Our main contribution in this chapter is a specialization of SEQUAL for business process models. In addition to specializing the existing SEQUAL framework, we have also extended it, taking into account aspects of other frameworks described in Chap. 2. As illustrated in Fig. 3.1, there are a number of specializations of SEQUAL. As with GoM, we have a first level that is meant to be relevant for all node-edge-oriented models (SEQUAL-GEN). A second level views a particular type of model (in this book and in this chapter in particular focusing on business process models). Finally, one can have specific guidelines on an even more detailed level, e.g., for business process modeling using BPMN. In real-world modeling activities, we find that all quality levels are important, but the weight on the different levels is different based on the different goals of modeling, as will be exemplified and discussed in more detail in Chap. 4.

In presenting the specialization of SEQUAL here, we will include not only aspects inherited from the general SEQUAL framework briefly described in Sect. 2.2.1, but also aspects specific to business process modeling, based on aspects noted in Sect. 2.3 in particular. For some areas, such as empirical and syntactic quality, we will also illustrate with aspects specific to process modeling language, using BPMN as the main example.

The other current specializations, explaining the abbreviations used in Fig. 3.1, are the following:

- SRS—software requirements specifications (Krogstie 2001)
- DM—data models (Krogstie 2013a; Krogstie 2015)
- DQ—data quality (Krogstie 2013b; Krogstie 2015)
- IM—interactive models (Krogstie and Jørgensen 2002)
- EM—enterprise models/modeling languages (Krogstie 2012a)
- BPM—business process models (Krogstie 2012b)
The high-level SEQUAL framework has many similarities with Krogstie (2012a), but we have added reference models and reference languages/ontologies due to their importance in the business process modeling area when we present the updated SEQUAL framework below.

### 3.1 Sets in the Quality Framework

**G, the Goals of the Modeling Task**
What goals are meant to be fulfilled through the modeling? In simple cases, there is one well-defined goal, whereas oftentimes (views and versions of), the same model is used to achieve many, often partly contradictory goals. As discussed in Chap. 1, conceptual models are used for a number of different purposes, and as observed from practice, even within the same project, different stakeholders can have different goals, i.e., the process models can be multivalent. In Chap. 4, we will present
more examples of this phenomenon linked to large case studies. The initial goals of modeling are normally defined before the modeling starts, but can often be changed and extended during a project, either in a planned or in an emergent fashion (Krogstie et al. 2006). Goals also include other organizational and economic issues, e.g., whether a requirements specification model will be constrained because one wants to produce a computerized information system based on the software requirements specifications under the given time and resource constraints.

A, the Audience
The audience is represented through the part of the model that they can access and indirectly through their explicit knowledge (K) and interpretation of the models (I). The audience is the union of the set of individual actors $A_1, \ldots, A_k$, the set of organizational actors (an organizational actor typically consists of a group of people with at least one shared goal) $A_{k+1}, \ldots, A_n$, and the set of technical actors $A_{n+1}, \ldots, A_m$ who need to relate to the model. The individuals who are members of the audience are called the participants of the modeling process. The participants $P$ are a subset of the set of stakeholders $S$ of the process of creating the model.

Those actively creating models (modelers) are a subset of the participants.

A technical actor is typically a computer program, e.g., a modeling tool, which must “understand” parts of the model at a certain level to automatically manipulate it to, e.g., perform code generation, model layout, or model analysis based on the models to which they have access.

The audience often changes during the process of developing and evolving the process model when people leave or enter the project or organization.

L, the Language Extension
The language extension is the set of all statements that are possible to make according to the vocabulary and syntax of the process modeling languages used. Several languages can be used in the same modeling effort, corresponding to the sets $L_1, \ldots, L_j$. One example is the different diagrams defined in BPMN. These languages can be interrelated (typically by sharing concepts across sublanguages). Sublanguages are related to the complete language by the limitations on the vocabulary, the set of allowed grammatical rules in the syntax of the overall language, or (typically) both. The statements in the language model of a formal or semiformal language $L_i$ are denoted as $M(L_i)$. This model is often called the meta-model of the language, a term that is appropriate only in connection with work on repositories for process models. Another term that could be used for this is language model.

The languages used in a modeling effort are often predefined, but it is increasingly common that one creates specific modeling languages or extensions to existing languages, using a meta-modeling environment for the modeling effort, in which case the syntax and semantics of the languages must be intersubjectively agreed upon by the audience as part of the modeling. If one is using an existing language, the “correct” syntax and semantics of the language (to the extent that it is formally
defined) can be regarded as predefined. One often chooses to apply only parts of the predefined process modeling languages for a given modeling effort and changes this subset during a project as appropriate (e.g., it is seldom that the whole of BPMN is used within one project (zur Mühlen and Recker 2008)), and as we will see in cases in the next chapter, companies often develop their own specific, reduced but also, in parts, extended versions of the standard process modeling languages.

**M, the Externalized Model**

This is the set of all statements in the explicitly represented model of part of the perceived reality expressed in a process modeling language. $M_E$ is the set of explicit statements in a model, whereas $M_I$ is the set of implicit statements, i.e., the statements not made but implied through the deduction rules of the modeling language. For example, in a process model, if activity A is before activity B and activity B is before activity C, then activity A is before activity C. A model written in language $L_i$ is denoted as $ML_i$. The meaning of $ML_i$ is established through the (intersubjectively) agreed-upon syntax and semantics of $L_i$. If the language has a formal operational or logical semantics, then the achievement of this agreement is easier to assure.

For each participant, the part of the externalized model that is considered relevant can be viewed as a projection of the total externalized model; hence, $M$ can be divided into projections $M_1, \ldots, M_k$, corresponding to participants $A_1, \ldots, A_k$. Generally, these projections will not be disjoint, but the union of the projections should cover $M$. The actor should at least have access to what is relevant, as will be discussed as part of physical quality. $M$ will clearly evolve during modeling as statements are inserted into and deleted from the model.

**D, the Modeling Domain**

The modeling domain is the set of all statements that can be made about the situation at hand. One can differentiate between domains along two orthogonal dimensions:

- **Temporal**: Is the model of a past, current (e.g., as-is), or future situation (e.g., to-be) as it is perceived by someone in the audience? The first two models are descriptive models, whereas the last type will typically be a prescriptive model, although it may also be of a future unwanted situation, as discussed in Chap. 1.
- **Scope**: Examples of different scopes are (a subset of) the physical world (a subset of) the social world, an organization, an information system, and a computerized information system (CIS).

The domains evolve during modeling, both through external changes outside the control of the modeling activity and through the deployment and activation of the model itself. Note that the precise delimitation of the scope often can be tricky. As discussed in Sect. 2.1.4, the scope might be blurred or hidden, and there might be disagreement among stakeholders for what is within scope. One can differentiate between the current domain $D$ and a perceived optimal domain $D^O$ (often represented by the ought-to-be model) that one attempts to achieve (e.g., through the information systems development).
**K, the Relevant Explicit Knowledge of the Participants**

The relevant explicit knowledge is the union of the set of statements, \( K_1, \ldots, K_k \), one for each participant. \( K_i \) is all possible statements that would be correct and relevant for addressing the problem at hand according to the knowledge of the participant \( A_i \). \( K_i \) is a subset of \( K^i \), the explicit knowledge of the social actor \( A_i \) that can be externalized. \( M_i \) is an externalization of \( K_i \) and is a model made based on the knowledge of the individual or organizational actor. Even if the internal reality of each individual will always differ, the internal reality that can be made explicit (externalized) concerning a constrained area may be equal for all practical purposes, especially within well-defined groups of participants (Gjersvik 1993; Orlikowski and Gash 1994). Thus, it can be meaningful to also speak in terms of the explicit knowledge of an organizational actor. \( M_i \) is an externalization of \( K_i \) and is a model made based on the knowledge of the individual or organizational actor. Even if the internal reality of each individual will always differ, the internal reality that can be made explicit (externalized) concerning a constrained area may be equal for all practical purposes, especially within well-defined groups of participants (Gjersvik 1993; Orlikowski and Gash 1994). Thus, it can be meaningful to also speak in terms of the explicit knowledge of an organizational actor.\( M_i \) is an externalization of \( K_i \) and is a model made based on the knowledge of the individual or organizational actor. Even if the internal reality of each individual will always differ, the internal reality that can be made explicit (externalized) concerning a constrained area may be equal for all practical purposes, especially within well-defined groups of participants (Gjersvik 1993; Orlikowski and Gash 1994). Thus, it can be meaningful to also speak in terms of the explicit knowledge of an organizational actor.\( M_i \) is an externalization of \( K_i \) and is a model made based on the knowledge of the individual or organizational actor. Even if the internal reality of each individual will always differ, the internal reality that can be made explicit (externalized) concerning a constrained area may be equal for all practical purposes, especially within well-defined groups of participants (Gjersvik 1993; Orlikowski and Gash 1994). Thus, it can be meaningful to also speak in terms of the explicit knowledge of an organizational actor.

**I, the Social Actor Interpretation**

The social audience interpretation is the set of all statements that the social audience perceives that an externalized model consists of. Precisely similar to the externalized model itself, its interpretation can be projected into \( I_1, \ldots, I_n \), denoting the statements in the externalized model that are perceived by each social actor.

**T, the Technical Actor Interpretation**

Similar to the above, \( I_{n+1}, \ldots, I_m \) denote the statements in the conceptual model as they are interpreted by each individual technical actor in the audience.

**M (D^0) Reference models**

As discussed earlier in Chap. 2, reference process models and reference process modeling languages (ontologies) have received a considerable interest in this field. Thus, in the description of the quality of business process modeling, we also include these in the main framework. A reference model can be viewed as a model of the same (or at least an overlapping) part of the domain that is of relevance in the modeling initiative. Contrary to the model, which is made to support a specific goal in an organizational setting, the reference model can be used to depict a generic solution in a certain area not being limited by the concrete situation in the organization.

**O, Reference Modeling Languages—Ontologies**

Similarly, we have reference modeling languages (often termed ontologies) that serve a similar role in the modeling language, where one often wants the language used to be compatible to this language, even if the organizational reality and goals often mandate the use of specialized modeling languages rather than the use of standard notations out of the box.

The overall framework is depicted in Fig. 3.2, with only some minor differences at this level from the general framework presented in Fig. 2.2. Throughout this
chapter, we will use part of a BPMN model from the domain of conference
organizing, as depicted in Fig. 3.3, to illustrate the different types of model quality.
The modeling goal in this limited case is to support communication among authors,
conference organizers, and reviewers with respect to the flow of a research paper
review. This figure (also used in Chap. 1) states that, first, a paper is written. Then,
it is submitted for review, is reviewed, and then is accepted or not. If the paper is
accepted, then a final version of the paper must be written and submitted.

In the next 7 sections, we present each of the seven core quality levels illustrated
by relationships between the model and the other sets in Fig. 3.2. For each quality
level, there are one or more quality characteristics. We describe the different means
that can be beneficially used to reach these goals. The means can be of different
types:

- Model properties, which are the subcharacteristics of the high-level goal.
- Beneficial existing qualities, i.e., other quality levels that one would normally
  attempt to address before addressing the quality at the given level. This points to
guidelines for a modeling methodology. Modeling methodology to achieve
high-quality models is discussed more in Chap. 5.
3.1 Sets in the Quality Framework

- Language properties, which are the characteristics of the modeling language being used.
- Modeling methods and techniques.
- Modeling tool functionality (often in combination with a modeling technique, but some modeling techniques are not dependent on tool support).

Metrics for these quality characteristics and subcharacteristics are described. The structure follows the same structure as in the description of the generic SEQUAL framework in Krogstie (2012a), and here, we also include generic aspects. In addition, we specialize the treatment to focus on business process models (and, in some cases, specific modeling notations such as BPMN), including material described in Sect. 2.3.

3.2 The Physical Quality of Business Process Models

Although information systems models are not typical of the physical (three-dimensional, tactile) type, any model must be represented physically somehow, e.g., on disk, on paper, or on a blackboard. An early version of the model in Fig. 3.3 is depicted in Fig. 3.4.

It is also represented in a more formal form on paper as in Fig. 3.3 (and naturally in the electronic source of the book, albeit in a version that cannot be edited for anyone). Originally, this model was made in the Signavio modeling tool; thus, I have access to also update this model (and can make it available to other users of Signavio for comments and additional work if interesting). The basic quality features on the physical level are that the externalized model is persistent, current, and available, enabling the audience to make sense of it (and for modelers to change it when necessary). Making sense of the model is not the same as the participants actually
internalizing the model; at the physical level, we only look at how it is made available for possible interpretation by different actors. Some aspects of this are as follows:

- **Persistence:** How persistent is the model and how protected is it against loss or damage? The method of storing the model should be efficient, i.e., not use more space than is necessary. A simple metric for persistence is the proportion of model statements that are electronically stored in a model repository. This aspect is particularly relevant when business process modeling is performed as part of enterprise modeling, where a large portion of the model statements are elicited through informal modeling techniques (e.g., participatory techniques using wall charts as discussed in more detail in Sect. 5.4), which needs to be transferred into the model repository at a later stage to be made generally available.

- **Currency:** How long ago was the model statements included in the model (assuming that the statements were current when entered). Depending on the type of model, the age of the model statements is of differing levels of importance. If the model is to be of a past situation, it needed only to be current at the time of modeling. For an as-is model, when the domain is changing rapidly (i.e., has high volatility), the currency of the stored model is of greater importance for the model to have appropriate timeliness. This aspect is particularly important in models on the instance level (for instance, event data in a process support environment). Metrics on currency can easily be devised and calculated if the model repository supports the time-stamping of the statements. This area will relate to semantic quality, not only in relation to the time a model statement was entered, but also in relation to the last time the model statement was validated.

- **Availability:** How available is the model to the audience? Clearly, this aspect is dependent on whether the model is externalized and made persistent in the first place. Availability also depends on distributability, especially when members of the audience are geographically dispersed. A model that is in an electronically distributable format will be more easily distributed than a model that must be printed on paper (or is only on paper in the first place). What exactly is distributed may also matter, e.g., the model in an editable form or merely in an output format or a format where one can add annotations but one cannot change the actual model. The entire model should be available to at least someone in the audience, i.e., $A = M$. Security aspects also come into play here, since not all statements should necessarily be available to anyone.

A metric for availability is the proportion of the model statements relevant for a member of the audience that is available for that audience member. In connection with currency and availability, the term “timeliness” is often used; i.e., the model is not only current but also available in time for events that correspond to its usage, which relates directly to the goal of modeling. Thus, timeliness is established as a deontic goal (see Sect. 3.8).

Many of the modeling techniques and tool functionalities in connection with physical quality are based on traditional database functionality, using a model repository solution for the internal representation of the model. In addition, it is
regarded as necessary for advanced tools for business process modeling (Wesenberg 2011) to include functionalities such as version control and configuration management, in addition to advanced concurrency control mechanisms that are not normally found in conventional DBMSs.

A more detailed list of general modeling tool mechanisms, most of them concerning availability and support with regard to meta-model evolution, is presented in Krogstie (2012a). Although most of these are also relevant for business process model repositories, we have not included this here.

### 3.3 The Empirical Quality of Business Process Models

As described briefly in Chap. 1, empirical quality addresses the variety of elements distinguished, the error frequencies when being written or read, coding (including the shapes of boxes), and ergonomics for both documentation and models as presented in modeling tools. The term is based on that this layer collects the traits of visual or textual communication, which has empirically been shown (e.g., through work in cognitive psychology) to result in models that are easier to understand in general.

Changes to improve the empirical quality of a model do not change the statements that are included in the model; thus, we have no set-theoretic definition of this quality characteristic.

For longer descriptions of concepts and informal textual models, several means for text readability have been devised, such as the different types of readability indices. This issue is discussed generically in (Krogstie 2012a). A specific example of this issue is presented in Sect. 4.2 related to one of the case studies.

For computer output specifically, many of the principles and tools used for improving human–computer interfaces are relevant at the empirical level (see, e.g., Shneiderman et al. (2009)). For the visual presentation of process models, one can also base the guidelines on work, e.g., in cognitive psychology and cartography, based on the fact that the models are meant to be useful in connection with communication between people. Going back to Shannon and Weaver (1963), communication entails both encoding by the sender and decoding by the receiver. Encoding has been discussed in detail, e.g., in the work of Bertin (1983). According to Bertin (1983), there are 4 different effects of encoding:

1. Association: The marks can be perceived as similar.
2. Selection: The marks can be perceived as different.
3. Order: The marks can be perceived as ordered.
4. Quantity: The marks can be perceived as proportional.

Eight different variables for conveying one or more of these meanings in a model are as follows:
- Planar variables: the horizontal position and the vertical position.
- Retinal variables: shape (association and selection), size (selection, order, and quantity), color (association and selection), brightness (value) (selection and order), orientation (association), and texture (association, selection, and order).

Rules for color usage are also useful in connection with evaluating diagrammatical models (if different colors are used). Approximately 10% of the male population and 1% of the female population suffer from some form of color vision deficiency (Ware et al. 2000); thus, many modeling notations (e.g., UML) explicitly avoid the use of colors as a part of the notation for conveying meaning. However, many modeling tools may give the modeler freedom to assign any color to the background, symbols, and labels and to the icon/shape used to represent the concept. Color is an important differentiator in other visual representations that is meant to be widely used (e.g., maps, see Bertin (1983)). Shneiderman et al. (2009) has listed a number of guidelines for the usage of color in visual displays in general.

- Use color conservatively.
- Limit the number of colors used. Many design guidelines suggest limiting the number of colors in a display to four, with a maximum limit of seven colors. According to the opponent process theory (Ware et al. 2000), there are six elementary colors, and these colors are arranged perceptually as opponent pairs along three axes: black-white, red-green, and yellow-blue.
- Red attracts the eye more than other colors.
- Ensure that the use of colors supports the task, i.e., makes useful differentiations between different parts of the model.
- Have color coding appear with minimal user effort.
- Place the application of color coding under (guided) user control.
- Use color to help in formatting.
- Be consistent in color coding.
- Be aware of common expectations about color codes. This issue can be dependent on the local culture.
- Be aware of problems with color pairings. If saturated (pure) red and blue appear on a display simultaneously, it may be difficult for users to absorb the information. Similarly, other combinations will appear difficult, such as yellow on purple and magenta on green. Too little contrast is also a problem (yellow letters on a white background or brown letters on a black background). Note that this phenomenon may be different on different screens and projectors.
- Use color changes to indicate status changes.
- Use color in graphic displays to enable greater information density.
- When using color coding, take into account that the model may need to be presented or distributed in gray scale (e.g., when printed). Although there are techniques to also ensure differentiation when transferring to a black and white printout, normal printers do not necessarily use the best algorithms to do so (Alsam 2009).
Overall, it may be better to have the use of colors under the control of the modeling language design rather than allowing it to be up to the individual modeler. The same is partly the case for other usages of emphasis. The use of emphasis can also be in accordance with the relative importance of the statements in a given model. Factors that have an important impact on visual emphasis are as follows:

- Size (the big is more easily noticed than the small).
- Solidity (e.g., **bold** letters vs. ordinary letters, full lines vs. dotted lines, thick lines vs. thin lines, and filled boxes vs. non-filled boxes).
- Difference from the ordinary pattern (e.g., *slanted* letters will attract attention among a large number of ordinary letters).
- Foreground/background differences (if the background is white, things will be more easily noticed the darker they are).
- Change (blinking or moving symbols attract attention).
- Position (when looking at a diagram, people tend to start at its middle).
- Connectivity (objects that connect to many others (having a high degree) will attract more attention than objects with few connections).

For diagrammatical models (diagrams), layout modification is a meaning-preserving transformation that can improve the comprehensibility of a model. A layout modification is a spatially different arrangement of the elements in the diagrammatical representation of the model.

Graph aesthetics has a long tradition, and general lists of guidelines for graph aesthetics are presented in Battista et al. (1994), Tamassia et al. (1988). These guidelines, summarized below, can act as a starting point for automatic layout modification techniques and metrics to be calculated to support the manual improvement of the model layout. Note that a model that is optimal according to one of these aesthetics is not necessarily optimal for another.

- **ANGLE**: Angles between edges going out from the same node should not be too small.
- **AREA**: Minimize the area occupied by the drawing.
- **BALAN**: Balance the diagram with respect to the axis.
- **BENDS**: Minimize the number of bends along the edges.
- **CONVEX**: Maximize the number of faces drawn as convex polygons.
- **CROSS**: Minimize the number of crossings between the edges.
- **DEGREE**: Place nodes with high degree in the center of the drawing.
- **DIM**: Minimize differences among nodes’ dimensions (given nodes of the same type).
- **LENGTH**: Minimize the global length of the edges.
- **MAXCON**: Minimize the length of the longest edge.
- **SYMM**: Have symmetry of sons in hierarchies.
- **UNIDEN**: Have a uniform density of the nodes in the drawing.
- **VERT**: Have verticality of hierarchical structures. The implication is that in a tree/hierarchy, nodes at the same level in the tree are placed along a horizontal line with a minimum distance between them.
These guidelines overlap the guidelines for process model presentation described in Sect. 2.3.4. In addition, a few BPMN/process modeling-specific guidelines are included there:

- Consider the use of partitions, e.g., pools and swimlanes.
- Specify task types, especially user (human task) and service (automated task) tasks.

A number of metrics can be produced relatively easily based on these guidelines, e.g., the number of crossing edges divided by the total number of edges in a model or compared with the minimum possible number of crossings, provided that one does not duplicate symbols. Similar metrics can be devised for the other aesthetics and be used during modeling to assess the potential for improving empirical quality. Based on such metrics, one could easily assess that the quality of Fig. 3.5 can be argued to be less than that of Fig. 3.3, although it contains the same statements. In particular, we can observe a worsening of the aspects in relation to the ANGLE, AREA, BALAN, CROSS, DEGREE, DIM, LENGTH, MAXCON, SYMM, UNIDEN, and VERT guidelines.

However, we should remember that aesthetics is a subjective issue; thus, familiarity with a diagram is oftentimes just as important for comprehension. As noted by Petre (1995), one of the main advantages of diagrammatic modeling languages appears to be the possibility of using the so-called secondary notation, i.e., the use of layout and perceptual cues to improve the comprehension of the model. Thus, one
oftentimes needs to constrain automatic layout modifications. Although it would be more accurate to place such constraints as a mean for pragmatic quality (see below), we include them here because they are used in techniques for automatic graph layout that it is natural to cover as part of empirical quality. A list of constraints used in connection with automatic graph layout is presented in Table 3.1. Additionally, manual diagram layout mechanisms should be available in a modeling tool to quickly make an existing model visually pleasing (including horizontal and vertical alignments, minimizing the number of alignment points, and equal spacing when selecting more than 2 nodes). Tool functionality to make the size and font size of selected elements in the model equal is also useful.

Clearly, it should be easy to retain the aesthetically pleasing diagram when we have to update the model at a later point in time. Doing so includes the possibility of selecting and moving a group of nodes as one, the moving of complete subtrees as one in a hierarchical model, rerouting connections when changing the relative position of two interconnected nodes, and tool functionality, such as snap to grid. In advanced modeling tools, one finds all of these mechanisms. In tools such as Troux Architect, one can also define layout strategies so that one automatically keeps, e.g., a collection of elements in a matrix format when adding new elements to the model.

Finally, there are structures that one has found beneficial in general (e.g., limiting the number of tasks on a decomposition level in a DFD).

A number of such guidelines were presented for BPMN models in Sect. 2.3.4, in relation to both the number of elements and the composition of the elements. Most of these are style guidelines (Silver 2012), whereas some of these point to potential violations of the syntax rules of BPMN and thus are more correctly positioned as part of syntactic quality. To provide a complete overview of the quality of business process models in this chapter, we thus repeat here those aspects relevant for process modeling.

**Number of elements**

1. The model contains a high number of elements (i.e., gateways, activities, and events) → Decompose models with more than 31 elements.

<table>
<thead>
<tr>
<th>Aspects</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTRE</td>
<td>Placing a set of given nodes in the center of the drawing</td>
</tr>
<tr>
<td>DIMENSION</td>
<td>Assigning the dimensions of symbols</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>Placing specified nodes on the external boundary of the drawing</td>
</tr>
<tr>
<td>NEIGH</td>
<td>Placing a group of nodes close together</td>
</tr>
<tr>
<td>SHAPE</td>
<td>Drawing a subgraph of selected nodes with a predefined shape</td>
</tr>
<tr>
<td>STREAM</td>
<td>Placing a sequences of nodes along a straight line (a specialization of SHAPE)</td>
</tr>
</tbody>
</table>

| Table 3.1 A taxonomy of constraints for graph layouts |
2. The model contains duplicate elements (e.g., identical start events, identical end events, and identical activities) or fragments, capturing the same control flow logic \(\rightarrow\) Avoid duplicate elements and fragments in the process models.

3. Models contain unnecessary elements (e.g., one empty arc between an AND split and an AND join) \(\rightarrow\) Avoid unnecessary elements.

4. A high number of events \(\rightarrow\) Avoid models with more than 7 events.

5. The model contains multiple start/end events \(\rightarrow\) Use no more than two start/end events in the top process level, use one start event in the subprocesses, and use two end events to distinguish success and fail states in subprocesses.

6. Do not omit start and end events \(\rightarrow\) Have at least one start and one end event.

7. A high number of intermediate events \(\rightarrow\) Avoid high numbers of intermediate events in the process model.

8. The model contains too many arcs \(\rightarrow\) Avoid models with more than 34 arcs.

9. The model contains too many gateways \(\rightarrow\) Avoid models with more than 12 gateways.

10. A high number of activities \(\rightarrow\) Minimize the number of activities.

11. A high number of routing paths per gateway \(\rightarrow\) Use no more than 3 routing paths per gateway.

12. Split/join gateways have more than one incoming and outgoing flows (i.e., two behaviors in the same gateway) \(\rightarrow\) Do not combine multiple inputs and multiple outputs in the same gateway.

13. There are too many outgoing sequence flows from an event \(\rightarrow\) Do not use more than 4 outgoing sequence flows from events.

**Composition of components**

1. The model has deeply nested structured blocks \(\rightarrow\) Avoid deeply nested structured blocks.

2. The model contains multiple cycles \(\rightarrow\) Avoid cycles in the process models if possible, especially unstructured cycles (i.e., cycles with multiple exit points).

3. Badly formed cycles: The backward connection of a loop construct does not begin in an XOR split or does not lead back to an XOR join \(\rightarrow\) When modeling cycles, the backward connection should begin in an XOR split and lead back to an XOR join.

4. Multiple exit points per cycle \(\rightarrow\) Avoid multiple exit points per cycle.

5. A high level of parallelism (the sum of the output degrees of AND and XOR gateways should be 8 at most) \(\rightarrow\) Avoid a high level of parallelism in the process models.

6. Bad parallelism: Parallel paths do not reach end events or do not synchronize \(\rightarrow\) Each parallel path must reach an end event or must be synchronized.

7. A high level of unstructuredness \(\rightarrow\) Every split gateway should match a respective join gateway of the same type.

8. The model contains a long path from the start node to the end node \(\rightarrow\) Keep the path from a start node to the end node as short as possible.
9 High gateway diversity → Minimize gateway diversity (i.e., avoid having a mix of “XOR,” “OR,” “AND,” and complex gateways in the same model.
10 The existence of inclusive OR gateways in the process models → Avoid the use of inclusive OR gateways.
11 High complexity in the model → Select the less complex alternative when modeling.
12 The model lacks modularity → There should be no more than 31 nodes in a diagram (cf. guideline 1) and no less than 5 activities in a subprocess.

Techniques to keep the model in an adequate structure as the model develops are called model refactoring. In programming, refactoring is “the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure” (Fowler and Beck 1999). Similar information-preserving model transformations can be defined for process models. Refactoring techniques are oftentimes specific to a modeling perspective or modeling language (as the example with the number of tasks on a given decomposition level above for functional models).

Refactoring will often support expressive economy and, as a result, indirectly improve many of the graph layout metrics. Many of the above guidelines could act as a basis for suggestions for refactoring. A comprehensive overview of model refactoring strategies is found in Conesa et al. (2011).

3.4 Syntactic Quality of Business Process Models

Syntactic quality is the correspondence between the model $M$ and the language extension $L$ of the language in which the model is written. Referring to the discussion on meta-levels in Sect. 1.4.1, $L$ is constrained by a model on a higher meta-level (the language model).

There is only one syntactic quality characteristic, syntactical correctness, which means that all of the statements in the model are according to the syntax and vocabulary of the language, i.e.,

$$M \models L = \emptyset,$$

which is illustrated in Fig. 3.6.

There are two types of syntax errors:

- **Syntactic invalidity**, in which words or graphemes that are not part of the language are used. An example of syntactic invalidity is given in Fig. 3.7, where an actor symbol (triangle with the label JK) that is not part of the chosen language (BPMN) is used.
- **Syntactic incompleteness**, in which the model lacks constructs or information to obey the language’s grammar. An example of syntactic incompleteness is given in Fig. 3.8, where the graph is not connected.
The degree of syntactic quality can be measured as one minus the rate of erroneous statements, i.e.,

$$\text{Syntactic quality} = 1 - \frac{#M_E \setminus L + M_{\text{missing}}}{#M_E},$$

where $M_{\text{missing}}$ is the number of statements that would be necessary to make the model syntactically complete (in Fig. 3.8 $M_{\text{missing}} = 2$).

To ensure the syntactic quality of the model, syntax checks should be provided as an integral part of the modeling support of a modeling tool, supported in either the modeling tools or the modeling techniques applied. The checks may be performed along two main directions:

- Error prevention: This type of check adapts the principles of syntax-directed editors. Thus, only the modeling constructs defined in the language’s vocabulary are available through the modeling tool. This includes having modeling palettes (the concepts available to choose for modeling) that are limited to the
allowed concepts of the language or sublanguage used in the given diagram or
decomposition and limitations on the types of relationships that can be estab-
lished between two or more concepts. Additionally, when a modeler violates a
syntax rule of the language, the modeling session should be temporarily inter-
rupted to restore a legal model. This type of check is controlled by the tool.

- Error detection: During a modeling session, some syntactical errors, particularly
ersors related to syntactic incompleteness, should be allowed on a temporary
basis. For instance, although BPMN requires that all activities must eventually
be linked to a sequence flow, it is difficult and/or inconvenient to draw an
activity and a flow simultaneously. In this case, syntactical completeness must
be checked at the user’s request. One may also imagine cases where one wants
to allow syntactic invalidity for a moment to be able to capture insights during a
modeling session that are at odds with the chosen language. An example of this
situation will be presented in the next chapter. Thus, in contrast to implicit
checks, where the tool “forces” the user to follow the language syntax, explicit
checks can only detect and report on existing errors. The user has to make the
corrections.

By distinguishing between these types of syntax checks, modeling freedom can
be encouraged. Throughout the modeling process, the tool will accept some syn-
tactical errors, but these can be detected at the user’s request. The developer is free to
construct the model to achieve semantic and pragmatic quality, as discussed below,
unless the syntax rules are directly violated. Although error-free models are the
ultimate goal of model quality assurance, it can be advantageous to have some errors
early in model development. Placing too much focus on syntactic quality at an early
stage may hamper the creativity of the modeling process. This idea is summed up in
what was originally termed “the Heisenberg Uncertainty Principle of CASE”
(Hewett and Durham 1989): “High levels of inconsistency and incompleteness are
permissible if they are confined to a small region of space and time.”

A third syntactic means is error correction. Error correction, i.e., replacing a
detected error with a correct statement, is more difficult to automate. When
implemented, it typically works as a typical spell-checker found in a word pro-
cessor, giving suggestions for the correct modeling structure or a term found in
some controlled vocabulary or ontology as discussed, e.g., in Lin et al. (2006), but
leaving it up to the modeler to perform the actual change.

All of the syntactic means are only meaningful to provide if the languages used
have a well-defined syntax. There are several ways to describe the syntax of lan-
guages for conceptual modeling, as described, e.g., in Krogstie (2012a), and we
refer to this for further details.

A model consists of both graphics and text. Although the diagrammatical aspects
are most focused in connection with conceptual models, there are also relevant
aspects linked to the textual parts of the models, which we will return to in one of
the cases reported in Chap. 4. The text can be in the form of labels (naming the
concepts) and longer descriptions. With respect to labels, for a long period of time,
there have existed simple guidelines for the labeling of particular types of concepts.
For example, in process modeling (returning to the definition of DFDs (see Gane and Sarson (1979))), there is the guideline that one should name a process in an “active verb, noun” manner (i.e., “register participant”). One reason for doing so is, clearly, to ensure that the process is actually named as a process (and not as an organizational unit, for instance). As reported in Mendling and Recker (2008), this practice also has a positive effect on the comprehension of the model. In addition, Hawryszkiewycz (2001) provides more detailed guidelines for naming processes, data stores, and flows in DFDs. Detailed metrics for labeling are very much dependent on the type of modeling language used (e.g., process modeling) and, in certain cases, for concrete modeling languages (e.g., DFD or BPMN). A general metric that can be specialized is the proportion of concepts that are labeled in a manner that does not conform to the guidelines for the labeling of the concrete concept. Oftentimes, simple linguistic techniques can be utilized to determine these labels automatically, although it is normally not supported in standard modeling tools. Labeling guidelines for BPMN models are found in Sect. 2.3.4.2:

- Labels do not follow the verb–object style.
- Labels are too long.
- The pools label is different from the process name.
- Timer events are not labeled with the duration or date/time parameter.
- Gateways are not labeled.
- A black box pool is not labeled with the participant’s name.
- There are constructions other than send/receive task types that are labeled as send or receive.

Figure 3.9 provides an example of poor labeling of our standard example model. An example of more concrete syntax rules for a specific process model notation is found in Sect. 4.2.

3.5 Semantic and Perceived Semantic Quality of Business Process Models

Semantic quality was originally defined as the correspondence between the model and the modeling domain (Lindland et al. 1994).

The framework contains two primary semantic quality characteristics: validity and completeness.

- Validity means that all of the statements made in the model are regarded as correct and relevant for the problem, i.e.,

\[ M \setminus D = \emptyset \]
A definition for the degree of validity could be:

\[
\text{validity} = 1 - \frac{(\#(M \setminus D))}{\#M}
\]

However, how useful such a metric may be can be called into question because it cannot typically be measured automatically due to the intractability of the domain. An example of invalidity is given in Fig. 3.10 because we believe that most persons would agree that the first task (Go party) is invalid (in the sense that it is not a relevant task to perform in connection with developing a scientific paper).

- Completeness means that the model contains all of the statements that would be correct and relevant about the domain, i.e.,

\[
D \setminus M = \emptyset
\]

Similarly, a definition for the degree of completeness could be:

\[
\text{completeness} = 1 - \frac{(\#(D \setminus M))}{\#D}
\]

Completeness would only be interesting in well-defined and limited domains, for instance temporarily deciding on a model of a new CIS. Then, one would like to view all of the statements in the model as also being part of the implemented CIS. However, \( D \) is not completely represented in the previous model in this case; thus, validity here is also more relevant. To summarize, a useful model will most likely still be a subset of the domain, as illustrated in Fig. 3.11. An important question, often answered by clarifying the goal of the modeling and the other modeling constraints as we discuss as part of deontic quality, asks what parts of the domain to leave out to avoid analysis paralysis. This issue is further discussed below under deontic quality.
An example of incompleteness can be the original as shown in Fig. 3.3, which is missing, e.g., an indication of who is performing the different tasks.

The primary goal of semantic quality is a correspondence between the externalized model and the domain, but this correspondence can be neither established nor checked directly: To build the model, one must go through the participants’ knowledge \((K)\) regarding the domain, and to check the model, one must compare this knowledge with the participants’ interpretation \((I)\) of the externalized model. Hence, what we observe at quality control is not the objective semantic quality of the model but what we term perceived semantic quality based on comparisons of the current knowledge with the current interpretation of the model.

Perceived validity and completeness, related to the individual performing validation, can be expressed as follows:

- **Perceived validity** of the model externalization: 
  \[
  I \setminus K_i = \emptyset
  \]

- **Perceived completeness** of the model externalization: 
  \[
  K_i \setminus I_i = \emptyset
  \]

These error classes are illustrated in Fig. 3.12. The metrics for the degree of perceived validity and completeness can be defined by means of the cardinalities in the same way as for semantic quality. Thus, we can define perceived validity in the following manner:

\[
\text{Perceived validity} = 1 - \frac{(\#(I_i \setminus K_i))}{\#I_i}
\]

That is, it is the number of invalid statements interpreted, divided by the total number of statements interpreted by the actor \(A_i\), and similarly, one can establish a sum over all of the participants to obtain an overall number. An example of a model with a perceived invalid statement is the example in Fig. 3.10, where I (the author of this book playing the role of the actor \(A_i\)), in the subjective role of an end user, may claim that the “Go party” task is not part of the paper-writing process.

Perceived completeness can be defined in the following manner:

\[
\text{Perceived completeness} = 1 - \frac{(\#(K_i \setminus I_i))}{\#K_i}
\]

That is, it is the number of statements regarded to be relevant but not seen in the model divided by the total number of relevant knowledge statements known by the actor \(A_i\), and similarly, a sum over all of the participants can be established to obtain an overall number. As in semantic quality, I (in the subjective role of an end user of a conference system) miss a number of tasks (such as “write paper” in Fig. 3.10).

Atomic modeling activities for establishing higher semantic quality are statement insertion and deletion. Conceptually, an update can be viewed as a deletion.
followed by an insertion. Statement insertions and deletions can also clearly result in lower semantic quality. Statement insertion and deletion can generally be viewed as meaning-updating transformations, which can be performed either manually or automatically.

Of specific importance is model reuse (which is a specific type of statement insertion), particularly the reuse of reference models (see Sect. 2.3.5). Either it can be the reuse of a previous model of a similar domain (e.g., a reference process model of the domain), or it may be a translation of a previous baseline model.

Consistency checking is another activity on the semantic level. Note that consistency can be argued to be subsumed by the combination of validity and completeness because an inconsistency must be caused by at least one invalid statement or the lack of a statement to sort out the inconsistency. To be able to perform consistency checking, the model must be made in a formal, preferably logical, language, and to enable and assess the impact of updates, the model should be modifiable. Properties such as structure, the locality of changes, and control over redundancy are included. Consistency checking can be viewed as one of the several types of model testing that are beneficial at this level. For process models, in particular, to ensure the possibility of consistency checking, two of the guidelines in 7PMG described in Sect. 2.3.3 are specifically important

- G3: Use one start event and one end event.
- G5: Avoid OR routing elements.

We noted that these were given low priority in the 7PMG report. However, if consistency and formal analysis (or the simulation vs. analysis of throughput (Kuntz et al. 1998)) are important, then one may want to prioritize these higher. There are two main approaches for formal specifications as the basis for consistency verification:

1. The algebraic specification approach specifies a system as a set of abstract data types (ADTs). The theory of an ADT consists of a set of symbols (sorts and operators, the signature) and a collection of formulae (the axioms of the theory); the interpretation of the theory is a multisorted algebra. The specification is a set of theories and the relationships among them.
2. On the other hand, the logical theory approach treats the complete model (including both structural and behavioral parts) as a logical theory.

Of particular interest in models with decomposition possibilities as is found in most process modeling languages is testing for constructivity. The notion of constructivity was introduced into the field of information systems by Langeforss
(1973). It entails that one can derive the properties of a system based on the properties of the subsystem and check whether the derived properties are the same as those previously specified for the system (if any). Thus, constructivity is necessary when we want to check the consistency of a hierarchical model, i.e., to check whether decompositions are correct. Krogstie and Solvberg (2003) provide an in-depth description of techniques for constructivity checking, which, incidentally, also depend on G5 (avoiding OR gates).

A wide range of conceptual modeling techniques begin by verbalizing cases. The verbalizations resulting from this step are then used for the development of the first version of the model. A technique for the further elaboration of the model is the use of driving questions based on the already existing model as used, e.g., in Tempora (Wangler et al. 1993) and 4EM (Sandkuhl et al. 2014). Driving questions can be both intralanguage and interlanguage. A simple example of an interlanguage driving question is ensuring that what is depicted in a store in a dataflow diagram is also found in the accompanying data model. More concrete guidelines for this type of technique will be linked to particular modeling languages and the combination of modeling languages.

In the area of semantic quality, general automated tools are difficult to develop for the simple reason that the domain and audience are beyond automatic manipulation as discussed above. The means of achieving a high perceived validity and completeness are similar to those for traditional validity and completeness, with the addition of participant training (in the modeling language and the domain). As we understand it, the actual checking of perceived semantic quality involves the view of the participants and cannot be totally automated, which is also why we place semantic quality into the social and not the technical realm in the discussion in Sect. 1.4.

By using a formal language, one can in a sense translate some semantic problems into syntactic problems, but doing so sets additional requirements to the domain appropriateness of the language used. In many cases, the modeling language chosen is not appropriate for representing the knowledge on the domain, thus making it very difficult to achieve semantic completeness. One important activity for addressing this issue is the adaptation of the meta-model of the modeling language used to suit the domain, not only by adding concepts but also by removing concepts (temporarily) from the language if they are not relevant for the modeling of the particular domain. This activity is treated in more detail in the case studies presented in Chap. 4. Not only domain appropriateness, modeler appropriateness, participant appropriateness, and organizational appropriateness are relevant dimensions in connection with this type of language development, but also comprehensibility appropriateness, tool appropriateness, and ontological appropriateness come into play as we will see in the cases.

When working on process improvements, one compares to an improved domain; thus, in relation to the validity of the model, one can devise a number of guidelines, as described in Sect. 2.3.1. We will not repeat all of these here but note only that the different guidelines/heuristics focus on improving some of the characteristics of the
process outcome. Which characteristic(s) to optimize is something that is treated at the level of deontic quality, underlining the important ways in which deontic quality (the goals of modeling) directly influences the semantic quality measures.

3.6 Pragmatic Quality of Business Process Models

As we define it, pragmatic quality concerns the comprehension of the model by participants. Two aspects can be distinguished:

- The interpretation of the model by human stakeholders is correct in relation to what is meant to be expressed in the model. Note that a model can be said to formally mean anything only if the syntax and semantics of the language used are intersubjectively agreed upon and are at least semiformal, but preferably formal (with an operational semantics); thus, one can trace or execute the model and experience the dynamic behavior of the process model. In addition, it will often be useful to have different metadata of the process models represented (e.g., who has made the model and when it was made). In particular, it can be useful to have the intention of making the model explicitly represented (because a model created to achieve one goal oftentimes may have little value in achieving a different goal). This issue is also important to take into account model reuse (e.g., of reference models), as described briefly in Sect. 3.5 on semantic quality.
- The tool interpretation is correct in relation to what is meant to be expressed in the model, making it possible to manipulate the model with tools, e.g., for code generation or process simulation.

Starting with the human comprehension component, pragmatic quality on this level is the correspondence between the process model and the audience’s interpretation of it. Not even the most comprehensive model would be of any use if no one was able to understand it. Moreover, not only that the process model has been understood but also who has understood (the relevant parts of) it are important.

Individual comprehension is defined to mean that the individual actor $A_i$ understands the part of the model available to that actor, i.e., $I_i = A^i$.

The corresponding error class is erroneous comprehension, meaning that the above formula does not hold. As illustrated in Fig. 3.13, it actually covers two aspects:

- Part of the model is not understood.
- The model is interpreted to contain statements that are not there.

It is important to observe that the pragmatic goal is stated as comprehension, i.e., that the model has been understood, not as comprehensibility, i.e., the model’s ability to be understood (which is what we treat under empirical quality in Sect. 3.3). There are several reasons for doing so. First, the ultimate goal is that the
process model is understood, not that it is understandable per se, although this characteristic can be an important means of achieving comprehension. Moreover, comprehension is very dependent on the process by which the process model is developed, the manner by which the participants communicate with each other and the various types of tool support.

From the technical actors’ perspective, that a model is understood means that all of the statements that are relevant to the technical actors to be able to perform, e.g., code generation or simulation, are comprehended by the relevant technical actors (modeling or development tools). In this sense, formality can be viewed as being a pragmatic means, and the formal syntax and formal (operational) semantics of the modeling language are the means of achieving pragmatic quality. This aspect illustrates that pragmatic quality is dependent on the different actors involved. This issue also applies to social actors. Whereas some individuals are familiar with formal languages from the outset and a formal model will also be best for them for comprehension, other people will find a mix of formal and informal statements to be more comprehensive, even if the set of statements in the complete model is redundant. Another important aspect is that familiarity with the domain will make a model of the domain easier to understand even if it is less complete than what is needed for a novice in the domain.

A process model can be difficult to comprehend due to the formality or unfamiliarity of the modeling language used, the complexity or size of the model, or the effort needed to deduce its important properties. A process modeling environment may make use of certain techniques to enhance user comprehension. Examining the linguistic aspects of process modeling, we can describe such strategies along the following four dimensions:

- **Language perception** concerns the user’s ability to understand the concepts of the process modeling language.
- **Content relevance** indicates the possibilities of distinguishing between irrelevant and relevant model properties so that at any point in time, one is able to focus on only the relevant parts based on previous domain knowledge.
- **Structured analysis** depends on the environment’s abilities to analyze and reveal the structural properties of the model.
- **Behavior experience** is related to the process model execution facilities offered.

Some of the activities to achieve pragmatic quality are as follows:

- **Participant training**: One should educate the audience in the syntax and semantics of the modeling languages used or on relevant aspects of the domain when needed.
Model inspection and walk-throughs: One should manually read a model, go through it in an orderly manner, and potentially explain it to others. Having stakeholders who have not developed the model themselves but who need to understand the model go through it and explain what it represents aloud is often a very good method of testing the current comprehension of the model. Other techniques from code inspections have been adapted to (requirements) model inspection (van Lamsweerde 2009). Useful support for this in a modeling tool supports navigation and the browsing of the model. This support also includes the possibility of scrolling through the model, either incrementally (pan) or one page at a time (page), and zooming.

Model transformations. Generally, these are performed to transform a model into another model in the same language. This activity can generally be expressed as follows:

\[ T : M_{L_1} \rightarrow M_{L_2} \]

The need to transform models arises for several reasons. First, models may be transformed to improve efficiency. In several approaches, an initial operational specification gradually evolves into the final implementation by a continuous replacement of real-world modeling constructs with more efficient constructs from the programming world. Second, models or programs may achieve improved readability through the use of transformations. This issue is discussed under layout modifications in Sect. 3.3 on empirical quality. As a final example, models may need to be changed if the modeling language evolves.

Model rephrasing is a meaning-preserving transformation where some of the implicit statements of the model are made explicit. It is related to model refactoring, discussed under empirical quality above.

Model filtering is a meaning-removing transformation that concentrates on and illuminates specific parts of a model. Filtering has been defined in Seltveit (1993) based on the notion of a views \( V \), which is a model that contains a subset of the statements of another model in the same language, i.e., \( V \) is a subset of \( M \), as illustrated in Fig. 3.14.

Another method of specifying a filter is to say that it is a set of not necessarily syntactically complete deletes of statements. Filters can be classified into two groups:

- Language/meta-model filters: Suppress details with respect to graphemes and symbols in the modeling language. An example is illustrated in Fig. 3.15, where we show only activities and the sequence flow between them (including also collapsing the gateway).
- Model/specification filters: Suppress details with respect to a particular model. An example is given in Fig. 3.16, where only the nodes that can be reached from the gateway are included.
One could also imagine specifically selected parts of the model. In process modeling, it may often be needed, for instance, to pick out the preferred sequence, the so-called happy path (see Fig. 3.17).

Other relevant aspects of filters include the following:

- **Inclusiveness/exclusiveness:** A filter can be defined by specifying the components to be included in the view or by specifying the components to be excluded in the view. These are referred to as the inclusiveness and exclusiveness properties of the viewspecs, respectively.

- **Query:** A filter can be defined as a query after elements that have certain attributes or attribute values.

- **Determinism/non-determinism:** A filter is deterministic if the resulting views of performing the filter on a model $M$ are the same each time, given that it operates on $M$ each time. If the result is not predictable, the filter is non-deterministic.
• Global/local effects: We distinguish between two cases: (1) The scope of the effects is local if there is no effect of the filter beyond the model upon which it operates and (2) it is global if the scope of the effects goes beyond the model upon which it operates. How to propagate changes to affected models is a challenging problem for filters with global effects.

Tools for addressing large models (e.g., enterprise architecture models) such as Troux Architect have good facilities for creating views of models, with different types of filters, where it is also possible to update the views, propagating changed values back to the main model.

• Model translation: A translation can generally be described as a mapping from a model in one language to a model containing all or some of the same statements in another language:

\[ T : M_{L_i} \rightarrow M_{L_j}, \ i \neq j \]

In paraphrasing, both \( L_i \) and \( L_j \) are textual languages. Often, this term is used more generally. In visualization, \( L_i \) is a textual language, whereas \( L_j \) is a diagrammatical language.

Translations between different diagrammatical languages can also be useful for comprehension in case different persons are fluent in different related languages. For example, for those who are familiar with UML activity diagrams, it may be easier to relate to a process model in this language than in a semantically equal BPMN model.

Finally, one may want to translate a diagrammatical model into a textual language, for instance to translate a BPMN model into BPEL so that the resulting model can be executed, or into a natural language for linearizing the model for improved comprehensibility. In this case, \( L_i \) is diagrammatical, and \( L_j \) is textual.

Although most translations and transformations will be easier and faster to perform when there is tool support, they can also be performed manually. Manual translations and transformations can also be used as part of participant training. However, participant training may also be enhanced by using tool support. Several specific applications of translations and transformations and combinations of these exist. Some examples are the following:

• Model execution: Translating or transforming the model into a model in an executable language, e.g., in a workflow environment. When manually performing this translation, we speak in terms of prototyping in the traditional sense.
• Animation: Making system dynamics explicit by using moving pictures. This may take the form of icons, such as a telephone ringing or a customer arriving at a registration desk, or it may apply the symbols of the modeling language. Recently, techniques applying virtual reality environments for enacting process models have been illustrated as a validation technique (Brown 2010).
• Explanation generation (Gulla 1996): This practice can be manual or tool-supported. An explanation generator can answer questions about a process model and its behavior.
• Simulation: Using statistical assumptions about the domain, such as the arrival rate of customers and the distributions of processing times, to anticipate how a system built according to the model would behave if implemented. It is not practical without tool support for large models. Simulation can be combined with execution, animation, and explanation.

The properties that a model and the languages it is made in must have to support these techniques include those for syntactic and semantic quality, in addition to executability (i.e., the execution of the model has to be efficient). The 7MPG Guideline G3 (only one start node and one end node) may also be relevant, making it possible to execute the process model. Other beneficial characteristics are expressive economy and aesthetics (empirical quality), as noted above.

3.7 Social Quality of Business Process Models

The goal defined for social quality is agreement. Six types of agreement can be identified according to the following dimensions:

• Agreement in knowledge versus agreement in model interpretation. In the case where two models are made based on the view of two different actors, we can also talk in terms of agreement in model.
• Relative agreement versus absolute agreement.

Relative agreement means that the various sets to be compared are consistent; hence, there may be many statements in the model of one actor that are not present in that of another, provided that they do not contradict each other. Absolute agreement, on the other hand, means that all of the models are the same.

Agreement in model interpretation will typically be a more limited demand than agreement in knowledge because the former means that the actors agree on what is stated in the model, whereas there may still be much they disagree on that is not stated in the model so far, even if it may be regarded as relevant by one or both participants for the current modeling task. The agreement of models will be easier to check in practice, especially if the languages have formal syntax or semantics, although this is limited to the situation described above. Hence, we can define the following:

• Relative agreement in interpretation: All $I_i$ are consistent.
• Absolute agreement in interpretation: All $I_i$ are equal.
• Relative agreement in knowledge: All $K_i$ are consistent.
• Absolute agreement in knowledge: All $K_i$ are equal.
• Relative agreement in the model: All $M_i$ are consistent.
• Absolute agreement in the model: All $M_i$ are equal.

Metrics can be defined for the degree of agreement based on the number of inconsistent statements divided by the total number of statements perceived or by the number of non-corresponding statements divided by the total number of statements perceived.
Because different participants will have their expertise in different fields, relative agreement is regarded as being more useful than absolute agreement. In process modeling, one typically wants to link the different areas of the organization, where the different stakeholders primarily have expertise on a limited number of activities. However, the different actors must have the possibility to agree and disagree on something, i.e., the parts of the model that are relevant to them should overlap to some extent. This would in a process model at least amount to the interfaces between tasks, i.e., what is produced in one task that is to be consumed in another. Since process improvements often span the whole process, achieving a good overall process often necessitates that the different persons get an understanding of and agree to the whole process, not only their own parts.

The pragmatic goal of comprehension is viewed as a means of achieving social quality. The reason is that agreement without comprehension is not very useful, at least not when democratic ideals are held. The area of model monopoly (Bråten 1973) discussed in Chap. 1 is related to this aspect; thus, one should be aware of the dangers of particular modelers consciously or (more likely) unconsciously misleading other participants.

Tool support related to social quality is most easy to establish based on achieving agreement in models created according to the internal reality of the participants who are to agree. Figure 3.18 provides an example of model merging, where one merges the paper process viewed from the perspective of both the author and the organizer. Accordingly, the main activities for investigating and hopefully achieving agreement are the use of model integration techniques, with a specific emphasis on the conflict resolution of the integrated models.

The general model integration process has many similarities with view integration, which has been a topic of much research in the database community over the last 20+ years. The process can be considered to consist of four subprocesses (Francalanci and Pernici 1993).

- **Preintegration:** When more than two models are used as inputs in the process, one must decide on how many models should be considered at a time. A number of strategies have been developed, such as binary ladder integration, N-ary integration, balanced binary strategy, and mixed strategies. The strategy chosen will often depend on the organizational setting for the modeling project. For instance, in the case of participatory modeling in modeling conferences described in Chap. 5 (Gjersvik et al. 2004), one first integrates the 2 or 3 individual models in the first 4 workshops and then integrates all of the resulting 4 models into the final model.

- **Viewpoint comparison:** This includes identifying correspondences and detecting conflicts among the viewpoints. Some types of conflict that may be detected are as follows:
  - **Naming conflicts:** Problems based on the use of synonyms and homonyms. When using different labeling styles, problems of detecting equal concepts are even larger, which is another reason to standardize on one style (e.g., “active verb and noun”).
- Type conflicts: That the same statements are represented by different concepts in different models.
- Value conflicts: An attribute has different domains in two models.
- Constraint conflicts: Two models represent different constraints on the same phenomena.

- Viewpoint conforming: This aims at solving the previously detected conflicts. Representations of statements in two different models can be classified as follows: identical, equivalent, compatible, and inconsistent. To address such conflicts, either traditional approaches are mostly based on transformational equivalence, or they trust the skills of the participants by only providing examples that are valid for the particular model. According to Francalanci and Pernici (1993), few of the early approaches addressed inconsistent statements. An exception is Leite and Freeman (1991).

In this regard, other useful techniques are goal modeling, particularly where one can explicitly represent conflicting goals, such as in EEML (Krogstie 2008), and the use of argumentation systems (Conklin and Begeman 1988; Conklin 2005; Hahn et al. 1990) for supporting the argumentation process. These systems use the issue-based information system (IBIS) approach originally proposed by Rittel (1972) or extensions thereof. IBIS focuses on the articulation of the key issues in the problem area. Each issue may have many positions, which are statements or assertions that resolve the issue. Each of the issue’s positions in turn
may have one or more arguments that either support or oppose the position. Going from one node type to another is done through the so-called rhetorical moves.

- Merging and restructuring: The different models are merged into a joint model and then restructured. The latter involves checking the resulting model against criteria for empirical, (perceived) semantic, pragmatic, and social quality. It is taken for granted that syntactic quality issues are taken care of automatically in the process.

Generally, it is not to be expected that apart from syntactic matching, matching can be performed in a manner that is fully automatic, although several modeling tools have this type of functionality. Matching business process models are discussed in Dijkman et al. (2010), Rittgen (2011). Three aspects of model similarity are discussed: node matching, structural similarity, and behavioral similarity. Node matching attempts to map the nodes from one model to the nodes of the other model by comparing the labels, attributes, and types of nodes. Node matching can be affected with semantic or syntactic measures. The latter is based on the string-edit distance, i.e., the number of letters that need to be added, replaced, or deleted to transform the label of an activity in one model to that of an activity in the other model. This is clearly easier when all models use the same labeling style. Semantic matching is based on a database of synonyms or an ontology (Lin et al. 2006). Based on the node matching, the two models can be compared with the help of structural similarity or behavioral similarity. The former uses only structural information on the model, i.e., the manner in which activities are connected with “arrows,” but does not examine their meaning in terms of the control flow. Behavioral similarity examines the actual execution of the processes described by the models. Here, two models are considered equivalent if, at any time during process execution, an activity that can be performed in one process can also be performed in the other and vice versa. A weakness with these types of similarity measures is that they typically do not focus on which areas of the model where similarity is the most important.

One should also be able to address inconsistencies and variants, in the sense that not all need to follow the same process in all areas. If two models are inconsistent, it would be useful to generate some type of information on the differences. Are the parts that do not match important parts of the model? How difficult is it to change parts to make them consistent? Are the inconsistent parts automated or manual processes? After the analysis, the tool could suggest which processes or process parts to change. Model merging can be supported in several ways, having computerized support for manual integration, possibly with the use of CSCW techniques. This issue is discussed in more detail for models in general in (Krogstie 2012a), techniques that can be applied also for business process models.
3.8 Deontic Quality of Business Process Models

Modeling is (normally) not performed for the fun of it but to achieve some goal (termed the goals of modeling, \( G \)) that is typically linked to some business and organizational goals. This linkage introduces the need to examine both the costs and benefits of modeling. Here, the means are related to the modeling of these goals and checking their fulfillment (i.e., that all goals are achieved and that there are no goals that are not achieved).

For everything apart from extremely simple and highly intersubjectively agreed-upon domains, total validity, completeness, comprehension, and agreement, as described above, cannot be achieved. Hence, for the goals in these areas to be realistic, they must be somewhat relaxed by introducing the idea of feasibility. Attempts to reach a state of total validity, completeness, comprehension, and agreement will potentially lead to an unlimited use of time and money in the modeling activity. The time to terminate a modeling activity is thus not when the model is “perfect” (which will never occur) but when it has reached a state where further modeling is regarded as being less beneficial than applying the model in its current state. This resonates well with the concept of “just enough method” found in agile methods. Accordingly, a relaxed type of these human-related goals can be defined, which we term feasible validity, feasible completeness, feasible comprehension, and feasible agreement.

- **Feasible validity**: \( M \setminus D = R, R \neq \emptyset \), but there are no statements \( r \in R \) such that the benefit of performing a syntactically valid delete of \( r \) from \( M \) exceeds the drawback from eliminating the invalidity \( r \). (A syntactically valid delete is the deletion of statements from the model in a manner that does not introduce syntactic errors in the model.)
- **Feasible perceived validity**: \( I \setminus K = R, R \neq \emptyset \), but there are no statements \( r \in R \) such that the benefit of performing a syntactically valid delete of \( r \) from \( M \) exceeds the drawback from eliminating the invalidity \( r \).
- **Feasible completeness**: \( D \setminus M = S, S \neq \emptyset \), but there is no statement \( s \in S \) such that the benefit of inserting \( s \) in \( M \) in a syntactically complete manner exceeds the drawback from adding the statement \( s \).
- **Feasible perceived completeness**: \( K \setminus I = S, S \neq \emptyset \), but there is no statement \( s \in S \) such that the benefit of inserting \( s \) in \( M \) in a syntactically complete manner exceeds the drawback from adding the statement \( s \).
- **Feasible comprehension** means that although the model may not have been correctly understood by all audience members, there is no statement in the model such that the benefit of rooting out the misunderstanding related to a statement here exceeds the drawback from making that effort.
- **Feasible agreement** is achieved if feasible perceived semantic quality and feasible comprehension are achieved and inconsistencies are resolved by choosing one of the alternatives when the benefits of doing so are greater than the drawbacks from working out an agreement.
Thus, feasibility introduces a trade-off between the \textit{value} and the \textit{drawbacks} of achieving a given model quality. We have used the term “drawback” here instead of the more usual “cost” to indicate that the discussion is not necessarily restricted to purely economic issues. Judging completeness with respect to some intersubjectively agreed-upon standard, as suggested by Pohl (1993), is one approach to feasibility. Additionally, reference models can be used in this manner. We see also from Sect. 2.3.5 that similar value/drawback deliberations must be done on the use of reference models.

By making the standard a part of the language, one can also transform this inherently semantic problem into a syntactic problem to enable automatic checks for conformance to the standard. Note that this is only workable if the use of the standard is appropriate for achieving the goals of modeling in the first place. Another relevant issue on this level is how to decide the timeliness of model updates, as discussed in Sect. 3.2.

As we can observe from the variety of goals of modeling discussed in Sect. 1.5.1, in addition to changing the models, these may also be meant to change other aspects, including the following:

- The participants learn as a result of the modeling (i.e., $K$ is increased).
- If the modeling is intended to bring about change (e.g., a process improvement), then the domain $D$ is changed, preferably in a positive direction in relation to the goal of modeling.

### 3.9 Summary

In this chapter, we have described the main parts of a specialization of SEQUAL for process model quality. Inspired by earlier discussions on the quality of conceptual models and requirements specifications, combined with semiotic theory, process model quality has been divided into seven areas:

- Physical quality: The persistence, currency, and availability of the process model.
- Empirical quality: The relationship between the process model and another process model that contains the same statements, which are somehow regarded as better through a different arrangement or layout.
- Syntactic quality: The relationship between the process model and the process modeling language. Is the modeling language used correctly?
- Semantic quality: The relationship between the process model and the domain of the modeling. Perceived semantic quality is the parallel relationship between the knowledge of the participants and their interpretation of the process model. Is the process model complete (containing all valid statements) and valid (not containing invalid statements)?
• Pragmatic quality: The relationship between the process model and the stakeholder’s interpretation of the model. Does the audience understand the implications of the part of the process model relevant to it? Do process modeling tools, e.g., for model execution “understand” the model?
• Social quality: The relationship between different process model interpretations. Do the different participants of the modeling agree on the semantic quality of the process model?
• Deontic quality: How do the process models contribute to fulfilling the overall goals of modeling?

Although the same levels developed for the quality of models in general are also relevant for the quality of business process models and they share many of the same means for improving model quality, we have also identified a number of specific aspects that are related to the use of process modeling languages, the use of reference models, and process improvement.

References


Langefors, B.: Theoretical Analysis of Information Systems, 1st edn. Studentliteratur, Auerbach (1973)

