Verified Software for Ubiquitous Computing[1]
— a Proposed *VSTTE/Ubiquitous Computing* Foothill Project —

Working Document, Draft, December 15, 2006: 14:22

Dines Bjørner

Computer Science and Engineering
Informatics and Mathematical Modelling
Technical University of Denmark
DK-28000 Kgs.Lyngby
Denmark

Graduate School of Information Science
Japan Advanced Institute of Science & Technology
1-1, Asahidai, Tatsunokuchi
Nomi, Ishikawa 923-1292
Japan

bjorner@gmail.com

July 4, 2006. Compiled December 15, 2006, Kanazawa, Japan

Abstract

We sketch a project that is to be pursued over the next 15 years and, hopefully, by partners on five continents: (i) the Pacific: Australia, New Zealand; (ii) Asia: China, India, Japan, Korea, Macau, Singapore; (iii) Europe (Austria, Denmark, England, France, Germany, Ireland, Italy, Portugal, Sweden, Switzerland) and (iv-v) North and South America (Argentina, Brasil, Canada, USA).

Sets of commensurate descriptions of transportation domains shall be developed and analysed. A domain theory of, it is proposed, transportation shall thus emerge. Based on these descriptions (formalised using a set of integrated formal notations) a set of commensurate prescriptions of requirements for a ubiquitous computing system for automated highways shall be developed. The "commensurateness" of the different requirements prescriptions and their relations to the commensurate domain descriptions shall likewise be proved. Eventually these requirements shall be programmed with program annotations relating program statements to respective requirements and domain models.

In this report we outline and discuss implications of the above.

There are two versions of this document.

A short version is the draft of an invited paper for the First Asian Working Conference on Verified Software, UNU-IIST, Macau, 29–31 October, 2006[2].

A long version aims at informing potential project partners in, we hope, sufficient detail so as to help them decide to join the project.

This is the long version.

Contents

[1]Invited talk for the First Asian Working Conference on Verified Software, UNU-IIST, Macau, 29–31 October, 2006.

[2]http://www.iist.unu.edu/www/workshop/AWCVS2006/

1 The Backgrounds

1.1 "A Gleam in the Eye"

Three strands appear to come together in this project proposal.

One initiation point for this research and advanced development proposal is the author's fascination with domain engineering as applied to transportation. Initially to the railway subdomain. More generally, and later, with the transportation domain. And, in this proposal, the instantiation of this transportation domain to the concept of "the automated highway".

Two other initiation points were Tony Hoare's similar obsession with VSTTE, "a million lines of verified code", and Robin Milner's "lifting" of the concerns of ubiquitous computing to elegant heights.

More than 30 years of quest for provably correct software, since the days of the birth of VDM at the IBM Laboratory, Vienna, Austria, is embodied in the project.

1.2 Grand Challenges of Informatics

Ref. 1 enumerates 13 criteria for a project to attain "grand challenge" status;

1. Criteria for a Grand Challenge
   Tony Hoare, Robin Milner, Martyn Thomas, and Alan Bundy;
   Revised by Tony Hoare, 30 May 2002.
   http://www.cra.org/Activities/grand.challenges/hoare.pdf

   The primary purpose of the formulation and promulgation of a grand challenge is to accelerate the advancement of science. The only purpose of this document is to clarify the concept of grandness as applied to a scientific challenge. The suggested criteria concentrate on those aspects of grandness that contribute towards the primary scientific goal of the challenge. They are the criteria that distinguish a grand challenge from the many other worthy kinds of challenge that are formulated to contribute to economic, political or other societal goals. Each criterion is intended to describe some property relevant for the comparison and evaluation of proposed grand challenges solely according to their degree of grandeur. No challenge, however grand or otherwise desirable, should be expected to meet all the criteria. The order of the criteria is not significant.

   (a) It arises from scientific curiosity about the foundation, the nature or the limits of the scientific discipline.
   (b) It has enthusiastic support from (almost) the entire research community, even those who do not participate.
   (c) It has international scope: participation would increase the research profile of a nation.
   (d) It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines.
   (e) It was formulated long ago, and still stands.
   (f) It goes beyond what is initially possible, and requires development of techniques and tools unknown at the start of the project.
   (g) It calls for planned co-operation among identified research teams and schools.
   (h) It encourages and benefits from competition among identified individuals and teams, with clear criteria on who is winning, or has won.
   (i) It necessitates collaboration of several identified research specialties, theoretical and/or practical.

(j) It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails.

(k) It should be rather obvious how far and when the challenge has been met (or not).

(l) It should lead to radical paradigm shift, breaking free from the dead hand of legacy.

(m) It is not likely to be met simply from commercially motivated evolutionary advance.

The word "challenge" is these days commonly applied to a survey of relevant topics in some general research area that promises some short-term social or economic benefit, often just to a single nation or region (provided, of course,that sufficient research funds are allocated to it immediately). The criteria listed above are not intended to apply to such challenges.

The tradition of Grand Challenges is common in many branches of Science. If you want to know whether a challenge is grand enough, compare it with:

- Put a man on the moon
- Cure cancer (in ten years)
- Prove Fermat's last theorem
- Map the Human Genome
- Map the Human Proteome
- Find the Higgs boson
- Find Gravity waves
- Unify the four forces of Physics
- Complete Hilbert's programme for the foundations of mathematics

Some of these have succeeded, some of them have failed, and some of them are still open. In computing, the following are listed, not because we recommend them, but because they are familiar to Computer Scientists. The reader is invited to evaluate any of the examples according to the criteria given later. Obviously, no challenge will meet all of them.

- Prove that P is not equal to NP
- The Turing test
- The Verifying Compiler
- A championship chess program
- A championship GO program.
- Automatic translation of scientific literature from Russian to English.
- A mathematical model of the evolution of the web.
- A wearable computer serving as a guide dog for the blind.

Some of these have succeeded, some of them have failed; some of them have lost the interest of the scientific community, and some are reliably conjectured to be impossible. They are quoted here only as illustrations of the property of grandeur. There is no implication that any of these will be found worthy of general support.

1.3 VSTTE: Verified Software: Theories, Tools and Experiments

The idea at the root of the VSTTE grand challenge is that of a verifying compiler. That compiler accepts annotated code. The annotations are, for example, like pre- and post-conditions (say in a Hoare-style logic). The annotations are meant to express requirements and assumptions about the machine and the environment of the machine on which the code is executed. The verifying compiler is then expected to prove, that is, to verify, that the annotated code indeed satisfies the annotations. We refer to [1, 2]

Reference 2 is an original source for the VSTTE effort; Ref. 1 is a position statement prepared for the conference otherwise announced in Ref. 2 while Ref. 3 elaborates on Ref. 1; Ref. 4 provides further elaboration in the form of a FAQ; Ref. 5 announces one kind of follow-up on the VSTTE conference in October 2005.

2. The Verifying Compiler: A Grand Challenge for Computing Research
   Tony Hoare, Journal of the ACM, Vol. 50, No. 1, January 2003
   http://www.csl.sri.com/ shankar/GC04/hoare-compiler.pdf

3. VSTTE: Verified Software: Theories, Tools, Experiments
   Conference home page, Oct. 10th-14th, 2005
   http://vstte.inf.ethz.ch/ or http://vstte.ethz.ch/

4. The IFIP Working Conference on Verified Software: Theories, Tools, Experiments; conference report
   Tony Hoare, Jayadev Misra, and N. Shankar, October 20, 2005
   http://vstte.ethz.ch/report.html

5. Verified Software: Theories, Tools, Experiments — Vision of a Grand Challenge project
   Tony Hoare and Jayadev Misra, July 2005.
   http://vstte.ethz.ch/pdfs/vstte-hoare-misra.pdf

6. Verified Software: Frequently Asked Questions
   Tony Hoare and Jayadev Misra, 19 September 2005
   http://wiki.se.inf.ethz.ch/vstte/index.php/Verified_Software:_Frequently_Asked_Questions

7. FLoC'2006 Workshop on Verified Software: Theories, Tools, and Experiments
   K. Rustan M. Leino, Summer 2006
   http://research.microsoft.com/~leino/vstte2006/

8. Verified Software Roadmap 2006
   A list of 16 references (in the form of URLs) are given to position statements on the topic of VSTTE. http://qpq.csl.sri.com/vsr/private/verified-software-roadmap-2006/
   http://qpq.csl.sri.com/vsr/private/verified-software-roadmap-2006/dines.pdf/view refers to an early version of the current proposal.

1.4 Ubiquitous Computing: The Automated Highway

By 'ubiquitous computing' we shall here understand the presence of computation (and communication) "everywhere". And by 'the automated highway' we shall specifically understand the use of computation and communication in every possible aspect of automating the orderly (safe and efficient) movement of cars along a net of highways. This latter implies the availability of sensors and actuators along and possibly above and/or "outside" the highway net and in all vehicles moving along this net of highways.

1.4.1 Ubiquitous Computing

9. Ubiquitous Computing Grand Challenge: Introduction

   http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/

   - There is burgeoning population of 'effectively invisible' computers around us, embedded in the fabric of our homes, shops, vehicles, farms and some even in our bodies. They are invisible in that they are part of the environment and we can interact with them as we go about our normal activities. However they can range in size from large Plasma displays on the walls of buildings to microchips implanted in the human body. They help us command, control, communicate, do business, travel and entertain ourselves, and these 'invisible' computers are far more numerous than their desktop cousins. How many computers will you be using, wearing, or have installed in your body, in 2020? How many other computers will they be talking to? What will they be saying about you, doing for you, or to you? By that time computers will be ubiquitous and globally connected. Shall we be able to manage such large-scale systems, or even understand them? How do people interact with them and how does this new pervasive technology affect society? How can non-computing people configure and control them? What tools are needed for design and analysis of these constantly adapting and evolving systems? What theories will help us to understand their behaviour?
   - These are the sort of issues which make Ubiquitous Computing a Grand Challenge; join us in addressing them.
   - The Ubiquitous Computing Grand Challenge (UbicompGC) is one of the 6 UKCRC Grand Challenges. It was formed by merging two of the original Grand Challenges GC2 "Science for global ubiquitous computing" which focused on theory and GC4 "Scalable ubiquitous computing systems" which focused on engineering aspects. UbicompGC is formulating a research manifesto which postulates the need for combined Science (theory) as well as addressing the Engineering and Social issues related to building Ubiquitous Systems. So far, most research in the UK and elsewhere has focussed on the Engineering with very little attention on the theory required to underpin the design and analysis of ubiquitous systems which are intrinsically large-scale and complex. Some of the work in the Equator project has addressed social aspects and how people will interact with Ubiquitous Systems. The overview page summarises the goals of the challenge.

     Background information on the Grand Challenges together with a report describing each of the challenges can be found on the UKCRC Grand Challenges site. We intend to hold a mini-workshop as part of the Grand Challenges Conference in Glasgow 22-24 March 2006.
   - UbicompGC activities are currently being promoted by the EPSRC funded UK-UbiNet Network grant which predated UbiCompGC and runs from April 2003 - March 2006. The Ubicomp steering committee are all members of the UK-UbiNet

management committees. See the UK-UbiNet for information on UK and worldwide activities on ubiquitous computing, workshops organised which addressed the issues raised by the UbiCompGC research manifesto, future workshops and information on events and conferences.

- We welcome discussion of all aspects of UbicompGC. For this purpose, please subscribe to the UbicompGC email list. We especially invite discussion on the final draft of the UbicompGC manifesto, which we aim to finalise by 1 September 2005.
- There will be a sequence of evolving Annexes to the manifesto, including descriptions of "foothill projects", also published for discussion with the manifesto. Two threads of discussion, called 'Manifesto' and 'Foothills', have been created for these discussions.
- More generally, wide-ranging discussion will greatly help the steering committee in building a community around the Grand Challenge, and in coordinating its activity.
- UbicompGC Steering Committee
  (a) Prof. Morris Sloman, Department of Computing, Imperial College London (Chairman)
  (b) Dr. Dan Chalmers, Informatics, University of Sussex
  (c) Prof. Jon Crowcroft, Computer Laboratory University of Cambridge
  (d) Prof. Marta Kwiatkowska, School of Computer Science, University of Birmingham
  (e) Prof. Robin Milner, Computer Laboratory University of Cambridge
  (f) Prof. Tom Rodden, Computer Science and IT, University of Nottingham
  (g) Prof. Vladimiro Sassone, Informatics, University of Sussex

10. Ubiquitous Computing Grand Challenge: Overview

    http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/overview.html

    - By 2020 how many computers will you be using, wearing, have in your home, or even in your body? Computers are ubiquitous and will soon be globally connected. Shall we be in control of the complex emerging behaviour arising from their aggregation in a "ubiquitous" global network, or even understand it? As these devices become smaller, more numerous, more independent from users and more deeply embedded in the world around us, they raise formidable scientific and engineering challenges.
    - We propose to develop scientific theory and the design principles of Global Ubiquitous Computing together, in a tight experimental loop.
    - At least since Leonardo, engineers have relied upon science when building their artifacts, whilst real-world challenges have often set the goalpost for scientists. Nevertheless, often in the past the two have developed independently from each other. Why are we advocating here for their integration as the key of our research method? The reason is in the very scale and nature of ubiquitous computers, and the way they are growing intimately embedded in our lives: we can hardly

afford "experimental" failures, as we allow them to control transport, monitor our health, regulate our bank accounts as well as our most delicate global financial mechanisms.

- The final goals of the research proposed here can be described succinctly as:
  - To define a set of design principles that pertain to all aspects of ubiquitous computing and are accepted, taught, and used in practice;
  - To develop a coherent information science that allows descriptive and predictive analysis of ubiquitous computing at many levels of abstractions, and to use it to derive, analyse, justify its constructions.
- Qualities of Global Computers

  In order to allow a ubiquitous computer to blend in the environment and act "invisibly" whilst being dependable and remain controllable, we will have to guarantee ("by design and construction") that it is:
  - Fluid: it structure will vary frequently, and evolve in the long term;
  - Purposive: each of its components has a purpose, which explains its actions;
  - Autonomous: it makes autonomous decisions based on its context and its "experience" of it;
  - Reflective: it can take actions (on itself) based on self-analysis;
  - Trustworthy: it can be trusted not to misuse information and resources;
  - Tactful: its impact on its surroundings is effective, but minimal
  - Scalable: the size of its subsystems can vary greatly, yet the same principles apply to them all; and
  - Efficient.
- Challenges of Global Computing

  We are already developing new theories, programming languages and experimental systems to help engineer ubiquitous computers. The challenges ahead (arising from the qualities required) are many, and of considerable depth. Amongst the engineering ones:
  - design devices to work from solar power, are aware of their location and what other devices are nearby, and form cheap, efficient, secure, complex, changing groupings and interconnections with other devices;
  - engineer systems that are self-configuring and manage their own exceptions;
  - devise methods to filter and aggregate information so as to cope with large volumes of data, and to certify its provenience.
  - business model for ubiquitous computing, and other human-level interactions.
- Amongst the science challenges we list:
  - discover mathematical models for space and mobility, and develop their theories; devise mathematical tools for the analysis of dynamic networks;
  - develop model checking, as well as techniques to analyse stochastic aspects of systems, as these are pervasive in ubiquitous computing;
  - devise models of trust and its dynamics;
  - design programming languages for ubiquitous computing.

- These and other ideas are developed in detail in the Manifesto of the UKCRC Grand Challenges proposal:
  - Global Ubiquitous Computing: Design and Science
  - http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/manifesto.html

11. Ubiquitous Computing, The Computer Journal.2006; 49: 390-399

    http://www.bcs.org/BCS/Awards/Events/cjlectures/ubiquitous.htm

    http://comjnl.oxfordjournals.org/cgi/content/full/49/4/390

    - The Computer Journal presents the first Computer Journal Lecture: "Ubiquitous Computing" by Robin Milner.
    - 23rd February 2006, Central London, 5:30 pm for a 6:15 pm start.
    - Lecture by Robin Milner - Computer Laboratory, University of Cambridge. The lecture will be followed by a debate on the lecture. The lecture itself, and the discussion following it, will be edited for the Computer Journal.

    The vision of ubiquitous computing (ubicomp) is that computing entities become an effective part of our environment, supporting our lives without our continual direction, so that we can be largely unaware of them. One of the UK Grand Challenges for Computing Research addresses not only the ubicomp vision, but also the design principles and theories that will support it. Ubicomp will entail hardware/software systems that exceed those that we know by orders of magnitude in size. There is little chance of extrapolating existing methods of software production to cope with them. Ubicomp offers an opportunity to develop a deeper science of computing that interweaves three ingredients - vision, design and theory - more intimately than ever before. This is the Grand Challenge; the lecture will explore how we may approach it.

12. Ubiquitous Computing: Shall we understand it?

    Follow-up discussion on the above:

    http://www.bcs.org/server.php?show=ConWebDoc.4708&setPaginate=No

13. Wikipedia's URL on Ubiquitous Computing

    http://en.wikipedia.org/wiki/Ubiquitous_computing

    Ubiquitous computing (ubicomp, or sometimes ubiqcomp) integrates computation into the environment, rather than having computers which are distinct objects. Other terms for ubiquitous computing include pervasive computing, calm technology, and things that think. Promoters of this idea hope that embedding computation into the environment and everyday objects would enable people to move around and interact with information and computing more naturally and casually than they currently do. One of the goals of ubiquitous computing is to enable devices to sense changes in their environment and to automatically adapt and act based on these changes based on user needs and preferences. We refer to [3]

14. Ubiquitous Computing Evaluating Consortium:

    http://ubiqcomputing.org/ — An effort led by SRI Intl.

15. Ubicomp: Conference announcements:

    http://ubicomp.org/ubicomp2005, http://ubicomp.org/ubicomp2006

16. IEEE Journal on Pervasive Computing

    http://www.computer.org/portal/site/pervasive/

1.4.2 The Automated Highway

Just very briefly: an automated highway system or Smart Road, is an advanced intelligent transportation system technology designed to provide for driverless cars on specific rights-of-way. It is most often touted as a means of traffic congestion relief, since it drastically reduces following distances and thus allow more cars to occupy a given stretch of road. We refer to [4]

17. Wikipedia: Automated Highway System

    http://en.wikipedia.org/wiki/Automated_highway_system

    An automated highway system (AHS) or Smart Road, is an advanced Intelligent transportation system technology designed to provide for driverless cars on specific rights-of-way. It is most often touted as a means of traffic congestion relief, since it drastically reduces following distances and thus allow more cars to occupy a given stretch of road.

    - How it works

      The roadway has magnetized stainless-steel spikes driven one meter apart in its center. The car senses the spikes to measure its speed and locate the center of the lane. Further the spikes can have either magnetic north or magnetic south facing up. The roadway thus has small amounts of digital data describing interchanges, recommended speeds, etc.

      The cars have power steering, and automatic speed controls, but these are controlled by the computer.

      The cars organize themselves into platoons of eight to twenty-five cars. The platoons drive themselves a meter apart, so that air resistance is minimized. The distance between platoons is the conventional braking distance. If anything goes wrong, the maximum number of harmed cars should be one platoon.

    - Deployments

      A prototype automated highway system was tested in San Diego County, California in 1991 along Interstate 15. However, despite the technical success of the program, investment has moved more toward autonomous intelligent vehicles rather than building specialized infrastructure. The AHS system places sensory technology in cars that can read passive road markings, and use radar and inter-car communications to make the cars organize themselves without the intervention of drivers.

18. Foothill Project: Automating the Highway:

    http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/fp-automatinghighway.html

    UKCRC Grand Challenges for Computing Research. Ubiquitous Computing: Science and Design

    Since this project is one of the bases for the present proposal we quote in extenso.

- Jon Crowcroft: http://www.cl.cam.ac.uk/users/jac22/
- Monitoring and control of private vehicles on the public highway is high on the political agenda; this is because it is becoming feasible, and may be desirable for at least two reasons: first, from the economic perspective, it may achieve more efficient use of road resources; second, from the safety perspective, it may achieve a significant drop in injury and death on the roads. Various prototypes exist, and various projects are current. Many technologies interact, and there are numerous legal, political and economic stakeholders. We propose a foothill project to study monitoring and control with particular concern for efficiency and safety, in the context of ubiquitous systems for transport. For efficiency (of road use) the monitoring and control may be either distributed or centralised, or a combination of the two. In a distributed system the car receives information from navigation systems and roadside monitors concerning routes, conditions and prices; it (or its driver) then makes a decision and pays. on the other hand a centralised system, such as the London congestion-charging scheme, depends entirely on a network of roadside monitors, recording data about vehicles, drivers and journeys on a central database used as the basis for billing.
- To improve safety, there a spectrum of possible solutions from distributed to centralised systems. At the centralised extreme, 'car-trains' have been proposed; vehicles joining trunk routes would be logically clumped, and controlled by a single aggregate unit. At the distributed extreme, each vehicle always chooses its own velocity, using data from on-board and remote sensors. There are many research problems; for example:
  - What are the design spaces for distributed and/or centralised systems in the two cases? Can they me mixed, e.g. distributed for efficiency of road-use but centralised for safety?
  - By what measures can each solution in the space be assessed for its contribution to both efficiency and safety?
  - In each possible design, what threats arise from neglect or malevolence? These threats may attack endanger correct technical function, or they may endanger privacy (for example, centralised records may be illegally mined to deduce driver habits).
  - Success in addressing these problems will involve a variety of theoretical or simulational models of distributed and mobile processes; and will prompt the further development of such models.

19. A longer paper addressing these issues of item 2 on page 33 is also available: http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/road.pdf
20. Carnegie Mellon AHS group

    http://www.cs.cmu.edu/Groups/ahs/
21. Automated Highway Systems

    http://scholar.lib.vt.edu/theses/available/etd-5414132139711101/unrestricted/ch2.pdf

    http://72.14.203.104/search?q=cache:ctx_HZkkonwJ:scholar.lib.vt.edu/theses/available/etd-5414132139711101/unrestricted/ch2.pdf+Automated+Highway&hl=en&ct=clnk&cd=3

1.5 Domain Engineering

Section 3.2.1 should probably be read first. By a domain model we understand an abstraction of some reality, something which exists, and as it is conceived to be. A domain model is typically expressed by both a precise, and as here English narrative and a formalisation. If the formalisation is in logic then one of the set of models which satisfy the logic is what we mean by "the domain model". If the formalisation is in some model-oriented specification language, such as B [5], VDM-SL [6, 7, 8] or Z, then the semantics of these languages explicitly designates the model(s). To create such a model, or, typically, set of models, one needs to acquire domain knowledge, analyse it, and then bring the acquisition and the analysis together in a description such as for example given in Sect. 3.2.1 and according, for example, to the principles, techniques and tools as outlined to some depth in Chap. 11 of [9].

22. Domain Engineering: Part IV, Chaps. 8–16 of [9].

    These chapters cover:

    8. Overview of Domain Engineering
    9. Domain Stakeholders
    10. Domain Attributes: Continuity, Discreteness and Chaos; Statics and Dynamics; Tangibility; One, Two, ... Dimensionality
    11. Domain Facets: Domain Facilitatos (Business Processes), Intrinsics, Support technologies, Management and Organisation, Rules and Regulations, Scripts, Human Behaviour
    12. Domain Acquisition
    13. Domain Analysis and Concept Formation: Incompleteness, Incosistency, Conflicts; Concepts
    14. Domain Verification and Validation
    15. Towards Domain Theories
    16. The Domain Engineering Process Model

23. "Domain Engineering", Chapter in forthcoming EATCS Series textbook covering a number of BCS FACS Evening Seminars, 2003-2005, Eds. Jonathan Bowen and Paul Boca, Springer, 2007. http://www.jaist.ac.jp/ bjorner/domain.ps

24. "On Domains and Domain Engineering, Prerequisites for Trustworthy Software, A Necessity for Believable Project Management", http://www.jaist.ac.jp/ bjorner/jaist-dom.ps, 182 pages

    > This is a somewhat lenghty document which summaries item 22, lists and answers a number of FAQ, and brings, in a number of appendices, reasonably sized examples of domain descriptions or references to such: Transportation, Manufacturing, Documents, "The Market", "Cyberrail" (a Japanese conception of a future railway system). In addition the document also surveys the concept of Business Processes (referred to in Chap. 11, see item 22, as well as a summary of the RSL language.

25. "Possible Collaborative 'Domain' Projects: Japanese Private and/or Public Institutions + JAIST/DEDR", The JAIST School of Information Sciences' Domain Engineering and Digital Rights Group, http://www.jaist.ac.jp/ bjorner/jaist-dom-projs.ps

    Based on the lengthy document referenced in item 24 on the preceding page, this short document outlines, for Japanese public and private institutions (government and industry) how possible collaborative domain engineering projects might be organised.

2 The Triptych Dogma

2.1 The Dogma

Before software can be designed its requirements must be understood. Before requirements can be prescribed the application domain must be understood.

Hence the "idealistic" triptych dogma:

- $\mathcal{D}$**:** First one must establish a proper, comprehensive domain theory for the application area — here transportation in general.
- $\mathcal{R}$**:** From such a domain theory one can, as we show in Chaps. 18–25 of Vol 3 of [10, 11, 9], i.e., [9], "derive" domain (i.e, functional) requirements.
- $\mathcal{S}$**:** From these again, as we show in Chaps. 26–29 [9], we can decide major structures of the desired software.

2.2 Verification

In proving software, i.e., program code, $\mathcal{S}$, correct with respect to requirements, $\mathcal{R}$, such a proof is always relative to assumptions about the domain, $\mathcal{D}$:

$$\mathcal{D}, \mathcal{S} \models \mathcal{R}$$

To do proper proofs we postulate that proper theories need first be established of the domain and of the requirements.

2.3 Decomposition of Project Effort

The present proposal hinges to some extent on the triptych dogma. But the proposal allows for research groups to focus their contributions "down stream" from domain descriptions and requirements prescriptions, or only on one of these.

Some "Verified Software for Ubiquitous Computing" project groups will wish to focus on one or more of:

- Domain Theories (descriptions and analysis) using one or another approach, including combinations of approaches, other groups on
- Requirements Prescription development from Domain Descriptions and proofs of development "transformations", other groups on

- Requirements Theories (descriptions and analysis), or on
- Software Design development from Requirements Prescriptions and proofs of development "transformations", or on
- Program Code Annotation and Verification — from Domain Descriptions and Requirements Prescriptions

Project management is then needed to secure that it all "hangs together", forms a whole.

3 The Project Proposal

3.1 Summary

Section 2.3 on the preceding page suggests a number of project components:

1. Sets of commensurate correlated informal and formal descriptions of transportation domains shall be developed and analysed. We refer to Sect. 3.2.
2. A theory of transportation shall thus emerge. That theory shall explain and prove what precisely is meant by commensurateness.
3. Based on these descriptions (formalised using a set of integrated formal notations) a set of commensurate prescriptions of requirements for a ubiquitous computing system for automated highways shall be developed.
4. The "commensurateness" of the different requirements prescriptions and their relations to the commensurate domain descriptions shall likewise be proved.
5. Eventually these requirements shall be programmed with program annotations relating program statements to respective requirements and domain models.
6. The programs with annotations shall be processed by a verifying compiler and the program code shall be automatically proved correct wrt. the program annotations.

3.2 Domain Theories

To get a grasp on what a domain description might entail we give in Sect. 3.2.1 an example domain description. From that the reader is expected to draw a number of conclusions, including whether this project proposal is of interest or not. We refer to Item 1 Sect. 3.1.

3.2.1 A Domain Model

Let us take a reasonably comprehensive look at an albeit simple domain description, namely that of a simple transportation net and its traffic. We refer to

- The BCS FACS Journal, December 1005
  http://www.bcs-facs.org/newsletter/facts200512.pdf

for a more detailed exposition of the below description.

(1.) A transportation net (N) consists of segments (S) and junctions (J). (2., 3.) That is, from a net one can observe the set of all segments and the set of all junctions. (4.) There is at least one segment and two (distinct) junctions in any net. (5.) Segments and junctions have unique identifications (Si, Ji)[3]. (6.,7.) One can observe segment and junction identifiers from segments, respectively junctions. (8., 9.) To avoid "junk" being models of our specification we must insist that the number of segments [junctions] of a net equals the number of segment [junction] identifiers of that net. (10., 11.) Segments are delimited by a pair of junctions, and from a segment one can (thus) observe the pair of junction identifiers of the delimiting junctions. (12., 13.) Junctions connect to one or more distinct segments, and from a junction one can (thus) observe the set of one or more respective segment identifiers.

type
1. N, S, J
value
2. obs_Ss: N $\rightarrow$ S-**set**
3. obs_Js: N $\rightarrow$ J-**set**
axiom
4. $\forall$ n:N • **card** obs_Ss(n) $\geq 1 \wedge$ **card** obs_Js(n) ≥ 2
type
5. Si, Ji
value
6. obs_Si: S $\rightarrow$ Si
7. obs_Ji: J $\rightarrow$ Ji
axiom
8. $\forall$ n:N • **card** obs_Ss(n) $\equiv$ **card** {obs_Si(s)|s:S • s $\in$ obs_Ss(n)}
9. $\forall$ n:N • **card** obs_Js(n) $\equiv$ **card** {obs_Ji(c)|j:J • j $\in$ obs_Js(n)}
type
10. Jip = {|{ji,ji′}:Ji-**set** • ji$\neq$ji′|}
value
11. obs_Jip: S $\rightarrow$ Jip
type
12. Si1 = {|sis:Si-**set**•**card** sis $\geq$1|}
value
13. obs_Sis: J $\rightarrow$ Si1
axiom
14. $\forall$ n:N, s:S • s $\in$ obs_Ss(n) $\Rightarrow$
15. **let** {ji,ji′} = obs_Jis(s) **in**
16. $\exists$! j,j′:J • {j,j′}$\subseteq$obs_Js(n) $\wedge$ j$\neq$j′ $\wedge$
17. obs_Si(s) $\in$ obs_Sis(c) $\cap$ obs_Sis(c′) **end**
18. $\forall$ n:N, j:J • j $\in$ obs_Js(n) $\Rightarrow$
19. **let** sis = obs_Sis(c), ji = obs_Ji(j) **in**
20. $\exists$! ss:S-**set** • ss$\subseteq$obs_Ss(n) $\wedge$ **card** ss=**card** sis $\wedge$
21. sis = {|obs_Si(s)|s:S•s $\in$ ss|} **end**

[3] Segment and junction identifiers can be thought of as summarising the distinct spatial location of that which they identify.

(14.–17.) For every segment of a net there exists exactly a set of distinct junctions of that net whose identifiers are those observable from that segment, and (18.–21.) – vice versa – for every junction of a net there exists exactly a set of distinct segments of that net whose identifiers are those observable from that junction.

The above description covers just the net of its junction-connected segments.

(22.) Paths are triples of (junction,segment,junction) identifiers, and routes are sequences of paths. (23.–25.) 'paths(n)' expresses all paths of a net. (27.–30.) A route is is a finite sequence of paths such that adjacent sequence triples designate adjacent paths, that is, paths sharing a junction identifier (well-formedness). (32.–35.) 'routes(n)' expresses all routes of a net, that is, a possibly infinite set since routes my be cyclic. Any path is a route. If r and r' are routes such that the last path of r shares last junction identifier with the first junction identifier of the first path of r', then their concatenation is a route.

```
type
22. P = Ji × Si × Ji
value
23. paths: N → P-set
24. paths(n) ≡
25.   {(ji,si,ji′)|s:S,ji,ji′:Ji,si:Si• s ∈ obs_Ss(n)∧{ji,ji′} ∈ obs_Jis(s)∧si=obs_Si(s)}
type
26. R = {|r:P*•wf_R(r)|}
value
27. wf_R: P* → Bool
28. wf_R(r) ≡
29.   ∀ i:Nat • {i,i+1}⊆inds(r) ⇒
30.     let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) in ji = ji′ end
31. routes: N → R-infset
32. routes(n) ≡
33.   let rs = {⟨p⟩|p:P•p ∈ paths(n)}
34.          ∪ {r^r′|r,r′:R•{r,r′}⊆rs∧wf_R(r^r′)} in
35.   rs end
```

(36.) The state of a segment is a either an empty set, or a singleton set, or a set of two "reversed" pairs of the identifiers of the two junctions to which the segment is connected. An empty set designates that the segment is closed for traffic. A single pair $(j_i, j_{i'})$ designates that the segment is open for traffic from the junction identified by j_i to junction identified by $j_{i'}$. A double pair designates that the segment is open for traffic in both directions. (37.) The state of a junction is a set of pairs of identifiers of the segments that are connected to that junction. If (s_{i_j}, s_{i_k}) are in the state then it means that traffic can traverse the junction from the segment identified by s_{i_j} to the segment identified by s_{i_k}. (38.–39.) From a segment [junction] one can observe the state of the segment [junction]. (40.–43.) One can speak of and observe the state space of a segment [junction]. (44.–45.) The current state of a segment [junction] lies in the state space of that segment [junction].

```
type
36. SΣ = {|jip:(Ji×Ji)-set•card jip≤2∧card jip=2⇒(ji,ji′)∈ jip⇒(ji′,jip)∈ jip∧ji≠ji′|}
37. JΣ = (Si×Si)-set
value
```

38. obs_SΣ: S → SΣ
39. obs_JΣ: J → JΣ
type
40. SΩ = SΣ-**set**
41. JΩ = JΣ-**set**
value
42. obs_SΩ: S → SΩ
43. obs_JΩ: J → JΩ
axiom
44. ∀ s:S • obs_SΣ(s) ⊆ obs_SΩ(s),
45. ∀ j:J • obs_JΣ(j) ⊆ obs_JΩ(j)

(46.) Time, vehicles and lengths are introduced. Time is considered a dense, ordered set. (47.) Fractions are reals in the inclusive range 0 to 1. (48.) A (vehicle) position is either at a junction or some fraction a segment (down, from fj_i to tj_i, implicitly) identified by this pair of identifiers of the junctions to which the segment is connected. (49.) Real [observed] traffic is a continuous function [discrete map] from time to pairs of nets and positions of vehicles. (50.) Segments have lengths. (51.–58.) The lengthy axiom expresses one aspect of traffic: that positions are indeed positions of the net.

type
46. T, V, L
47. F = {|f:**Real**•0≤f≤1|}
48. P == mkP_at_J(ji:Ji) | mkP_along_S(fji:Ji,f:F,tji:Ji)
49. rTF = T → (N × (V $\overrightarrow{m}$ P))
50. oTF = T $\overrightarrow{m}$ (N × (V $\overrightarrow{m}$ P))
value
51. obs_L: S → L
axiom
52. ∀ tf:(rTF|oTF) •
53.   ∀ t ∈ **dom** tf •
54.   **let** (n,vps) = tf(t) **in**
55.   ∀ p:P • p ∈ **rng** vps ⇒
56.   **case** p **of**
57.   mkP_at_J(ji) → ji ∈ obs_Jis(n),
58.   mkP_along_S(jf,_,jt) → {jf,jt}⊆obs_Jis(n)
      end end

The formalisation of other aspects of traffic are left as an exercise: (i) that real [observed] traffic is truly continuous [monotonic] (including: moving from open segments through "open" junctions onto open segments and: vehicles do not "jump around"), (ii) that there are no "ghost" vehicles (that is, that a vehicle which is in the traffic at two distinct times are in the traffic at all times in between these, etcetera.

3.2.2 Real Highway Transportation Nets

The model above has been simplified to fit this report. A more realistic model, one that can serve as a domain model for the requirements of an automated highway, would model (i) multi-lane highway segments and junctions, (ii) vehicle lane positions, (iii) vehicles changing lanes, etc.

3.3 Requirements

3.3.1 General: Co-ordination and Function Requirements

Now, depending on requirements for ubiquitous automated highway computing, a number of co-ordinated requirements prescriptions need be studied, developed, the resulting formal prescriptions analysed — including establishing a theory of the requirements. By "co-ordinated requirements prescriptions" we mean a "set and a co-ordination" which, as a whole, prescribe requirements to an automated highway computing system, but such that individual members of the set ("function requirements") each solve a well-prescribed problem in the context of the co-ordination software (prescribed by the "co-ordination").

We think of the co-ordination requirements and the corresponding software to represent the essence of ubiquity.

3.3.2 Ubiquitous Requirements: Some Speculations

In this section we shall sketch some rather preliminary ideas on some aspects of requirements to automated highways — such aspects, it seem, which may cast light on what we may be an issue of, an "ingredient" in requirements to ubiquitous computing systems.

Let us perform an experiment. Assume traffic "rolling" down a highway. Traffic consists of a number of vehicles passing down a number of opposite lanes, possibly at different velocities, possibly changing lane, overtaking, possibly slowing or speeding up.

Now what might "automation" mean in this case? Well it could mean either of many different things. All vehicles could be coerced into travelling at "same" velocity, or "as much as possible" close to (from below) a desired mean velocity, or be "free to change velocity", and/or be constrained to "near same" (mean) acceleration and deceleration. Etcetera.

Whichever of these are chosen, it seems that each vehicle is expected to register, as is expected done by their drivers today, some composite "picture", a "local state", of the traffic "around" that vehicle, to "some depth", that is, to some distance from that vehicle. Based on such a local state the driver today makes driving (steering, acceleration, deceleration, etc.) decisions. Today drovers do that — thus basically without consulting other drovers' plans (including their local states). In an automated highway computing system one might expect individual as well as communal driving decisions and hence actions to be based on summaries of local states "around" each vehicle, or even global such summaries. Classically one might be able to express such decision outcomes as the result of solving a large number of for example differential equations, or, alternatively, some similar sets of fuzzy control expressions.

One is reminded here of:

- Victor I. Varshavski and Dmitrii Aleksandrovich Pospelov: *Puppets without strings: Reflections on the evolution and control of some man-made systems*, Mir Publishers, Moscow, 1984 (English edition: Mir Publishers, 1988, 294 pages).

An analysis of what is thought to be needed to express (i.e., to monitor and, later, control) local (and "less local") states in domain descriptions and, subsequently in requirements prescriptions should give a rather precise idea about the need for computing system sensors and actuators.

Everyone can speculate: mention this and that kind of sensors and actuators, but only a thorough domain analysis and a "therefrom derived" requirements analysis can tell us exactly the properties we need of those sensors and actuators.

3.3.3 Further on Requirements

In this paper we shall not further attempt to formulate further requirements. To do this properly requires, in our mind, very close collaboration with the Cambridge University group of Prof. Jon Crowcroft. We refer to:

- [18] http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/fp-automatinghighway.html
- [3] http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/road.pdf

It is hoped that some of the groups involved in Domain Theories will also want to become involved in th Requirements-related projects.

And it is expected that this, the Requirements effort, will be closely collaborating with the Ubiquitous Computing Automated Highway project as referred to above.

3.4 Software

We postulate, as we did in Sect. 3.3 on the facing page, that the resulting computing system software consists of the co-ordination (the ubiquitous) software and its many "isolated" packages. We speculate that these "isolated" software packages can each be developed in isolation — provided that a stable interface to the co-ordination software can be established — and that they can each be verified in, i.e., assuming this context.

We shall not speculate further on the software possibly emanating from this proposed project other than the above single paragraph.

3.5 Overview of Project Components

4 Challenges

There is an overall challenge:

> **Challenge # 0. Verified Software for Ubiquitous Computing:** The overall challenge is to both understand the concept of ubiquitous computing and contribute to the verified software concept.

And there are a number of constituent challenges. The latter contribute to the former. We formulate some of the latter as we expect different research groups around the world to more readily "catch on" to the latter challenges than to the former.

4.1 Commensurate Specifications

The example given in Sect. 3.2.1 on page 14 was expressed in the RSL formal specification language [12, 13, 10, 11, 9]. There are other formal specification languages. To wit: Alloy [14], B [5], CafeOBJ [15, 16, 17], Casl [18, 19, 20], CSP [21, 22, 23, 24], VDM-SL [6, 7, 8], RSL [12, 13, 10, 11, 9], or Z [25, 26, 27, 28] are some of the better known.

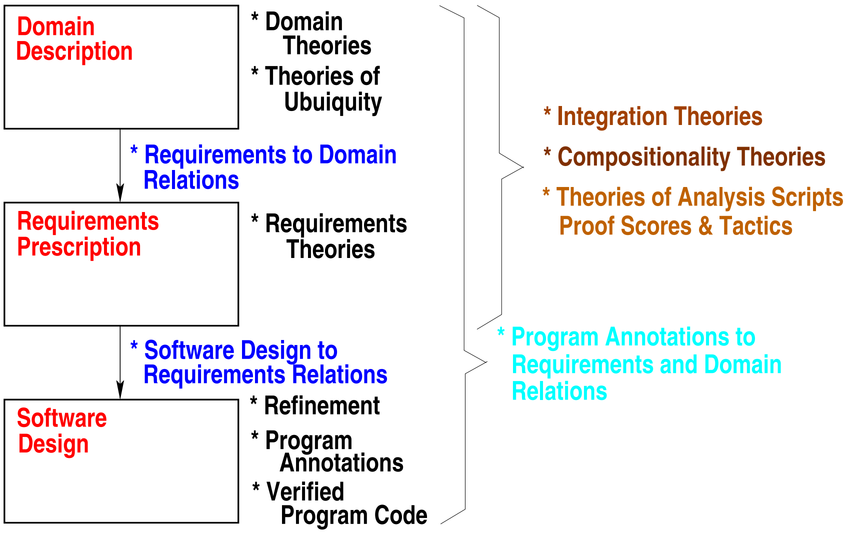

Figure 1: Project Components

Some (verified software for ubiquitous computing project groups) will specify their domains or requirements in one of these languages. Others in another of these languages.

> **Challenge # 1. Commensurate Specifications:** It is therefore a challenge to show relations between some pairs of specifications purporting to describe "the same thing", that is, to embody the comparable theories. How exactly such co-existing different notation formal descriptions can be so compared is an open question.

4.2 Integrated Specifications

The example given in Sect. 3.2.1 on page 14 captures only some basic entities of transportation nets and some basic characteristics of traffic (a behaviour).

To capture such phenomena as the individual behaviour of a vehicle, the monitoring and control of traffic and of traffic signals (i.e., the changing of segment and, in particular junction states, would probably be a mistake. More suitable formal notations should be used. They should then be used in conjunction with some formal textual notation, such as those mentioned in Sect. 4.1 on the previous page.

Which other such possible, "more suitable" notations exists, today, Summer 2006? We suggest that perhaps either one or more of the following may be used: DC (for duration calculus, a temporal logic specification language) [29, 30], message or live sequence charts (MSCs, [31, 32, 33], or LSCs, [34, 35, 36]), Petri nets [37, 38, 39, 40, 41], statecharts [42, 43, 44, 45, 46], TLA+ [47, 48, 49], or other. Some of these other may "be in the works" as of now: being researched and put forward as means for modelling, say, statistical, continuous, chaotic or other phenomena.

Whenever the textual notations of Alloy [14], B [5], CafeOBJ [15, 16, 17], Casl [18, 19, 20], VDM-SL [6, 7, 8], RSL [12, 13, 10, 11, 9], Z [25, 26, 27, 28], or other, are combined with those of, for example, DC [29, 30], MSC [31, 32, 33], LSC [34, 35, 36], Statecharts

[42, 43, 44, 45, 46], TLA+ [47, 48, 49], or other, we say that the result is a set of two or more interfacing specifications. The interface may be manifest in for example the following way: In specification $S_A^{L_i}$ some identifiers $id_1, id_2, \ldots, id_n$ are claimed to somehow "equivalize" with identical identifiers in specification $S_B^{L_j}$. When this is the case, an obligation arises, namely to show that a suitable equivalence (or satisfaction) relation can be established. Reasoning in one notation "must carry over" into "interfacing" parts expressed in another specification.

Substantial research is done in this are as witnessed by, for example, [50, 51, 52, 53, 54].

A real core issue for this project to study is the relationship between the kind of textual specification languages extensively listed and referenced above and the calculus of classical mathematics — as for example used in automotive engineering: differential equations and the like. It is a challenge to provide as seamless transition from specification in the form of classical applied mathematics, including operations research, and the newer formal specification languages.

> **Challenge # 2. Integrated Specifications:** For specific such interfacing (and thus commensurate) specifications or for general uses of their specification languages, suitable equivalence (or satisfaction) relations must be established (formulated and formally proven).

4.3 Domain Theories

By a domain theory we understand a domain description and a number of theorems that can be proved to hold of what the description models. These theorems are (obviously) proved in the proof system of the notations in which the domain description is expressed.

What could such "interesting" theorems be about? Well, let's try some possible, or potential examples:

- *Kirchoff's Law for Railways:* Assuming that train timetables are modulo 24 hours and that trains run on time, then for any 24 hour contiguous period (i.e., interval) and for every train station it is the case that the number of trains arriving at a station minus the number of trains ending their journey at that station plus the number of trains starting their journey at that station equals the number of trains leaving that station.
- *"No Ghost Trains":* In a domain description of highway traffic along a segment, then for any consecutive times t and t'' and for any vehicle which is in the traffic at those times it is the case that the vehicle is also in the traffic at all times between t and t''.
- *"God does'nt Play Dice":* We assume that the model for traffic is continuous, or at least monotonic over time (either a dense set of points, or, respectively, a discrete set of points with clearly defined "next" time successors. We assume a single lane, one traffic direction segment. If at time t vehicle v_i is "after" vehicle v_j in the traffic along the segment, then at a "next" (i.e., the successor) time t' vehicles v_i and v_j have not reversed their position.

This leads us to formulate:

> **Challenge # 3. Domain Theories:** So, for the domain of transportation (incl. highway transportation) the challenge is to identify and prove "interesting" theorems.

A source of inspiration as to what kinds of theorems to look for in any domain theory seems to be [55]. It is hard for me to think that laws of man-made domains are that fundamentally different from those nature.

4.4 Analysis Scripts: Proof Scores and Tactics

In connection with research around CafeOBJ some insight is gained with respect to "standard" scripts according to which CafeOBJ specifications can, or should be analysed. These are called proof scores. Proof scores are first class CafeOBJ objects. In that sense they are different proof tactics pursued as early as the early days of LCF [56, 57, 58, 59, 60] and is also a core subject of PVS [61, 62] proof work [63].

> **Challenge # 4. Analysis Scripts: Proof Scores and Tactics:** So, for any domain or requirements formalisation the challenge is to find a suitable, reusable set of analysis scripts.

4.5 Ubiquity

"How are the computers in the automated highway system configured and controlled? What tools are needed for design and analysis of constantly adapting and evolving computing system? What theories will help us to understand its behaviour? How do we manage such systems, or even understand them? How do people interact with them and how, in which ways, does this new pervasive technology affect us?"

These are some of the many questions that the Ubiquitous Computing — Science and Design — Grand Challenge initiative is raising.

> **Challenge # 5. Ubiquity:** The proposed Verified Software for Ubiquitous Computing project is expected to contribute to at least the tools are needed for design and analysis of constantly adapting and evolving computing system and to their underlying theory.

4.6 Compositionality

We have made some vague, non-committed references to 'decomposition' of the domain description, the requirements prescription and the software design tasks (the latter in terms of 'isolated' software packages composed with 'co-ordination software'.

> **Challenge # 6. Compositionality:** Does the established computer science work on compositionality, for example as illustrated by [64, 65, 66, 67, 68, 69], apply here? Or must new concepts be developed?

This version of the compositionality concept apply specifically to components and objects.

4.7 Formalisation of Machine Requirements

Machine requirements such as performance, dependability, maintainability, platforms, etc. are not easily formalisable, and, when formalisable (with some difficulty) it is not yet well understood how one "derives" implementations correct wrt. the machine requirements.

Challenge # 7. Formalisation of Machine Requirements: Ubiquitous computing has very much to do with machine requirements so a special challenge, in the small, and, it appears, as many distinct challenges, is to discover, study and make practically useful principles, techniques and tools of machine requirements and their implementation.

4.8 Requirements to Domain Relations

We claim, and show in Chap. 19 of Vol. 3 [9], that core issues of requirements, what we term the domain requirements and the interface requirements, can be systematically deduced, in interaction between the domain description, the requirements engineers and the client.

Challenge # 8. Requirements to Domain Relations: Although the engineering seems clear it also seems desirable to have a more formal understanding of the relations between requirements prescriptions and domain descriptions, and for a suitable variety of cases — for example such as appears to be offered in the context of software for ubiquitous computing.

4.9 Software Design to Requirements Relations

Many of the relations that are implied here are those that show up, it appears — and to this author certainly — in for example Michael Jackson's seminal work on 'Problem Frames' [70].

Challenge # 9. Software Design to Requirements Relations: So we pose as a challenge to more precisely understand these relations — including formally.

4.10 Program Annotation to Requirements and Domain Relations

A major cornerstone of the VSTTE ideal of program correctness is that of being able to have a machine prove correctness of program code with respect to program code annotations.

We assume (0) that programs are annotated, (1) that the annotations contain elements that clearly relate to $\mathcal{R}$equirements prescriptions, (2) that other (therefrom possibly distinct (?)) annotations that "clearly" relate to $\mathcal{D}$omain descriptions, and (3) that yet other elements (and perhaps the three kinds "overlap") that clearly relate to the programming language. The three, as we postulate, "clearly" identifiable parts appears (and this is yet a postulate) to express properties that then appears to only be provable in relation to respective semantics: ($1'$) The domain theory,($2'$) the requirements theory and ($3'$) the programming language semantics or proof system.

In view of our suggestion:

$$\mathcal{D}, \mathcal{S} \models \mathcal{R}$$

we pose the challenge:

Challenge # 10. Program Annotations: Do the postulates above ($1,1'$–$2,2'$–$3,3'$) hold. How can we express the annotations so as to [most?] clearly designate the three relations? And how is the verifying compiler going to handle all this?

5 Some Observations

Rather unusual for a paper of this kind we bring some extensive comments on an earlier version. Michael Jackson's comments were based in [2].7.

5.1 Michael Jackson's 22 March, 2006 Observations

I would be inclined to go further, in a direction that may offer some insight but would surely be fruitless for influencing the GC. A major difficulty of developing a software-intensive system that interacts with a non-formal physical or human domain is that formalisation is always only approximate. If the system is critical it becomes vital to obtain a very good approximation indeed, and this cannot, in general, be achieved by elaborating the description of domain properties. Instead it becomes necessary to structure a set of domain descriptions, with their attendant requirements and subproblem machines, so that they form an assemblage that in total is a good enough approximation. A simple case is a cascading structure in which the highest element in the cascade describes an idealised domain in which everything works 'as it should' and other elements describe a less idealised domain in which various 'faults' invalidate parts of the assumptions on which the highest element is based.

More generally still, a realistic problem is an assemblage of many subproblems, each addressing a partial requirement on the basis of a partial view of the problem world. The composition of these subproblems raises many concerns such as conflict, interference, incommensurability, and so on, which must be addressed if the subproblems and their solutions are to be combined into the desired system. This complexity of composition inevitably leads to a complexity of software structure. Without addressing this aspect of software structure the developer cannot know whether or not the behaviour of the programmed machine has a good likelihood of ensuring satisfaction of the problem requirements.

One manifestation of this difficulty is that the 'verifying compiler' version of the GC can be seen to depend on the program having an appropriate structure. If the specification is to be embedded in the program the form of pre- and post-conditions and invariants, the program must possess a structure that can accommodate the specification without introducing so many specification variables that the specification itself becomes unintelligible. To take a trivial and grossly simplified example: if a word processor specification includes an invariant on the document that is to be processed, there ought to be a program component corresponding to the whole document. If there are, instead, only components corresponding to operations on the document there will be no place in the program where the document invariant can be written.

5.2 Tony Hoare's July 31, 2006 Observations

The first subsection below is from "random paragraphs" drafted by Tony Hoare, July 2006: Theory for Verified Software, received in an e-mail.

The second subsection below is from an e-mail of It predates "random paragraphs". The bulleted items of the first subsection were part of the June e-mail from which we only bring the advice.

5.2.1 Random Paragraphs

In any practical software project, theory begins to play a role long before the project starts. Its initial role is to construct a mathematical model of the real world domain in which the projected software will be deployed, without making any commitment to a particular choice of functionality or of implementation technology.

An academic example of a useful domain model is one that describes a range of procedures for the organisation of different kinds of scientific meeting [Fisler]. An industrial example is the domain of scheduling algorithms [Doug Smith]. Domain models (often called ontologies) are of wide application in various e-sciences, to standardise terminology and logical structure of observable data, and to help solve consistency problems between experimental data bases. A domain model may make a contribution towards the analysis of systems composed of other systems (including commercial off-the shelf software). A domain model can provide a framework for classification of a range of software and hardware standards, relating them to each other and to their common purposes. A good example is the ISO Reference Model for Open Distributed Processing. Other serious applications (perhaps too serious for experimentation!) are in the service industries, including retail sales, transport, the health service, and even parts of e-government.

- *A well-structured domain model will describe all aspects of the real world that are relevant for any good software design in the area. It describes possible places to define the system boundary for any particular project.*
- *A domain model makes explicit the preconditions about the real world that have to be made in any embedded software design, especially one that is going to be formally proved.*
- *A domain model describes the whole range of possible designs for the software, and the whole range of technologies available for its realisation. It is a suitable basis for the product line approach to software design and construction.*
- *A domain model provides a logical framework for formalisation and full analysis of a range of possible customer requirements; it reduces the risk of contamination of requirements by the choice of a particular technology of implementation.*
- *A domain model enumerates and analyses the decisions that must be taken earlier or later any design project; it identifies those decisions that are independent and those that conflict, those decisions that must be taken early, and those that can be postponed. Late discovery of feature interactions can be forestalled.*

Research on domain modelling dates back at least as far as the development of VDM, Z, RAISE, and there are now many mathematical formalisms available for describing both the whole and various aspects of a domain model. For example, there are formalisms based on set theory (Z), process calculus (Pi), abstract algebra (ASM), graphical notations (UML), and mixtures of these. Each of these will be suitable for describing different aspects of the domain. It will be a challenge to use them in an appropriate combination, with formal checks of consistency between them. Work on combined use of formalisms may motivate and guide research toward unifying theories of programming.

An even more important challenge will be to validate a domain model. This may be done by a broad survey of existing implementations, showing that they can all be classified as examples (at least, in approximation) of the model. The model may suggest useful experiments on the real world. The extra effort devoted to the generality of a domain model will be very worth while, because it can be done long in advance of any particular project, and because domain models are much more re-usable than any of their specific implementations.

5.2.2 Comments, I of III

You will find the above issues covered extensively in [9].

5.2.3 Advice

I suggest you give more prominence to these (ed.: the above bulleted) advantages.

A large part of the serious work of domain modelling is a check of the correspondence between the model and the real world. I think your plans should include careful design of the earlier experiments that make this check. This would increase the value of your project far more than actual implementation of any particular one of the possible designs. This particular example project would have to be conducted in close (maybe too close) association with the automobile industry, who are already developing the technology and conducting the experiments – and even delivering the products!

I think your project will have more impact and attract more researchers if you put forward a small range of example projects. They should include a domain taken from the home, one taken from the law (perhaps election law), and perhaps a fragment of an application in (local) government. Different researchers will find different domains more attractive.

The whole point of your example projects is that each of them will deliver results that are much more general than any particular implementation (whether verified or not). Don't spoil this by implementing just one of the possibilities. If you are going to implement anything it should be a program development tool or generator that can be applied to the whole class of implementations, including perhaps an experimental environment for evaluating alternative designs.

Your document contains an example of a fragment of a model written in your favourite language (presumably). Different aspects of the model and design might be well expressed in different conceptual frameworks, for example, process calculus, the pi calculus, algebra, UML, bigraphical systems, etc. Even pictures can be quite helpful. The experienced mathematician is quite skilled in living simultaneously in a variety of modelling presentations. Such diversity should be welcomed, and theorists should be invited to develop techniques and tools that ensure consistency and coherence of the various aspects of the model. Thus domain modelling will provide motivation and source material for another Grand Challenge – Unifying Theories.

In your next step, I would advise you to 'choose your team'. Who are the world-wide leaders in the field whom you would most like to recruit to work on the project? Who are those whose antagonism to the project would be most damaging? The biblical number of apostles is twelve. When you have the leaders helping you, then you can cast your net to the whole community.

5.2.4 Comments, II of III

The current paper has already presupposed this advice.

5.3 Robin Milner's July 31, 2006 Observations

5.3.1 The E-mail Header

Thank you for putting so much effort into joining two GCs – GC6 and GC2/4 – in the context of automated traffic. I largely agree with what you are saying, and in the note at the end I try to map your way of thinking onto how we have already been thinking in GC2/4. I also comment inconclusively on how we would coordinate this junction; and finally I mention a few things that occurred to me while reading your piece.

I hope we can push this further. I'm puzzled at present as to who on the GC2/4 side will be proactive in the coordination, as opposed to proactive in their own research strand. This whole thing depends on the chance of finding the right individuals!

Here's one thought (but read my remarks below): we could avoid setting up a coordination team at the start, and instead – by luck or judgement – find just ONE (sub)-project that can arise between us because the right individuals need each other. What do you think?

5.3.2 Automated Traffic

You propose a project of 7-10 years on automated traffic. This is a valuable specialisation of the Ubicomp Grand Challenge; the GC must surely involve such special goals. The GC also aims for generic engineering principles and theories that cover a range of specialisations, and this is evident from the general form in which its goals are stated. But I can't see any way to reach these goals other than specialisations like the traffic one.

*The Ubicomp GC proposes that we identify foothill 'topics', or better 'activities', and for each activity there would be many (coordinated we hope, but loosely so) foothill 'projects', each perhaps lasting 3-4 years and tackling a specific aspect of the topic, taking care to involve at least some mix of the three GC themes: Experience, Design and Science. So I see your 'foothill project' more as a 'foothill activity'. This isn't just a matter of what words we choose; it emphasizes that to advance understanding of the 'topic' (by coordinating the *activity*) we need to identify many 'projects' with aims achievable in 3-4 years.*

With this in mind, we can think of your proposal in Section 3 as needing to be specialised to specific projects. (See Sect. 5.3.3.) Formulating achievable goals for these (sub)-projects is surely a creative task in itself. But, for example, Jean Bacon has one such project already running, involving traffic in Cambridge. I don't know the project as well as I should, but I believe it involves knitting together at least two of the GC themes: Experience (what do citizens and the City Council actually want?) and Design (what event-based model can express alternative designs to achieve this experience?)

One can think of this kind of project as helping to formulate an appropriate dynamic model, rather than as presupposing an already perfect such model.

A different project (prompted by Jon Crowcroft) could involve the comparison of two different design choices for a specific application: 'distributed control' (each car has its say) with 'centralised' control (the highway makes all the decisions). To do this comparison one still has to postulate a model in which both designs can be expressed; so the project can be regarded proposing and testing the Model as well as identifying the differences in Experience provided by the two designs.

Other projects can involve theoretical aspects of such models: how to express designs (in some mix of logics, process calculi, differential equations, and languages based on these); how to analyse the state-space of a design (by pencil and paper, or by model-checking) to *ensure that some requirements are met; how all of this can be done stochastically; etc. These theoretical aims pertain to ALL ubicomp systems of course; but theories, like engineering principles. designs, get established through specific application.*

5.3.3 Coordinating a Foothill Activity

In GC2/4 there is debate about how a foothill activity can be coordinated, and woven into the coordination of the whole GC activity. This is hard; and presumably it's even harder to do between two GCs! You already address this in 6.4.2 where you expect '20 to 30 groups' to be enlisted. The big question is: how do they interact? I'm sure it depends, as you are hinting, on gathering a steering group who really believe in the coordination as much as in their own specific research. This group ought to be more or less equally represent the two GCs. What will not work is to try to do this top-down! It follows that, like democracy, it sometimes cannot be done at all. But we can continue to try.

5.3.4 Specific Comments on DB's Early Draft of This Paper

- *At the start of 1.5 you introduce domain models as abstracting reality. I think one of our main problems is that they have to model 'future' reality, or future possible realities. That's where we have not the luxury of a natural science – where the reality is not (or is only very slowly) modified by the model!*
- *This raises a question-mark about the Dogma (Sect. 2.1), first bullet, where you say "First one must establish a ... domain theory". In GC2/4 we imagine that the model (or theory) will only be arrived at by the research; so it's one of the ultimate aims of the GC, not a first step.*

5.3.5 Comments, III of III

You will find the specific comments addressed in [9]:

6 Project Management

6.1 Self-funding

This project frame proposal does not imply central funding. That is, each partner must find own funding. Each partners is expected to contribute at least 6 person months of research per year — and typically 6-18 such months per year.

6.2 A Patchwork of Overlapping, Individualistic Contributions

Challenges # 1.–# 10. indicate rather clearly that there is room for individual groups to contribute to just one of them. It is then hoped, through "gentle coercion and persuasion" to get these or other individual groups interested in "filling the gaps".

> **Challenge # 11. Management:** It is indeed a management challenge to get as many groups to work in order to "fill" the programme, so that all challenges (# 1.–# 10.) are taken care of, and it is indeed a management challenge to make sure that the patchwork hangs together, that is, to fullfull challenge # 0.

6.3 Suggested Willing & Courageous Project Partners

It is planned to contact individuals at at least the 87 places mentioned in the below list.

- **Argentina:**
  1. ???
- **Australia:**
  2. University of New South Wales
  3. University of Sydney
  4. University of Queensland, SVC
- **Austria:**
  5. Technical University of Graz
  6. Technical University of Vienna
- **Brasil:**
  7. University of Pernambuco, Recife
- **Bulgaria:**
  8. Sofia University
  9. Bulgarian Academy of Science
- **Canada:**
  10. McMaster University, Hamilton
  11. University of Toronto
- **China:**
  12. Peking University
  13. Beijing University of Aero- and Astronautics
  14. Shanghai Jaiotong University
  15. Software Institute, Academia Sinica
- **Croatia:**
  16. University of Zagreb
- **The Czech Republic:**
  17. Charles University, Prague
  18. Czech Academy of Science, Institute of Computer Science
  19. Ths.Mazaryk University, Brno
- **Denmark:**
  20. Aalborg University
  21. Århus University, DAIMI/BRICS
  22. IT University, Copenhagen
  23. Technical University of Denmark, IMM, CSE
- **England:**
  24. Cambridge University
  25. London Southbank University
  26. University of Newcastle
  27. Oxford University
  28. University of Southampton
  29. York University
- **Estonia:**
  30. Institute of Cybernetics
  31. Technical University, Tallinn
- **France:**
  32. Grenoble University, IMAG
  33. University of Nancy
  34. University of Rennes
- **Germany:**
  35. Technical University of Berlin
  36. DFKI, Saarbruecken
  37. Humboldt University, Berlin
  38. Kiel University
  39. RWTH, Aachen
  40. Technical University of Munich
  41. Oldenburg University
  42. University of Saarbruecken
- **Greece:**
  43. University of Athens
  44. Aristotle University of Thessaloniki
- **Holland:**
  45. Amsterdam Free University
  46. CWI
  47. Technical University of Eindhoven
  48. University of Leiden
- **India:**
  49. India Institute of Technology, Mumbai
  50. TIFR
- **Ireland:**
  51. University of Limerick
  52. Trinity College Dublin

Italy:

  53. CNRS
  54. University of l'Aquila
  55. University of Florence

56. University of Pisa

- **Japan:**
  57. JAIST
  58. Kyushu University
- **Korea:**
  59. Posttech
- **Latvia:**
  60. University of Latvia
- **Lithuania:**
  61. Vilnius University
  62. University of Kaunas
- **Macau:**
  63. University of Macau
  64. UNU-IIST

New Zealand:

65. University of Auckland
66. University of Waikato

- **Portugal:**
  67. University of Minho
- **Romania:**
  68. Bucharest University
  69. Alexandru Ioan Cuza University of Iasi
  70. Institute of Mathematics "Simion Stoilow" (IMAR)
  71. Informatics Department Scoala Normala Superioara
- **Russia:**
  72. Institute of Systems Programming, RAS, Moscow
  73. A.P. Ershov Institute of Informatics, RAS, Akademgorodok
- **Serbia:**
  74. University of Belgrade
- **Singapore:**
  75. National University of Singapore
- **Sweden:**
  76. University of Uppsala
  77. SICS
  78. KTH, Royal Institute of Technology, Stockholm
- **Slovakia:**
  79. Comenius University, Bratislava
- **Slovenia:**
  80. University of Ljublana
- **Switzerland:**
  81. EPUL
  82. ETH
  83. Università della Svizzera italiana
- **Ukraine:**
  84. National Taras Shevchenko University of Kyiv
- **USA:**
  85. CMU
  86. MIT
  87. Stanford University

6.4 Project Plan

The project plan can only be very loose: This project is not centrally funded, is a research project with very many open issues, requires individuals of widely different skills that must be "fused", etcetera.

The best we can suggest at this time is for the reader to take a look at Fig. 1 on page 20 and at our sketch "milestone" plan, Sect. 6.6 on the next page.

6.5 Resource Requirements

The project is characterised by a diversity of issues, from domains via requirements to software; from integration and composition of medium to large scale specifications expressed in many "styles", transitions from domain via requirements to software design specfications, the establishment of formal relations (for these transitions), and the verification of annotated code. This calls for many groups to contribute to the project, each with their specialty, some working already on other foothill projects of either of the two underlying grand challenges.

We therefore expect a number from 20 to 30 groups to be enlisted in this project for it to succeed.

Each such group may be from one person working at least half time to several (3–4) such persons. And for typically 3–4 years.

It is expected to hold an annual "Verified Software for Ubiquitous Computing" workshop — probably as part of some event of the two underlying Grand Challenges. It is deemed mandatory that each group participate three or four consecutive times in these workshops.

6.6 "Milestones"

These are the short, medium and long term goals to be reached:

- **Short Term Goals:** Summer 2007

  The gathering of at least some 20 or more reasonably committed and reasonably strong research individuals (groups), the holding of a kick-off workshop, and hence the start of the project.

- **Medium Term Goals:**

  In chronological order we list one set of "deliverables", one could call them the "on the path from domains to verified software" deliverables:

  - **Domain Specifications** 2008–...
  - **First Set of 1–2 Requirements** 2009–...
  - **Next Set of 4–5 Requirements** 2010–...
  - **Requirements to Domain Relations** 2011–...
  - **First Set of 1–2 Software Designs** 2012–...
  - **Next Set of 4–5 Software Designs** 2013–...
  - **Software Design to Requirements Relations** 2013–...
  - **Program Annotations to Requirements and Domain Relations** 2015–...
  - **First Set of 1–2 Verified Program Code** 2016–...
  - **Next Set of 4–5 Verified Program Code** 2018–...

  We then list supporting theory "milestones". These are more trhought of as individual contributions (i.e., papers) that are produced in a hopefully steady flow:

  - **Domain Theories** 2009–...
  - **Requirements Theories** 2011–...
  - **Integration Theories** 2009–...
  - **Compositionality Theories** 2009–...
  - **Theories of Ubuiquity** 2009–...
  - **Theories of Proof Score/Scripts &c.** 2009–...

- **Long Term Goals:**

  - **Verified Software for Automated Highways** 2020–...

6.7 Project Board and Management

Obvious one must have “such a thing”: board, project leader, etc. Tony Hoare’s and Robin Milner’s comments in Sects. 5.2.3 and 5.3.3 are all relevant in this context.

7 Conclusion

We have sketched a foothill project within the two grand challenges of ‘VSTTE: Verified Software, Theories, Tools and Experiments’, and ‘Ubiquitous Computing’, the latter specifically in the context of ‘Automated Highways’. Both of the two grand challenges had set somewhat different scopes and spans for their expected activities. The current proposal attempts to bring the two together on “one hat”.

8 Acknowledgements

Discussions over the last six months with (in chronological order) Tony Hoare, Cliff Jones, Michael Jackson, Kokichi Futatsugi and Robin Milner are much appreciated.

References

[1] Tony Hoare. The Verifying Compiler: A Grand Challenge for Computing Research. Journal of the ACM, Vol. 50, No. 1, January 2003

[2] We lump a number of VSTTE papers, reports and Web pages:

1. VSTTE: Verified Software: Theories, Tools, Experiments. Conference home page, Oct. 10th-14th, 2005. http://vstte.inf.ethz.ch/ or http://vstte.ethz.ch/
2. Tony Hoare, Jayadev Misra, and N. Shankar. The IFIP Working Conference on Verified Software: Theories, Tools, Experiments; conference report. October 20, 2005. http://vstte.ethz.ch/report.html
3. Tony Hoare and Jayadev Misra. Verified Software: Theories, Tools, Experiments — Vision of a Grand Challenge project, July 2005. http://vstte.ethz.ch/pdfs/vstte-hoare-misra.pdf
4. Tony Hoare and Jayadev Misra. Verified Software: Frequently Asked Questions, 19 September 2005. http://wiki.se.inf.ethz.ch/vstte/index.php/Verified_Software:_-Frequently_Asked_Questions
5. K. Rustan M. Leino. FLoC’2006 Workshop on Verified Software: Theories, Tools, and Experiments, Summer 2006. http://research.microsoft.com/~leino/vstte2006/
6. Verified Software Roadmap 2006. A list of 16 references (in the form of URLs) are given to position statements on the topic of VSTTE. http://qpq.csl.sri.com/vsr/private/-verified-software-roadmap-2006/.
7. An early version of the current proposal was: http://qpq.csl.sri.com/vsr/private/-verified-software-roadmap-2006/dines.pdf/view.

[3] We lump a number of Ubiquitous Computing papers, reports and Web pages:

1. Ubiquitous Computing Grand Challenge: Introduction http://www-dse.doc.ic.ac.uk/-Projects/UbiNet/GC/
2. Ubiquitous Computing Grand Challenge: Overview http://www-dse.doc.ic.ac.uk/-Projects/UbiNet/GC/overview.html
3. Ubiquitous Computing, The Computer Journal.2006; 49: 390-399. http://-www.bcs.org/BCS/Awards/Events/cjlectures/ubiquitous.htm and http://comjnl.-oxfordjournals.org/cgi/content/full/49/4/390
4. Ubiquitous Computing: Shall we understand it? Follow-up discussion on the above: http://www.bcs.org/server.php?show=ConWebDoc.4708&setPaginate=No
5. Wikipedia's URL on Ubiquitous Computing http://en.wikipedia.org/wiki/Ubiquitous_-computing

[4] We lump a number of Automated Highway papers, reports and Web pages:

1. Wikipedia: Automated Highway System. http://en.wikipedia.org/wiki/Automated_-highway_system.
2. Foothill Project: Automating the Highway. http://www-dse.doc.ic.ac.uk/Projects/-UbiNet/GC/Manifesto/fp-automatinghighway.html.
3. A longer paper addressing these issues of item 2 is also available: http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/road.pdf

[5] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.

[6] Dines Bjørner and C.B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer–Verlag, 1978. This was the first monograph on *Meta-IV*. .

[7] Dines Bjørner and C.B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[8] John S. Fitzgerald and Peter Gorm Larsen. *Developing Software using VDM-SL*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.

[9] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[10] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[11] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.

[12] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[13] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[14] Daniel Jackson. *Software Abstractions Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[15] K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. *CAFE: An Industrial–Strength Algebraic Formal Method*, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.

[16] Kokichi Futatsugi and Razvan Diaconescu. *CafeOBJ Report The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. AMAST Series in Computing – Vol. 6. World Scientific Publishing Co. Pte. Ltd., 5 Toh Tuck Link, SINGAPORE 596224. Tel: 65-6466-5775, Fax: 65-6467-7667, E-mail: wspc@wspc.com.sg, 1998.

[17] Ražvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical Foundations and Methodology. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [71, 72, 20, 73, 49, 28] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[18] Michel Bidoit and Peter D. Mosses. CASL *User Manual*. LNCS 2900 (IFIP Series). Springer, 2004. With chapters by T. Mossakowski, D. Sannella, and A. Tarlecki.

[19] CoFI (The Common Framework Initiative). CASL *Reference Manual*, volume 2960 of *Lecture Notes in Computer Science (IFIP Series)*. Springer–Verlag, 2004.

[20] Till Mossakowski, Anne E. Haxthausen, Don Sanella, and Andzrej Tarlecki. CASL — The Common Algebraic Specification Language: Semantics and Proof Theory. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [71, 72, 17, 73, 49, 28] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[21] Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[22] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997.

[23] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.

[24] Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.-usingcsp.com/cspbook.pdf`, 2004. Second edition of [21]. See also `http://www.-usingcsp.com/`.

[25] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, January 1988.

[26] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[27] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

[28] Martin C. Henson, Steve Reeves, and Jonathan P. Bowen. Z Logic and its Consequences. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [71, 72, 17, 20, 73, 49] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[29] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real––time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

[30] Chao Chen Zhou, Charles Anthony Richar Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5), 1992.

[31] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.

[32] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.

[33] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.

[34] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19:45–80, 2001. Early version appeared as Weizmann Institute Tech. Report CS98-09, April 1998. An abridged version appeared in *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-based Distributed Systems* (FMOODS'99), Kluwer, 1999, pp. 293–312.

[35] David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[36] Jochen Klose and Hartmut Wittke. An automata based interpretation of Live Sequence Charts. In T. Margaria and W. Yi, editors, *TACAS 2001*, LNCS 2031, pages 512–527. Springer-Verlag, 2001.

[37] Kurt Jensen. *Coloured Petri Nets*, volume 1: Basic Concepts (234 pages + xii), Vol. 2: Analysis Methods (174 pages + x), Vol. 3: Practical Use (265 pages + xi) of *EATCS Monographs in Theoretical Computer Science*. Springer–Verlag, Heidelberg, 1985, revised and corrected second version: 1997.

[38] Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[39] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.

[40] Wolfang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.

[41] Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag, December 1998. xi + 302 pages.

[42] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[43] David Harel. On visual formalisms. *Communications of the ACM*, 33(5), 514–530 1988.

[44] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark B. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *Software Engineering*, 16(4):403–414, 1990.

[45] David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.

[46] David Harel and Eran Gery. Executable object modeling with Statecharts. *IEEE Computer*, 30(7):31–42, 1997.

[47] Leslie Lamport. The Temporal Logic of Actions. *Transactions on Programming Languages and Systems*, 16(3):872–923, 1995.

[48] Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.

[49] Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [71, 72, 17, 20, 73, 28] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[50] Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors. *IFM 1999: Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*, York, UK, June 1999. Springer. Proceedings of 1st Intl. Conf. on IFM.

[51] Wolfgang Grieskamp, Thomas Santen, and Bill Stoddart, editors. *IFM 2000: Integrated Formal Methods*, volume of *Lecture Notes in Computer Science*, Schloss Dagstuhl, Germany, November 1-3 2000. Springer. Proceedings of 2nd Intl. Conf. on IFM.

[52] Michael J. Butler, Luigia Petre, and Kaisa Sere, editors. *IFM 2002: Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, Turku, Finland, May 15-18 2002. Springer. Proceedings of 3rd Intl. Conf. on IFM. ISBN 3-540-43703-7.

[53] Eerke A. Boiten, John Derrick, and Graeme Smith, editors. *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, London, England, April 4-7 2004. Springer. Proceedings of 4th Intl. Conf. on IFM. ISBN 3-540-21377-5.

[54] Judi M.T. Romijn, Graeme P. Smith, and Jaco C. van de Pol, editors. *IFM 2005: Integrated Formal Methods*, volume 3771 of *Lecture Notes in Computer Science*, Eindhoven, The Netherlands, December 2005. Springer. Proceedings of 5th Intl. Conf. on IFM. ISBN 3-540-30492-4.

[55] Richard Feynmann, Robert Leighton, and Matthew Sands. *The Feynmann Lectures on Physics*, volume Volumes I–II–II. Addison-Wesley, California Institute of Technology, 1963.

[56] R. Milner. Models of LCF. In K. Apt, editor, *Mathematical Centre Tracts 82: Found. of Comp. Sci. II, part 2*, pages 49–63. Mathematisch Centrum, 1976.

[57] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[58] R. Milner. LCF: A way of doing proofs with a machine. In *Proceedings: Math. Found. of Computer Sci.*, pages 146–159. Vol. 74 of Lecture Notes in Computer Science, Springer-Verlag, 1979.

[59] M. Gordon, R. Milner, and C. Wadsworth. Edinburgh LCF. In *Lecture Notes in Computer Science, Vol. 78*. Springer-Verlag, 1980.

[60] Lawrence Paulson. *Logic and Computation: Interactive proof with Cambridge LCF*. Cambridge University Press, 1987.

[61] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[62] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[63] Sam Owre and N. Shankar. Writing PVS proof strategies. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number CP-2003-212448 in NASA Conference Publication, pages 1–15, Hampton, VA, September 2003. NASA Langley Research Center. The complete proccedings are available at http://research.nianet.org/fm-at-nia/STRATA2003/.

[64] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Number 54 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, November 29, 2001. ISBN 0 52180608 9, xxiv+776 pages, 5 tables, 84 figures, and 156 excercises.

[65] Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul De Roever, editors. *FMCO 2002: Formal Methods For Components and Objects, I*, volume 2852 of *Revised Lecture Series, Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, November 5–8, 2002.

[66] Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul De Roever, editors. *FMCO 2003: Formal Methods For Components and Objects, II*, volume 3188 of *Revised Lecture Series, Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, November 4–7, 2003.

[67] Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul De Roever, editors. *FMCO 2004: Formal Methods For Components and Objects, III*, volume 3657 of *Revised Lecture Series, Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany, November 2–5, 2004.

[68] Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul De Roever, editors. *FMCO 2005: Formal Methods For Components and Objects, IV*. Fairleigh Dickinson University Press, 1–4 November, 2005.

[69] Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul De Roever, editors. *FMCO 2006: Formal Methods For Components and Objects, V*, volume to be assigned. to be assigned, 7–10 November, 2006.

[70] Michael A. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems*. ACM Press, Pearson Education. Addison–Wesley, Edinburgh Gate, Harlow CM20 2JE, England, 2001.

[71] Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [72, 17, 20, 73, 49, 28] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[72] Dominique Cansell and Dominique Méry. Logical Foundations of the B Method. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [71, 17, 20, 73, 49, 28] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[73] Chris W. George and Anne E. Haxthausen. The Logic of the RAISE Specification Language. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [71, 72, 17, 20, 49, 28] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.