

Domains

Their Simulation, Monitoring and Control

— A Divertimento of Ideas and Suggestions —

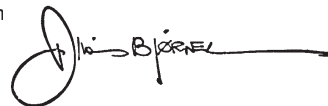
Dines Bjørner
Fredsvvej 11, DK-2840 Holte, Denmark
bjorner@gmail.com

No Institute Given

Abstract

This divertimento – on the occasion of the 70th anniversary of [Prof., Dr Hermann Maurer](#) – sketches some observations over the concepts of domain, requirements and modelling – where abstract interpretations of these models cover both a priori, a posteriori and real-time aspects of the domain as well as 1–1, microscopic and macroscopic simulations, real-time monitoring and real-time monitoring & control of that domain. The reference frame for these concepts are domain models: carefully narrated and formally described domains. I survey more-or-less standard ideas of verifiable development and conjecture product families of demos, simulators, monitors and monitors & controllers – but now these “standard ideas” are recast in the context of core requirements prescriptions being “derived” from domain descriptions.

A Laudatio: This paper is dedicated to Hermann Maurer and is presented on the occasion of his 70th birthday. Hermann and I both spent years at the legendary IBM Labor in Vienna, Austria. Hermann in the 1960s, I in the early 1970s. Hermann went on to do other things than what the Labor became famous for – and contributed significantly to his chosen, foundational and theoretical science — and then, suddenly, Hermann changed somewhat: into highly applications-oriented and almost exclusively technology-oriented work. Again with very significant contributions and now also with decisive industrial and societal impact. I was deeply influenced – and remain so since my days in the early 1970s – by the Vienna work: formal semantics, first of languages, later of systems understood through the languages they exhibit. I take pride and have joy in developing and presenting, to others, the careful English narration and the formalisation the professional languages, i.e., one, crucial aspect of the domains of air traffic, banking, commodities exchange, container lines, the market, railways, etc., etc. Maurer, I am sure, likewise takes pride in the wonderful universes he and his co-workers create for us inside and on the surface of the computing machine, interacting in sometimes unforeseen but always exciting ways. Congratulations Hermann. We never met at Vienna. But I have enjoyed all the m Vienna – across several continents.



1 Introduction

A background setting for this paper is the concern for professionally developing the right software, i.e., software which satisfies users expectations, and software

that is right: i.e., software which is correct with respect to user requirements and thus has no “bugs”, no “blue screens”.

The present paper must be seen on the background of the main line of experimental research around the topics of domain engineering, requirements engineering and their relation. For details I refer to (6, Chaps. 9–16: Domain Engineering, Chaps. 17-24: Requirements Engineering).

The aims of this paper is to present (1) some ideas about software that (1a) “demo”, (1b) simulate, (1c) monitor and (1d) monitor & control domains; (2) some ideas about “time scaling”: demo and simulation time versus domain time; and (3) how these kinds of software relate.

The paper is exploratory. There will be no theorems and therefore there will be no proofs. We are presenting what might eventually emerge into (α) a theory of domains, i.e., a domain science (7; 17; 10; 15), and (β) a software development theory of domain engineering versus requirements engineering (16; 9; 11; 14).

The paper is not a “standard” research paper: it does not compare its claimed achievements with corresponding or related achievements of other researchers – simply because we do not claim “achievements” which have been fully, or at least reasonably well theorised – etcetera. But I would suggest that you might find some of the ideas of the paper (in Sect. 3) worthwhile publishing. Hence the “divertimento” suffix to the paper title.

The structure of the paper is as follows.

In Sect. A we discuss what a domain description is. It would consume too many pages to give a realistic example. Instead we refer to the literature.

In Sect. 3 we then outline a series of interpretations of domain descriptions. These arise, when developed in an orderly, professional manner, from requirements prescriptions which are themselves orderly developed from the domain description¹. The essence of Sect. 3 is (i) the (albeit informal) presentation of such tightly related notions as *demos* (Sect. 3.1), *simulators* (Sect. 3.2), *monitors* (Sect. 3.3) and *monitors & controllers* (Sect. 3.3) (these notions can be formalised), and (ii) the conjectures on a product family of domain-based software developments (Sect. 3.5). A notion of *script-based simulation* extends demos and is the basis for monitor and controller developments and uses. The script used in our example here is related to time, but one can define non-temporal scripts – so the “carrying idea” of Sect. 3 extends to a widest variety of software. We claim that Sect. 3 thus brings these new ideas: a tightly related software engineering concept of *demo-simulator-monitor-controller* machines, and an extended notion of *reference models for requirements and specifications* (22).

2 Domain Descriptions

By a domain description we shall mean a combined narrative, that is, precise, but informal, and a formal description of the application domain **as it is**: no reference to any possible requirements let alone software that is desired for that

¹ We do not show such orderly “derivations” but outline their basics in Sect. 3.4.

domain. (Thus a requirements prescription is a likewise combined narrative, that is, precise, but informal, and a formal prescription of what we expect from a machine (hardware + software) that is to support simple entities, actions, events and behaviours of a possibly business process re-engineering application domain. Requirements expresses a domain **as we would like ti to be.**)

We bring in Appendix A an example domain description.

We further refer to the literature for examples: (4, *railways* (2000)), (5, *the 'market'* (2000)), (11, *public government, IT security, hospitals* (2006) chapters 8–10), (9, *transport nets* (2008)) and (14, *pipelines* (2010)). On the net you may find technical reports (8) covering “larger” domain descriptions. Recent papers on the concept of domain descriptions are (14; 15; 12; 17; 9; 7; 13).

To emphasize: domain descriptions describe domains as they are with no reference to (requirements to) possibly desired software. Domain descriptions do not necessarily describe computable objects. They relate to the described domain in a way similar to the way in which mathematical descriptions of physical phenomena stand to “the physical world”.

3 Interpretations

3.1 What Is a Domain-based Demo?

A *domain-based demo* is a software system which “*present*” (1) simple entities, (2) actions, (3) events and (4) behaviours of a domain. The “*presentation*” abstracts these phenomena and their related concepts in various computer generated forms: visual, acoustic, etc.

Examples A domain description might, as that of Appendix A, be of transport nets (of hubs [street intersections, train stations, harbours, airports] and links [road segments, rail tracks, shipping lanes, air-lanes]), their development, traffic [of vehicles, trains, ships and aircraft], etc. We shall assume such a transport domain description below.

(1) Simple entities are, for example, presented as follows: (*a*) transport nets by two dimensional (2D) road, railway or airline maps, (*b*) hubs and links by highlighting parts of 2D maps and by related photos – and their unique identifiers by labelling hubs and links, (*c*) routes by highlighting sequences of paths (hubs and links) on a 2D map, (*d*) buses by photographs and by dots at hubs or on links of a 2D map, and (*e*) bus timetables by, well, indeed, by showing a 2D bus timetable.

(2) Actions are, for example, presented as follows: (*f*) The insertion or removal of a hub or a link by showing “instantaneous” triplets of “before”, “during” and “after” animation sequences. (*g*) The start or end of a bus ride by showing flashing animations of the appearance, respectively the flashing disappearance of a bus (dot) at the origin, respectively the destination bus stops.

(3) Events are, for example, presented as follows: (*h*) A mudslide [or fire in a road tunnel, or collapse of a bridge] along a (road) link by showing an animation

of part of a (road) map with an instantaneous sequence of (α) the present link, (β) a gap somewhere on the link, (γ) and the appearance of two (“symbolic”) hubs “on either side of the gap”. (i) The congestion of road traffic “grinding to a halt” at, for example, a hub, by showing an animation of part of a (road) map with an instantaneous sequence of the massive accumulation of vehicle dots moving (instantaneously) from two or more links into a hub.

(4) Behaviours are, for example, presented as follows: (k) A bus tour: from its start, on time, or “thereabouts”, from its bus stop of origin, via (all) intermediate stops, with or without delays or advances in times of arrivals and departures, to the bus stop of destination (ℓ) The composite behaviour of “all bus tours”, meeting or missing connection times, with sporadic delays, with cancellation of some bus tours, etc. – by showing the sequence of states of all the buses on the net.

We say that behaviours (3(j)-4(ℓ)) are *script-based* in that they (try to) satisfy a bus timetable (1(e)).

Towards a Theory of Visualisation and Acoustic Manifestation The above examples shall serve to highlight the general problem of visualisation and acoustic manifestation. Just as we need sciences of visualising scientific data and of diagrammatic logics, so ***we need more serious studies of visualisation and acoustic manifestation — so amply, but, this author thinks, inconsistently demonstrated by current uses of interactive computing media.***

3.2 Simulations

“Simulation is the imitation of some real thing, state of affairs, or process; the act of simulating something generally entails representing certain key characteristics or behaviours of a selected physical or abstract system” [Wikipedia] for the purposes of testing some hypotheses usually stated in terms of the model being simulated and pairs of statistical data and expected outcomes.

Explication of Figure 1 Figure 1 attempts to indicate four things: (i) Left top: the rounded edge rectangle labelled “The Domain” alludes to some specific domain (“out there”). (ii) Left middle: the small rounded rectangle labelled “A Domain Description” alludes to some document which narrates and formalises a description of “the domain”. (iii) Left bottom: the medium sized rectangle labelled “A Domain Demo based on the Domain Description” (for short “Demo”) alludes to a software system that, in some sense (to be made clear later) “simulates” “The Domain.” (iv) Right: the large rectangle (a) shows a horizontal time axis which basically “divides” that large rectangle into two parts: (b) Above the time axis the “**fat**” rounded edge rectangle alludes to the time-wise behaviour, a *domain trace*, of “The Domain” (i.e., the actual, or real, domain). (c) Below the time axis there are eight “**thin**” rectangles. These are labels S1, S2, S3, S4, S5, S6, S7 and S8. (d) Each of these denote a “run”, i.e., a time-stamped “execution”, a *program trace*, of the “Demo”. Their “relationship” to the time

axis is this: their execution takes place in the real time as related to that of “The Domain” behaviour.

A *trace* (whether a domain or a program execution trace) is a time-stamped sequence of states: domain states, respectively demo, simulator, monitor and monitor & control states.

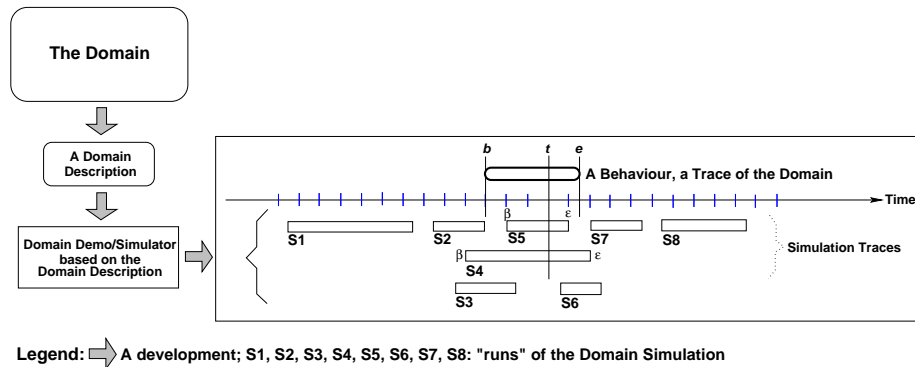


Fig. 1. Simulations

From Fig. 1 and the above explication we can conclude that “executions” S4 and S5 each share exactly one time point, t , at which “The Domain” and “The Simulation” “share” time, that is, the time-stamped execution S4 and S5 reflect a “Simulation” state which at time t should reflect (some abstraction of) “The Domain” state.

Only if the domain behaviour (i.e., trace) fully “surrounds” that of the simulation trace, or, vice-versa (cf. Fig. 1[S4,S5]), is there a “shared” time. Only if the ‘begin’ and ‘end’ times of the domain behaviour are identical to the ‘start’ and ‘finish’ times of the simulation trace, is there an infinity of shared 1–1 times.

In Fig 2 we show “the same” “Domain Behaviour” (three times) and a (1) simulation, a (2) monitoring and a (3) monitoring & control, all of whose ‘begin/start’ (b/β) and ‘end/finish’ (e/ϵ) times coincide. In such cases the “Demo/Simulation” takes place in real-time throughout the ‘begin...end’ interval.

Let β and ϵ be the ‘start’ and ‘finish’ times of either S4 or S5. Then the relationship between t, β, ϵ, b and e is $\frac{t-b}{e-t} = \frac{t-\beta}{\epsilon-t}$ — which leads to a second degree polynomial in t which can then be solved in the usual, high school manner.

Script-based Simulation A script-based simulation is the behaviour, i.e., an execution, of, basically, a demo which, step-by-step, follows a script: that is a prescription for highlighting simple entities, actions, events and behaviours.

Script-based simulations where the script embodies a notion of time, like a bus timetable, and unlike a route, can be thought of as the execution of a demos

where “chunks” of demo operations take place in accordance with “chunks”² of script prescriptions. The latter (i.e., the script prescriptions) can be said to represent simulated (i.e., domain) time in contrast to “actual computer” time. The actual times in which the script-based simulation takes place relate to domain times as shown in Simulations S1 to S8 in Fig. 1 and in Fig. 2(1–3). Traces Fig. 2(1–3) and S8 Fig. 1 are said to be *real-time*: there is a one-to-one mapping between computer time and domain time. S1 and S4 Fig. 1 are said to be *microscopic*: disjoint computer time intervals map into distinct domain times. S2, S3, S5, S6 and S7 are said to be *macroscopic*: disjoint domain time intervals map into distinct computer times.

In order to concretise the above “vague” statements let us take the example of simulating bus traffic as based on a bus timetable script. A simulation scenario could be as follows. Initially, not relating to any domain time, the simulation “demos” a net, available buses and a bus timetable. The person(s) who are requesting the simulation are asked to decide on the ratio of the domain time interval to simulation time interval. If the ratio is 1 a real-time simulation has been requested. If the ratio is less than 1 a microscopic simulation has been requested. If the ratio is larger than 1 a macroscopic simulation has been requested. A chosen ratio of, say 48 to 1 means that a 24 hour bus traffic is to be simulated in 30 minutes of elapsed simulation time. Then the person(s) who are requesting the simulation are asked to decide on the starting domain time, say 6:00am, and the domain time interval of simulation, say 4 hours – in which case the simulation of bus traffic from 6am till 10am is to be shown in 5 minutes (300 seconds) of elapsed simulation time. The person(s) who are requesting the simulation are then asked to decide on the “*sampling times*” or “*time intervals*”: If ‘*sampling times*’ 6:00 am, 6:30 am, 7:00 am, 8:00 am, 9:00 am, 9:30 am and 10:00 am are chosen, then the simulation is stopped at corresponding simulation times: 0 sec., 37.5 sec., 75 sec., 150 sec., 225 sec., 262.5 sec. and 300 sec. The simulation then shows the state of selected entities and actions at these domain times. If ‘*sampling time interval*’ is chosen and is set to every 5 min., then the simulation shows the state of selected entities and actions at corresponding domain times. The simulation is resumed when the person(s) who are requesting the simulation so indicates, say by a “resume” icon click. The time interval between adjacent simulation stops and resumptions contribute with 0 time to elapsed simulation time – which in this case was set to 5 minutes. Finally the requestor provides some statistical data such as numbers of potential and actual bus passengers, etc.

Then two clocks are started: a domain time clock and a simulation time clock. The simulation proceeds as driven by, in this case, the bus time table. To include “unforeseen” events, such as the wreckage of a bus (which is then unable to complete a bus tour), we allow any number of such events to be randomly scheduled. Actually scheduled events “interrupts” the “programmed” simulation

² We deliberately leave the notion of chunk vague so as to allow as wide an spectrum of simulations.

and leads to thus unscheduled stops (and resumptions) where the unscheduled stop now focuses on showing the event.

The Development Arrow The arrow, \Rightarrow , between a pair of boxes (of Fig. 1) denote a step of development: (i) from the domain box to the domain description box it denotes the development of a domain description based on studies and analyses of the domain; (ii) from the domain description box to the domain demo box it denotes the development of a software system — where that development assumes an intermediate requirements box which has not been show; (iii) from the domain demo box to either of a simulation traces it denotes the development of a simulator as the related demo software system, again depending on whichever special requirements have been put to the simulator.

3.3 Monitoring & Control

Figure 2 shows three different kinds of uses of software systems (where (2) [Monitoring] and (3) [Monitoring & Control] represent further) developments from the demo or simulation software system mentioned in Sect. 3.1 and Sect. 3.2.

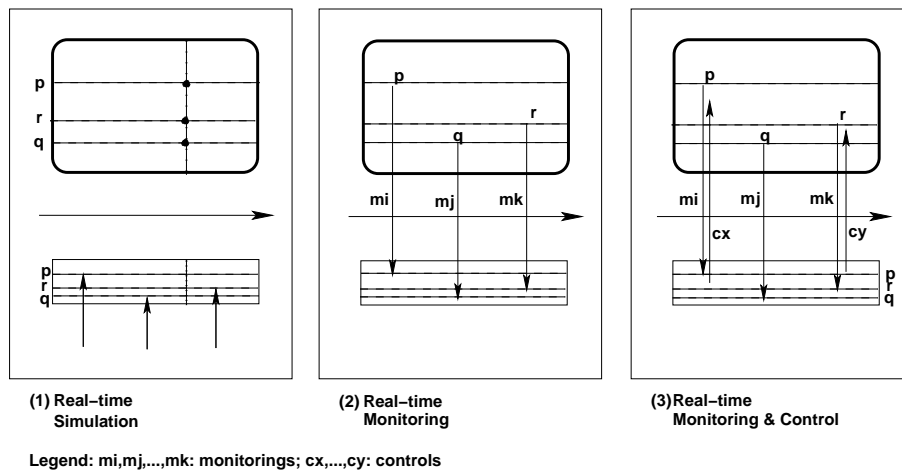


Fig. 2. Simulation, Monitoring and Monitoring & Control

We have added some (three) horizontal and labelled (p, q and r) lines to Fig. 2(1,-2,3) (with respect to the traces of Fig. 1). They each denote a trace of a simple entity, an action or an event, that is, they are traces of values of these phenomena or concepts. A (named) action value could, for example, be the pair of the before and after states of the action and some description of the function (“insertion of a link”, “start of a bus tour”) involved in the action. A (named)

event value could, for example, be a pair of the before and after states of the entities causing, respectively being effected by the event and some description of the predicate (“mudslide”, “break-down of a bus”) involved in the event. A cross section, such as designated by the vertical lines (one for the domain trace, one for the “corresponding” program trace) of Fig. 2(1) denotes a state: a domain, respectively a program state.

Figure 2(1) attempts to show a real-time demo or simulation for the chosen domain. Figure 2(2) purports to show the deployment of real-time software for monitoring (chosen aspects of) the chosen domain. Figure 2(3) purports to show the deployment of real-time software for monitoring as well as controlling (chosen aspects of) the chosen domain.

Monitoring By *domain monitoring* we mean “*to be aware of the state of a domain*”, its simple entities, actions, events and behaviour. Domain monitoring is thus a process, typically within a distributed system for collecting and storing state data. In this process “observation” points — i.e., simple entities, actions and where events may occur — are identified in the domain, cf. points p, q and r of Fig. 2. Sensors are inserted at these points. The “downward” pointing vertical arrows of Figs. 2(2–3), from “the domain behaviour” to the “monitoring” and the “monitoring & control” traces express communication of what has been sensed (measured, photographed, etc.) [as directed by and] as input data (etc.) to these monitors. The monitor (being “executed”) may store these “sensings” for future analysis.

Control By *domain control* we mean “*the ability to change the value*” of simple entities and the course of actions and hence behaviours, including prevention of events of the domain. Domain control is thus based on domain monitoring. Actuators are inserted in the domain “at or near” monitoring points or at points related to these, viz. points p and r of Fig. 2(3). The “upward” pointing vertical arrows of Fig. 2(3), from the “monitoring & control” traces to the “domain behaviour” express communication, to the domain, of what has been computed by the controller as a proper control reaction in response to the monitoring.

3.4 Machine Development

Machines By a *machine* we shall understand a combination of hardware and software. For demos and simulators the machine is “mostly” software with the hardware typically being graphic display units with tactile instruments. For monitors the “main” machine, besides the hardware and software of demos and simulators, additionally includes *sensors* distributed throughout the domain and the technological machine means of *communicating* monitored signals from the sensors to the “main” machine and the processing of these signals by the main machine. For monitors & controllers the machine, besides the monitor machine, further includes actuators placed in the domain and the machine means of computing and communicating control signals to the actuators.

Requirements Development Essential parts of Requirements to a Machine can be systematically “derived” from a Domain description. These essential parts are the *domain requirements* and the *interface requirements*. Domain requirements are those requirements which can be expressed, say in narrative form, by mentioning technical terms only of the domain. These technical terms cover only phenomena and concepts (simple entities, actions, events and behaviours) of the domain. Some domain requirements are *projected*, *instantiated*, made more *deterministic* and *extended*³.

(a) By *domain projection* we mean a sub-setting of the domain description: parts are left out which the requirements stake-holders, collaborating with the requirements engineer, decide is of no relevance to the requirements. For our example it could be that our domain description had contained models of road net attributes such as “the wear & tear” of road surfaces, the length of links, states of hubs and links (that is, [dis]allowable directions of traffic through hubs and along links), etc. Projection might then omit these attributes.

(b) By *domain instantiation* we mean a specialisation of entities (simple, actions, events and behaviours), refining them from abstract simple entities to more concrete such, etc. For our example it could be that we only model freeways or only model road-pricing nets – or any one or more other aspects.

(c) By *domain determination* we mean that of making the domain description cum domain requirements prescription less non-deterministic, i.e., more deterministic (or even the other way around!). For our example it could be that we had domain-described states of street intersections as not controlled by traffic signals – where the determination is now that of introducing an abstract notion of traffic signals which allow only certain states (of red, yellow and green).

(d) By *domain extension* we basically mean that of extending the domain with phenomena and concepts that were not feasible without information technology. For our examples we could extend the domain with bus mounted GPS gadgets that record and communicate (to, say a central bus traffic computer) the more-or-less exact positions of buses – thereby enabling the observation of bus traffic.

Interface requirements are those requirements which can be expressed, say in narrative form, by mentioning technical terms both of the domain and of the machine. These technical terms thus cover shared phenomena and concepts, that is, phenomena and concepts of the domain which are, in some sense, also (to be) represented by the machine. Interface requirements represent (i) the initialisation and “on-the-fly” update of simple machine entities on the basis of *shared* domain entities; (ii) the interaction between the machine and the domain while the machine is carrying out a (previous domain) action; (iii) machine responses, if any, to domain events — or domain responses, if any, to machine events cum “outputs”; and (iv) machine monitoring and machine control of domain phenomena. Each of these four (i–iv) interface requirement facets themselves involve projection, instantiation, determination, extension and fitting.

³ We omit consideration of *fitting*.

Machine requirements are those requirements which can be expressed, say in narrative form, by mentioning technical terms only of the machine. (An example is: visual display units.)

3.5 Verifiable Software Development

An Example Set of Conjectures (A) From a domain, \mathcal{D} , one can develop a domain description \mathbb{D} . \mathbb{D} cannot be verified. It can at most be validated. Individual properties, $\mathbb{P}_{\mathbb{D}}$, of the domain description \mathbb{D} and hence, purportedly, of the domain, \mathcal{D} , can be expressed and possibly proved

$$\mathbb{D} \models \mathbb{P}_{\mathbb{D}}$$

and these may be validated to be properties of \mathcal{D} by observations in (or of) that domain.

(B) From a domain description, \mathbb{D} , one can develop requirements, \mathbb{R}_{DE} , for, and from \mathbb{R}_{DE} one can develop a domain **demo** machine specification \mathbb{M}_{DE} such that

$$\mathbb{D}, \mathbb{M}_{\text{DE}} \models \mathbb{R}_{\text{DE}}.$$

The formula $\mathbb{D}, \mathbb{M} \models \mathbb{R}$ can be read as follows: in order to prove that the Machine satisfies the Requirements, assumptions about the Domain must often be made explicit in steps of the proof.

(C) From a domain description, \mathbb{D} , and a domain demo machine specification, \mathbb{M}_{DE} , one can develop requirements, \mathbb{R}_{SI} , for, and from such a \mathbb{R}_{SI} one can develop a domain **simulator** machine specification \mathbb{M}_{SI} such that

$$(\mathbb{D}; \mathbb{M}_{\text{DE}}), \mathbb{M}_{\text{SI}} \models \mathbb{R}_{\text{SI}}.$$

We have “lumped” $(\mathbb{D}; \mathbb{M}_{\text{DE}})$ as the two constitute the extended domain for which we, in this case of development, suggest the next stage requirements and machine development to take place.

(D) From a domain description, \mathbb{D} , and a domain simulator machine specification, \mathbb{M}_{SI} , one can develop requirements, \mathbb{R}_{MO} , for, and from such a \mathbb{R}_{MO} one can develop a domain **monitor** machine specification \mathbb{M}_{MO} such that

$$(\mathbb{D}; \mathbb{M}_{\text{SI}}), \mathbb{M}_{\text{MO}} \models \mathbb{R}_{\text{MO}}.$$

(E) From a domain description, \mathbb{D} , and a domain monitor machine specification, \mathbb{M}_{MO} , one can develop requirements, \mathbb{R}_{MC} , for, and from such a \mathbb{R}_{MC} one can develop a domain **monitor & controller** machine specification \mathbb{M}_{MC} such that

$$(\mathbb{D}; \mathbb{M}_{\text{MO}}), \mathbb{M}_{\text{MC}} \models \mathbb{R}_{\text{MC}}.$$

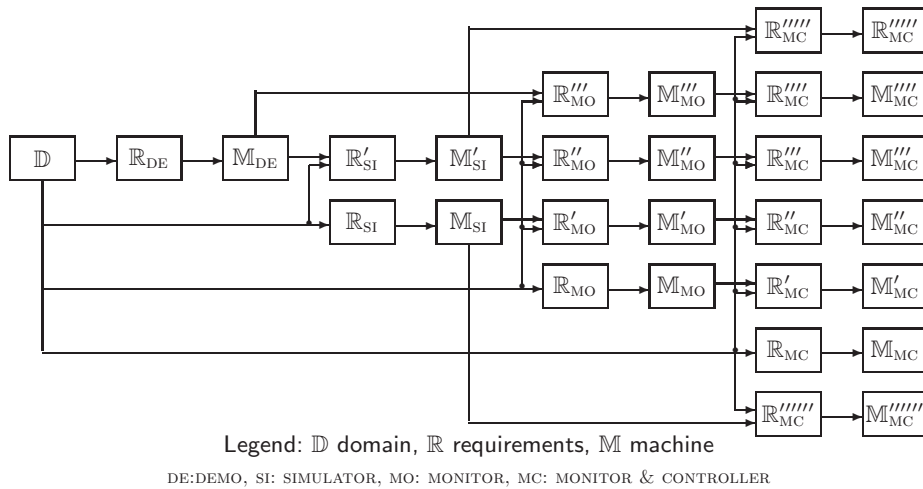


Fig. 3. Chains of Verifiable Developments

Chains of Verifiable Developments The above illustrated just one chain of development. There are others. All are shown in Fig. 3. The above development is shown as the longest horizontal chain (third row).

Figure 3 can also be interpreted as prescribing a widest possible range of machine cum software products (18; 25) for a given domain. One domain may give rise to many different kinds of DEMO machines, SIMULATORS, MONITORS and Monitor & Controllers (the unprimed versions of the \mathbb{M}_T machines (where T ranges over DE, SI, MO, MC)). For each of these there are similarly, “exponentially” many variants of successor machines (the primed versions of the \mathbb{M}_T machines).

What does it mean that a machine is a primed version? Well, here it means, for example, that \mathbb{M}'_{SI} embodies facets of the demo machine \mathbb{M}_{DE} , and that \mathbb{M}''_{MC} embodies facets of the demo machine \mathbb{M}_{DE} , of the simulator \mathbb{M}'_{SI} , and the monitor \mathbb{M}''_{MO} . Whether such requirements are desirable is left to product customers and their software providers (18; 25) to decide.

4 Conclusion

Our divertimento is almost over. It is time to conclude.

4.1 Discussion

The $\mathbb{D}, \mathbb{M} \models \mathbb{R}$ (‘correctness’ of) development relation appears to have been first indicated in the Computational Logic Inc. **Stack** (1; 21) and the EU ESPRIT **ProCoS** (2; 3) projects; (22) presents this same idea with a purpose much like ours, but with more technical details and full discussion.

The term ‘domain engineering’ appears to have at least two meanings: the one used here (7; 13) and one (23; 20; 19) emerging out of the Software Engi-

neering Institute at CMU where it is also called *product line engineering*⁴. Our meaning, is, in a sense, more narrow, but then it seems to also be more highly specialised (with detailed description and formalisation principles and techniques). Fig. 3 illustrates, in capsule form, what we think is the CMU/SEI meaning. The relationship between, say Fig. 3 and *model-based software development* seems obvious but need be explored.

What Have We Achieved We have characterised a spectrum of strongly domain-related as well as strongly inter-related (cf. Fig. 3) software product families: *demons*, *simulators*, *monitors* and *monitor & controllers*. We have indicated varieties of these: simulators based on demons, monitors based on simulators, monitor & controllers based on monitors, in fact any of the latter ones in the software product family list as based on any of the earlier ones. We have sketched temporal relations between simulation traces and domain behaviours: *a priori*, *a posteriori*, *macroscopic* and *microscopic*, and we have identified the real-time cases which lead on to monitors and monitor & controllers.

What Have We Not Achieved — Some Conjectures We have not characterised the software product family relations other than by the $\mathbb{D}, \mathbb{M} \models \mathbb{R}$ and $(\mathbb{D}; \mathbb{M}_{xyz}), \mathbb{M} \models \mathbb{R}$ clauses. That is, we should like to prove conjectured type theoretic inclusion relations like:

$$\wp(\llbracket \mathcal{M}_{x_{\text{mod ext.}}} \rrbracket) \supseteq \wp(\llbracket \mathcal{M}'_{x_{\text{mod ext.}}} \rrbracket), \quad \wp(\llbracket \mathcal{M}'_{x_{\text{mod ext.}}} \rrbracket) \supseteq \wp(\llbracket \mathcal{M}''_{x_{\text{mod ext.}}} \rrbracket)$$

where x and y range appropriately, where $\llbracket \mathcal{M} \rrbracket$ expresses the meaning of \mathcal{M} , where $\wp(\llbracket \mathcal{M} \rrbracket)$ denote the space of all machine meanings and where $\wp(\llbracket \mathcal{M}_{x_{\text{mod ext.}}} \rrbracket)$ is intended to denote that space modulo (“free of”) the y facet (here *ext.*, for extension).

That is, it is conjectured that the set of more specialised, i.e., n primed, machines of kind x is type theoretically “contained” in the set of m primed (unprimed) x machines ($0 \leq m < n$).

There are undoubtedly many such interesting relations between the DEMO, SIMULATOR, MONITOR and MONITOR & CONTROLLER machines, unprimed and primed.

What Should We Do Next This paper has the subtitle: *A Divertimento of Ideas and Suggestions*. It is not a proper theoretical paper. It tries to throw some light on families and varieties of software, i.e., their relations, and. It focuses, in particular, on so-called DEMO, SIMULATOR, MONITOR and MONITOR & CONTROLLER software and their relation to the “originating” domain, i.e., that in which such software is to serve, and hence that which is being *extended* by such software, cf. the compounded ‘domain’ $(\mathbb{D}; \mathbb{M}_i)$ of in $(\mathbb{D}; \mathbb{M}_i), \mathbb{M}_j \models \mathbb{D}$. These notions should

⁴ http://en.wikipedia.org/wiki/Domain_engineering.

be studied formally. All of these notions: requirements projection, instantiation,

Bibliography

- [1] W.R. Bevier, W.A. Hunt Jr., J Strother Moore, and W.D. Young. An approach to system verification. *Journal of Automated Reasoning*, 5(4):411–428, December 1989. Special Issue on System Verification.
- [2] Dines Bjørner. A ProCoS Project Description. *Published in two slightly different versions: (1) EATCS Bulletin, October 1989, (2) (Ed. Ivan Plander:) Proceedings: Intl. Conf. on AI & Robotics, Strebse Pleso, Slovakia, Nov. 5-9, 1989, North-Holland, Publ., Dept. of Computer Science, Technical University of Denmark, October 1989.*
- [3] Dines Bjørner. Trustworthy Computing Systems: The ProCoS Experience. In *14'th ICSE: Intl. Conf. on Software Eng., Melbourne, Australia*, pages 15–34. ACM Press, May 11–15 1992.
- [4] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, VDI-Gesellschaft für Fahrzeug- und Verkehrstechnik. Invited talk.
- [5] Dines Bjørner. Domain Models of "The Market" — in Preparation for E-Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press. .
- [6] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
- [7] Dines Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.
- [8] Dines Bjørner. Domain Descriptions (technical reports):
 1. *On Development of Web-based Software. A Divertimento of Ideas and Suggestions*: <http://www2.imm.dtu.dk/~db/wdfdfp.pdf>
 2. *XVSM: A Narrative and a Formalisation*:
<http://www2.imm.dtu.dk/~db/xvsm-p.pdf>
 3. *The Tokyo Stock Exchange* <http://www2.imm.dtu.dk/~db/todai/tse-1.pdf>
and <http://www2.imm.dtu.dk/~db/todai/tse-2.pdf>
 4. *What is Logistics*: <http://www2.imm.dtu.dk/~db/logistics.pdf>
 5. *A Domain Model of Oil Pipelines*:
<http://www2.imm.dtu.dk/~db/pipeline.pdf>
 6. *A Container Line Industry Domain*:
<http://www2.imm.dtu.dk/~db/container-paper.pdf>
 7. *A Railway Systems Domain*: <http://www.railwaydomain.org/PDF/tb.pdf>
 8. *CoMet: Comparative Methodology, A Technical Note. Transport Systems*:
<http://www2.imm.dtu.dk/~db/comet/comet1.pdf>.R&D Experiments, Bjørner, Fredsvej 11, DK-2840 Holte, Denmark, 2007–2010.
- [9] Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, pages 1–30, Heidelberg, May 2008. Springer.

- [10] Dines Bjørner. An Emerging Domain Science – A Rôle for Stanisław Leśniewski's Mereology and Bertrand Russell's Philosophy of Logical Atomism. *Higher-order and Symbolic Computation*, 2009.
- [11] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. Research Monograph (# 4); JAIST Press, 1-1, Asahidai, Nomi, Ishikawa 923-1292 Japan, 2009. This Research Monograph contains the following main chapters:
1. *On Domains and On Domain Engineering – Prerequisites for Trustworthy Software – A Necessity for Believable Management*, pages 3–38.
 2. *Possible Collaborative Domain Projects – A Management Brief*, pages 39–56.
 3. *The Rôle of Domain Engineering in Software Development*, pages 57–72.
 4. *Verified Software for Ubiquitous Computing – A VSTTE Ubiquitous Computing Project Proposal*, pages 73–106.
 5. *The Triptych Process Model – Process Assessment and Improvement*, pages 107–138.
 6. *Domains and Problem Frames – The Triptych Dogma and M.A.Jackson's PF Paradigm*, pages 139–175.
 7. *Documents – A Rough Sketch Domain Analysis*, pages 179–200.
 8. *Public Government – A Rough Sketch Domain Analysis*, pages 201–222.
 9. *Towards a Model of IT Security — – The ISO Information Security Code of Practice – An Incomplete Rough Sketch Analysis*, pages 223–282.
 10. *Towards a Family of Script Languages – – Licenses and Contracts – An Incomplete Sketch*, pages 283–328.
- Bjørner will post this 507 page soft cover book (with 77 fine photos of “all things Japanese”, in full colours, taken by Dines in 2006) to you provided you e-mail your name and address and post international reply postage coupons, in the total amount of: Denmark 60.50 Kr., Europe 126.00 Kr., elsewhere 209.00 Kr.
http://en.wikipedia.org/wiki/International_reply_coupon.
- [12] Dines Bjørner. On Mereologies in Computing Science. In *Festschrift for Tony Hoare, History of Computing* (ed. Bill Roscoe), London, UK, 2009. Springer.
- [13] Dines Bjørner. Domain Engineering. In *BCS FACS Seminars, Lecture Notes in Computer Science, the BCS FAC Series* (eds. Paul Boca and Jonathan Bowen), pages 1–42, London, UK, 2010. Springer.
- [14] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, (2), May 2010.
- [15] Dines Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny analiz*, (2), May 2010.
- [16] Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.
- [17] Dines Bjørner and Asger Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.
- [18] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley, New York, NY, 2000.
- [19] F. Buschmann, K. Henney, and D.C. Schmidt. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. John Wiley & Sons Ltd., England, 2007.

- [20] R. Falbo, G. Guizzardi, and K.C. Duarte. An Ontological Approach to Domain Engineering. In *Software Engineering and Knowledge Engineering*, Proceedings of the 14th international conference SEKE'02, pages 351–358, Ischia, Italy, July 15–19 2002. ACM.
- [21] Don I. Good and William D. Young. Mathematical Methods for Digital Systems Development. In *VDM '91: Formal Software Development Methods*, pages 406–430. Springer-Verlag, October 1991. Volume 2.
- [22] Carl A. Gunter, Elsa L. Gunter, Michael A. Jackson, and Pamela Zave. A Reference Model for Requirements and Specifications. *IEEE Software*, 17(3):37–43, May–June 2000.
- [23] M. Harsu. A Survey on Domain Engineering. Review, Institute of Software Systems, Tampere University of Technology, Finland, December 2002.
- [24] C.A.R. Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: <http://www.usingcsp.com/cspbook.pdf> (2004).
- [25] K. Pohl, G. Bockle, and F. van der Linden. *Software Product Line Engineering*. Springer, Berlin, Heidelberg, New York, 2005.

A An Example Domain Description

A domain description is a specification of the domain as it is, without any reference to requirements, let alone required software.

A.1 Nets

We first describe abstraction of nets, hubs (street intersections, train stations, airports, harbours) and links (street segments, rail tracks, air lanes, sea lanes):

Hubs and Links

1. There are nets, hubs and links.
2. A net contains zero, one or more hubs.
3. A net contains zero, one or more links.

```

type
  1. N, H, L
value
  2. obs_Hs: N → H-set
  3. obs_Ls: N → L-set
axiom
  2. ∀ n:N • card obs_Hs(n) ≥ 0
  3. ∀ n:N • card obs_Ls(n) ≥ 0

```

Hub and Link Identifiers

To express the mereology (12): how parts compose into a whole, the connections of hubs and links, we introduce the abstract concepts of hub and link identifiers.

4. There are hub identifiers and there are link identifiers.
5. Hubs of a net have unique hub identifiers.
6. Links of a net have unique link identifiers.

```

type
  4. HI, LI
value
  5. obs_HI: H → HI
  6. obs_LI: L → LI
axiom
  5. ∀ n:N, h, h':H • {h, h'} ⊆ obs_Hs(n) ∧ h ≠ h' ⇒
      obs_HI(h) ≠ obs_HI(h')
  6. ∀ n:N, l, l':L • {l, l'} ⊆ obs_Ls(n) ∧ l ≠ l' ⇒
      obs_LI(l) ≠ obs_LI(l')

```

Observability of Hub and Link Identifiers

We postulate reasonable observer functions: such which a person with a reasonably good sight could "implement".

7. From every hub (of a net) we can observe the identifiers of the zero, one or more distinct links (of that net) that the hub is connected to.
8. From every link (of a net) we can observe the identifiers of the exactly two (distinct) hubs (of that net) that the link is connected to.

```

value
  7. obs_LI: H → LI-set
axiom
  7. ∀ n:N, h:H • h ∈ obs_Hs(n) ⇒ ∀ li:LI • li ∈ obs_LI(h) ⇒ L_exists(li)(n)
value
  8. obs_HI: L → HI-set
axiom
  8. ∀ n:N, l:L • l ∈ obs_Ls(n) ⇒
      card obs_HI(l) = 2 ∧ ∀ hi:HI • hi ∈ obs_HI(l) ⇒ H_exists(hi)(n)
value
  L_exists: LI → N → Bool
  L_exists(li)(n) ≡ ∃ l:L • l ∈ obs_Ls(n) ∧ obs_LI(l) = li
  H_exists: HI → N → Bool
  H_exists(hi)(n) ≡ ∃ h:H • h ∈ obs_Hs(n) ∧ obs_HI(h) = hi

```

If we had chosen an ability to observe from a hub its connected links and from a link its connected hubs, then it would follow that from any hub (or any link), without "moving" one could observe the entire net; we find that kind of "observability" to be problematic and to, potentially leading to inconsistencies.

Net Descriptors

9. A net descriptor, ND, associates to each hub identifier a possibly empty link-to-hub identifier map, LHIM, from the identifier of a link emanating from a hub to the identifier of the connected hub.

type
 9. $ND = HI \xrightarrow{m} LHIM$
 10. $LHIM = LI \xrightarrow{m} HI$

The hub identifiers of the definition set of net descriptors are called the defining occurrences of hub identifiers. The hub identifiers of the range of link-to-hub identifier map are called the using occurrences of hub identifiers.

11. Wellformedness of a net descriptor is simple.
 a) The set of using occurrences of hub identifiers must be a subset of the set of defining occurrences of hub identifiers.
 b) If in $nd:ND$ an hi maps into some li which in turn maps into hi' , then in $nd:ND$ hi' , amongst other link identifiers maps into li which in turn maps into hi .

value
 11. $wf_ND: ND \rightarrow \mathbf{Bool}$
 11. $wf_ND(nd) \equiv$
 11a. $\{ (nd(hi))(j) \mid hi:HI,li:LI \bullet hi \in \mathbf{dom} \ nd \wedge li \in \mathbf{dom} \ nd(hi) \} \subseteq \mathbf{dom} \ nd$
 11b. $\bigwedge \forall hi,hi':HI,li:LI \bullet hi \in \mathbf{dom} \ nd \wedge nd(hi)=li \Rightarrow$
 11c. $(nd(hi))(li)=hi' \Rightarrow (nd(hi'))(li)=hi$

12. From a net one can extract its net descriptor.

value
 12. $xtr_ND: N \rightarrow ND$
 12. $xtr_ND(n) \equiv$
 12. $[hi \mapsto [li \mapsto hi'] \mid li:LI,li':LI,hi,hi':HI \bullet$
 12. $li \in \mathbf{obs_Ls}(n) \wedge li = \mathbf{obs_Ll}(l) \wedge \{ hi, hi' \} = \mathbf{obs_Hls}(l)]$
 12. $\cup [hi \mapsto [] \mid h:HI \bullet h \in \mathbf{obs_Hls}(n) \wedge \mathbf{obs_Ll}(h) = \{ \}]$

Routes

We first define a concept of paths.

13. A path is a triple:
 a) a hub identifier, h_i , a link identifier, l_j , and another hub identifier, h_k , distinct from h_i .
 b) such that there is a link ℓ with identifier l_j in a net n such that $\{h_i, h_k\}$ are the hub identifiers that can be observed from ℓ .

type
 13. $Pth = HI \times LI \times HI$
axiom
 13a. $\forall (hi,li,hi'):Pth \bullet \exists n:N,li:L \bullet li \in \mathbf{obs_Ls}(n) \Rightarrow$
 13b. $\mathbf{obs_Ll}(l) = li \wedge \mathbf{obs_Hls}(l) = \{hi,hi'\}$

14. From a net one can extract all its paths:
 a) if l is a link of the net,
 b) l_j its identifier and
 c) $\{h_i, h_k\}$ the identifiers of its connected hubs,
 d) then (h_i, l_j, h_k) and (h_k, l_j, h_i) are paths of the net.

value
 14. $paths: N \rightarrow Pth\text{-set}$
 14a. $paths(n) \equiv$
 14d. $\{ (hi,lj,hk), (hk,lj,hi) \mid l,lj:LI,hi,hk:HI \bullet li \in \mathbf{obs_Ls}(n) \wedge$
 14b. $lj = \mathbf{obs_Ll}(l) \wedge$
 14c. $\{hi,hk\} = \mathbf{obs_Hls}(l) \}$

15. From a net descriptor one can (likewise) extract all its paths:
 a) Let h_i, h_k be any two distinct hub identifiers of the net descriptor (definition set),
 b) such that they both map into a link identifier l_j ,

- c) then (h_i, l_j, h_k) and (h_k, l_j, h_i) are paths of the net.

value
 14. $paths: ND \rightarrow Pth\text{-set}$
 14. $paths(nd) \equiv$
 15a. $\{ (hi,lj,hk), (hk,lj,hi) \mid hi,hk:HI,lj:LI \bullet hi \neq hk \wedge \{hi,hk\} \subseteq \mathbf{dom} \ nd \Rightarrow$
 15b. $lj \in \mathbf{dom} \ nd(hi) \cap \mathbf{dom} \ nd(hk) \}$

Now we can define routes.

16. A route of a net is a sequence of zero, one or more paths such that
 a) all paths of a route are paths of the net and
 b) adjacent paths in the sequence "share" hub identifiers.

type
 16. $R = Pth^*$
axiom
 16. $\forall r:R, \exists n:N \bullet$
 16a. $\mathbf{elems} \ r \subseteq \mathbf{paths}(n) \wedge$
 16b. $\forall i:Nat \bullet \{i,i+1\} \subseteq \mathbf{inds} \ r \Rightarrow$
 16c. $\mathbf{let} \ (_ _)=r(i), (_ _)=r(i+1) \ \mathbf{in} \ hi=hi' \ \mathbf{end}$

From a net, n , we can generate the possibly infinite set of finite and possibly infinite routes:

- a) $\langle \rangle$ is a route (basis clause 1);
 b) if p is a path of n then $\langle p \rangle$ is a route of n (basis clause 2);
 c) if r and r' are non-empty routes of n
 d) and the last h_i of r is the same as the first h_j of r' then the concatenation of r and r' is a route (induction clause).
 e) Only such routes which can be formed by a (finite, respectively infinite) application of basis clauses Items 17a and 17b and induction clause Items 17c–17d are routes (extremal clause).

value
 17. $routes: N \mid ND \rightarrow R\text{-infset}$
 17. $routes(nond) \equiv$
 17a. $\mathbf{let} \ rs = \{ \langle \rangle \} \cup$
 17b. $\{ \langle p \rangle \mid p:Pth \bullet p \in \mathbf{paths}(nond) \} \cup$
 17c. $\{ r \frown r' \mid r,r':R \bullet \{ r,r' \} \subseteq rs \wedge$
 17d. $\exists hi,hi',hi'',hi''':HI,li:LI,lj,lj',lj'',lj''':R \bullet \{ r,r' \} \subseteq rs \wedge$
 17e. $r = r' \frown \langle (hi,li,hi') \rangle \wedge r' = \langle (hi'',li'',hi''') \rangle \frown r'' \wedge hi'' = hi'''$
 17e. $rs \ \mathbf{end}$

A.2 Buses, Bus Stops and Bus Schedules

Buses

We now consider buses and routes and schedules related to buses.

18. Buses have unique identifiers and are further undefined.
 19. Bus identifiers can be observed from buses.

type
 18. B, BI
value
 19. $\mathbf{obs_Bl}: B \rightarrow BI$

Bus Stops

20. A link bus stop indicates the link (by its identifier), the from and to hub identifiers of the link,
 21. and the fraction "down the link" (from the hub of the from to the hub of the to hub identifiers) of the bus stop position.

type
 20. $BS = mk_L_BS(\mathbf{sel_fhi}:HI,\mathbf{sel_li}:LI,\mathbf{sel_f}:F,\mathbf{sel_lhi}:HI)$
 20. $F = \{ |f:Real \bullet 0 < f < 1 \}$

Bus Stop Lists and Routes

22. A bus stop list is a sequence of two or more bus stops, bsl .
23. A bus route, br , is a pair of a net route, r , and a bus stop list, bsl , such that route r is a route of n and such that bsl is embedded in r .
24. bsl is embedded in r if
 - a) there exists an index list, il , of ascending indices of the route r and of the length of bsl
 - b) such that the i th path of r
 - c) share from and to hub identifiers and link identifier with the $il(i)$ th bus stop of bsl .
25. We must allow for two or more stops along a bus route to be adjacent on the same link — in which case the corresponding fractions must likewise be ascending.

```

value
  n:N
type
  BSL = {|bsl:BS*•len bsl≥2|}
  BR = {|(r,bsl):(R×BSL)•r ∈ routes(n)∧is_embedded_in(r,bsl)|}
value
  24. is_embedded_in: BR → Bool
  24. is_embedded_in(r,bsl)(n) ≡
  24a. ∃ il:Nat*•len il=len bsl ∧
  24b. inds il⊆inds r ∧ ascending(il) ⇒
  24c. let (hj,lj,hk) = r(il(i))
  24d. (hj',lj',fk) = bsl(i) in
  24e. hi=hi' ∧ lj=lj' ∧ hk=hk' end ∧
  25. ∃ i:Nat•{i,i+1}⊆inds il ⇒
  25. let (hj,lj,f,hk)=bsl(i), (hj',lj',f',hk')=bsl(i+1) in
  25. hi=hi' ∧ lj=lj' ∧ hk=hk' ⇒ f<f' end
  24a. ascending: Nat* → Bool
  24a. ascending(il) ≡ ∃ i:Nat•{i,i+1}⊆inds il ⇒ il(i)<il(i+1)

```

Bus Schedules

26. Let us introduce a net. It is referred to in some subsequent wellformedness predicates.
27. A timed bus stop is a pair of a time and a bus stop.
28. A timed bus stop list is a sequence of timed bus stops.
29. A bus schedule is a pair of a route and an embedded timed bus stop list where
30. position-wise “earlier” bus stops occur at earlier times than Position-wise “later” bus stops.

```

value
  26. n:N
type
  TBS :: sel_T:T sel_bs:BS
  TBSL = TBS*
  29. BusSched = {|(r,tbsl):(R×TBSL)•r ∈ routes(n)∧wf_BusSched(r,tbsl)|}
  30. SimBusSched = {|tbsl:TBSL•wf_TBSL(tbsl)|}
value
  29. wf_BusSched: BusSched → Bool
  29. wf_BusSched(r,tbsl) ≡
  29. is_embedded_in(r,(sel_BS(tbsl(i))|i:[1..len tbsl])) ∧
  30. wf_SimpleBusSched(tbsl)
  30. wf_SimpleBusSched: TBSL → Bool
  30. wf_SimpleBusSched(tbsl) ≡
  30. ∃ i:Nat•{i,i+1}⊆inds tbsl ⇒ sel_T(tbsl(i))<sel_T(tbsl(i+1))

```

A.3 Timetables

31. A bus b that plies a bus schedule starting at time t has a unique bus number, b_t ; colloquially it is bus b at departure time t , or, even more colloquially: the t o'clock bus b — but henceforth we do not “encode” such bus “numbers”.
32. A [time]table maps bus numbers to bus schedules.
33. A bus timetable is a pair of a net descriptor and a table.

```

type
  31. BNo
  32. TBL = BNo → BusSched
  33. BTT = ND × TBL

```

Denotations

What are routes and bus timetables scripting (i.e., prescribing)? Routes (lists of connected link traversal designations) script that one may transport people or freight along the sequence of designated links. Bus timetable scripts denote (at least) two things: the set of bus traffics on the net which satisfy the bus timetable, and information that potential and actual bus passengers may, within some measure of statistics (and probability), rely upon for their bus transport. Here, we shall now develop the idea of bus timetables denoting certain traffics.

A.4 Bus Traffic

34. Bus traffic is here considered a discrete function from time into bus positions on the net.
35. From (such) a bus we can observe its bus number.
36. A bus is at any time positioned either at a hub or a fraction of a distance along a link.
37. Fractions are reals in the open interval between 0 and 1.
38. We shall not define necessary bus traffic wellformedness conditions.

```

type
  34. BTF = T → B (B → BP)
  35. BP = atH(hi:HI) | onL(li:LI,f:F,li':LI)
  37. F = {|f:Real•0<f<1|}
value
  36. obs_BNo: B → BNo
  38. wf_BTF: BTF → Bool

```

Bus Traffic versus Bus Timetable

In expressing generation of bus traffics and whether a bus traffic satisfies a bus timetable, we shall make the following assumptions: buses must not depart from a bus stop earlier than its scheduled time; and buses, when “late” must not be “too late”, that is, must not be further away than the nearest previous hub or approaching the bus stop along its link. These assumptions are encoded by the “multiplier” and “fraction increment” constants m and δ introduced now.

39. Let m be a positive natural number (a time interval multiplier, say, of value 2,3,4).
40. Let δ be a “tiny” (position) fraction increment.
41. Satisfaction of a bus traffic with respect to a bus timetable is expressed in terms of
 - a) a predicate over buses, represented by their bus numbers bn ;
 - b) we consider only the timed bus schedule;
 - c) for all bus stops we express a predicate over bus traffic positions;
 - d) namely that there exists a time, t' , of the traffic which is equal to or some small time interval before the time of the scheduled stop,
 - e) at which time (t') some buses have traffic positions bp such that
 - f) the bus being considered, namely bn , is recorded in the traffic,
 - g) among those bus positions as having
 - h) being either at the previous hub or
 - i) on the appropriate link, either at the bus stop ($f' = f$) or shortly before that bus stop ($f' - \delta$).

```

value
  39. m:Nat, axiom 0<m≤5
  40. δ:Real, axiom 0<δ≪1
  41. satisfy: BTF × BTT → Bool
  41. satisfy(btf,btt:(nd,tbl)) ≡
  41a. ∃ bn:BNo•bn ∈ dom tbl ⇒
  41b. let (tbl) = tbl(bn) in
  41c. ∃ (t:bp,mkL_BS(hi',li',f',hi')):TBS•(t,ps) ∈ elems tbsl ⇒
  41d. ∃ t':T•t' ∈ dom btf ∧ t-m*t'<t ≤t'
  41e. let bp = btf(t') in
  41f. bn ∈ dom bp ∧
  41g. case bp(bn) of
  41h. atH(hi) → hi=hi'
  41i. onL(hi,li,f,li') → li=li' ∧ hi=hi' ∧ f'-δ ≤ f ≤ f'
  41. end end end

```

In the above satisfaction relation we do not consider where the buses are at times properly “between” bus stop times (other than when very “close” — as expressed by the proposition $t-m*t'<t'≤t$).