Interfaces for timed model architectures - a pragmatical approach to freshen the discussion

Jochen Wittmann
HTW Berlin, FB2, Umweltinformatik, Wilhelminenhofstr. 75A, 12459 Berlin
wittmann@htw-berlin.de

Abstract:

The paper tries by its praxis-oriented approach to bring some more transparency for the old, very important, and not yet solved problem of model coupling. It knowingly disclaims a complete classification for the potential configurations but it concentrates on a minimum sized set of terms necessary for the discussion. This necessary set of terms consists of the differentiation of real world time, system time, model time, calculation time, and presentation time with the according information about the scale and the mutual relations between these times. Starting with this as a base, it is shown that a complete classification of all possible combinations for coupling will simply by the high number of possibilities to configure a software architecture using modules for these time levels not lead to any constructive solution but at the most to a descriptive one. Classifications found in litterature concentrate on special aspects and find with such a restriction in objectives to a systematical solution. For the design of a software architecture in general the view on the overall systems is essential, however. Thus, a restriction of the solution space will only be possible with regard on the particular case with its particular objectives and determining factors. This implies a precise specification of the demands concerning the interaction between the components of the software system, first. It is outlined that a main danger for writing such specifications is the amalgamation of the level of model description that specifies how the modelled objects communicate and interact as analogon to their behaviour in the real world system with the level of implementation that specifies the communication and the interaction of the components of the simulator that make the model components running but have their own distributed and/or parallel behaviour as well but on their distinct implementation level.

1 What's the problem?

The problem to represent the real physical time [Zöbe08] of the real world (in the sense of Schmidt [Schm84]) within a simulation model and to calculate an according simulation on a computer is a major topic of investigations since the 70s of the last century. In this context, there are the following main steps to observe that figure milestones in the understanding of simulation methods triggered by new developments in computer science techniques, each.

1. model time versus calculation time
   The process paradigm of SIMULA [Rohl73] differentiates between processes that model an active phase within the model but that are paused or sleeping concerning the consumption of calculation time in the CPU of the computer. In this context the term and the semantics of an "event" become important: The event describes a sudden switch in the value of a model quantity going from one model state to another without any delay or time consumption. In reality the execution of the so-defined events lasts per definition 0 time units exactly. However, the simulation, i.e. the execution of an event by calculation the event specification with the aid of a computer will take some calculation time, obviously.

2. Parallel-distributed simulation
   The next step in history of model description and simulation technique was not only to calculate models on a single CPU but to execute them in a distributed and/or parallel architecture (see e.g. [Fuji00]). This implies the existence of a distinguished system time on every calculation node and makes a common control instance necessary to provide correct handling of the common model time in spite of different times on the collaborating nodes. The solution idea consists in sophisticated synchronisation algorithms that consolidate the potentially diverging model times on the calculation nodes to a common model time supposed by the common model specification.

3. Object-oriented model paradigms
   Whereas the second phase of development is mainly interested in improvements in the computer architecture the third phase concentrates on software specific topics with great impact on model specification: the object oriented programming paradigm is introduced. Models are constructed modular- hierarchical now, with an autarcy for the model components as known from objects of a programming language. This change in model description causes new tasks for the simulation runtime system: The cooperation of the model components (or the autonomous objects of the model descriptions) during model simulation has to be organized (see e.g. [Esch90]). This includes the information exchange between the model objects on the one hand, but also their (autonomous) behaviour in time. Within one model the task to coordinate the behaviour of the set of model components included mirrors the task of synchronisation a parallel-distributed execution of a closed, one-component-model executed on several (autonomous) calculation nodes.

4. Modularisation of the simulation system
   The next phase concerns the simulation system level instead of the level of model. There are growing demands for graphical model construction on the one side, and for a very detailed, and sophisticated model data analysis on the other. In this context it seems reasonable to divide the so far monolithic simulation software into modules with specialized functionality that work together to offer support for the common ask "model description, simulation, and analysis of results". The advantage of such a point of view is the possibility to reuse already existing modules specialized on their particular set of functions. Often used combinations contain MatLab as a module for data analysis or a CAD-system as module for graphical model specification. Doing so, the data interfaces get a central importance. The main phases of a simulation study follow one to each other like in a pipeline, such making the data flow quite simple by reducing interdependencies between the modules completely excepted a pure input-output coupling to the neighbours in the data processing chain. However, there are more complex cases to handle, too: A typical example is the animation of simulation results parallel to the running model calculation. Now, the architecture has to care about the temporal relation between the simulatorr and the animation, for example, if the simulation is slower than the animation or if the user has the possibility to interact in the animation and influences thereby the course of the simulation.

5. Cloud Computing
   A culmination of the problems described in the phases 2, 3, and 4 can be predicted when the concept of „cloud-computing" will have reached the area of modeling and simulation. Working in the cloud in combination with object-oriented model specification concepts and a parallel and distributed simulation runtime system allows an arbitrarily configurable architecture of components for model generation, autonomously simulation by arbitrary runtime system configurations coupled with modules for data analysis and data presentation.

In this situation the literature offers detailed investigations to specify and to classify the problems mapping the real world time to a simulation time correctly. This research can be summarized to two main trends that differ substantially in their objectives: The works of Eschenbacher [Esch90], Zeigler [Zeig90], Uhrmacher [Uhrm01], that basically concentrate on a precise specification of modular-hierarchical, parallel and distributed model objects or model components and their proper treatment by an adequate simulation runtime system. On the other hand, there are those approaches that primarily deal with the embedding of simulation models into a complex software architecture and thereby mainly concentrate on the phases 4 and 5. As an example for this approach the work of the group of Straßburger (e.g. [Stras06]) can be mentioned.

This paper aligns itself with these classification and synchronisation tasks but it does not intend to give an new, own solution. It simply tries to give an easily understandable description of the problem first to enable the practitioner to oversee potential problems in a very early state of his project and to give a classification and base for discussion that needs no special mathematical and theoretical background but helps to come to an effective and pragmatical solution that nevertheless does not lack theoretical distinctiveness.

2 What different times are to be distinguished? - Definitions

This section will try to give some precise but informel definitions to specify the different time axes the modeler and the system architect has to care about to integrate the corresponding modules into an integrated software system:

1. real world time
   This means the physical time course of the real world we are living in.
2. system time
   In dependence on the objectives the model has to fulfill, the modeler cuts away parts of the real world and the remaining parts are called the „Real System" according to Schmidt [Schm84]. The understanding of time in this real system has not necessarily to be the same as the time in the real world, e.g. it might differ in scale, it might be just an interval of time, …
3. model time
   The next step determines the time concept valid within the model. Of special interest will be the granularity – the scale – of model time with regard on the implementation of events on the one hand and with regard on the implementation of the model on a computer with its restricted number range on the other hand.
4. calculation time
   This means the time for evaluation of the model description by the simulation runtime system.
5. presentation time
   is the last time to be distinguished and means the time during which the results of the calculated model are presented.

Problems for an implementation of these different times arise by differences in attributes of these times and especially by the relations these times have to each other. It is obvious that all the times have a relation to the real world time because all activities are part of the pysical world. The other pairwise existing relations are potentially unrestricted, however, some restrictions can be recognized by simple practical deliberations: For example should the relation between calculation time and model time allow to answer the questions on the model "in time". This does not necessarily mean the model has to be earlier at a certain point in time than the real world will arrive there because it could easily be imagined to build a model with the objective not to control or adjust the real world but to explain some phenomena that happened in the past … An other classical relation is the slow-motion or fast motion, but a more precise look on the situation opens some lack of specification in this apparently simple relation: Slow and fast motion can be interpreted as a relation between presentation time and system time as well as model time. This very simple example demonstrates the confusion by the commonly used terminology and shows the importance of a correct interpretation for effective solutions.

Furtheron, another interesting relation is that between presentation time and model time (and/or calculation time!). With respect on the scale of these times it might be imagined that the scale of presentation offers more intersecting points than the scale of the model calculation. In this case the presentation can use interpolation methods to fill these intermediate points in time. However, to use a certain interpolation method means nothing else than to use a model assumption for the values of unknown intermediate points. Such models should be written down in the model description section and not be introduced indirectly and intransparently somewhere within the presentation method. In combination with object oriented software architectures that explicitely intend an exchange of the submodules of the architecture, such a concept whould allow the exchange of the presentation module and would result in different model data in dependence on the used presentation. The origin of the differences neither would be transparently found in the model description nor would it be accessible for a proper data interpretation.

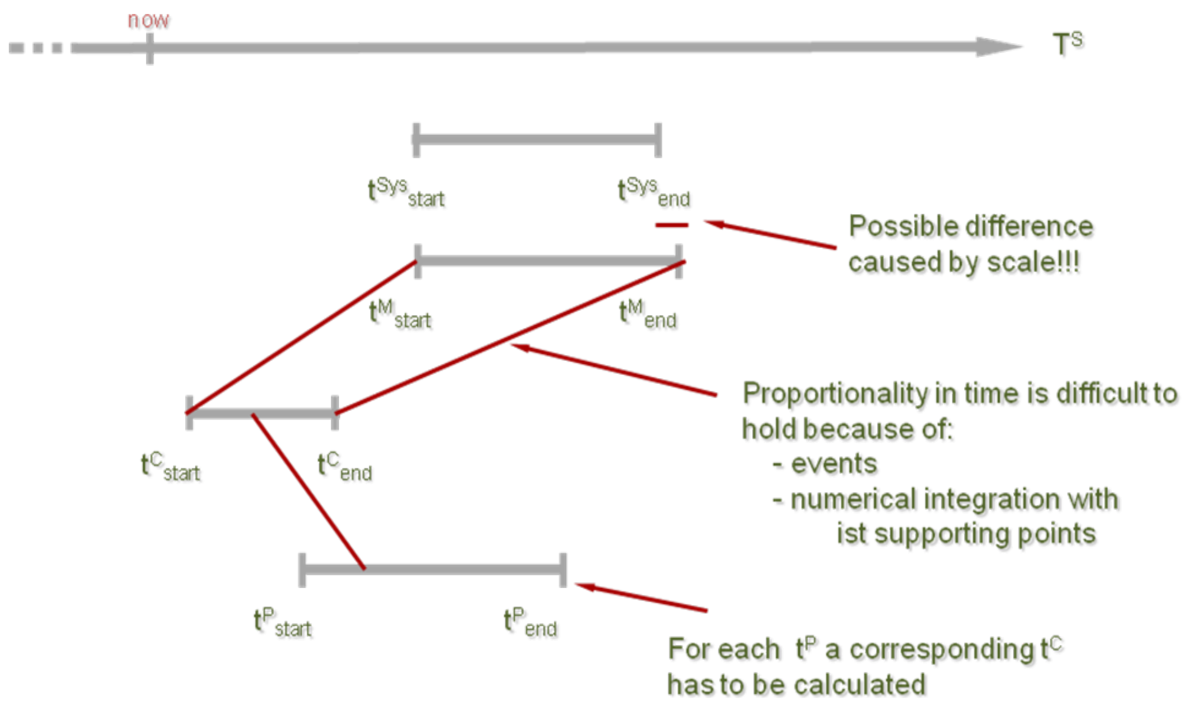
now
T^S
t^{Sys}_{start}
t^{Sys}_{end}
Possible difference caused by scale!!!
t^M_{start}
t^M_{end}
Proportionality in time is difficult to hold because of:
- events
- numerical integration with ist supporting points
t^C_{start}
t^C_{end}
t^P_{start}
t^P_{end}
For each t^P a corresponding t^C has to be calculated

Abb.1: relations between the different time for an example

Figure 1 shows for an example the usage of the terms and definitions and emphasizes the typical problems at the interfaces between the different time levels that are:

- the difference generated by the different scales of the time axes
- the problem of proportionality in time caused by the timeless state transitions given by events and by the execution of numerical integration algorithms that normally use additional intermediate points to integrate over the given time interval deltaT.
- The problem caused by interpolation if the presentation scale does not fit to the model and/or calculation scale.

3 What are the standard situations for the software developer?

With these deliberations the theoretical part of the paper shall be ended already, because the author is definitely convinced that a complete classification of all cases possible on base of the definitions introduced before would be of small benefit or the practical work to be done. Instead of theoretical completeness two representative szenarios will explain where the problems lie to sensibilize a software developer for potential shortcomings of software architectures with embedded simulation models.

A single software component

The easiest case consists in a single software component that handles all the time leves introduced. In this case the problem of the different scales dominates the implementation. It is normally solved by some transformation functions between the scales and doing so these functions are quite obvioous and will not lead to practical complications. However, adding a transformation means always adding additional model assumptions. These model assumptions should not be hidden “somewhere in the interface fuction” but should be explicitely specified, visibly, and transparently be part of the model description itself. Otherwise the architecture realizes an amalgamation of (model-)specification and implementation that strongly should be avoided by means of software technology.

Two software components

The case for n=2 software components demonstrates already by its complexity and number of differentiations necessary that a reasonable common solution will be barely possible. The software developers should be guided through their decisions by the following questions:

- On which of the time levels the two components work on?
- What scales do they have?
- Is there any synchronisation necessary?
  Is a synchronisation by waiting satisfactory or is a reset-mechanism necessary?
- Do the components communicate by messages?
  Do they exchange those messages on the level of the model or on the level of the implementation of the communication between the object-oriented software components?
  Are the messages time-stamped?
  Has a delay by the channel that implements the message passing on technical level to be considered?
  Is the time for message passing part of the model?
  …
- Is there any interpolation necessary?
- …

Main focus has to be the clear separation between the level of model specification on the one hand and the level of the implmentation on the other hand. Such a separation can (by the confirmed opimon of the author) only be realized with regard to the concrete objectives of the model and the experimental design and the restrictions that can be derived there from.

4 Scenarios for typical software architectures

Often practical examples do not need an equal high level of differentiation for the complete task but have to be highly differentiated for a decided spot only. In the following four of those typical spots shall be discussed by applying the terms of the section 2 to prototypical examples of software architectures.

Example: measured value acquisition as a model input

A frequently asked standard situation consists in a online measured value acquisition component for model parametrization.

Thus, the problem of the scales is represented in this example by the treatment of the – technically forced – pulsing of the input. But there the conceptual problems will start: Does the model model the input as a continuous input stream or as a step function? Does it suppose equidistant measured values?

In addition: Independently of the questions and the specification of the pulsing the semantical question has to be answered whether the measured process is continuous or discrete in its time as well as in its value axis.

Furthermore: What is the relation between the intervals for value acquisition and the frequency or resulation that is given or that might be achived by the scale in calculation time. Has to be waited to each other? If yes, how will deal the remaining software components with this waiting time induced by the value acquisation?

Here we have a list of interdepending questions that only will be answered with reasonable effort by a pullback on the level of problem specification and a view on the restricting objectives for the overall project. A general classification seems me to be too elaborate and too less efficient with regard on the high number of cases for differentiation necessary and the need to be familiar with them for a broad number of developers to assure a effective exchange of ideas.

Example: message mechanism between object oriented model components

The main danger in this scenario is a amalgamation between the level of model specification and the technical level of model implementation.

This danger appears first concerning the description of time handling for the message passing: Is it enough to assure a before/after-relation for the messages or are real time stamps necessary? To implement the demand for time proportionality time stampe will be always necessary!

Secondly, the problem comes up with the realization of the message passing in the channel between the model components: Is this message passing part of the model and has to be modelled explizitely therefore or is this message exchange an exchange on implementation level only that has no time consuming equivalent within the model time although it obviously takes some amount of calculation time to execute the message exchange. For this case it becomes difficult to assure the reproducibility of the model results. On the other hand it has to be clarified whether such a reproducibility is necessary for the given task: individual-based simulation games might dispense with the strict reproducibility without constrictions.

Again, a reasonable solution for this scenario can only be found with regard to the functionality for the architecture derived from the objectives of the project in common, in comparison with the effort for the implementation of a proper and adequate coupling.

Example: decoupling and feedback in a model pipeline

Another example (explained in more detail by Himstedt und Wittmann in [Hims10] demonstrates that the analysis of demands might lead to an unusual, but nevertheless well accepted solution in the context of a cooperation project.

The way of a passenger has to be modeled from the home to the take-off of the aircraft at the airport. The different phases of this way are city-traffic for the way to the airport, pedestrian model within the terminal, airtraffic model on the pron and flight coordination at the airport. These phases are handled by different specialized working groups, each of them delivering an autarkic model realized on different model specification paradigms. These models are to be coupled, a task that is quite straight-forward solvable because of the low connectivity and the objective of the project to confine oneself on the observation of scenarios during an one day period: so the models for the phases are basically linked by a list with the number of passengers per hour arriving at the border of the model and each model delivers such a list to its own successor.

Hower, there are few situations where a restricted feedback between the model in pipeline is necessary. With respect to the time scale of 1 hour as the scale for the interface between the models in the pipeline a heuristical approach is proposed and implemented that bases on the concept of an “iteratively propulsed simulation” and realizes the feedback as needed with a low level effort for interface specification and implementation. Thus the black-box charakter of the models of the pipeline can be kept, a fact that is of importance for the general organisation of the cooperation project, but these advantages are bought by a substantial increase in calculation time in case of feedback. For the project context, this deal was sufficient and the higher calculation time was accepted with regard on the main setting “scenario-analysis” without any real-time-requirements.

Example: a physical system as a component „in the loop“

The last example deals with the integration of a physical system into the model architecture as it is useful for instance if a control unit has to been tested in a virtual environment.It is obvious that the timed behaviour of the controller can not be influenced from outside the component, especially it can nor be stopped nor be resetted by those components representing the modelled environment.

That point is the main difference between the controller and the other exemplarily scenarios and connotes a restriction that has to be considered carefully for system design, nterpretation of interfaces and the iimplementation of the system. By the author’s opinion, the concept of object orientation comes to a point where the principle of capsulation and the mutual exchange of components (for the example a modelled control unit by a real physical controller) can not be hold in its most strict formulation for a larger software architecture. Complete modularisation might be realized by high conceptual, spezificatorical, and implementational effort but the costs to do so will be high and the question for efficiency will arise. On the other hand would it also be the wrong way to realize the coupling pragmatically as some piece of code deep in the interface functions because –again- important informations about semantics and/or model specification are described and stored on the wrong level of the software architecture and thereby become hardly accessible, less understandable, and barely discussible.

5 Conclusions

The conclusions shall emphasize the following aspects of the argumentation:

The paper tries by its praxis-oriented approach to bring some more transparency for the old, very important, and not yet solved problem of model coupling. It knowingly disclaims a complete classification for the potential configurations but it concentrates on a minimum sized set of terms necessary for the discussion.

This necessary set of terms consists of the differentiation of real world time, system time, model time, calculation time, and presentation time with the according information about the scale and the mutual relations between these times.

Starting with this as a base, it is obvious that a complete classification of all possible combinations for coupling will simply by the high number of possibilities to configure a software architecture using modules for these time levels not lead to any constructive solution but at the most to a descriptive one. Classifications found in litterature concentrate on special aspects and find with such a restriction in objectives to a systematical solution (e.g. parallelization, individual-based approaches …). For the design of software architecture in general the view on the overall systems is essential, however.

Thus, a restriction of the solution space will only be possible with regard on the particular case with its particular objectives and determining factors. This implies a precise specification of the demands concerning the interaction between the components of the software system, first. It was outlined that a main danger for writing such specifications is the amalgamation of the level of model description that specifies how the modelled objects communicate and interact as analogon to their behaviour in the real world system with the level of implementation that specifies the communication and the interaction of the components of the simulator that make the model components running but have their own distributed and/or parallel behaviour as well but on their distinct implementation level.

The author presents a reduced set of terms and definitions that allows a familiarization with the problem field even for practical oriented modelers and software architects. He focuses on the ubiquitous difficulties in building coupled software systems with modeling and simulation functionalities by giving representative example scenarios and doing so he finally hopes to help to decrease the number of malfunctions within such architectures and to increase at least a little the quality of system specification.

6 References

[Esch90] Eschenbacher, P.: Entwurf und Implementierung einer formalen Sprache zur Beschreibung dynamischer Modelle; Dissertation an der Technischen Fakultät der Universität Erlangen; 1990

[Fuji00] Fujimoto, R. M.: *Parallel and Distributed Simulation Systems*, John Wiley & Sons, 2000

[Hims10] Himstedt, K.; Wittmann, J.; Möller, D.P.F.: Modellkopplung und Szenarioanalyse am Beispiel des Projektes “Effizienter Flughafen 2030”, in: Wittmann, J.; Maretis, D.K. (Edts.): Simulation in Umwelt- und Geowissenschaften: Workshop Osnabrück 2010, AM129, Shaker Verlag, Aachen 2010, pp 91-104

[Rohl73] Rohlfing, H.: SIMULA, Bibliographisches Institut, Mannheim 1973

[Schm84] Schmidt, B.: Systemanalyse und Modellaufbau – Grundlagen der Simulationstechnik, Springer, Berlin 1984

[Stras06] Straßburger, S.: The Road to COTS-Interoperability: From Generic HLA-Interfaces Towards Plug-And-Play Capabilities. In: Perrone, L. F.; Wieland, F. P.; Liu, J.; Lawson, B. G.; Nicol, D. M.; Fujimoto, R. M. (Hrsg.): Proceedings of the 2006 Winter Simulation Conference. Monterey (2006) 1111-1118

[Uhrm01] Uhrmacher A.M.: Dynamic Structures in Modeling and Simulation - A Reflective Approach. ACM Transactions on Modeling and Simulation, Vol.11. No.2:206-232, 2001

[Zeig90] Zeigler, B.P.: Object-Oriented Simulation with Hierarchical, Modular Models. Academic Press, London, 1990

[Zöbe08] Zöbel, D.: *Echtzeitsysteme - Grundlagen der Planung*, 1. Aufl., Springer Verlag, 2008