# KOMEGA REQUIREMENTS No.1
# Basic Application Scenario

*

Gerd Doeben-Henisch
gerd@doeben-henisch
in cooperation with the INM KOMeGA-Teams

July-25, 2020

**Abstract**

As described in the uffmm eJournal (URL: `https://www.uffmm.org/`) the wider context of this software project is a *generative theory of cultural anthropology [GCA]* which is an extension of the engineering theory called *Distributed Actor-Actor Interaction [DAAI]*. The more detailed considerations to the GCA theory are described in the section *Case Studies* of the uffmm eJournal. There is also a section about *Python co-learning* – mainly dealing with python programming – and a section about a web-server with Dragon. This document will be part of the Case Studies section.

## 1 Basic Application Scenario

Before starting any kind of programming one has to consider, which *application scenario* is the context of the software and what are the detailed *functional and non-functional requirements* which have to be fulfilled to match the intended case. Figure 1 gives a first starting point for the intended application scenario.

**Main Actors:** The main actors in the intended application scenario are some *experts* working as a *group* with the *common intention* to solve a *given problem* $P$ (a task, a question, ...).

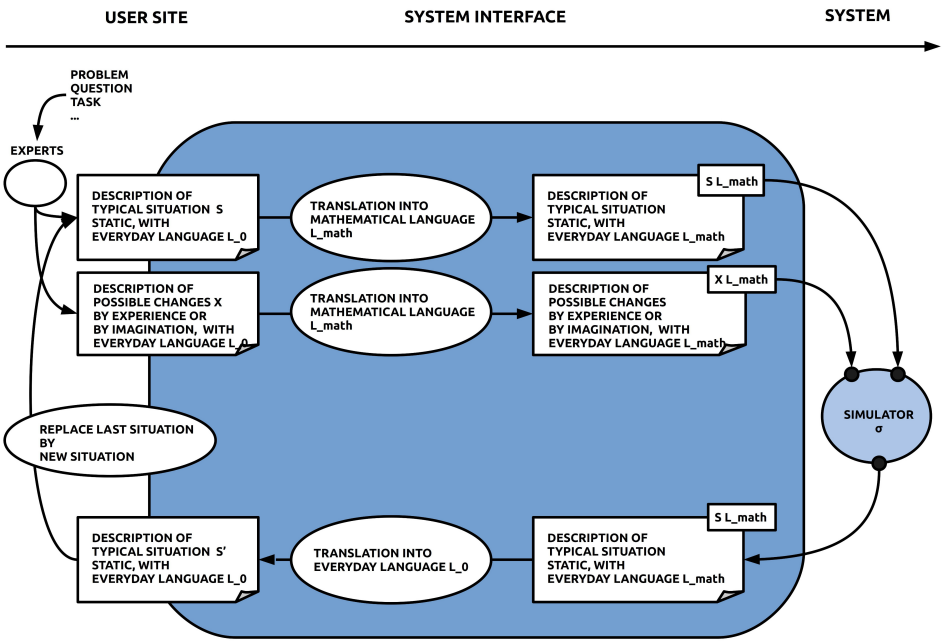


Figure 1: Overview application scenario base-case, version 1

**How to Proceed:** At the *beginning* of the process every *expert* $A_i$ (with 'i' as an index in the range of the number 'n' of experts) has its own *experience*, therein embedded the individual *knowledge*. These experiences are usually mostly *unconscious*. Thus it is needed to start a *process of common communication* to *activate* as much experience as possible from the unconsciousness which is *related* to the problem $P$ in question.

**Target:** This activation process of available knowledge *reaches* its first *end* if the experts could *write down* two *texts* in *everyday language* $L_0$:

1. A description of at least one *static state* $S$ which is a typical part of the intended *problem* $P$.

2. A collection of rules representing a set of *known possible changes* $X$ related to this static state as well as – in some cases – a collection of rules representing a set of *new possible changes* enabling new static states related to the problem $P$.

These two texts are close to the understanding of the experts measured in the light of the used everyday language $L_0$. But for to use a computer to support the thinking of the experts one needs a sufficient *formal language* $L_f$ which can be processed by the computer. To prevent a deep *semantic gap* between the

text of the experts and the formal text for the computer one needs a formal language $L_f$ which fulfills at least the following requirements:

1. The used *formal language* $L_f$ for the computer must be *readable and understandable* for *every speaker-hearer* of the used everyday language without a special training.

2. It must be possible to define a *mapping (translation)* $\tau$ from the text in everyday language $L_0$ to the text written in a *formal language* $L_f$, written as $\tau : L_0 \longmapsto L_f$.

**Translation $\tau$:** It is assumed here that a *subset* $L_{\epsilon*}$ of the *set-theoretical language* $L_\epsilon$ in its full form is sufficient for this task.[1] Thus the translation requirement reduces to the requirement to construct a mapping $\tau$ from the everyday language $L_0$ to a defined subset of the set-theoretical language $L_{\epsilon*}$, written as

$$\tau \quad : \quad L_0 \longmapsto L_{\epsilon*} \tag{1}$$

To make things a little bit more convenient one can define a *hierarchy* of subsets of the everyday language $L_{0.1} \subset L_{0.2} \subset ... \subset L_0$ in a way that one defines the mapping $\tau$ first for the smallest subset $L_{0.1}$, then for the next, and so on. The *final target* should be to map a subset of the everyday language to the set theoretical subset as large as possible.

**Simulator $\sigma$:** If the static state description $S$ as well as the set of *change rules* $X$ has successfully been translated in texts written in the reduced set-theoretical language $L_{\epsilon*}$ then these texts can be processed by the simulator as follows: take the set of change rules $X$ and apply these to a given static state description $S$ in a way that (i) either *nothing will change* in the new state $S'$ or (ii) some *old facts* will be *deleted* before the next state $S'$ or (iii) some *new facts* will be *created* for the next state $S'$. Finally a new state description $S'$ is given which results from the schema $S' = S - Deleted + Created$.

**Re-Translation $\tau^{-1}$ :**

To make the understanding for the experts more convenient an additional translation has to be done:

$$\tau^{-1} \quad : \quad L_{\epsilon*} \longmapsto L_0 \tag{2}$$

[1] The set-theoretical language $L_\epsilon$ is the main language for modern mathematics and is an instance of the general *logic language* $L_{PL}$.

Thus we have a *re-translation* from a formal language to an everyday language. As soon as the new follow-up state $S'$ has been composed the follow-up state will *replace* the old state $S$ and the new state $S'$ will become the new actual state $S$. To allow afterwards some evaluation the whole sequence of computed states can and will be stored as a *history* $H$.