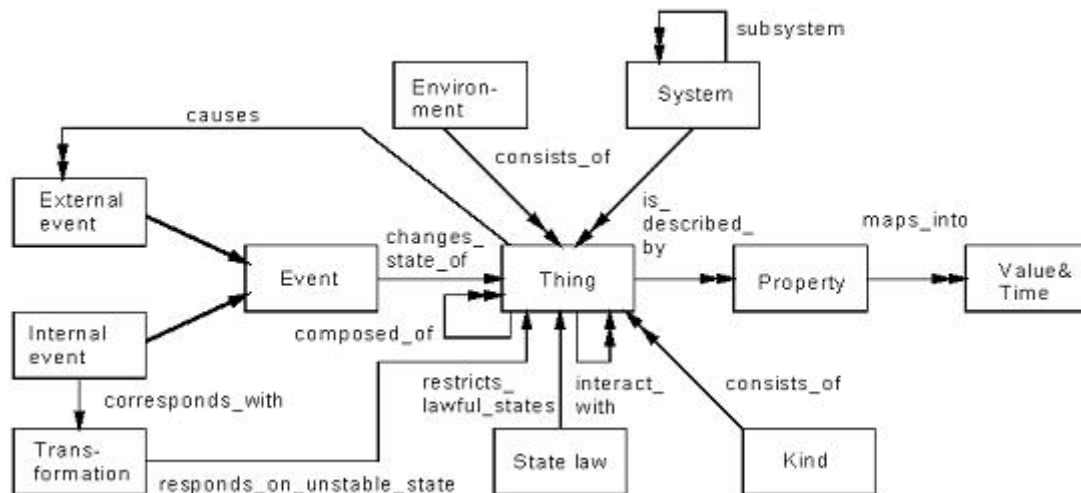


Figure 10: A fragment of Bunge-Wand-Webers world ontology

that are useful in such an evaluation and further how this was incorporated in the evaluation of BWW vs. UML.

Possible mapping types for the BWW

Construct overload

One UML-concept covers several BWW concepts

Problematic

Construct redundancy

Several UML concepts for the same BWW concept

Not necessarily problematic

Construct excess

One UML concept has no correspondance in BWW

Not necessarily problematic

Construct deficit

A BWW concept has no correspondance in UML

Problematic

Mapping types incorporated in BWW vs. UML evaluation

Representation mapping; BWW \rightarrow UML

Identify redundancy

Identify deficit

Interpretation mapping; UML \rightarrow BWW

Identify excess

Identify overload

Precisely define the meaning of each UML concept

The evaluation performed gave a more precise definition of UML concepts, discovered weaknesses in UML and a need to define concepts more precisely relative to the intended use. However the evaluation focused purely on the concepts and not on the diagram notation (eg. graphical representation).

There are several other methods that supports evaluation of a modeling language. Other ontologies can be used (philosophical or domain-centric) or a more empirical driven evaluation can be performed. An empirical approach can easily also include the diagram notation as well as the meta-model. We will discuss another approach for evaluation of both model and language quality in the section of SEQUAL (section 8).

7.3 Destroying the silos

A problem that often occur in both organization models as well as for information models is *interoperability*. Traditionally this phenomena of closing information is coined *information silos* or the *silo perspective*, where information is kept within the silo and not throughout the “silos” working together.

This is also a problem that occurs in conceptual modeling, where *modeling languages*, and their respective models, can be considered to be silos. However employing *meta-modeling* and the knowledge of ontologies there are possibilities to avoid information silos.

Unified Enterprise Modeling Language (UEML), proposed by Opdahl and Sindre (2007), is one such approach that aim at *leveraging the modeling interoperability* issues. Mainly there are two different “architectures” for interoperability:

Model level

Model-to-model

Repository based

Language level

Language-to-language

Repository based

In the UEML approach they try to *not* propose a new language, rather *integrate* existing ones. They also focus on the distinction between *syntax* and *semantics* by *seperating them* additionally they provide both mathematical and ontological semantics aiming at executional abilities. The semantic/referential integration is performed through a *common ontology*, which is essential for the approach. Figure 11 shows the overall idea for the approach, where diagrams, languages, presentation and modeling constructs are *interrelated* and *mapped* to the *common ontology*.

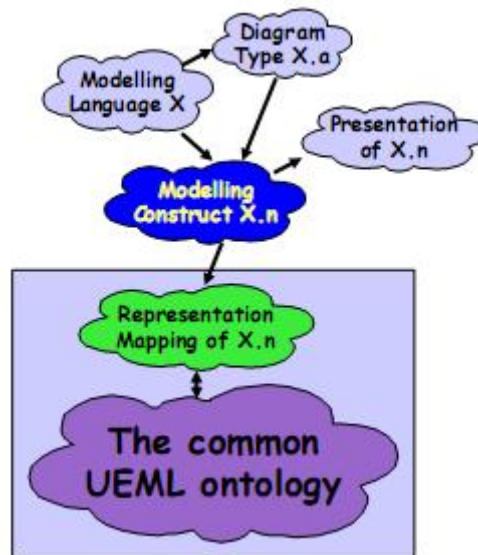


Figure 11: UEMLs central principle

UEML also defines a *meta-meta model* which is used for structuring information in a common way. The meta-meta model can be found in figure 12.

Comprised UEML is an attempt to define a set of principles to ease interoperability between models and modeling languages. To support this they define a common meta-meta-model, a common ontology to map concepts onto and templates for describing information. This makes UEML into a *dynamically growing* ontology for modeling languages, enabling to easy interoperate between them.

8 SEQUAL Quality framework

This section is copied entirely from Nossum (2007).

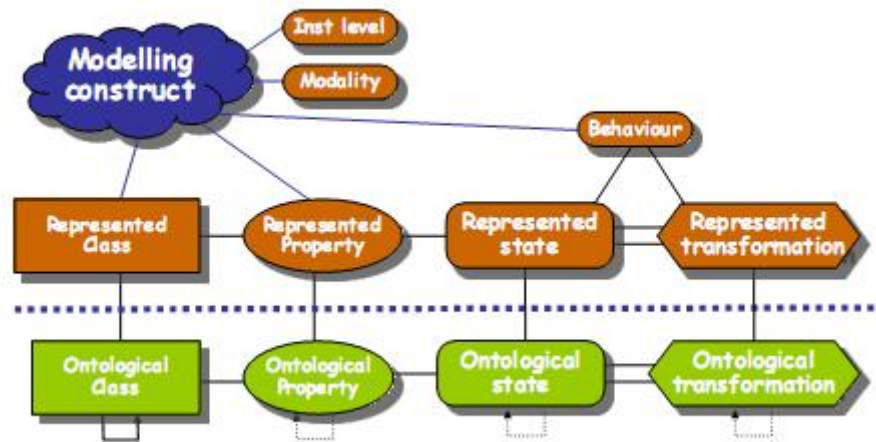


Figure 12: UEML Meta-Meta model for a common way of structuring information

SEQUAL is a framework for evaluating the quality of both models and languages in the context of software development. The framework base itself on a set of elements involved in a model and in a language. Key concept for the framework is to evaluate the relationship between these elements, which typically includes some form of perception of knowledge. The elements are:

- Social actor explicit knowledge
- Social actor interpretation
- Language extension
- Model externalization
- Technical actor (tools) interpretation
- Modeling domain
- Goals of modeling

Further on, we discuss the quality evaluations based on relationship between these elements. With the evaluation of models and evaluation of languages.

8.1 Quality of models

8.1.1 Physical quality

The physical quality of a model is in essence how the model is externalized or in a sense manifested. Whether it is on paper, electronic ... and how this affects

attributes such as; ability of distribution, possibility for interpretation, change management, backup etc. These attributes can be grouped into two key components: *Persistence* and *Availability*.

Persistence is in essence how well the physical model is saved. Including different versions, changes and so on.

Availability describes how easy it is for the audience to get hold of the model. An important factor is the ease of distribution - which is heavily dependant on the externalization of the model.

Physical quality of a model relates directly to the *Social actor, explicit knowledge* element of the framework.

8.1.2 Empirical quality

Empirical quality is the evaluation of how easy it is to gain knowledge (externalize) from the model. The externalization of knowledge is dependant on how the actual model is designed. Several factors are relevant for this, symbology, use of colors, layout of symbols (crossing lines, squared vs. wobbly boxes) and so on. The empirical quality is heavily dependant on subjective factors hence, there are no formal rules only guidelines for evaluating the quality.

As implied, the empirical quality relates to the *model externalization*. Other attributes relates the externalized model to the actual perception value for the audience. Social quality (8.1.6) and Pragmatic quality (8.1.5) handles the concrete perception and values that are obtained by the model audience.

8.1.3 Syntactic quality

Syntactical quality of a model relates to the relationship between; *Language extension* and *Model externalisation*, and is as such, the ability the model has for externalization (manifestation/capturing).

Since the syntactic quality derives from the language extension it can be formalized from the syntactic rules of the language in use. The overall, *and only*, goal is: **syntactic correctness**. If the model is in complete compliance with the syntactic rules of the language - the model obtains a high syntactic quality.

8.1.4 Semantic and perceived semantic quality

Semantic quality is the completeness between *the domain* and *the model* in a sense;

The completeness of the captured knowledge between domain and model.

There are mainly two goals for semantic quality; *Validity* and *completeness*. Since a model can capture knowledge that actually does not exist in the domain, we need *validity between the domain and model*. On the other hand the main interest for a model is to capture the whole, correct, knowledge area of the domain - ensuring that *the completeness between domain and model are fulfilled*.

Semantics are not easy to formalize since it base itself mostly on subjective factors. The introduction of perception and thus *perceived semantic validity and completeness* is necessary for the evaluation of the model from the audience. Perceived semantic quality is hence not related directly to the domain \Leftrightarrow model but *between the Social actor explicit knowledge and the Social actor interpretation*.

8.1.5 Pragmatic quality

In the evaluation of pragmatic quality - the practical considerations are taken into account. A main goal for achieving pragmatic quality is **comprehension** from the different actors. The actors are of different kind, both human and abstract. This relates the pragmatic quality from the **model externalization** to; **modeling domain, social actor explicit knowledge, social actor interpretation and technical actor interpretation**. Induced from this we see that *interpretation* is a key concept in pragmatic quality and hence the *concrete comprehension*. The learning aspect is therefore an important factor for evaluating and improving the pragmatic quality.

Focusing on the elements, mentioned earlier, that are related with the pragmatic quality, we can introduce some concepts that are necessary in obtaining pragmatic quality.

Social actor interpretation is essentially human understanding of the model.

How well is the human actor able to understand the model.

Technical actor interpretation is the ability for tools to interpret the model.

Typical the correctness between the tool interpretation and the actual meaning described in the model.

Modeling domain is the models ability for changing/affecting the domain it models. Both positive and negative directions are included in the term change.

8.1.6 Social quality

Social quality is related to the *social actor interpretation* by itself. The overall goal for obtaining social quality is **agreement**. Agreement in the sense that social actors agree on their interpretations on different levels of agreement.

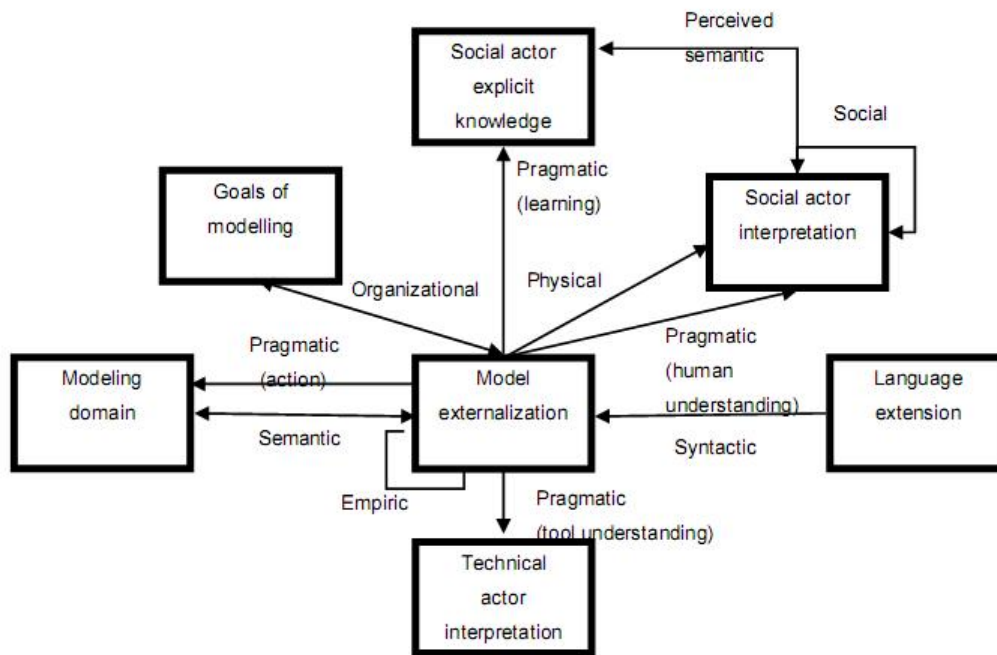


Figure 13: SEQUAL framework for evaluating model quality

8.1.7 Organizational quality

The organizational quality is related to the element; **goals of modeling**. Evaluating whether the overall goals for the modeling performed is achieved. Overall goals, including organizational goals and concrete modeling goals.

In real life, the ability in obtaining all these goals perfectly in performing the modeling task is *unrealistic*. Taken this into account, the term **feasibility** is introduced. Feasibility, used as a realistic level of “perfectness” in obtaining **enough perfection** of the goals.

8.2 Quality of languages

The language is the main basis for any model and it’s important that one can evaluate the language in terms of quality. SEQUAL proposes a framework for this. Using the same approach as for the evaluation of models, with qualities related to the relationship between elements. The relationships are of some greater intuitivity when it comes to languages, with less abstract evaluation terms. The proposed evaluating factors are:

Organizational appropriateness

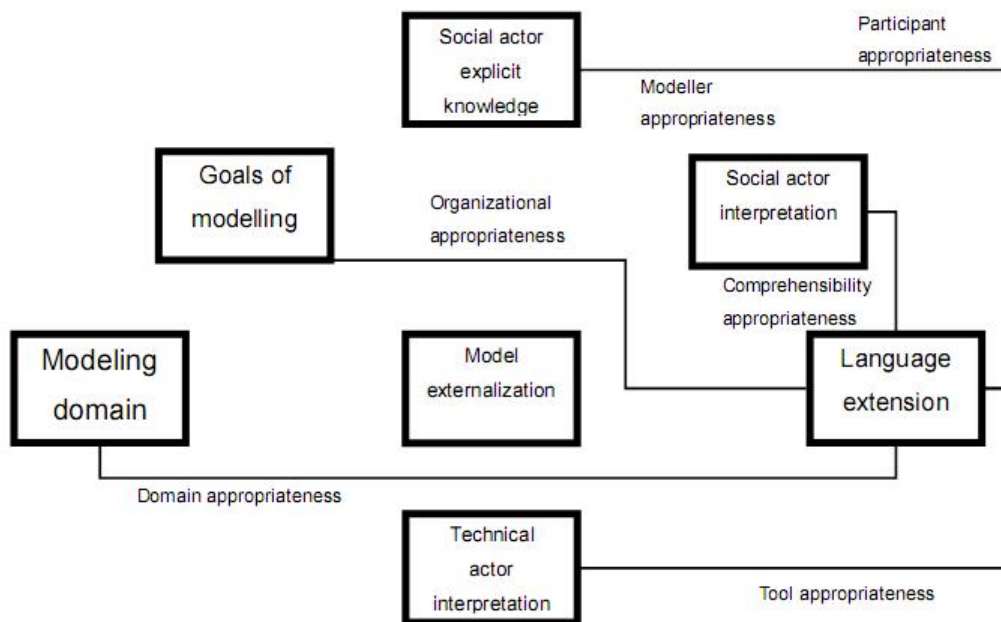


Figure 14: SEQUAL framework for evaluating language quality

Tool appropriateness

Participant and modeler appropriateness

Comprehensibility appropriateness

Domain appropriateness

All factors are spun from the *language extension* element in the framework. This is very natural since it is the language itself that is under evaluation. Further on we see that all factors are based on *appropriateness*. From this we can derive that the main driver for quality in a language, is how well it is suited for the task. The framework only elaborates more specifically this question.

9 Participatory modeling

We will in this paper not go into any discussion on participatory modeling since this is (thoroughly) discussed in Nossum (2008), among others. Although we believe this is probably one of *the* most interesting topics in the area of conceptual modeling.

10 The future

A vision for the future is to leverage the modeling task to be a natural part of everyday work. Thus enabling for a broader specter of people to perform modeling and benefit from it. In this section we will *briefly* discuss some of the current research topics in conceptual modeling and some thoughts on what the future holds.

10.1 Enabling the user

Currently, modeling has been accepted in the industry as a technique well suited for primarily documentation of knowledge, such as requirements, software analysis and design, and organisational constellations.

The trend however is to be able to have executable models. Either generate pure code, generating services or have the model *be executable itself* - thus reaching *interactive models*, where the division between code and design is leveraged or completely removed.

Executable models are not a new phenomenon and could be seen as a “solved” problem. However to get the full benefits of such models one must look at the actors involved in the modeling process. Usually the responsible for making the model and executing it are primarily professional software developers. With the idea of interactive models the modeling actor should be replaced with the *user actually using the final product*. This idea leverages the complexity of tailoring and adapting a system to the specific end-users needs and the resulting methodology have been termed; Model Generated WorkPlaces (MGWP) or Model Designed User Environments (MDUE). In figure 15 we try to give an overview of this vision and the roles of the actors involved.

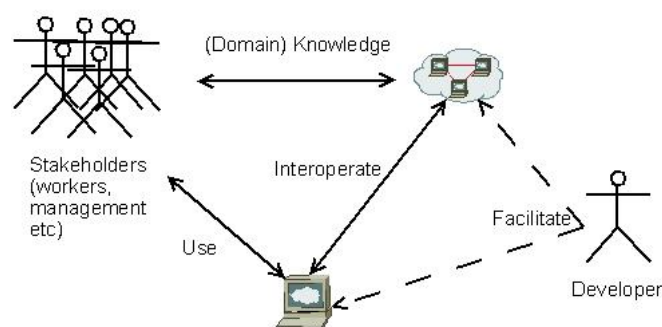


Figure 15: The role of actors involved in interactive modeling and MGWP

10.2 Richer than reality

Models are looked upon as an abstraction of reality with an isolated view, either being *to-be*, *as-is* or *wanted*. A new approach to what models actually are is proposed by Krogstie (2007). The new approach aims at models *being richer than reality* with integrating models with knowledge on *past*, *present* and *possible futures* and enabling *model-views* to be the equality to the now traditional model and thus an abstraction of reality. This introduces a orientation-shift from OO-/Process-orientation towards *view-orientation*.

Enabling view-oriented modeling introduces semantic issues. The traditional semantics are *atomic* while a paradigm shift should aim on a *holistic semantics*. Holistic meaning, in its essence, to include contextual information so *the concepts as a whole determines the semantics of the model* (eg. what it represents). The idea is fairly visionary and still on a very early stage, however inclusion of contextual information is a trend in computer science and will probably be more applied in the years to come.

We will in this paper not go into further details on these visionary topics but leave it up to the interested reader to explore it further.

10.3 The downside

Why isn't these visions already incorporated in the modeling industry? Well, modeling is in general looked upon "as difficult" and an experts task. We argue however that *type-level* modeling is difficult, but *instance-level* can be performed by any symbolic analyst (eg. humans). Techniques that supports instance-level modeling as a main way of modeling is however very limited.

Interoperability, as mentioned earlier in the paper, is an evergrowing problem that needs to be solved. There are research on making languages interoperable by for example a common meta-model/ontology (UEML) or "discrete" mapping between common languages to create a "new" language. However there is a huge challenge in making modeling infrastructures interoperable.

References

- Firesmith, D. G.: 2003, Engineering security requirements, *Journal of Object Technology* .
- Krogstie, J.: 2007, Modelling of the people, by the people, for the people, *Conceptual Modelling in Information Systems Engineering* .
- Krogstie, J. and Sølvsberg, A.: 2000, *Information Systems Engineering: Conceptual Modeling in a Quality Perspective*, Draft.
- Maiden, N. A., Jones, S. V., Manning, S., Greenwood, J. and Renou, L.: 2004, Model-driven requirements engineering: Synchronising models in an air traffic management case study, *CAiSE* .
- Nossum, A.: 2007, Model driven development - tdt4250 a summary, *Whitepaper* .
- Nossum, A.: 2008, Modeling 2.0? state-of-the-art in participatory modeling, *whitepaper* .
- Opdahl, A. L. and Henderson-Sellers, B.: 2002, Ontological evaluation of the uml using the bunge-wand-weber model, *Softw. Syst. Model* .
- Opdahl, A. L. and Sindre, G.: 2007, Interoperable management of conceptual models, *Conceptual Modelling in Information Systems Engineering* .
- Sindre, G. and Opdahl, A. L.: 2005, Eliciting security requirements with misuse cases, *Requirements Engineering* .
- Sindre, G. and Opdahl, A. L.: 2007, Capturing dependability threats in conceptual modelling, *Conceptual Modelling in Information Systems Engineering* .