SEKI-PROJEKT

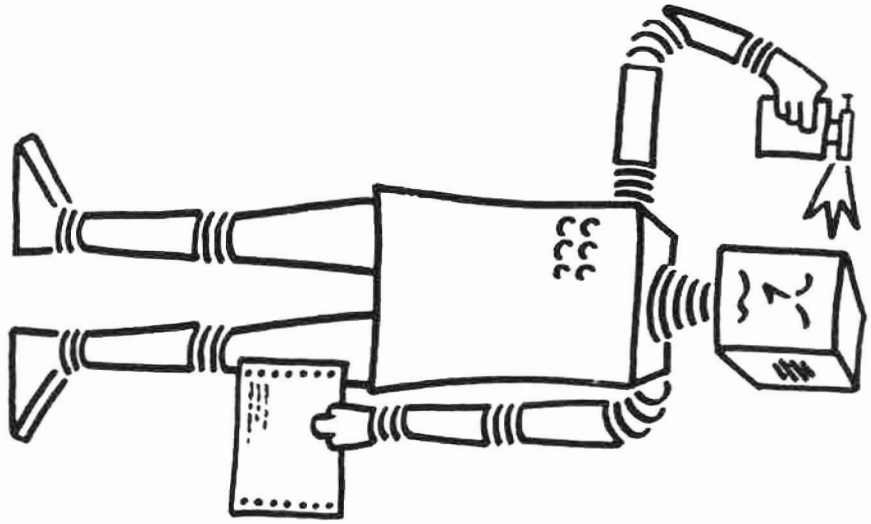
SEKI MEMO

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

FOUNDATION OF EXPERT SYSTEMS FOR CONCEPTIONAL DESIGN IN MECHANICAL ENGINEERING

Peter Raulefs

MEMO SEKI-84-08 Oktober 1984



CONTENTS

Abstract

A theory of qualitative reasoning on physico-technical systems is developed. Particular results include

- an approach to reasoning on time and causality combining state- and event-based approaches allowing concurrent time structures.
- the multiple view technique which allows to represent, reason about, and relate different views of time scales and other physical dimensions with respect to e.g. precision and granularity. Precise quantitative models appear as particular views, so that conventional mathematical reasoning can be incorporated in our approach.
- representing physico-technical systems in terms of objects with internal states, processes denoting consecutive and/or concurrent structures of state transitions (events), and meta-processes for expressing teleological considerations. This approach is summarized in terms of the qualitative reasoning language QRL which, in addition, provides various aggregation and abstraction mechanisms on objects and processes.
- an approach for applying non-monotonic reasoning techniques to the design of physico-technical systems.

These results provide a foundation towards mechanizing the construction process in mechanical engineering: QRL supports the phases of requirements definition and specification, as well as conceptional design in a way which is refinable to material design as supported by CAD-systems.

1. Introduction

The central methods for building expert systems differ for different fields of applications, allowing the following taxonomy of fields:

(1) INTERPRETATION SYSTEMS are aimed at interpreting findings, i.e. sensory or other observations about the state of a system to derive propositions about past, present, and future states of the system. Examples of such propositions are identifying faults and their causes (diagnosing), justifying why particular states or state sequences occur, and what future developments can be predicted.

(2) CONFIGURATION SYSTEM put together configurations of actions (planning) or objects to achieve a particular goal, such as a procedure transforming initial into goal states, or assembling parts to a machine which performs some given task.

(3) CONSTRUCTION SYSTEMS perform the task of devising methods to solve problems described in terms of intended functional behaviors. Such a method may be realized in terms of a program as well as a machine in mechanical engineering.

(4) TUTORING SYSTEMS help to acquire a particular body of knowledge which is to be incorporated into the thinking structures of the students.

The fact that much more rapid progress in developing expert systems has been achieved for interpretation rather than configuration and construction systems is apparently due to a simpler control structure of interpretation systems:

- Interpretations of findings are derived from chaining primarily local observations which associate findings and intermediate stages of interpretations with consequences. Such associations either express compiled knowledge, i.e. heuristic resp.

judgmental knowledge based on shallow models, or they are based on deep models in which functional and causal relationships are explicitly represented.

- Configuring and construction requires additional mechanisms on top of chaining local decisions: There is always some overall goal, or even hierarchies of overall goals so that local associations must be made with simultaneously pursuing ways for achieving the overall goals. If local and global decision making is in conflict, additional mechanisms such as backtracking, constraint propagation and reasoned revisions of beliefs are required.

The purpose of this paper is to contribute to the development of expert systems for mechanizing the process of construction in mechanical engineering. Present approaches towards a systematic construction process can be summarized in four successive phases [Hansen 74, Koller 71, Pahl/Beitz 76, Rodenacker 76, Roth 68, Yoshikawa 83]:

I. Problem clarification
II. Conceptional design
III. Detailed design
IV. Elaboration of the detailed design

Phases III and IV are currently supported by CAD/CAM-systems which mechanize the drawing board together with catalogues of pre-fabricated parts. Our work strives for providing mechanical support of phases I and II which incorporate the core of what is creative about construction, i.e. the invention of new technical mechanisms and the synthesis of such mechanisms for achieving a desired functional behavior. As far as it is known to us, there is no previous work in this area, except for attempts in mechanical engineering to systematize the construction process.

Conceptional design is exclusively concerned about the qualitative properties of technical systems and ignores quantitative aspects such as geometrical measures, material

properties, etc. Hence, progress in this area requires solutions for the following problems:

- Qualitative reasoning: How do we represent
  - the way a technical system is assembled from its parts,
  - qualitative properties of parts and the way such properties interact to forming properties of the entire system, and
  - the functioning of parts and their interaction towards the functioning of the whole?

How can we reason on such representations to derive, judge and explain properties detailing how and why a technical system functions? As such qualitative representations and reasoning form a basis towards mechanizing phases I and II of the construction process, it is essential that they can be refined into quantitative representations and reasoning as done in CAD/CAM-systems and mechanical engineering anyway.

- Non-monotonic reasoning (NMR). Conceptional design is very much a trial and error process. Approaches are pursued under assumptions which may turn out to be wrong as a result of elaborating them. Techniques of non-monotonic reasoning must therefore be extended to the representations and mechanisms of qualitative reasoning.

2. A Model of Construction and Conceptional Design

Construction in mechanical engineering is a task which has methodological similarities with program development in software engineering. Some of the standard textbooks in construction theory of mechanical engineering [Hansen 74, Koller 71, Pahl/Beitz 76, Roth 68] decompose the construction process into phase models which we very roughly summarize in three major phases:

A. Requirements definition and specification. The task to be solved is clarified in terms of functional, performance, material, and economical requirements. Except for structured charts and tables, no formal means (for mechanical processing) for expressing such requirements have been developed so far.

B. Conceptional design. This phase produces principle solutions exhibiting the required functional behavior qualitatively, i.e. without detailing geometrical measures, materials, etc. Different solution variants are evaluated, and decisions for selecting an optimal one are made.

C. Material design. The result of this phase is a detailed design containing all information required for a subsequent material realization. As this phase is a major working area for mechanical engineers, it is often elaborated into more detailed phases. The current widespread CAD-systems support this phase by transferring the drawing board to the computer.

It is interesting to note that in its short period of existence, software engineering has come up with much more sophisticated, precise, and mechanizable models of program development. For an objective to contribute towards a mechanization of construction in mechanical engineering, it turned out that concepts from

software engineering can be fruitfully applied. This is briefly explored in section 2.1. Section 2.2 then concentrates on the central issue of this paper and introduces a model for computer-aided conceptional design.

2.1. Construction in Mechanical and Software Engineering

We consider the software development cycle as indicated in Fig. 2-1 [Beierle et al. 83]. In contrast to traditional approaches, it has been the contribution of AI programming methodology to show that methods of "structured top-down design" are unfeasible for complex systems, and that program development should include all phases simultaneously, supported by appropriate programming environments. We therefore propose in [Raulefs 84] to realize the development cycle of Fig. 2-1 by a spiral of program development triads as indicated in Fig. 2-2.

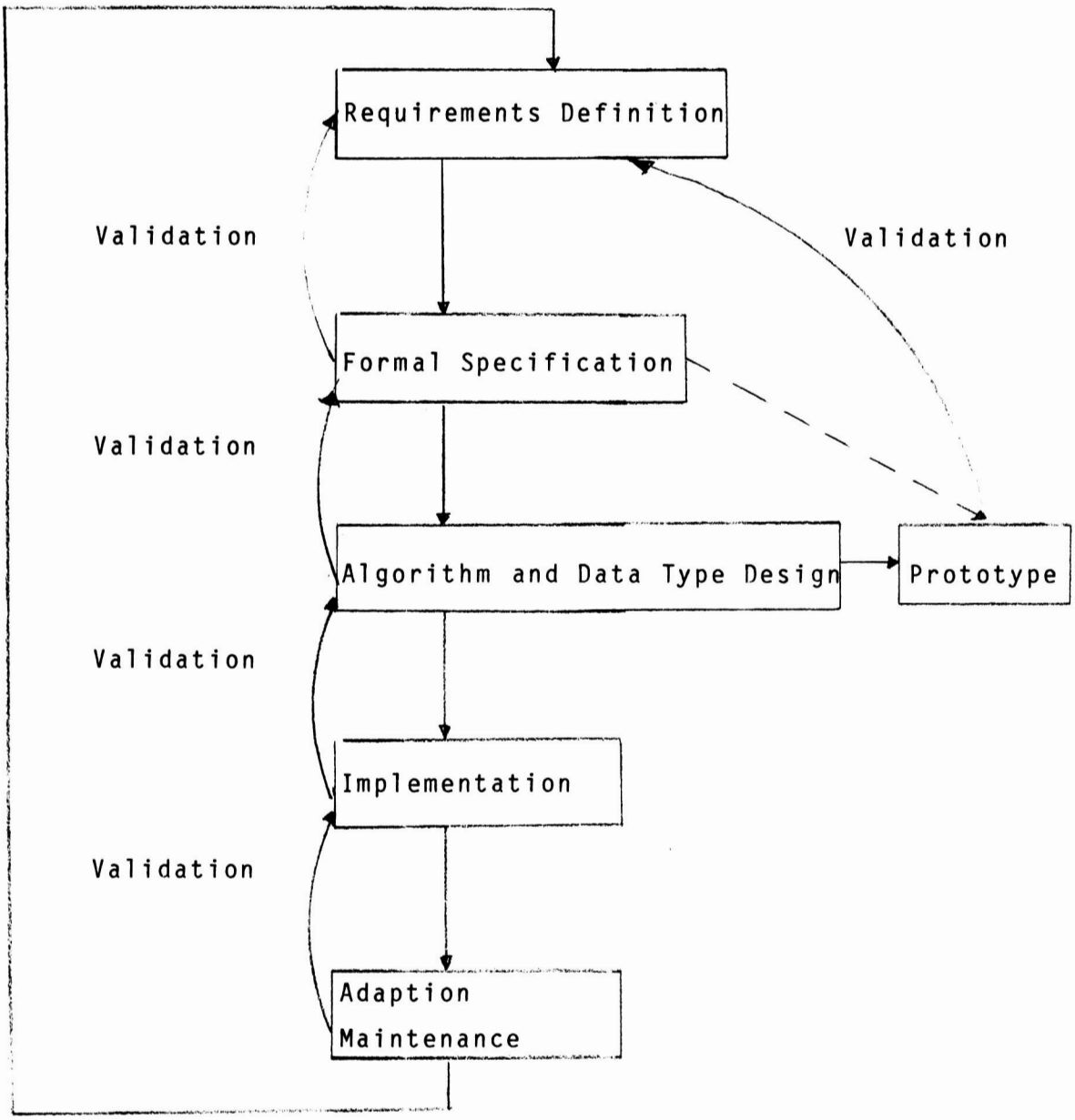


Fig. 2-1 Software Development Cycle

Methods exist for supporting the mechanization of all phases of the above development cycle [Beierle at al. 83], using both formal and knowledge-based techniques.

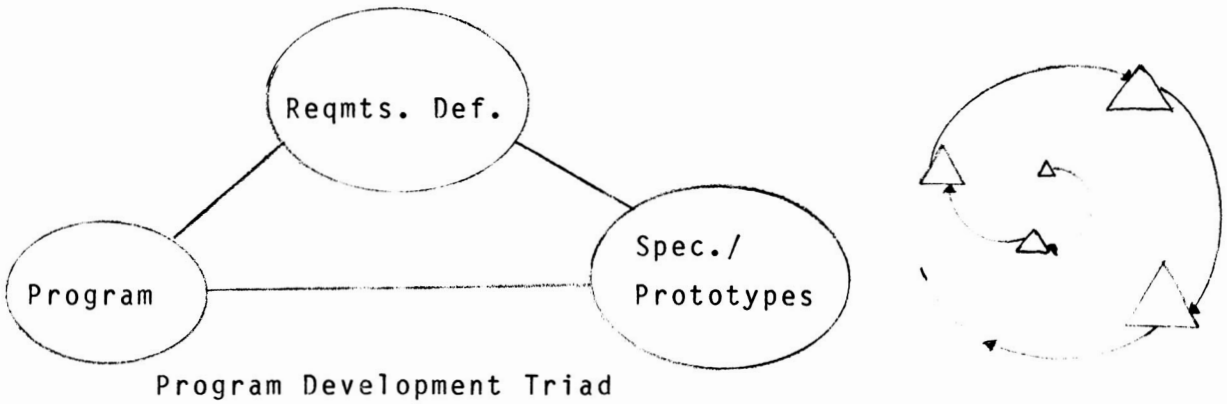


Fig. 2-2. Program Development Triad and Spiral

A crucial reason for the failure of the "structured top-down approaches" is the fact that the development itself leads to revisions in earlier stages. In particular, requirements and specifications usually fail to anticipate all what is expected from the program under construction, as this is often revealed when observing the behavior of an initial version under operating conditions. In other words, software development is an inherently non-monotonic process.

Similar observations apply to mechanical engineering. The construction process has the following analogies to software development:

- Requirements definition and specification is common to both areas.
- Algorithm and (abstract) data type design corresponds to conceptional design.
- Implementation corresponds to a combination of material design and manufacturing.

Apparently, there are in particular two areas where construction methodology in mechanical engineering could benefit from software construction methods:

(1) Requirements definition and specification: Mechanical engineering currently completely lacks mechanisms to formally represent the architecture, structure, function of components and their interaction, and properties of technical devices in an integrated framework so that both such concepts as well as relationships can be mechanically reasoned about. We claim and provide evidence that algebraic software specification languages provide excellent facilities for developing such representations.

(2) Conceptional design: A language for qualitative representations of architecture and function of technical devices together with mechanisms for reasoning about such representations would provide a facility to
- decompose a problem definition into subtasks, resulting in a functional model of the problem structure (subproblems and their functional relationships),
- represent and extract basic solution approaches from appropriate knowledge bases,
- integrate principle solutions of subtasks to conceptional variants (corresponds to module integration),
- improve variants by concretizations and optimizations,
- evaluate and decide on conceptional variants.

2.2. A Model of Computer-Aided Conceptional Design

The above considerations suggest to learn from mechanisms developed for knowledge-based program construction when setting up a framework for conceptional design in mechanical engineering. Such a framework is based on a number of observations and principles which we discuss first.

2.2.1. Priniciples

(1) The paradigm of proceeding from non-constructive to constructive descriptions. We describe technical systems in terms of objects constituting their components, and processes denoting histories i.e. sequences of state changes in the objects involved. A non-constructive description details objects and processes with their constituents without saying how state changes and histories are actually executed. In contrast, a constructive specification can be executed, producing a history from a given initial state.
A requirements definition (informal) and a specification (formal) provide a non-constructive problem definition in describing devices in terms of their functional behavior and intended properties of their performance. The objective of conceptional design is to develop constructive solutions which can be obtained by gradually refining non-constructive descriptions into constructive designs.

(2) The requirement of a vocabulary of technical concepts. To describe the problem of e.g. designing a transmission, we need a sufficcient vocabulary of notions to say what a transmission is actually about: A transmission transforms one form of motion into another. Hence, we need to describe both forms of motion and the way both motions should be related. Moreover, both forms of motion are carried by concrete physical objects, such as a shaft and a wheel. This implies that part of the problem definition is a description of the physical environment into which the device

to be constructed is to be incorporated. In summary, a problem definition describes a technical system where some components (the existing environment) are described constructively, and others (to be designed) are specified non-constructively. Both require a vocabulary of technical concepts and mechanisms for reasoning about them.

(3) Qualitative-to-quantitative refinement. Conceptional design is primarily concerned about qualitative descriptions and qualitative reasoning about them. Mechanisms must be provided for refining qualitative into detailed quantitative designs so that conceptional design systems provide input into CAD-systems.

(4) Abstraction requirement. As conceptional design is intended to produce principle solution, problem descriptions resp. solutions must be general enough to
- allow to devise a variety of design variants
- allow tailoring both problem definitions and solution variants to specific situations.

2.2.2. A Framework for Conceptional Design

Fig. 2-3 summarizes our model for conceptional design. The table merely lists the major building blocks without reflecting the control structure providing substantial facilities for interative revisions and refinements. It is a crucial requirement to conduct the entire design process within the uniform language environment of QRL which is expressive enough to support all levels from abstract problem descriptions down to representations of concrete systems.

Language	Phase	Result	Tool(s)
graphics, technical NL	System Analysis, Requirements Definition	Description of intended system	(editors)
QRL	Specification and Abstraction	Specification (abstraction and potential refinements)	QRL-system (editor, interpreter, reasoner, explainer)
QRL	Decomposition into Subtasks	Specification	QRL-system
QRL	Design of principle solutions for subtasks	Conceptional subtask-variants	domain specific expert systems
QRL	Evaluation of subtask variants	Properties about subtask variants	QRL-system and critic
QRL	Subsystem integration	Conceptional variants of entire system	QRL-system
QRL	Evaluation of system variants and variant selection	Selected principle solution	critic with QRL-system
QRL	Optimization	Optimized principle solution	knowledge-based optimizer with QRL-system

Fig. 2-3 Model of Conceptional Design

3. Reasoning on Time and Causality

3.1 Requirements on Representing Time and Causality

Time and causality are closely related. Basic to our model is the elementary notion of an event denoting observable changes of the state of a system. Time becomes observable because of events occurring at different instances of time. An event e_1 may cause another event e_2 so that it is observable that e_1 occurs before e_2. However, any ordering of time depends on observable causal relationships between events: if e_2 is not caused by e_1, the only way to observe that e_1 occurs before e_2 is to observe e_1 simultaneously with some event $e\bar{1}$ which causes another event $e\bar{2}$ observed simultaneously with e_2 (note that e_1 and $e\bar{1}$, and e_2 and/or e_2 and $e\bar{2}$ may be equal). Hence, different unrelated systems necessarily have independent time scales, and events occur concurrently.

Previous work on modelling and reasoning about time consists of two categories:

- State-based approaches take states to be elementary and observable. Events are caused by actions described as state transformations which are not observed. Hence, time is observed as a sequence of states usually attributed to instants of time [Sacerdoti 77]. This has been extended to situations [McDermott 82] and states persisting over time intervals having a more or less certain extension [Allen 83].

- Event-based approaches take events to be elementary and observable, e.g. by describing time as a sequence of chains of state transitions denoting the state change effected by the events having occurred [Kahn, Gorry 77].

[Allen 83] postulated three major requirements for models of time:

(1) The postulate of uncertainty. The representation must allow for uncertainty w.r.t. two aspects:
   - The time scale need not consist of precisely quantifiable dates and relationships between dates. Instead, vague notions for relationships between instances and intervals such as "long before", "much later", etc. must be accounted for.
   - Vague notions for time spans such as "a pretty short time" must be available.

(2) The postulate of variety in granularity. Depending on the context being reasoned about, particular spans of time may be short or extremely long, like a millenium being looked at as an instant in paleontology although it is longer than all of modern history.

(3) The postulate of persistency. The model must account for properties being persistent, i.e. if it holds over a period of time, it will also hold later on unless explicitly negated. Persistency substantially reduces storage requirements when maintaining all of, or only aspects of states over a period of time.

The above dicussion indicates the importance of additional postulates:

(4) The postulate of time-causality linkage. Any ordering on time scales must be related to causal relationships between events s.t. one event causing another implies that the cause occurs before the effect.

(5) The postulate of concurrency. The model must represent concurrency of events and time scales.

We claim that the model presented in this section fully satisfies all of these requirements.

3.2. Interval Structures

To satisfy the above postulates except for (3-persistency), time is represented by a set Intv of intervals which is

- doubly partially ordered by the partial orders
  "$\prec$" s.t. $I_1 \prec I_2$ stands for "interval I_1 is earlier than interval I_2", and
  "$\underline{c}$" s.t. $I_1 \underline{c} I_2$ stands for "interval I_1 is during interval I_2".
  Intervals not comparable under $\prec$ are <u>concurrent</u>.

- associated with functions relating intervals to extensions and distances.

Interval structures are obtained by composing intervals sequentially and/or concurrently.

<u>Notation</u>. $I_{(n)}$ stands for the sequence $(I_1,I_2,\ldots,I_n)$ resp. the set $\{I_1,I_2,\ldots,I_n\}$ as it is evident from the corresponding context.

3.2.1. Intervals

(Intv,$\prec$,$\underline{c}$) is a doubly partially ordered set, and

- beg,end: Intv $\rightarrow$ Intv map intervals to their beginning resp. end.
- dist: Intv $\times$ Intv $\rightarrow$ Ext measures the distance between $\prec$-comparable intervals.
- ext $\underline{c}$ Intv $\times$ Ext relates intervals to notions of temporal extensions.

<u>3.2.1.1. beg-end laws.</u> The functions beg and end are required to satisfy the following laws for any intervals $I,I_1,I_2 \in$ Intv.

[beg-end 1] $beg(I) \prec end(I)$
{the beginning of an interval is earlier than its end}

[beg-end 2] $I_1 \prec I_2 \iff beg(I_1) \prec beg(I_2)$ & $end(I_1) \prec end(I_2)$
{If interval I_1 is earlier than interval I_2, then I_1 and I_2 are in one of three relations:

seperated $end(I_1) < beg(I_2)$

consecutive $end(I_1) = beg(I_2)$

overlapping $\quad beg(I_2) < end(I_1)\}$

[beg-end 3] $a(b(I)) = b(I)$ for $a,b \in \{beg,end\}$.
{The beginning/end of the beginning/end of an interval is the beginning/end itself. I.e. beginning and end of intervals are instances in the sense that their beginning and end coincide}.

[beg-end 4] $I_1 \subseteq I_2 \iff beg(I_2) \leq beg(I_1)$ & $end(I_1) \leq end(I_2)$
{$I_1 \subseteq I_2$ expresses that I_1 is during I_2}

Note that beg(I) = end(I) is sufficient to characterize I as an instance without having any notion on the extension on intervals. By [beg-end 2], $I_1 = I_2$ iff each beginnings and ends of I_1 resp. I_2 coincide, as "$\leq$" is a partial order.

3.2.1.2. Time lines. A sequence $I_{(n)}$ with $\forall i \in (n-1).I_i < I_{i+1}$ is a time line. We require Intv to be gap-free on time lines, i.e. for any intervals I_1, I_2:
$I_1 < I_2$ & $end(I_1) < beg(I_2) \Rightarrow \exists I.I_1 < I < I_2$
& $end(I_1) = beg(I)$ & $end(I) = beg(I_2)$

In this case, we call I to be the gap between I_1 and I_2:
$I = gap(I_1, I_2)$.

3.2.1.3. Extensions. Intervals describe both instances and temporally extended time spans. The postulates (1-uncertainty) and (2-granularity) are taken care of by introducing appropriate notions of the extension of intervals. Leaving the choice of particular notions up to specific applications, we postulate the following minimal requirements to be satisfied by a set Ext of notions of extensions:

[Ext-1] (Ext,$\leq$) is a partially ordered set of notions of extensions.
{Extensions should at least be partially comparable.}

[Ext-2] (0,pos) $\subseteq$ Ext and 0 is minimum of Ext.
{Ext contains at least the extensions 0 and pos ("positive") with 0 < pos.}

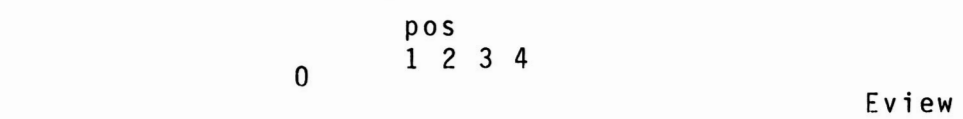
[Ext-3] +: Ext × Ext → Ext is an addition operation s.t.
- + is associative and commutative
- x + 0 = x

- + is monotonic
- all $\leq$-comparable elements of Ext are closed under +
  {+ is closed under any single view.}

[Ext-4] ext $\subset$ Intv × Ext is the extension relation satisfying

(1) ext is partitioned into multiple extension views
Eview = $\{ext_1, ext_2, \ldots\}$ s.t.
- each extension view ext is a function, and ext = U Eview.
- the range of each extension view is linearly ordered.
- (I,x) ε ext & x $\neq$0 => (I,pos) ε ext
  {if some view assigns I a non-0 extension then it is a "refinement" of pos}

The partitioning of ext into extension views may therefore look like this:



(2) ext(I) = 0 <=> beg(I) = end(I) for any IεIntv.
{beg and end take the "finest" view, i.e. if beginning and end of an interval coincide, then this interval has extension 0 under any view.}

(3) (a(I),0) ε ext for IεIntv, aε{beg,end}
{Under some view, beg/end map to instances, but not necessarily under every view.}

[Ext-3] ensures that addition of extensions of different intervals is defined if they conform to the same view. Partitioning ext into arbitrary views in [Ext-4] allows to have different granularities and degrees of imprecision to exist. Here, we leave open in what way different views might be combined.

3.2.1.4. For $\leq$-comparable intervals, it is interesting to determine how far they are apart. We therefore introduce a distance function mapping pairs of intervals to temporal extensions:

dist $\subset$ Intv × Intv x Ext is a relation s.t.

[dist-1] $I_1 \not\sim I_2 \Rightarrow dist(I_1,I_2)$ is undefined.
{Distances can only be determined for intervals of the same time line.}

[dist-2] For any intervals $I_1,I_2 \in Intv$,
- <u>if</u> $I_2 < I_1$ <u>then</u> $dist(I_1,I_2) := dist(I_2,I_1)$.
{A distance is always determined from an earlier to a later interval.}
- <u>if</u> $I_1 \leq I_2$
<u>then</u> a. $beg(I_2) \leq end(I_1) \Rightarrow dist(I_1,I_2) = 0$, and
{overlapping intervals have distance 0}
b. $end(I_1) \leq beg(I_2)$
$\Rightarrow \forall ext_i \in Eview.ext_i[gap(I_1,I_2)] \in dist(I_1,I_2)$
{the distance is determined as the extension of the gap between intervals}

[dist-3] dist satisfies the triangle inequality under any extension view:
$dist(I_1,I_3) \leq dist(I_1,I_2) + dist(I_2,I_3)$
for any intervals I_1, I_2 and I_3 on the same time line.

<u>Note:</u> dist is only a quasi-metric on any particular view of ext in the sense that $dist(I_1,I_2) = 0$ even if $I_1 \neq I_2$.

<u>3.2.1.5. Interval structures.</u> Intervals on the same time line can be merged to intervals covering them. Intervals on different time lines can be combined into an expression just describing their concurrency. This gives us two combinators for <u>merging</u> and <u>conjoining</u> ("join concurrently") which can be used to construct expressions describing interval structures. The merge and conjoin operations are defined in such a way that they require to introduce the empty interval Inil as a neutral element w.r.t. both operations.

<u>The empty interval Inil.</u> We require Intv to contain the empty interval Inil with the following properties:

[Inil-1] $\forall I \in Intv.\ Inil \not\sim I$
{Inil cannot be placed on any time line}

[Inil-2] $beg(Inil) = end(Inil) = Inil$
{implies $ext(Inil) = 0$ by [Ext-4(3)]}

[Inil-3] dist(Inil,I),dist(Inil,I) are undefined for any interval

$I \neq Inil$
{any artificial definition for dist(Inil,I) results in conflicts with the triangle inequality [dist-3].}

Merge. The merge operation ;: Intv × Intv → Intv is defined to be a function with the following properties for any intervals I, I_1 and I_2:

[Merge-1] $I_1;I_2$ is undefined if $I_1 \not\leq I_2$ unless I_1=Inil or I_2=Inil
{only intervals on the same time line can be merged}.

[Merge-2] Inil;I = I;Inil = I
{Inil is the identity for merging}

[Merge-3] If $I_1 \leq I_2$ with $I_1, I_2 \neq$ Inil then $I_1;I_2 \in$ Intv s.t.
(1) $beg(I_1;I_2) = beg(I_1)$, $end(I_1;I_2) = end(I_2)$.
{$I_1;I_2$ is an interval beginning with the begin of I_1 and ending with the end of I_2}

(2) if $end(I_1) \leq beg(I_2)$ {I_1 ends before I_2 begins}
then $ext(I_1;I_2) = \{e_1+e+e_2 \mid ext_i \in EView$ and $e_1 = ext_i(I_1)$, $e_2 = ext_i(I_2)$, $e = ext_i[gap(I_1,I_2)]\}$
{the merged interval $I_1;I_2$ extends not only over I_1 and I_2, but over the gap between I_1 and I_2, too}

(3) if $beg(I_2) \leq end(I_1)$ {both intervals overlap}
then $ext(I_1;I_2) = ext[beg(I_1);end(I_2)]$.

[Merge-4] ; is associative, i.e. $I;(I_1;I_2) = (I;I_1);I_2$

Notation: $I_{(n;)}$ stands for $I_1;\ldots;I_n$ for any time line $I_{(n)}$.

Conjoining concurrent intervals and interval structures.
Intervals on different time lines are called concurrent, i.e. intervals I_1 and I_2 are concurrent iff $I_1 \not\sim I_2$. The conjoin operation (,): Intv × Intv → IS combines concurrent intervals to concurrent interval structures.

The domain IS of interval structures is inductively defined to be the least set s.t.
(1) Intv $\subseteq$ IS {each interval is an interval structure}

(2) For any time line $I_{(n)}$, $I_{(n;)} = I_1;\ldots;I_n \in$ IS

is a consecutive interval structure.
Note: By [Merge-3], consecutive interval structures are gap-free.

(3) For any concurrent intervals $I, I' \in Intv$, $(I, I') \in IS$ is a concurrent interval structure with

$$beg[(I,I')] := \underline{if}\ beg(I) = beg(I')\ \underline{then}\ beg(I)\ \underline{else}\ undefined$$

$$end[(I,I')] := \underline{if}\ end(I) = end(I')\ \underline{then}\ end(I)\ \underline{else}\ undefined$$

(4) The merge and conjoin operations are both extended from intervals to interval structures, so that $;, (\ ,\ ): IS \times IS \to IS$ satisfy the following laws:

- The conjoin of the operation (,) is commutative and associative, and has Inil as its identity on IS.

- Merging distributes over conjoining, i.e.
  $I;(I_1,I_2) = (I;I_1,I;I_2)$ and
  $(I_1,I_2);I = (I_1;I,I_2;I)$,
  assuming the respective subexpressions to be proper interval structures.

3.3 Event and Situation Structures

Remembering that solely a state-based as well as an event-based approach has both advantages and disadvantages when compared with the other, we combine both approaches. States are described by assertions about values bound to observables, and events are taken to denote transitions between states referring to intervals on the same time line in increasing order.

3.3.1 Event structures. Let E be a set of events. ES = $(E,\leq,\#)$ is an event structure (on #) iff

- $(E,\leq)$ is a partially ordered set (a causal structure), and
- $\# \subseteq E \times E$ is a symmetric and irreflexive conflict relation on E s.t. for any events $e_1,e_2,e_3,e_4 \in E$,
  $e_1 \leq e_2 \leq e_3 \;\&\; e_2 \leq e_4 \Rightarrow e_2 \# e_4$

Clearly, the partial order "$\leq$" describes the causal relationships between events in the sense that $e_1 \leq e_2$ iff e_1 causes e_2. If an event e_2 causes both the events e_3 and e_4, the effects e_3 and e_4 of e_2 are said to be in conflict. These notions originate form Petri nets [Genrich, Lautenbach 79], where causal structures and conflicts are modelled in occurrence nets shown to be closely related to event structures in [Winskel 81].

3.3.2. Situation structures. Our objective is to relate states and events via the temporal intervals in which they occur. First, we relate event and interval structures: time $\subseteq E \times Intv$ relates events to intervals in which they occur.
Actually, time involves particular views w.r.t. precision and granularity to which the interval chosen to describe the time span containing the occurence of an event is determined.

Given an event structure ES = $(E,\leq,\#)$, time is required to satisfy the following compatibility relation between $(Intv,\leq)$ and ES for events $e_1,e_2 \in E$:

- $e_1 \leq e_2 \Rightarrow \forall I_1,I_2 \in Intv.\ e_1 \text{ time } I_1 \;\&\; e_2 \text{ time } I_2 \Rightarrow I_1 \leq I_2$
  {If e_1 causes e_2 then e_1 occurs earlier than e_2.}

Note: $e_1 \# e_2$ does not imply $\text{time}(e_1) \neq \text{time}(e_2)$, i.e. two events in conflict may occur on the same time line.

Notation: For an interval $I \in Intv$ and events $e_1,\ldots,e_n$, $(I|\ e_1,\ldots,e_n)$ denotes the assertion

$\{e_1,\ldots,e_n\} \subseteq time^{-1}(I)$, i.e. "the events $e_1,\ldots,e_n$ occur in I".

A <u>situation</u> $(I|e_1,\ldots,e_n|p)$ consists of
- an interval $I \in Intv$
- a set $\{e_1,\ldots,e_n\}$ of events s.t. $(I|e_1,\ldots,e_n)$
- a first-order property p about the state of the system being modelled s.t. p holds throughout the interval I.

Let $intst \in IS$ be any interval structure s.t. intv(intst) is the set of intervals occurring in intst. If each interval in intv(intst) is the image of some set of events under time, then replacing each interval in intst with a corresponding situation yields a <u>situation structure</u>. (Sit,$\prec$,$\subseteq$) denotes the domain of all situation structures, where $\prec$ and $\subseteq$ are the partial orders extended from Intv to Sit in an obvious way, but respecting the causality order of events.

A situation $(I|e_1,\ldots,e_n|p)$ is called <u>persistent</u>, <u>iff</u> for any interval $I' \in Intv$ containing I, the properties of the situation persist through I':

- $(I|e_1,\ldots,e_n|p)$ & $I \subseteq I'$ => $(I'|e_1,\ldots,e_n|p)$

Note that persistency is attributed to situations and not to temporal intervals as in [Allen 83]. This is because persistency relates to properties of states extending over time despite the occurrence of events. Ascribing persistency to temporal intervals which a variety of different and unrelated states and events might refer to simultaneously is a misleading "overloading" of time.

3.4. Reasoning Techniques

This section briefly introduces several approaches to causal and temporal reasoning, based on the concepts introduced in the previous sections.

3.4.1. Relations on intervals. For intervals on the same time line, Allen 83 discussed the following comprehensive set of relationships between intervals shown in Fig. 3-1.

Relation	Graphical Illustration	Definition
before	xxx..xxx	$I_1 < I_2$, i.e. $end(I_1) < beg(I_2)$
equal	xxxxx	$I_1 = I_2$
overlaps	xxxxx	$beg(I_2) \leq end(I_1)$ & $end(I_1)$ $end(I_2)$
during	xxx xxxxxx	$I_1 \subseteq I_2$, i.e. $beg(I_2) \leq beg(I_1)$ & $end(I_1) \leq end(I_2)$

Fig. 3-1. Allen's relations on intervals.

A refinement of the "during" relationship are the relations

starts	$beg(I_1) = beg(I_2)$ & $end(I_1)$ $end(I_2)$
between	$I_1 \subseteq I_2$ & $I_1 \neq I_2$
finishes	$beg(I_2) \leq beg(I_1)$ & $end(I_1 = end(I_2)$

Allen 83 presents procedures for computing the transitive closures of these relations, their inverses, and combinations of them.

3.4.2. Magnitudes. The above relations lack qualitative, refineable notions such as "immediately before", "long after", "slightly overlapping" etc., i.e. the postulates (1-uncertainty) and (2-granularity) of section 3.1 are not satisfied. We call such qualifications of the temporal relations magnitudes. Similarly as for extensions, we assume to have different views on

magnitudes, obtaining a partially ordered domain (Magn,$\leq$) with a structure similar to Ext:

[Magn-1] (Magn,$\leq$) is a partially ordered set of magnitudes.

[Magn-2] $0 \in$ Magn is the minimal magnitude.

[Magn-3] $+$: Magn $\times$ Magn $\rightarrow$ Magn is an addition operation s.t.
- $+$ is associative, commutative, and monotonic
- $x + 0 = x$
- all $\leq$-comparable magnitudes are closed under $+$

[Magn-4] The relation magn $\subseteq$ IReln $\times$ Intv $\times$ Intv $\rightarrow$ Magn ascribes relations on intervals a magnitude so that
magn is partitioned into multiple views
Mview = {$magn_1$, $magn_2$,...} with
- each magnitude view is single-valued, and magn = $\cup$ Mview
- the range of each magnitude view is linearly ordered.

When choosing particular notions and views of magnitudes, one may introduce rules for combining them and make inferences.

3.4.3. Reasoning on extension and magnitude views. One obvious way for determining magnitudes is to relate them to particular extension views.

Example: Let (0, immediately, shortly, long) $\subseteq$ Magn and (0, short, medium, long, very long) $\subseteq$ Ext be the ranges of two particular magnitude and extension views $magn_i$ and ext_j. One could then base the magnitude ascribed to the "before"-relation on ext_j:

$$magn_i(\text{before}, I_1, I_2) := \begin{cases} 0 & \text{if } ext_j[gap(I_1,I_2)] = 0 \\ \text{immediately} & \text{if } ext_j[gap(I_1,I_2)] = \text{short} \\ \text{shortly} & \text{if } ext_j[gap(I_1,I_2)] = \text{medium} \\ \text{long} & \text{if } ext_j[gap(I_1,I_2)] \in \{\text{long, very long}\} \end{cases}$$

So taking the gap between two intervals I_1 and I_2 to be 'short' resp. 'medium' is translated into I_1 being 'immediately before' resp. 'shortly before' I_2. Clearly, one must be careful to define the addition operations on Ext and Magn in a compatible way.

Rules may be introduced for reasoning about extension and magnitude views. Two particularly important ways of reasoning

about views are translating one view into another, and inferring one view from another view or even several other views.

3.4.4. Concurrency reasoning. [Winskel 81] has shown that event structures and occurrence nets are equivalent in the sense that they are isomorphic. This allows to apply the wealth of techniques for reasoning about occurrence nets to event and interval structures. For example, techniques for detecting or ruling out deadlocks, livelocks, liveliness, etc. become applicable.

3.4.5. Causal-temporal reasoning. Clearly, temporal relationships between intervals carry over to events, e.g.

e_1 occurs "long before" e_2 iff $(I_1|e_1)$ & $(I_2|e_2)$ & $magn_i(before, I_1, I_2)$ = long (for some magnitude view $magn_i$).

A historian might refuse to impose causal relationships, saying that $e_1 \neq e_2$. However, within severly restricted, e.g. partially restricted, contexts it may be possible to find conditions allowing to infer causal from temporal relationships.

4. Non-Monotonic Reasoning(NMR)

Conceptional design is often based on assumptions and goals which turn out to be unsuitable or misleading in the course of developing the design. If such assumptions and goals need to be revised, all dependent conclusions and design decisions have to be revised, too. Conceptional design therefore involves non-monotonic reasoning (NMR).

Previous work on NMR has been based on extensions of propositional and predicate logic (for summaries, see e.g. [Doyle 83], [Etherington 83], [Moore 83]). As we are not only concerned with logical assertions in general, but machine designs in particular, we introduce a special NMR-calculus to support the non-monotonic development of conceptional designs. This system is based on Doyles' Reason Maintenance System (RMS) [Doyle 83, Goodwin 82], and extends arbitrary formal systems, not just logical calculi.

4.1. Formal systems.

A formal system FS = (L,R) consists of a language L and a set $R \subseteq L \times L$ of rules which induce a derivation relation $\rightarrow$ on L, where $\rightarrow = R^*$. For denoting rules, we employ conditional rule schemata $(c \mid p \rightarrow q)$, where p and q are expressions with variables instantiating to elements of L, and c is a first-order formula containing at most the variables occuring in p and q as free variables. The rule schema $(c \mid p \rightarrow q)$ applies to an element $w \in L$ iff there is a substitution σ s.t. $\sigma p = w$ and σc = true, allowing to infer σq from w.

4.2 NMR-calculi on formal systems.

We apply Doyle's "reason maintenance"-notation to an arbitrary formal system FS = (L,R) with a set RS of rule schemata denoting R by constructing an NMR-calculus based on the following abstract syntax:

```
Attitude:        :=  Belief|Drive
                     {attitudes are either justified
                      assertions called beliefs, or
                      justified rules called drives}
```

```
Belief              :=  (Name L Justification-set)
                        {a belief is an assertion associated
                         with a set of justifications; each
                         belief may be referred to by a unique
                         name}

Drive               :=  SL-Justification | P-Justification |
                        Justification-set

SL-Justification    :=  (SL In-set Out-set)

CP-Justification    :=  (CP Conseq In-set Out-set)

In, Out, Conseq     :=  Atom

Atom                :=  Name | (GOAL Name) | (non Atom)

Meta-Drive          :=  MCond → MAction
                        {meta-drives act as meta-rules which
                         fire when an assertion on
                         justifications, called meta-condition,
                         holds; the effect of fixing a meta-
                         drive is executing its meta-action}

MCond               :=  is a domain of assertions on
                        justifications

MAction             :=  JustifyAction | SelectAction
                        {meta-actions may alter justifications
                         or propose names of drives for further
                         execution}.
```

The semantics of this system is an obvious extension of Doyle's RMS [Doyle 83], extended in [Mina 84].

5. A Theory of Qualitative Reasoning (QRT)

This section introduces a theory of qualitative reasoning (QRT) particularly suited for physico-technical systems. We proceed by summarizing the goals and requirements for any such theory in section 5.1, and introduce the basic concepts of our theory in section 5.2. The theory is formalized in terms of the language QRL (Qualitative Reasoning Language) presented in section 5.3.

5.1. Objectives

The overall goals of a theory of QR go far beyond applications for conceptual design. A theory of QR must provide for representation formalisms and reasoning mechanisms allowing to express and utilize all kinds of knowledge about physico-technical systems which is refineable into quantitative models.

5.1.1. Epistemological domains. Reasoning on physico-technical systems comprises a substantial variety of knowledge domains suggesting different knowledge representation and reasoning techniques. Therefore, it is a crucial problem to find an appropriate decomposition of the entire universe of discourse into epistemological domains. Imposing such structure should be guided by two principles:

- epistemological uniformity: each domain should be uniform and succinct w.r.t. employing a minimal diversity of techniques for representing its contents adequately. Note that this is not necessarily a matter of formalism: e.g. Horn clauses of first-order predicate calculus are similarly adequate for representing the state transformations effected by (AB)STRIPS operations as the PLANNER-like SOUP-functions [Sacerdoti 77].
- hierarchy formation: knowledge of one domain is often based on resp. is the basis for knowledge in other domains, resulting in use-relationships according to which domains utilize others. The easiest way of constructing and maintaining such relationships is to form use-hierarchies of domains.

For physico-technical systems that have been studied so far, it turned out to be useful to cluster the epistemological domains into three different levels:

(1) The object level comprises domains relating to the physical appearance and properties of objects, such as geometry with

decomposition and aggregation of parts, and materials with their properties.

(2) The function level consists of domains containing knowledge about operations and actions performed by, resp. executed upon objects and their parts described in the object level. Such operations are meant to be elementary, i.e. not composed of temporally extended functions on the respective level of aggregation. For example, the flight of a plane between cities is taken to be an elementary function at the aggregational level of flight schedules, although each single flight is composed of many subactions.

(3) The process level combines domains with knowledge about processes, i.e. temporally extended, successive and/or concurrent sequences of operations introduced in the function level, and acting on objects described in the object level.

5.1.2. Goals of QR. Based on the structure and extent of knowledge as outlined above, QR should allow to

- describe and explain conceptual, spatial, and temporal structure as well as causal and functional relationships on all three levels of epistemological domains.
- interpret observable phenomena, such as measurement data and sensory perceptions, as effects of processes.
- infer new causal and functional relationships, both by knowledge-based inference and formation of abstractions.
- predict future phenomena.
- synthesize properties from properties of aggregational components such as subparts or subprocesses.
- configure sequences of actions ("planning") and aggregations of parts ("design", "construction") to reach particular goals resp. obtain a specific behavior.

For each of these activities, QR should allow for appropriate quantitative refinements.

This wide range of objectives for a theory of QR makes it basic to all forms of representing and processing knowledge about physico-technical systems. Hence, a theory of QR is actually a foundation for all application areas of this subject, i.e. it is what expert systems for various kinds of applications discussed in section 1 are based on.

5.2. Principles

QR for physico-technical systems requires specific consideration of four aspects:

- qualitative reasoning should be refinable to quantitative reasoning in mathematical models.
- the frame problem has to be solved.
- QR must comprise mechanisms of aggregation and decomposition of spatially, temporally, and functionally composite objects and processes.
- QR must incorporate causal, temporal, teleological, and non-monotonic reasoning.

This section discusses the principles of our approach towards these aspects.

5.2.1. Refinement principle. Mathematical models in science and engineering are abstractions which assume, but do not explicitly model the nature of objects with the processes operating on them, based on causal relationships determined by physical laws. Quantitative models describe systems in terms of spatial and temporal changes of observable quantities as expressed in differential equations. Qualitative models only describe such changes in terms of qualitative notions such as "large speed", "small distance", etc. Clearly, such qualitative notions are made precise by quantitative data. The refinement principle postulates this relationship in a consistent and complete way:

- Refinement principle. QR must admit refinements into quantitative models satisfying the following three conditions:
  (1) Consistency. Qualitative notions are refined to quantitative domains of values s.t. orderings are preserved. Likewise, qualitative functions are refined into quantitative ones, again preserving continuity resp. monotonicity properties. This consistency requirement extends to explanations, interpretations, inferences, predictions, synthesized properties, plans, and designs.
  (2) Completeness. Each qualitative notion must have a quantitative refinement, and each quantitative value has a corresponding qualitative coarsening. Again, this completeness requirement extends to functions and reasoning procedures.
  (3) Disambiguation. Qualitative notions are necessarily vague so that a qualitative value may have several, even an

infinity of quantitative refinements. A quantitative model should allow to decide on any qualitative ambiguities and vagueness.

5.2.2. Frame problem. The frame problem [McCarthy, Hayes 69] is to maintain the current state under changes affecting only small parts of extremely complex state descriptions. It arises regardless of whether reasoning is done qualitatively or quantitatively. However, the fact that QR addressed functional and causal relationships provides approaches to solving the frame problem.

Early approaches to solve the frame problem (as e.g. in STRIPS/ABSTRIPS [Sacerdoti 77]) consisted of describing state changes in situation calculi by axiomatizing the effects of actions. However, an increasing number of actions results in an astronomical growth of axioms prohibiting this approach from being practical. [Hayes 79] has shown that stepping through isolated state changes without reasoning on inherent connections is principally insufficient. To solve this deficiency, Hayes [Hayes 78] introduced histories which are composed from episodes, where each episode comprises a conceptually aggregated sequence of state transitions. Confining the extent of state changes is achieved by imposing the requirement that a state change of an object may only affect another object if both objects touch each other. Forbus [Forbus 81, 82, 83] discovered that histories describe the effects of the underlying mechanisms without representing and allowing to reason about the mechanisms themselves. Forbus therefore proposed processes as basic entities desribing changes in physical systems in his qualitative process theory. A process describes a mechanism whose repeated execution may effect a multitude of histories.

Such processes may operate on different objects, and the changes of particular observables can only be determined from all, usually both spatially and temporally distributed processes, affecting it. A complementary view is taken by the envisionings [de Kleer, Brown 83], where confluence laws on qualitative variables are based on device and machine models. [Kuipers 82; Kuipers, Kassirer 83] discuss such confluence laws without explicitly relating them to either processes and device resp. machine models.

Both qualitative process theory and envisioning have identified

(see [Forbus 83] and [de Kleer, Brown 83]) two basic problems which must be solved for solving the frame problem with an approach based on histories and processes:

[Construction Problem] Given an observable behavior as a goal, how can one construct a machine model with processes being executed on the machine(s) leading to the required behavior?

[Interaction Problem] Interactions between histories can be considered on different levels of abstraction:

Process level: How can one determine and describe interactions between histories from descriptions of different processes generating these histories?
This is the problem of finding combinators on processes, leading to mechanisms for aggregating processes to another level of abstraction.

History level: Given different histories interacting via particular states resp. objects or parts, how can one infer and explain such interactions with interactions among objects in terms of physical laws?
This is the problem of how to find mechanisms modelling and explaining interactions among histories.

A third substantial problem closely related to the preceding ones is to find appropriate aggregations:

[Aggregation Problem] How can one combine spatially related objects to composite spatial aggregates, and, simultaneously, how can one combine temporally distributed events to composite temporal aggregates so that both spatial and temporal aggregates make up meaningful functional aggregates?

We will show in the sequel how solving these problems leads to solving the frame problem, too.

5.2.3. Conceptual network for QR on physico-technical systems.

The approach of qualitative reasoning theory (QRT) is based on the conceptual network [Raulefs 83] given in Fig. 5-1.

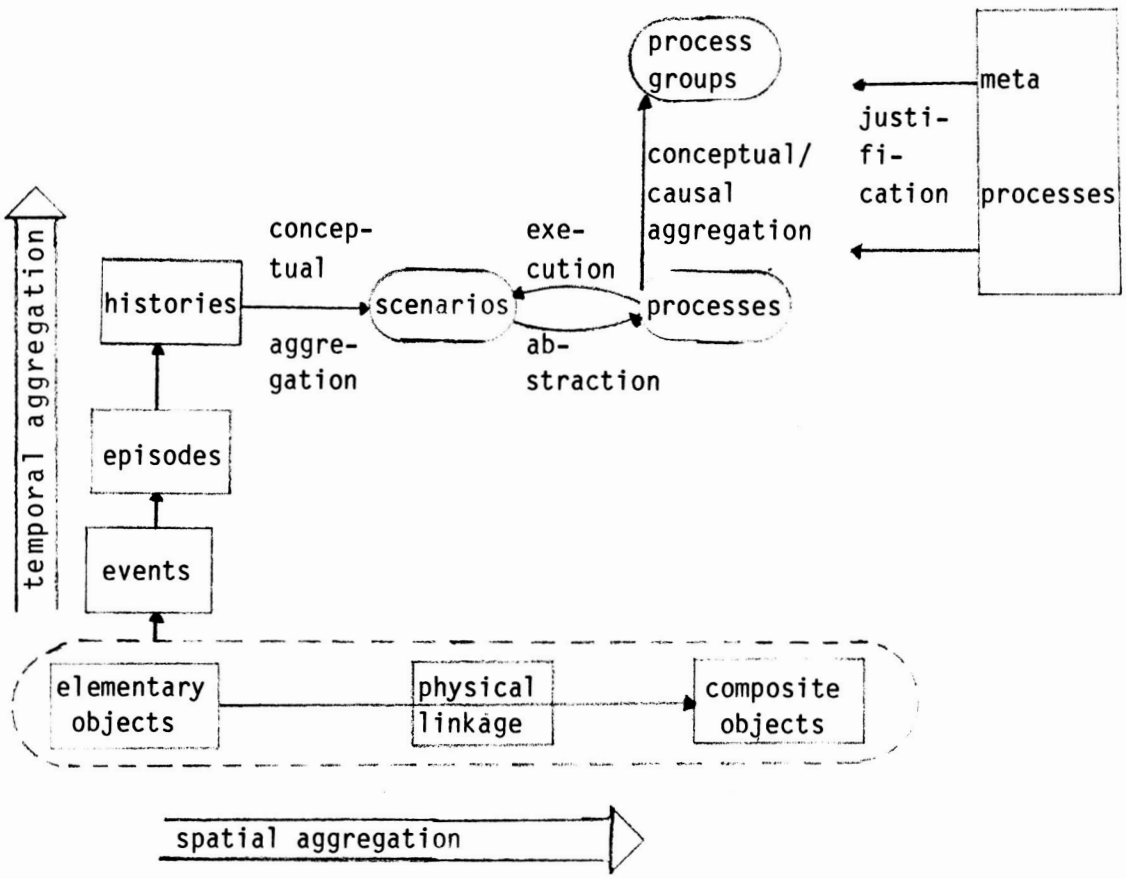


Fig 5-1. Conceptional network for QRT.

The nodes of this network are linked by four kinds of aggregation:

- Spatial aggregation is done by physically linking objects s.t. state changes of one object are physically transmitted to the linked objects. Objects which cannot be decomposed into physically linked parts are elementary objects, and spatial aggregation of elementary and composite objects results in composite objects. Note that an object must not be solid, but may very well be a gas or a liquid with a spatial extension which rapidly changes in time.
  Clearly, what is taken to be elementary is a matter of viewpoint. For describing a car, the engine may be taken to be an elementary part, although another view considers the engine to be a composite object aggregated from thousands of parts.

- Temporal aggregation takes events to be elementary implying that events cannot be attributed any temporal extension (time is discrete). Situations that may contain events may be

aggregated to episodes, and episodes can be composed to histories and furthermore to scenarios. Any temporal aggregate spreads into a situation structure so that episodes and histories may consist of concurrent (sub-)episodes, particulary for (sub-)episodes occurring on physically not connected objects.

- Abstraction of histories or scenarios to processes is a particular kind of conceptual aggregation. A process denotes the set of all histories which it can execute, similarly to a program denoting under its operational semantics the set of all execution traces obtained from executing it on all possible input data.

- Conceptual aggregation clusters histories to scenarios as well as processes to process groups whenever there appears to be a compelling reason to consider them under a common point of view.

Meta-processes provide justifications and allow for teleological reasoning.

There are close analogies between programming languages and QRT:
- Spatial aggregation of composite objects corresponds to forming data structures.
- Temporal aggregation of events to histories corresponds to combining isolated state transitions to execution traces of programs which, in turn, correspond to processes.
- Abstraction of histories resp. scenarios to processes corresponds to the design of abstract data types comprising data structures and characteristic sets of operations denoting sets of execution traces.
- Conceptional aggregation of scenarios to processes resp. process groups corresponds to incorporating different operations (giving rise to execution traces) in an abstract data type, whereas conceptual aggregation of processes to process groups corresponds to "use"-hierarchies of abstract data types.

5.2.4. Qualitative physics refining into "ordinary" physics is an important part of QRT. A physical system is described in terms of states and state changes in space-time, where a state is a set of values taken by physical quantities. A physical quantity consists of a numerical value referring to a particular scale, and a

dimension denoting that scale. Physical functions relate physical quantities in space-time, and thus contribute to descriptions of the spatially and temporally extended behavior of physical systems. As such behavior is recorded in terms of state changes, differential equations play a central role in physical models. It is therefore important to devise qualitative analogues which refine into the above notions.

5.2.4.1. Qualitative quantities. A dimension D is an arbitrary set of qualitative values which may decompose into different views required to refine each other. Such a multiple view dimension consists of

- a set D of qualitative values which is partitioned into
- a set Dview = $\{D_1, D_2, \ldots\}$ of views with D = UDview, and
- a relation refine $\subseteq$ D × D which refines views in a way which is compatible with orderings on views.

A qualitative quantity q denotes a pair q = (val(q),dim(q)) consisting of the value val(q) and the dimension dim(q) of q, where val(q) ε dim(q).

Dimensions can be composed to form new dimensions. As an example, we take

D_l = (0, short, medium, long) to be a dimension for measuring length, and

D_t = (0, immediate, very-short, short, a-while, some-time, long, very-long) to be a dimension for measuring time spans.

We want to relate both dimensions to the new dimension for velocities

D_v = (0, slow, medium, fast).

This is done by setting up a division table for dividing the qualitative values from D_l by the values for time spans from D_t:

D_v	0	short	medium	long
0	-	-	-	-
immediate	0	medium	fast	fast
very-short	0	medium	medium	fast
short	0	slow	medium	fast
a-while	0	slow	slow	medium
some-time	0	slow	slow	medium
long	0	slow	slow	slow
very-long	0	slow	slow	slow

We require such compositions of dimensions to be compatible with multiple views as defined below.

5.2.4.2. Qualitative functions map qualitative quantities to qualitative quantities. View compatibility is a property of particular importance for qualitative functions. We define this property for monadic functions producing single quantities only, as this easily generalizes to arbitrary vector functions:

> Let D and D´ be two multiple view dimensions with views Dview = $\{D_1, D_2, \ldots\}$ and Dview´ = $\{D_1', D_2', \ldots\}$, and view refinement relations refine and refine´.
> A qualitative function f: D → D´ is view compatible iff
> ∀ x,y ε D . x refine y => f(x) refine´ f(y).

As qualitative functions primarily describe changes of qualitative quantities, we need a notion for derivatives. Previous approaches on qualitative derivatives simply took difference quotients as qualitative derivatives:

> If f:D→D´ is qualitative function, and dist:DxD→Ext, dis´:D´xD´→Ext are distances on D resp. D´ (see 3.2.1.3.4), then
> $$dist'[f(x),f(x_0)]/dist(x,x_0)$$
> is the qualitative derivative of f at x_0, assuming a division operation on Ext.

Clearly, this approach fails uniqueness, because arbitrarily chosen values for x could produce wildly different values of the derivative at x_0. It is obvious that no such concept could do without topologies on D and D´.

Let D,D´ be sets with distances dist and dist´ as above.

(1) A subset A $\subseteq$ D´ is called open iff $\forall x \varepsilon A. \exists r \varepsilon Ext. r \neq 0$ & $\{y \mid dist(x,y) \leq r\} \subseteq A$

(2) A function f:D→D´ is continuous at $x_0 \varepsilon D$ iff for any open set $V' \subseteq D'$ with $f(x_0) \varepsilon V'$, there is an open set $V \subseteq D$ s.t. $x_0 \varepsilon V$ and $f(V) \subseteq V'$,

f is continuous on D iff εD, f is continuous at x for every x εD.

(3) A continuous function f:D→D´ is qualitatively differentiable for $x_0 \varepsilon A \subseteq D$ for an open subset A of D, iff there is a linear function u:Ext→D´ s.t. for φ:D→D´ with $\forall x \varepsilon D. \phi(x) := f(x_0) + u[dist(x,x_0)]$, the following property of qualitative differentiability holds:
$$\forall x \varepsilon A.\ dist'[f(x),\phi(x)]/dist(x,x_0) = 0$$
$$\Rightarrow \forall x' \varepsilon A.\ dist(x',x_0)/dist(x,x_0) = 0$$

$\Rightarrow dist'[f(x'),\phi(x')]/dist(x',x_0)=0$

If f is qualitatively differentiable at $x_0 \varepsilon A$ then $\phi(x_0)$ is called the qualitative derivative of f at x_0.

Notation. The qualitative derivative of f at x_0 is denoted by $QDf(x_0)$.

Note that $QDf(x_0)$ is defined as a linear form rather than a value, with the derivative obtained from a local linear approximation. This has many advantages over the approach taken in most elementary analysis texts, as pointed out in e.g. [Dieudonne 60]

It is far beyond the scope of this paper to further pursue a qualitative analysis, as it is currently being developed by the author.

5.3. QRL: A Language for Qualitative Reasoning

The language QRL (Qualitative Reasoning Language) combines our current principles and approaches to describe and reason about physico-technical systems. QRL has drawn from concepts in qualitative process theory [Forbus 83], envisionings [de Kleer, Brown 83], and the algebraic software specification language ASPIK [Beierle, et. al. 83]. These developments provided some of the ideas incorporated in the QR-language QARG [Raulefs 83, Mina 84], of which QRL is a substantially revised and extended language.

QRL combines the object and function level (see section 5.1.1) of epistemological domains in the notion of objects and their abstractions in terms of object-types. Processes and process-types comprise the process level.

Spatial aggregation is described by relating objects in terms of the part-/subpart relations, and describing their composition with material (e.g. "glued together..") and spatial relationships. The execution of processes produces histories, which can be decomposed into episodes, and, ultimately, to events. Explicit justifications allowing for teleological reasoning and reason revision are introduced by meta-processes.

QRL is object-oriented. Subsystems of physico-technical systems can be generalized as types for objects and processes. Object- and process-types can be instantiated to objects resp. processes so that any such type denotes a set of objects resp. processes.

5.3.1. Objects

Objects are described in terms of (QRL-keywords capitals)

- structure and geometrical composition by specifying the PARTS objects are composed of. The way an object is incorporated into the external world is described by referring to other external objects (REFOBS-clause). Geometrical properties and relations as well as indications about how PARTS and external objects (REFOBS) are composed are specified in WHERE-clauses.
- an internal state consisting of the qualitative values bound to qualitative variables (QVAR) being explicitly declared. Each QVAR-declaration consists of the name and the range of the qualitative variable.
- Operations that can be executed by the objects. Operations may

denote transitions of internal states. For each operation, at least a non-constructive specification is given by describing properties for initial states required to hold for carrying out the operation, and relationships between initial and final states constraining the effect of the operation. Note that the precision of a non-constructive specification may vary between weak constraints and a unique function from states to states.

- properties describing all properties of the objects which can be expressed in terms of the visible names.

Fig. 5-2 shows the definition of the object-type "wheel" as an example. wheel is declared to be a constant name (permanent binding) bound to the object-type defined in this declaration. The individual clauses express the following descriptions of wheels:

REFOBJ: Wheels are considered to roll on smooth surfaces. ´surface` must be the name of an object(type) which is defined with a qualitative variable ´evenness´ containing the qualitative value ´smooth´ in its range. TOUCHES is a built-in relation in QRL.

PARTS: Wheels consist of spokes and a ring. The number of spokes is left open up to an instantiation, where at least 3 spokes must be parts of a wheel. Spokes are particular bars and the ring is a particular circle with a radius equal to the length of the bars. All spokes have the same length, originate in the center of the ring, and end on the ring.

QVAR: The qualitative variables setting up the state of a wheel are described here with their ranges. Ranges are denoted by

- explicit enumerations of finite sets, as e.g. {metal, plastic, gum} and (small, med, large), "{...} refers to ordinary set notion, and "(...)" constructs a linearly ordered set s.t. small < med < large.
- built-in ranges such as NAT.
- names of qualitative variables with ranges already defined being taken to be the ranges of newly declared variables as well.

Any variable declared with the keyword CONST cannot change its value once the object of the corresponding type has been created. Hence, only the identifiers declared to be variable actually contribute to the internal state of an object. For objects of type ´wheel´, the velocity and direction of rotation, ´rot-vel´ and

'rot-dir', as well as the speed and direction on the surface, 'speed' and 'direction', are variable identifiers, whereas all others are constants.
Views. 'direction' is declared to have a multiple view dimension, i.e. values are either degrees between 0 and 360, or geographical directions, or local directions. Connections between views, and view refinement are expressed in the WHERE-clause.

OPS: In this example, operations are specified non-constructively with PRE- and POST-conditions. Qualitative variables always refer to the internal state before executing the operation, unless they are marked with an apostrophe indicating the internal state after completing the operation.
OP-TRACE is a clause under which the required sequences of operations are specified. Note that this clause is redundant if the operation-sequences given here can be inferred from the pre- and post-conditions of the operation specifications.

PROPS: Properties required for all objects of this type are specified in a logical first-order notation on all visible names.

Instantiation of an object-type is simply done by giving values for all identifiers which are not ex- or implicitly bound. A partial instantiation of an object-type results in another object-type inheriting all components and properties already defined.
The micro-world defined in Fig. 5-2 consists of two wheels w1 and w2 with the same radius, with w1 having three and w2 having seven spokes, and a suface s17 that both w1 and w2 touch. Many other properties such as both wheels being perpendicular to surface s17, their initial rotating velocity and direction, etc. have been left open as it is obvious how to specify them.

```
CONST wheel = OBJECT-TYPE
              REFOBJ surface[evenness = smoth]WHERE wheel TOUCHES surface.
              PARTS  spoke[1..$n: NAT & n>3], ring
                     WHERE ring IS circle[rad=radius], ring TOUCHES surface,
                           spoke IS bar[length=radius, origin=center.ring,
                                 end ON ring], spoke INSIDE ring.
              QVAR CONST radius:(small,med,large), #spokes: NAT& #spokes>3,
                   ring-material: {metal,plastic,rubber};
                   VAR rot-vel: (0,small,med,fast,very-fast),
```

```
                rot-dir: {left,right,undet}, speed:rot-vel,
                direction: [(0..360): NAT, {E,SE,S,SW,W,NW,N,NE},
                            {up,down,left,right}
                            WHERE E IS (340..360)+(0..30),
                                  SE IS (30..70), S IS (70..110),...,
                                  NE,E IS left, SE,S IS UP,... ].

OPS       stop = PRE rot-vel ≠ 0, rot-dir: {left,right}
                 POST rot-vel'=0, rot-dir'=undet

        move(d) = PRE rot-vel=0, rot-dir=undet
                  POST rot-vel'≠0, rot-dir: {left,right}
   accelerate = PRE 0<rot-vel<very-fast, rot-dir≠undet
                POST rot-vel<rot-vel',rot-dir'=rot-dir

          slow = PRE rot-vel≠0
                 POST 0<rot-vel'<rot-vel,rot-dir'=rot-dir.

OP-TRACE  (move;(accelerate|slow)*;stop)*.

PROPS     $n=#spokes, rot-vel=0 <=> rot-dir=undet,
          {equations relating rot-vel and speed,
                              rot-dir and direction, etc.}.

CONST w1 = OBJECT wheel[$n=5,surface=s17,radius=med,ring-material=metal,...];
VAR w2 = OBJECT wheel w1[$n=7];
```

Fig. 5-2. Example of object-type and object definitions.

5.3.2. Processes

Processes denote sets of histories, where each history is a situation structure as introduced in section 3.2.2. Before introducing processes, we need two additional concepts for describing processes:

- history views because of the fact that situations may be refined into different situation structures, and
- interval-types combining time-intervals with similar events and properties.

5.3.2.1. Histories and history views. A history is a situation structure referring to a specific set of objects s.t. all events are names of transitions of internal states in these objects, and properties are propositions about states of the involved objects as well as spatial and physical relationships among them.

As situations may cover intervals containing several events, a situation may be refined into a situation structure, e.g.

$(I|e_1,\ldots,e_5|p)$ refines into
$(I_1|e_1|p_1),(I_2|e_2|p_2),(I_3|e_3|p_3),(I_4|e_4|p_4),(I_5|e_5|p_5)$,
where $beg(I)=beg(I_1)$ and $end(I)=end(I_5)$, and
$\forall i \in (1..5) \,.\, (I_i \| p)$, i.e. p is an invariant on all intervals "covered" by I.

Therefore, histories may refer to different more or less refined history views. Depending on the way elementary (= no further refineable) situations are combined, histories ultimately refined to the same structures of elementary situations may not be refinements of each other, i.e. refinement only constructs a partial ordering on histories.

5.3.2.2. Interval types. Very often, histories contain repetitions of similar situations. For example, a wheel rolling up and down inside a U-shaped surface periodically undergoes similar motions from a left turning point down to the center, up to the right turning point, down to the center, etc.

We collect time intervals containing similar events and/or properties for states in interval types. An interval type denotes the set of all intervals declared to be of that type. Properties and events ascribed to interval types are automatically ascribed to all intervals of that type. Interval types may be used instead of intervals in expressions denoting interval and situation structures. Then, interval types in such an expression may be substituted with intervals of the respective type within an expression. Any such expression containing interval types denotes the set of all interval resp. situation structures obtained by proper substitutions.

5.2.3.2. Example. QRL-mechanisms for describing processes are illustrated in the micro-world of Fig. 5-3: The wheel "w1" of Fig. 5-2 rolls up and down in the interior of a circular surface. Friction causes the wheel to slow down until it stops ultimately at the center point.

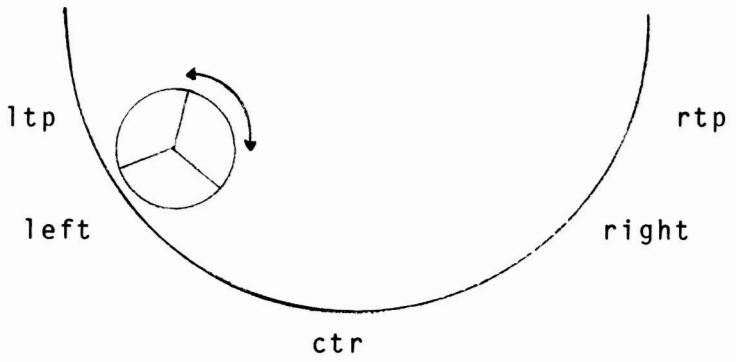


Fig. 5-3. Rolling wheel mirco-world

The particular mechanism of processes are explained at the example of the rolling wheel process (Fig. 5-4) which runs in the micro-world of Fig. 5-3.

OBJ — The wheel 'w1' as defined in Fig. 5-2 and a semicircular surface are involved. The identifier 'w1' is made visible via the REFerence qualification, and the surface object s is created here.

QVAR — Five positions are considered for the wheel (see Fig. 5-3). Because of speed-reduction due to friction, the left and right turning points 'ltp' and 'rtp' gradually move closer to the center 'ctr', starting with a 'very-high' position. 'last-speed' and 'slowdown' serve to take care of speed losses due to friction.

INTV-TYPES — The entire motion of the wheel is broken down into seven phases ascribed to seven different types of intervals:

(1) The motion starts at the left or right turning point, 'ltp' or 'rtp'. The start instance is an interval of type IT-lt resp. IT-rt. As such intervals occur as begin/end of others, they must have extent 0.

(2) The up and down motion takes place in the intervals of type IT.side-up/down with side ε {left,right} so that these intervals start resp. end with the turning point and the center intervals (which are instances).

INTV-STRUCT — This clause specifies a finite but unbounded sequence of intervals for the respective phases of the motion as indicated under INTV-TYPES.

EVENTS — The events listed in this clause must occur in intervals of the specified type, where (...|...) denotes mutually exclusive alternatives. Events

are names for state transitions effected by executing operations as specified in the OPS-clause. The order in which events occur follows from the order in which the correspondiong operations are executed, as it is specified in the OP-TRACE-clause.

OPS

Each operation is specified in three parts:

1. The PREcondition contains all conditions which must hold for the operation to be executable. Conditions consist of properties of the internal state of both the process and any objects involved, an indication of the time interval for which the properties about states apply, and an indication of events effecting the state change described by the operation. Hence, the first part of an operation specification has the form PRE <conditions> IN <time interval> WITH <events>.
2. Actions effecting the state transition. Here, all actions consist in executing operations of the wheel.
3. The POSTcondition specifies conditions which must hold upon completing the operation. Apostrophes after qualitative variables indicate that this identifier denotes values upon starting the operation, whereas identifiers without apostrophes denote values upon starting the operation. Time intervals and events apply analoguously to the prediction.

```
CONST rolling-wheel =
  PROCESS-TYPE
    OBJ  REF w1:wheel, s:surface.w1[shape=semicircular].

    QVAR position: {ltp,rtp,left,right,ctr} INITIAL (ltp|rtp),
         ltp,rtp: {ctr,close-to-ctr,near-ctr,middle,upper-middle,high,very-high}
                                                    INITIAL very-high,
         last-speed: speed.w1, slowdown: boolean

    INTV-TYPES IT-left-down, IT-left-up, IT-right-down, IT-right-up,
               IT-left-turn,IT-right-turn,IT-ctr
               WHERE beg(IT-left-down), end(IT-left-up): IT-left-turn,
                     beg(IT-right-down), end(IT-right-up): IT-right-turn,
                     beg(IT-left-down), end(IT-right-up): IT-ctr.

    INTV-STRUCT (IT-left-down; IT-right-up; IT-right-down; IT-left-up)*.

    EVENTS      start IN (IT-left-turn|IT-right-turn),upturn IN IT-ctr,
                left-turn IN IT-right-turn,right-turn IN IT-left-turn.

OPS           initiate(pos): LET side := (pos=ltp→left, pos=rtp→right) IN
                             PRE position=posε{ltp,rtp} IN IT-side-turn WITH start
                             move.w1(side)
                             POST position'=side,slowdown'=true IN IT-side-turn
        start-rolling(side): PRE positionε{ltp,rtp}, position≠ctr,
                                      IN IT-side-turn WITH side-turn
                             move.w1(side)
                             POST position'=side IN IT-side-down
            roll-down(side): PRE position=side IN IT-side-down
                             accelerate.w1
                             POST position'=ctr, slowdown'=¬slowdonw'IN IT-ctr
              roll-up(side): PRE position=ctr,speed.w1≠0 IN IT-ctr WITH upturn
                             slow.w1
                             POST position'=side
                 turn(side): PRE position=side IN IT-side-up
                             stop.w1
                             POST position'=(side=left→ltp,side=right→rtp),
                                  slowdown→position'<position
                                  IN IT-side-turn WITH side-turn.
    OP-TRACE(initiate;roll-down;roll-up;turn);(start-rolling;roll-down;
             rollup;turn)*.
    PROPS    positionε{ltp,rtp} => rot-vel.w1=0
```

Fig: 5-4. Type of a rolling wheel process

5.3.3. Meta-Processes

For specifying design goals and intentions, we simply use the same mechanisms already introduced for describing systems also for specifying requirements still open for design.

As an example, consider designing a "friction-clock" from the following principle: Friction causes a moving object to stop after a specific amount of time which depends on the initial position and speed of the object. The time elapsing between begin and end of a motion is taken for determining time spans. To specify this principle, we need the concepts of

- a moving object which is slowed down by friction until it ultimately stops.
- a process causing the object to move from particular initial states, determining the time elapsing between begin and end of the motion.

Before specifying our design goal in a meta-process, we specify the notions required to describe it:

```
CONST moving-object =

OBJECT-TYPE QVAR         position, speed:{0;≠0}.
            INTV-TYPES   IT-rest,IT-start-moving,IT-in-motion,IT-stop
                         WHERE beg(IT-in-motion): IT-start-moving,
                               end(IT-in-motion): IT-stop.
            INTV-STRUCT (IT-rest;IT-in-motion)*; IT-rest.
            EVENTS       start IN IT-start-moving, stop IN IT-stop.
            OPS          move     PRE speed=0  POST speed´≠0,
                         slowdown PRE speed≠0  POST speed´<speed,
                         stop     PRE speed≠0  POST speed´=0.
            OP-TRACE     (move;slowdown;stop)*.

CONST fading-motion =

PROCESS-TYPE OBJ      mob: moving-object.
         INTV-TYPES   IT-start, IT-motion-(1..$n), IT-stop
                      WHERE beg(IT-motion-1):IT-start,end(IT-motion-n):IT-stop
         INTV-STRUCT IT-motion-1;...;IT-motion-n.
         EVENTS       start IN IT-start, stop IN IT-stop.
         OPS          start(speed) PRE mob.speed=0 IN IT-start WITH start
                                   move.mob
                                   POST mod.speed´=speed≠0 IN IT-motion-1.
                      slowdown     PRE mob.speed≠0 IN IT-motion-i
```

```
                                    WHERE i IN (1..n-1)
                               slowdown.mob
                               POST mod-speed´≤mod.speed IN IT-motion-(i+1),
                    stop       PRE mod.speed≠0 IN beg(IT-motion-n)
                               stop.mob
                               POST mob.speed´=0 IN IT-stop WITH stop.
          OP-TRACE    start;slowdown;stop.
```

These notions now allow to specify what we want:

```
CONST friction-clock =

META-PROCESS    OBJ     m: moving-object.
                PROC    fm: fading-motion.
                QVAR    elapsed-time.
                OPS     timer(ip:position.m):elapsed-time
                             PRE speed.m=0, position.m=ip
                             start.fm;slowdown.fm;stop.fm
                             POST elapsed-time´=ext[intv(history(fm,speed.m))]
```

Actually, the ´friction-clock´ is a specific approach towards designing a clock. The meta-process clock specifies what we expect from a clock:

```
CONST META-PROCESS clock =

PROCESS  proc(INIT-COND)
   QVAR  elapsed-time
    OPS  clocking(INIT-VAL): elapsed-time
         PRE INIT-VAL IN RANGE
         POST elapsed-time´=ext{intv[history(proc,INIT-COND)]}
```

{INIT-COND,INIT-VAL are reserved words to be substituted with a condition resp. value}

The reasoning system observes that a ´friction-clock´ satisfies what is required from a ´clock´ and comes up with that statement

```
   friction-clock SATISFIES clock FOR fm/proc,f(ip:position.m)/speed.m,
                                      timer/clocking,
                                      ip:position.m/INIT-COND,
                                      speed.m/INIT-VAL
```

where f is an auxiliary function converting initial positions of the moving objects into initial speeds.

A first attempt to realize a 'fading-motion' process is to let a wheel roll down a slope onto a plane until it is stopped by friction. A critic might observe that a disadvantage of this approach is that the plane must possibly of considerable extensions if the friction is low; if the friction is high, however, the precision of timing will decrease. This difficulty is removed by bending the plane upwards so that the wheel will roll back and forth. It is considerably beyond the scope of this paper to discuss the intricacies of automating this kind of reasoning.

Acknowledgements and Ongoing Research

The results reported on qualitative and non-monotonic reasoning have been motivated by the work of Brown, Doyle, Forbus, de Kleer, and Kuipers. Earlier approaches in Kaiserslautern have been explored, revised and elaborated in [Mina 84]. This paper summarizes the state in mid-84 after substantial revisions and extensions, as well as a first experimental implementation of QRL and an earlier prototype version of the conceptual design system CD have been completed and applied in experiments.

Current work, to be completed in early 1985 includes

- a prototype of the QRL-system serving not only for descriptions of, but also for qualitative simulations of and reasoning (incl. NMR) about physico-technical systems.
- experimental applications in conceptual design and production planning in mechanical engineering. Another application concerns portfolio management and planning for capital investments.
- using QRL for deep modelling in fault diagnosis and signal interpretation under real-time conditions.
- foundations of qualitative analysis and qualitative mechanics.

6. References

1. Allen, J.F. Maintaining knowledge about temporal intervals. CACM:26(1983) 832-843.

2. Beierle, Ch., Gerlach, M., Göbel,R., Olthoff,W., Raulefs, P., Voß, A., 1983. Integrated Program Development and Verification in H.-L. Hansen(ed.) Symp. on Software Validation. Amsterdam: North-Holland Publ.Co.

3. Dieudonnē,J. 1960. Foundations of Modern Analysis. New York: Academic Press.

4. Doyle, J. 1979. A Truth Maintenance System. Artificial Intelligence. 12(1979)231-272.

5. Doyle, J. 1980. A Model of Deliberation, Action and Interpretation. TR-581, AI-Lab, MIT.

6. Doyle, J. 1983. Some Theories of Reasoned Assumptions. TR CS-83-125, Dept. of Computer Science, Carnegie-Mellon-University.

7. Etherington, D.W. 1983. Formalizing non-monotonic reasoning systems. Tech. Rept. CS-TR-83-1. Computer Sci. Dept., Univ. of British Colombia.

8. Fikes, R., Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence: 2(1971)189-208.

9. Forbus, K.D. 1981. Qualitative Reasoning about Physical Processes. Proc. 7th IJCAI-81:326-330.

10. Forbus, K.D. 1982. Qualitative Process Theory. AI-Lab Memo AIM-664, MIT.

11. Forbus, K.D. 1983. Measurement Interpretation in Qualitative Process Theory. Proc. 8th IJCAI-83:315-320.

12. Genrich, H.A., Lautenbach, K. 1979. The analysis of distributed systems by means of predicate-transition-nets. Proc. Symp. Semantics of Concurrent Computation (Evian), Springer, LNCS-70.

13. Goodwin, J.W. 1982. An improved algorithm for non-monotonic dependency net update. Tech. Rept. LITH-MAT-R-82-23, Linköping Univ.

14. Hansen, F. 1974. Konstruktionswissenschaft. München:Hanser-Verlag.

15. Hayes, P. 1978. Naive Physics I: Ontology for Liquids. (Working Paper) Inst. of Semantic and Cognitive Studies, Geneva.

16. Hayes, P. 1979. The Naive Physics Manifesto. in D. Michie (ed.) Expert System in the Micro Electronic Age. Edinburgh: Edinburgh Univ. Press.

17. Kahn, K.M., Gorry, A. G. Mechanizing temporal knowledge. Artificial Intelligence: 9(1977)87-108.

18. de Kleer, J., Brown, J.S. 1983. The Origin, Form and Logic of Qualitative Physical Laws. Proc. 8th IJCAI-83:1158-1169.

19. Koller, R. 1971. Konstruktionsmethode für den Maschinenbau--- Heidelberg:Springer-Verlag.

20. Kuipers, B. 1982. Getting the Envisionment Right. Proc. AAAI-82:209-212.

21. Kuipers, B. Kassirer, J.P. 1983. How to Discover a Knowledge Representation for Causal Reasoning by Studying an Expert Physician. Proc. 8th IJCAI-83:49-56.

22. McAllester,D.A. 1980. An Outlook on Truth Maintenance. Memo AIM-551, AI-Lab, MIT.

23. McCarthy, J., Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Machine Intelligence 4(1969), 463-502.

24. McDermott, D., Doyle, D. 1980. Non-Monotonic Logic I. Artificial Intelligence:13(1980).

25. McDermott, D. 1982. A temporal Logic for Reasoning about Processes and Plans. Cognitive Science:6(1982).

26. Mina, I. 1984. Ph.D. - Dissertation (submitted).

27. Moore, R.C. 1983. Semantical Considerations on non-monotonic logic. Proc. 8th IJCAI-83(1983) 272-279 and SRI International, AI-Ctr.TN 284 (1983).

28. Pahl, G. Beitz, W. 1976. Konstruktionslehre. Heidelberg:Springer-Verlag.

29. Raulefs, P. 1983. Basic Ideas on Conceptional Design and Qualitative Reasoning for Physico-Technical Systems. Working Paper, Kaiserslautern Univ.

30. Raulefs, P. 1984. Knowledge-based software engineering. Tech. Rept., FB Informatik, Univ. Kaiserslautern.

31. Rodenacker, W.G. 1976. Methodisches Konstruieren. Heidelberg:Springer-Verlag.

32. Roth, R. 1968. Gliederung und Rahmen einer neuen Maschinen-Geräte-Konstruktionslehre. Feinwerktechnik:72(1968)521-528.

33. Sacerdoti, E.D. 1977. A Structure for Plans and Behavior. New York:Elsevier.

34. Smullyan, R.M. 1970. Formal Systems. New York:Harper and Row.

35. Winskel, G. 1981. Events in Computation (Ph.D.- Thesis). Dept. of Computer Science, Univ. of Edinburgh.

36. Yoshikawa, H. 1983. Automation of Thinking in Design. Proc. CAPE-83(1983):405-407.