SeeMe in a nutshell – the semi-structured, socio-technical Modeling Method

Thomas Herrmann

This introduction explains the basis concepts of the modeling notation SeeMe (semi-structured, socio-technical Modeling Method). We assume that most of the readers know other types of diagrammatic modeling languages, and are now interested in a method which is especially designed for representing socio-technical work processes and structures. The following description is mainly focused on the syntactical and semantical aspects of the modeling notation SeeMe. Only the last section provides guidelines about how diagrams can be developed.

SeeMe was developed in 1997 after it had revealed that the available diagrammatic modeling methods were not feasible for socio-technical systems and for the purpose of smooth communication processes about socio-technical solutions. An important difference of SeeMe in comparison with other methods was the handling of vagueness, which was emphasized in the first publications on SeeMe [Herrmann&Loser, 1999; Herrmann et al., 1999]. Further publications gave reports about the practical usage and usefulness of this method [Herrmann et al. 2000; Herrmann et al. 2004a]. It turns out that SeeMe should be used in the context of a deliberately organized series of workshops including facilitated communication processes which we call Socio-technical walkthrough – STWT [Herrmann et al., 2004b]. To find collections of SeeMe-Models the following literature can be exploited: Loser, 2005; Kunau, 2006; Stefanides 2006.

1 The Background of SeeMe: Modeling socio-technical systems

If individual software development is projected for a company or if existing software has to be configured and introduced, several aspects and perspectives have to be taken into account to document the concept of integrating the software into an organization. **Typical aspects of documentation** are: Features of the technical components and their interplay; conditions, events or exceptions; resources; roles, actors and their competencies and skills; work procedures; communication and cooperation, human-computer interaction, power relations and interests etc. **Typical stakeholders**, whose perspectives should be represented, are: management, software-engineer, workers, project-manager, user advocates, or citizens. A socio-technical modeling method must be able to handle these aspects and perspectives, although not every diagram needs to mirror all of them. The diagrams should be a collective resource which can be used by the stakeholders to express and to document their points of view.

The purpose of SeeMe is to support the early phases of developing concepts for **socio-technical solutions** and to document them. The socio-technical approach is especially appropriate if the interactions between people are supported and shaped by ICT in the case of cooperative work settings or processes such as workflow management, knowledge management, cooperative design etc. Nearly every kind of cooperative work which is coordinated with a shared database (e.g.

between truck drivers and dispatchers), can be considered as a socio-technical system.

We suggest an underlying scenario where the relevant stakeholders "come together" (e.g. in a series of workshops) to contribute their requirements for a socio-technical solution. These contributions may include a variety of the aspects listed above which have to become visible in the documentation. Our approach is based on communication theory which suggests that communicators do only make explicit what is not already obvious by their context [Kienle & Herrmann, 2003] or common ground. Therefore a modeling notation which represents a rich variety of aspects must allow the modeler to focus on the essential aspects. An "early-phases" notation must not enforce the depicting of all details as they are needed for context-free tasks of programming, configuration or formulation of regulations. It must be possible to represent incomplete or uncertain information and to indicate those aspects of a model which are only incompletely specified. If misunderstandings are observed with respect to this incompleteness it can be gradually reduced by making the diagrams more explicit and formal.

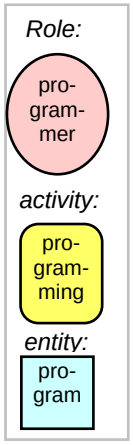
Therefore, for the early phases of designing socio-technical systems or processes it is reasonable to use a modeling notation to create diagrams which

- visualize the complex interdependencies between people, between human and computers, and between technical components
- do not focus on selected views such as processes, functions, tasks of objects but allows modelers to combine these views,
- can integrate overview sketches of the planned solution with the representation of rich details, if a contributor wants to introduce them. Subsequently it is not necessary to switch between different diagrams to see varying degrees of details.
- integrate formal and informal structures as well as technical and social aspects
- handle incompleteness and vagueness (e.g. if it is not clear which sub-activities are part of a task or under which conditions these sub-activities are carried out.)
- and represent conventions, interests, and multiple perspectives.

2 The basic elements of SeeMe

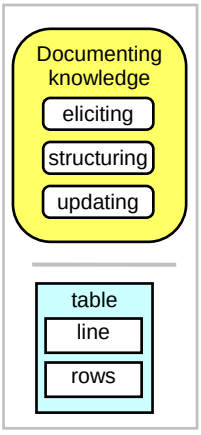
SeeMe helps to describe the interaction between people and between humans and physical or technical objects of the world. Therefore, SeeMe differentiates between three basic elements:

- ***Roles*** which represent a set of rights and duties as they can be assigned to persons, teams, or organizations. Eventually, the characteristics of a role are based on the expectations of other roles. These kind of reciprocal relationships are typical for social systems – roles are a means to introduce social aspects into the models.
- ***Activities*** which are (usually) carried out by roles and stand for the dynamic aspects which represent change, such as completing of tasks, functions etc.
- ***Entities*** representing passive phenomena; e.g. resources being used or modified by activities, such as documents, tools, programs, items of the physical world. They can represent containers (e.g. a box, a warehouse) or ephemeral phenomena (e.g. an utterance).

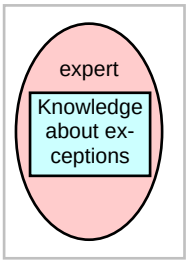


For the layout of a socio-technical diagram we recommend that **roles** are **in the top, activities in the middle and entities in the bottom** of a diagram. However, this is not a strict syntactical or semantical rule, and it can be reasonable to prefer another way of ordering the elements. Basic elements are abstract symbols (on the **level of classes**) and do not represent concrete persons or actions, since SeeMe is used for concepts and plans which represent not only one but a variety of concrete cases. There are multiple ways how a basic element can be *instantiated*.

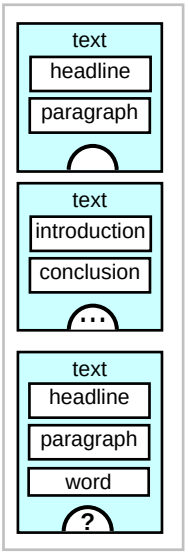
Elements can be embedded into other elements; we say "a sub-element is part of a super-element". Sub-roles can represent parts of the organizational structure of a more complex role, e.g. a `department`, sub-activities may describe the steps of a task (like `documenting knowledge`); entities can contain their components as sub-entities (such as a `table` consists of `rows` and `columns`). Sub-elements can contain further sub-elements.



Sub-elements can be **of another type** than their super-elements: A role may contain an entity which – for instance – represents its intellectual capital or its competence (see the `expert` example). An activity may include the depiction of tools which are exclusively assigned to it. If an element has its main relevance for only *one* certain other element, it is recommendable to embed it into this element to denote a kind of encapsulation.



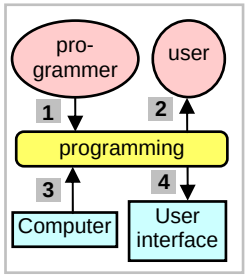
We want to differentiate whether a super-element is completely described by its sub-elements or only partially. **Incompleteness** is indicated by a semi-circle which we metaphorically call “mouse hole”. It is *empty* if the incompleteness is intentionally; *three dots* indicate that we do not know enough to complete the specification, and that further research is required. A *question mark* indicates doubts about the correctness of the used sub-elements. If an element includes “incompleteness-indicators”, it is only partially specified.



A mouse hole does only indicate that a sub-element of the *same* type as the super-element may be missing – e.g. a step of a task. To express that one or more **sub-elements of another type** are not specified, an *unnamed* sub-element of this other type is needed. The fig. about the role `team` expresses with the mouse hole that more sub-roles than the `leader` are relevant for the `team`. The mouse hole is exclusively referring to unspecified sub-roles, but not to other types of unspecified elements, such as activities or entities. However, we might want to express that also entities are needed to specify the `team` and that they should be a subject of further investigation; therefore, an additional, unnamed entity with three dots should be inserted. This entity is not referred to by the mouse hole.

3 Relations between basic elements

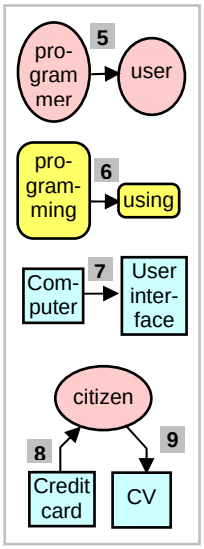
SeeMe offers nine standard relations depending on the types of elements being connected and on the relation's direction. Relations are depicted with directed arcs. They have a starting- and an ending-point which are anchored in basic elements. The "reading-direction" mirrors the direction of the arcs in the following definitions:



- The role (`programmer`) ***carries out*** [1] the activity (`programming`);
- the activity ***influences*** [2] the role (`user`);
- an entity (`computer`) ***is used by*** [3] the activity,
- which ***produces*** *or* ***modifies*** [4] an entity (`interface`).

While *modifying* (from activity to entity) represents a technical, deterministical relationship, *influencing* (from activity to role) indicates a contingent relationship [cf. Herrmann et al., 2004a]. Activities such as teaching, communicating, advising etc. can have an impact on roles, but they can not determine how the roles change their characteristics – therefore we use the term *influencing* to describe this relation

Elements of the same type can be related to each other:

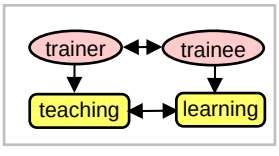


- A role (`programmer`) can ***have expectations towards*** **[5]** another role (`user`) – the content of the expectation can be expressed with an attribute (cf. section 6, description-on-relation);
- an activity ***is followed by*** **[6]** another one;
- and an entity can ***belong to*** **[7]** another one. *Belonging to* is a very abstract term which covers more concrete relationships such as that one entity is *the pre-requisite of* another one (like `computer`→ user interface or `original`→`copy`).

Furthermore,

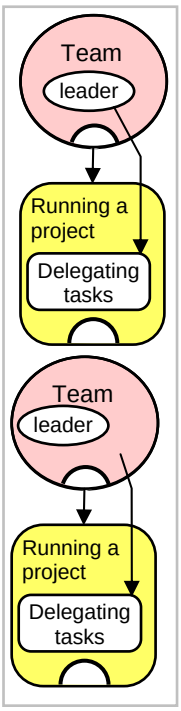
- a role can *be* ***described by*** **[8]** an entity (`CV, curriculum vitae`) to which the arrow points; this relation is relevant to express privacy issues; for instance, it can be used to indicate all kinds of traces which are left by a user in a computer system.
- An entity (`credit card`) points to a role with an arc, if this entity ***is possessed by*** **[9]** the role. This relation can especially be used to express access rights. (Attention: this relation does never mean that the entity triggers the behavior of a role, this needs always an activity).

As the examples with the relations of type [8] and [9] show, a relation does not need to be presented by a straight line. The arcs contain waypoints where they change their direction. It should be noticed that relations can connect an element with itself and they can maximally connect two elements – the waypoints must never be used to connect three or more elements.

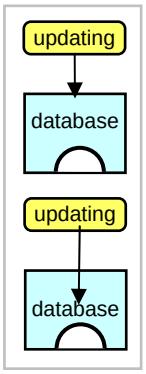


It is possible to use **double arrows** e.g. if two activities are alternating or if roles have expectations towards each other.

Relations can be connected to super-elements or **to** one of its **sub-elements** (that means crossing the border of the super-element). If a relation is pointing to a super-element, it is also referring to all of its sub-elements: while the `team` is responsible for the whole runtime of the project, only the `team leader` can `delegate tasks`.

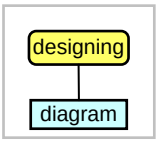


Relations can be **incomplete**: If it is not clear whether a relation should be connected with the whole super-element or only with its sub-elements (and with which of them), the **relation crosses the super-element** (`team`) and is not connected with a distinctive sub-element. In the example, we can express with the upper diagram that the `team leader delegates tasks`, while the lower diagram expresses that it is unclear which sub-roles of the `team` can be in charge with `delegating tasks`. This is a way to depict that it is not exactly clear who is in charge with the sub-activity `delegating tasks`.

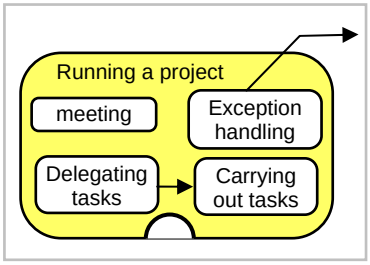


All types of relations can be used in this manner, e.g. the type "modification" in the example `updating a data base`. While in the upper case the whole `data-base` is `updated`, the lower case shows that the `data-base` is only partially `updated`.

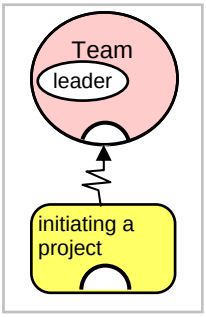
There are further kinds of incompleteness: a relation needs **not to be directed** by an arrow head if its direction is unknown or can not be identified (e.g. between the activity `designing` and the entity `dia-gram`).



Relations **can be left out**, e.g. between sub-activities if it is not clear in which sequence they occur. While the sequence between `delegating` and `carrying out` is clear, `meeting` and `exception handling` should not be brought in a pre-defined sequence. An arrow can also start or end in an undefined space if the element to which it is anchored is unknown or not represented in the diagram – in the case of the example, it is not clear how the reaction on `exception handling` looks like.

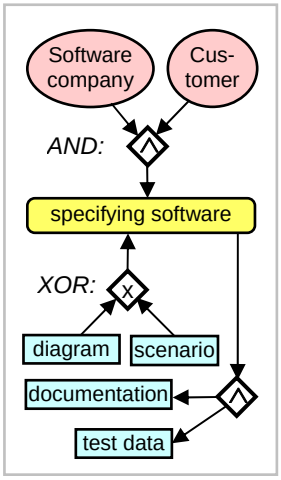


Meta-relations (zig-zag-arrow) help to express that a basic element is involved in influencing or modifying an element's structure, although the structure is part of the diagram. Since we model dynamic processes, every element has dynamic characteristics which can be influenced or modified by the activities in the diagram, and it has characteristics which usually remain stable when the process is executed. However it may happen that we have to indicate that even the structures, which are underlying the model, might not be stable but can be a subject of change – in this case a meta-relation is appropriate. This can be illustrated by a little story: Imagine an element (e.g. the role `team`) had only been partially specified during a workshop. Afterwards, the modeler tries to completely specify the role with a set of sub-roles, although he had been recommended not to do so. As foreseen, in the next workshop the participants can bring an example that the activity (`initiating the project`) can modify this role in a way that the modeler's proposed set of sub-roles is not appropriate

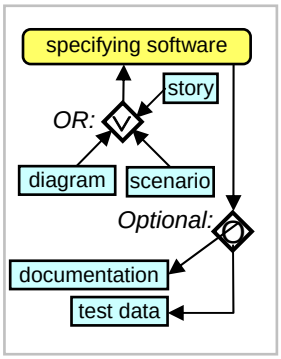


and should be remodeled after the activity was instantiated. However, re-modeling is not really reasonable, since the future might reveal situations where the role has again to be remodeled in another way. The conclusion of this story that it is reasonable to keep the role incompletely specified so that it is open for all kinds of variants, and to use the meta-relation to express that there is an activity which has the function to specify the role with sub-roles which comply with the `team's` task. Therefore, incompleteness and meta-relations are closely related. They can be applied between all types of basic elements. However, they are not often used from the viewpoint of our practical research.

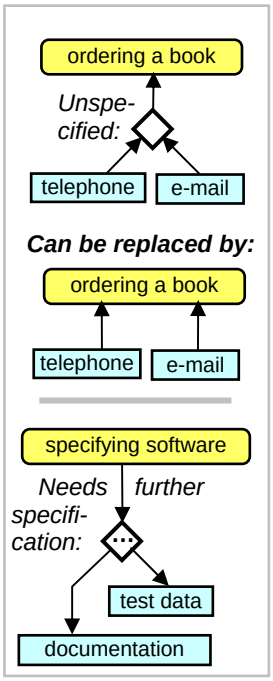
4 Connected relations



If two or more relations are assigned to the same element with their starting- or end-points, the question is, how the interdependency between them can be described. These kinds of dependencies are expressed with **connectors**. We can differentiate between two main cases: The **AND-Connector** should be used, if *all* of the relations have always to be instantiated *together* (in the example, the `software company` as well as the `customer` have to work on the `software specification`). The **XOR-connector** expresses that exactly *one* of the connected relations can be instantiated, both relations together are not possible (either `diagrams` or `scenarios` are used for the `specification`). With respect to the direction of the relations, a connector can lead to a joining or to a splitting of the relations (splitting is given in the example when the software-specification can produce a `documentation` as well as a set of `test data`). Beyond these examples, also more than two relations can be connected (see next example with `diagram`, `scenario` *and* `story`).



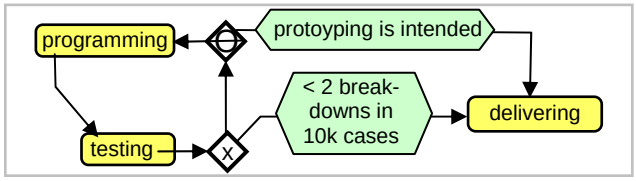
Further types of connectors are the **OR-connector** which expresses that either one, a subset or all of the connected relations can be instantiated (the revised example shows that it is possible that a `scenario` AND a `diagram` are used, and also a `story`) The **OPTIONAL-connector** expresses that one certain relation (which goes through the connector) is mandatory while the other one is optional (while the activity `specifying software` must produce a `documentation`, it is optional whether a set of `test data` is produced.)



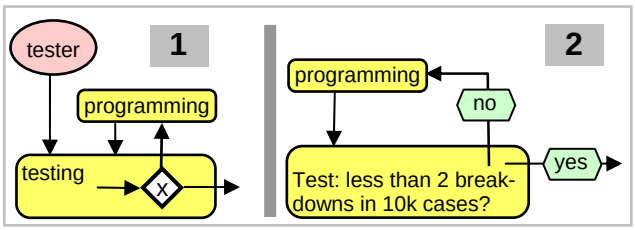
The **logical type** of a connector can be left **unspecified**, if its meaning is clear by the context of a diagram or if we do not want to be more precise. In the example it remains intentionally undefined whether both or only one of the entities `telephone` and `e-mail` must be used or can be used for `ordering a book`. Such an empty connector can also be left out: if two or more relations of the same type are directly connected with an element, this represents a short cut for a diagram section which uses an unspecified connector. However, the **empty connector** should be used when it helps to avoid the drawing of several parallel relation lines, or if it will possibly be specified or completed with vagueness indicators: If we want to indicate that further research and decision making is required to specify the connector, we should fill in three dots, as in the example.

5 Modifiers: Conditions and Events

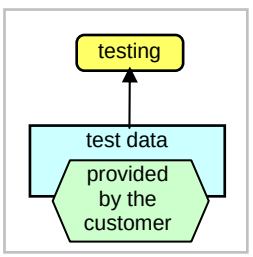
If a connector requires an OR-connection, the question arises under which **condition** which relation can be instantiated. In many case, these conditions can be clearly derived from the context (as well as the type of connectors is implicitly clear). In the example it is assumed that the `programming` is continued if the `quality check` is negative while `delivering` is instantiated in the positive case. If the contextual specification is not clear enough for the relevant stakeholders, we **annotate** so called **modifiers** as green hexagons to the relations as illustrated in the next example. **Note**: Relation arcs do never end or start at modifiers but go invisibly through them. This is reasonable because the type of the relation is defined by the types of the basis elements being connected (the connection between a basic-element and a modifier does not specify a new type of relation). The modifiers contain the conditions or events which are assigned to the instantiation of a relation. The example illustrates that the software is only `delivered` if there are `less than 2 breakdowns in 10.000 cases`, otherwise the programming has to be continued – however this incomplete software can yet be optionally delivered to use it as a prototype, if `prototyping is intended`. It should be noted, that testing has not to be completed when the programming goes on. (The semantic implications of modifiers can be very complex since it may happen that an activity of a larger process model can not be instantiated because of a modifier which is depicted in an "earlier" phase of the process.)



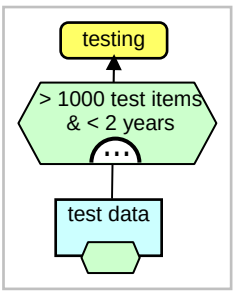
The question may arise **who decides** (and when) whether a **condition is fulfilled** or not. This decision is usually assigned to the activity (and the role carrying out this activity) which is connected to the modified relations. If this assignment is contextually unclear, the connector can be embedded into the activity where the decision is made (case 1). Another way to make the place of the decision clear, is to formulate a question within an activity and to add modifiers with YES or NO – in this case we have again the construction which abbreviates the usage of an empty connector (cf. the example with `ordering a book`), and the type of the connector (XOR) is clear by the context and needs not to be depicted (Case 2). A modeler would use case 1 if the conditions can not be clearly specified as it is the case with `YES` vs. `NO`. In contrast, case 2 is preferred if we do not want to confront the recipients of the diagrams with the logical complexity of connectors – it is an advantage of SeeMe that connectors can be completely avoided in the early phases of getting to know these modeling method.



In SeeMe, **modifiers** can not only be **annotated to** relations but also to all kinds of **basic elements**. This is helpful if the modification has nothing to do with any relations. The example shows that `test data` is always used if they are available, but they are not always available (only if `provided by the customer`). This is helpful, if the model is not so complete that it documents the production or the



pre-requisites of the element where a modifier is annotated (e.g. `test data`).

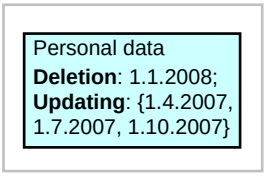


Modifiers can also **be incomplete**: they can be empty if we only know that the instantiation of an element or relation depends on a condition but the condition is unknown or contextual available. Three dots indicate, that the condition has possibly to be specified later, a question mark signals doubts whether the usage of a condition is appropriate. We can also add a mouse hole (empty or with "…" or "?") to indicate that the specification of the condition is incomplete. The example depicts that we do not know under which condition the `test data` is available and that we have to specify further criteria under which it is used (beside the fact that it includes more than `1000 items` and that the data is `younger than a year`).

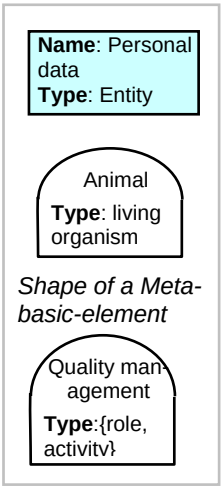
SeeMe offers also **modifiers as super-elements** which can contain a web of conditions and connectors as sub-elements. From the viewpoint of our practical research experience, these super-modifiers are only seldom needed. They can be useful to collect criteria and conditions during a discussion.

6 Attributes and additional specifications of relations

The elements of a modeling notation such as SeeMe have to be completed with text and numbers. It is possible to **add attributes** to the elements. An attribute has usually a name followed by a colon and one or more values, e.g. "`date of deletion: 1.1.2008`; `dates of update: {1.4.2007, 1.7.2007, 1.10.2007}`. In the example, the words "date of" are left out in the attributes' names, since it is clear by the type of values what is meant. Basic elements have usually a name which is also an attribute; the name-attribute of the example is: "`personal data`" – here, the prefix *name* is left out. Typical useful attributes are: duration of an activity; type of media used for communication; values of contracts; entry-fields of a form; characteristics of a role, such as a formal qualification etc.

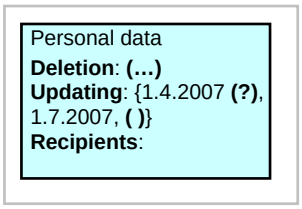


The possibility to be **incomplete** can also be applied to **attributes**. Mostly the incompleteness is compensated by the context. Names of attributes can be left out. The name of the attribute *`name`*, for instance, can be omitted since it is obvious which text is used as a name. *`Type`* (such as role, activity, entity) is also an attribute which is assigned to every element but is not made explicit because it is clearly indicated by the geometrical shape of the elements. The example shows an element with completely specified standard attributes (`name: personal data; type: entity`).

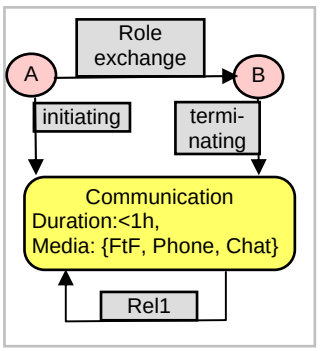


The attribute *`type`* can also be used, if an element of a special kind (such as *living organism)* has to be modeled which is not part of the set of basic-element-types. In this case, a so called **meta-basic-element** is used and specified with a certain type, such as "`type: living organism`". This construction can also be used, if an element's type is unclear (e.g. `quality management` which may be a `role` as well as an `activity`). Meta-basic-elements are abstract, generalized representations of the basic-elements as they were introduced in section 1. This Meta-basic-element helps modelers in exceptional situations where they want to vary the set of pre-specified basic elements.

To indicate **incompleteness** of the textual attributes, parentheses are used instead of semi-circles: "( )" represents an empty mouse hole; (…); (?). The parentheses are related to the round shape of the mouse hole; it is also possible to use angle brackets "<,>" to express conditions. The example means: the `date of deletion` has still to be specified; `updating` at the `1.4.2007` may be incorrect; further updating after the 1.7. might happen, but we do not want to specify the date; the value for recipients is also unspecified.



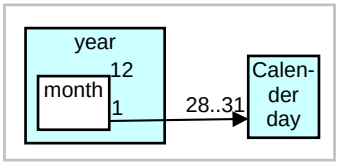
Names and types as well as attributes can also be annotated to relations. For this purpose, we use grey rectangles which we call ***description-on-relation***. Names (which can be represented by a number) are helpful to refer to the relation, e.g. in additional textual descriptions (beside the diagram). Usually it is clear by the context, whether a type or name is added to a relation; otherwise *type* or *name* have to be explicitly depicted. The example introduces special types of relations such as `initiating` or `terminating` an activity or `role exchange`. `Rel1` is a name which can be used as reference in an extra text to explain how and why the repetition of communication takes place.



Furthermore, we can use **names of relations in conditions** (a condition with the name of a relation is considered as fulfilled, if the relation has been instantiated; this enables us to define conditions

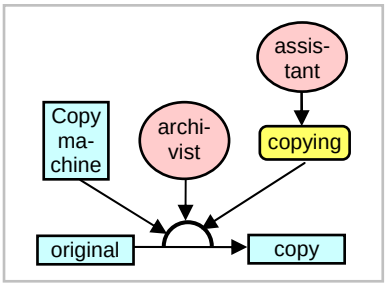
which refer to previous events in a process). Typing of relations is helpful if the standard meanings of relations are not sufficient, e.g. if persons can `exchange roles` etc. Especially, *aggregation* and *generalization*, as used in class diagrams, can be important non-standard types. Since they are used for object oriented modeling, the special graphical representations as they are used for aggregation and generalization in UML-class diagrams can also be used in SeeMe.

Special attribute are **cardinalities** as they are known from entity relationship modeling – they can be annotated to elements (which is helpful for sub-elements) and to the start or end of a relation. The examples expresses that a `year` has `12 months`, which consists of `28 to 31 calendar days` and that every `day` belongs to exactly one `month`.



Another special attribute indicates the **probability** with which a certain condition is fulfilled; they can be **added to the modifiers** (in the example in section 7, the probability is 2% that an `exception handling` takes place).

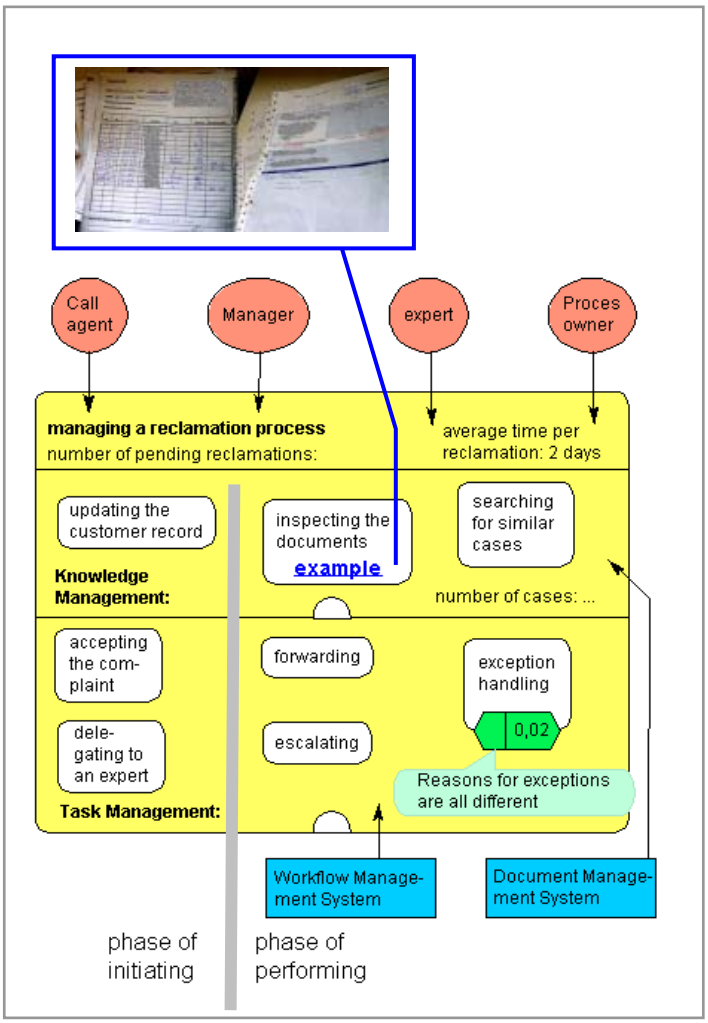
Since relations indicate the dynamics of a model, we offer the additional possibility to **assign basic-elements to relations** which "help" (or are needed) to instantiate them. They are connected via an arc and a semi-circle to the relation. We assign **roles which are interested** in the relation, and **activities and entities which are needed** to instantiate the relation, as shown in the example. This construction usually represents side aspects which help to understand the modeled phenomena but are not of central interest.



7 SeeMe's little helpers

SeeMe-modelers can use elements which help to make the diagrams easier to comprehend. Not all of these helpers are part of the formal specification of SeeMe.

Segment lines are part of the formal notation. They separate super-elements into segments which help to sort sub-elements or attributes according to different perspectives (in the example we differentiate between `knowledge management` and `task management`). The top segment usually contains the name of the segmented super-element as well as attributes which apply to the whole segment (i.e. to every of its segments, such as `average time per reclamation`). Relations which point into the top-segment are related to the whole super-element (as the roles in our example); if they point into another segment they are exclusively assigned to it (as `workflow management system` and `document management system`). Names can be assigned to segments by using attributes. Vertical segment-lines in modifiers help to differentiate between conditions (on the left side) and probabilities (on the right) – see the example of `exception handling`).



Other Elements can primarily be used with the SeeMe-editor and are not a formal part of the SeeMe-notation. The SeeMe-editor is a software-based tool which is especially designed to support the drawing as well as the presentation of SeeMe-diagrams. Switching between drawing and presenting the diagrams can take place seamlessly.

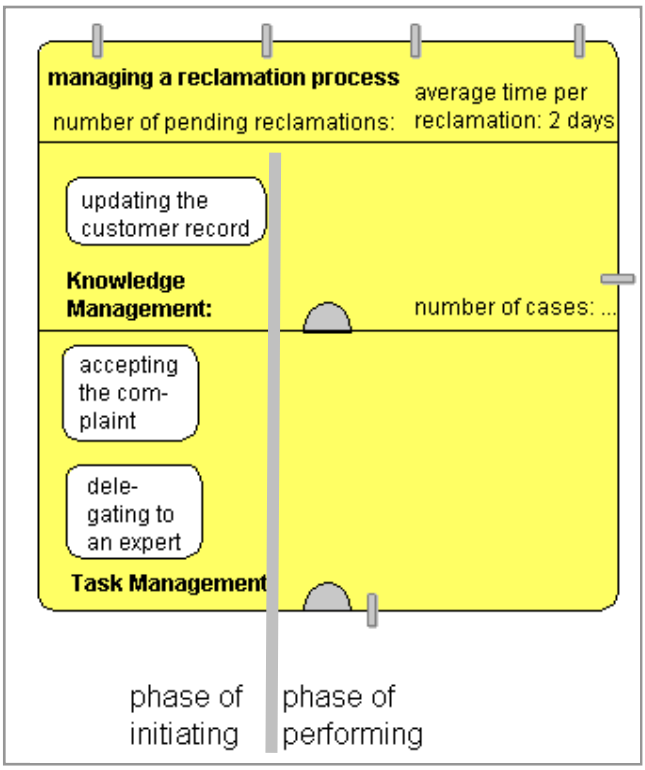
Typical examples of the editor's features are **dividers**. These are thick, usually grey lines which divide a diagram into different areas to help the recipient of a diagram to recognize the different aspects of content or topics of a larger model. E.g. the start of phases of a process or a project can be indicated with these dividers. Dividers can be freely used since they are not a part of the formal notation; therefore, they should not be confused with segment lines. For instance, it is possible to draw "swim lanes" with them as they are used within some modeling methods to differentiate between role, activities and tools.

The SeeMe-editor offers specific attributes which represent a hyperlink. These hyperlinks can be used with the editor to start an applica-

tion by clicking on them. With this feature, the displaying of pictures, other SeeMe-diagrams, webpages, screenshots of prototypes etc. can be started. The pictures or additional documents help to illustrate the abstract element containing the hyperlink as shown in the example with `documents`.

Furthermore, it is possible to **add free text to** diagrams, which is not connected with certain elements. This text can be used to add a headline to a diagram or to make explanations which refer to the whole diagram. Text can also be added to geometrical dividers, e.g. to indicate their meaning, since it is not formally specified (such as `initial phase` and `performing phase`). Special text can also be connected to basic-elements or relations to serve as a **comment** which is exclusively assigned to the annotated element – this text appears as bubble (such as `reasons for exceptions are all different`).

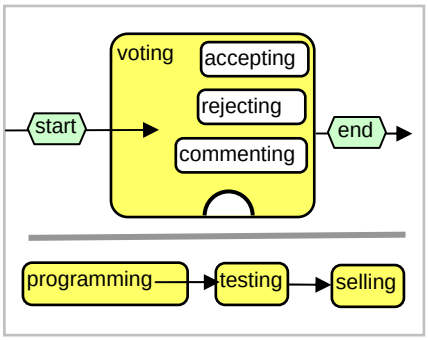
The SeeMe editor offers another useful feature which helps to focus onto the essential elements which are under discussion in a communicative process. For this purpose, basic-sub-elements and/or relations can be temporally hidden and be shown again if they are needed. This concept of **hiding and showing** is outlined in Herrmann, 1998. If a sub-element has been made disappearing, this is indicated by a grey mouse hole, a hidden relation appears as a grey, thickened residue of the arc. The example below shows how the diagram `managing the reclamation process` looks like if all elements are hidden which do not belong to the activities of the `phase of initiating`. The hidden elements can be shown again by clicking on the grey mouse holes or residues.



8 Guidelines for modeling with SeeMe

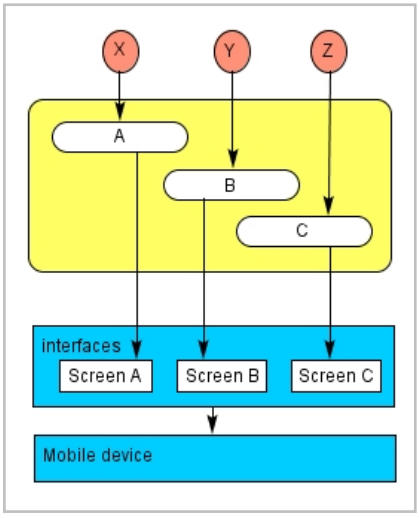
The **creation of a model** of a socio-technical work process may consist of the following steps:

- It can start with the collection of one type of relevant basic-elements: roles, activities, or entities. If there is no clear preference, we recommend starting with activities.
- The next question is, whether there are sequences between the collected activities – for sub-sets of non-sequential activities it might be reasonable to assign them to a super-element as sub-elements.
- Next step is to ask which entities are produced / modified or used. The usage of entities can especially focus on the support by information and communication technology.
- Another type of questions refers to the roles which carry out the activities or are influenced by them (the influence-relation is not often used; if a role R reacts on an activity A with an own activity, the influence of A on R is implicitly clear).
- Every entity has to be created somehow and/or be used somewhere – and it has to be asked whether this should be modeled or not.
- The collecting of roles and entities may give reason to integrate additional activities or sub-activities into the model.



Note that all **the sub-activities of an activity** must **be carried out**, if an incoming arc ends at its border instead of crossing into the activity. In the case of crossing, only one or some sub-activities are to be carried out (in the `voting` example, an incoming arc which ends at the border is nonsense, since `rejecting` and `accepting` are never performed together, while `commenting` can be added to each of these votes. If a leaving arc crosses the border, this activity will stay active although the next one is already started. Starting a leaving arc at the border of an activity means that it is completed. `Programming` might go on after `testing` has started, but the software should not be `sold` before `testing` is completed. Process diagrams can start and/or end with relations instead with activities – these relations can be indicated with **start- or end-modifiers**.

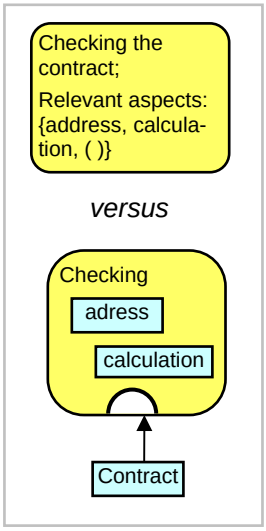
Be aware that whenever somebody or something is influenced / modified, an activity is needed, and the traces of activities, which are observable, are presented with entities – since entities are abstract classes they can even represent ephemeral phenomena such as verbal utterances, the ringing of a mobile phone etc. (that means that entities do not always represent persistent objects).



When the diagram develops step by step it should sometimes be **aesthetically improved**. Basic elements of the same relevance or embedding level should be of the same size if possible and aligned to a common row or column. However, it can be reasonable to align sub-activities in a stair case pattern to facilitate the connecting of relations as shown in the example. It is most important to keep the number of relations low. Furthermore, avoid long relation arcs which change their direction several times. Intersections of arcs or arcs which run parallel

can also be confusing. This can be achieved by using connectors, by aggregating elements into a super-element (which does not need to have a name), or by making basic elements geometrically so large that the relations can be short. The example with the stair case pattern can also be considered as a recommendable template: roles on top, activities in the middle, tools and computers are at the bottom, and interface entities between the technical system and the activities. The standard (good case) sequence of activities should be displayed with a left-to-right order.

The same aspects of a diagram are sometimes described by textual attributes or by an extended name of a basic element, while other diagram fragments use more basic-elements for the same description. The strategy of choosing between these options is to use **basic-elements instead of textual attributes** if they might have to be connected with other elements by relations in the further design. Elements which are not connected with any relation can be transformed into text. This strategy is related to the question about the appropriate **degree of granularity** which should be chosen: it is reasonable to introduce those sub-elements which are needed to understand the character of the super-element, which have to be connected with relations or which cannot be derived from the context.



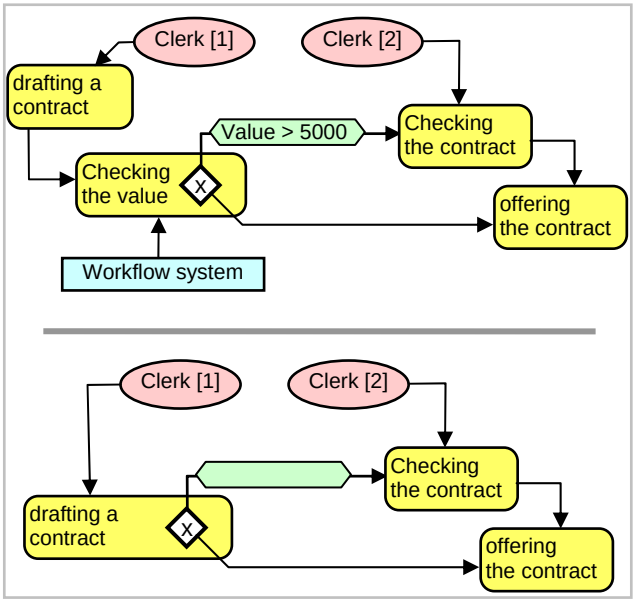
There are two - but closely related - basic concepts of incomplete specification. The crossing relation is always a means to indicate that not the whole element is used, modified, following, influencing, carrying out etc., but (mostly) only a part of it. We gave examples above about the advantage of this construction, especially for activities as explained with the `voting` example or with the sequence of `programming` and `testing`.

The other concept of incompleteness is represented by the mouse holes (or parentheses in the case of attributes). Whether a mouse hole is added or not to a basic element or not is triggered by the following question: “Is the set of the depicted sub-elements complete or are there sub-elements imaginable which have to be added to the basic-element to make its specification complete?” If the set of sub-elements is imaginably incomplete, they should be completed or a **mouse hole should be added in the following cases**:

- It is clear from the context which sub-elements are omitted
- It is not interesting to add further sub-elements for the purpose of the model
- Further specification should not be anticipated but be left to those ones who once will be in charge with instantiating the diagram during their work
- The specified element is a subject of dynamic change and it is therefore not reasonable to add further specification
- We do not know enough at the given moment to complete the set of sub-element – but this can be the case after further research (this is to be indicated by the three dots)
- There might be doubts about the correctness of the chosen set of specified sub-elements – this should be indicated with a “?”.

Those parts of the diagram which are not needed to program or to formally regulate a solution should make extensive use of being incomplete.

A specific relevance has the usage of incompleteness to express **freedom of decision** as shown by the example dealing with `checking a contract`. In the first case it is decided by a `workflow system`, whether a second `clerk` (the numbers in brackets indicate that different persons should instantiate the roles) will `check the contract`. In the second case, it is decided by clerk [1], whether a checking of the contract is reasonable or not. The condition is left unspecified to express that it is ad-hoc specified by `clerk` [1]. Actually the depicting of the empty condition is not needed, since it is clear by the contest of the diagram elements how the decision is made. However, depicting the empty hexagon emphasizes the message, that it is willingly decided to leave the decision with the `clerk` [1].



9 References

Herrmann, Th. (1997): Communicable Models for Cooperative Processes. In: Salvendy, G. et al. (eds.) (1997): Design of Computing Systems: Cognitive Considerations. Amsterdam et al. pp 285 -288.

Herrmann, Th.; Loser, K.-U.: (1999): Vagueness in models of socio-technical systems. In: Behaviour and Information Technology. Vol. 18, no.5, pp 313-323.

Herrmann, Th.; Hoffmann, M.; Loser; K.-U. (1999): Modellieren mit SeeMe - Alternativen wider die Trockenlegung feuchter Informationslandschaften. In: Desel, J.; Pohl, K.; Schürr, A. (eds.): Modellierung'99. Stuttgart: Teubner. pp 59-74.

Herrmann, Th., Hoffmann, M., Loser, K.-U., Moysich, K. (2000): Semistructured models are surprisingly useful for user-centered design. In: Dieng, R.; Giboin, A., Karsenty, L., De Michelis, G. (Hrsg.): Designing cooperative systems. Amsterdam: IOC press. pp 159 –174.

Herrmann, Th. (1999): Flexible Präsentation von Prozeßmodellen. In: Ahrend, E.; Eberleh, E.; K. Pitschke (eds.): Software-Ergonomie '99. Stuttgart: Teubner. pp 123 - 136.

Kienle, A., Herrmann, T. (2003): Integration of Communication, Coordination and Learning Material – a Guide for the Functionality of Collaborative Learning Environments. In: Proceedings of HICSS-36.

Herrmann, Th.; Kunau, G.; Loser, K.-U.; Hoffmann, M. (2004a): A Modeling Method for the Development of Groupware Applications as Socio-Technical Systems. In: Behaviour and Information Technology. Vol. 23, Nr.2. S. 119 – 135.

Herrmann, Th. Kunau, G.; Loser, K.-U.; Menold, N.; (2004b): Socio-technical Walkthrough: Designing Technology along Work Processes. In: Clement, A.; de Cindio, F.; Oostveen, A.-M.; Schuler, D.; van den Besselar,P.: PDC 2004 Proceedings. Artful Integration. Interweaving Media, Materials and Practices. pp 132-142.

Loser, Kai-Uwe (2005): Unterstützung der Adoption kommerzieller Standardsoftware durch Diagramme. [http: //hdl.handle.net/2003/ 21659]

Kunau, Gabriele (2006): Facilitating computer supported cooperative work with socio-technical self-descriptions.
[http://hdl.handle.net/2003/22226]

Stefanides, Marek (2006): „Gestaltung kooperativer, technisch-unterstützter Arbeitsprozesse in einer Röntgenpraxis". Diplomarbeit, Universität Dortmund