Communicating Process Architectures 2000
P.H. Welch and A.W.P. Bakkers (Eds.)
IOS Press, 2000

CSP: Arriving at the CHANnel Island
(an Industrial Practitioner's Diary: in Search of a New Fairway)

Øyvind Teig
Navia Maritime AS, division Autronica
7005 Trondheim, Norway
oyvind.teig@autronica.no

Abstract. This paper is a non-academic hands-on log of practical experiences of software engineering problems, where the process leading to the decision to use CSP to program real-time systems is on trial. A new and hopefully objective decision process is instigated. What we have previously learnt by using CSP in embedded real-time process control is used as subjective basis (or bias?) for the new process. The conclusion seems to be that CSP should be sufficiently future-proof to justify its use even in new projects. The term "CSP" is here used as shorthand for both the CSP language proper and different implementations of subsets.

1. Introduction

This paper came about after I read Hammond & Keeney's article "Making smart choices in engineering" [1], which states that making decisions is the most important thing an engineer does. The article says, "many poor choices result from falling back on the *default alternative*". In our case using CSP [2] is the default alternative. Starting from scratch, would it be possible to choose CSP another time, now when OO ("Object Orientation") and other techniques flourish? Could I write a paper outlined with the same 8 points that Hammond's article outlines? The following is an attempt. Since it mainly describes a process it does not describe each theory or procedure mentioned, but a thorough reference list is provided.

This paper targets real-time programmers who, like myself, often feel that acronyms and techniques pass by like pictures in a flickering movie. I cannot guarantee that no new acronyms will be added, but an attempt at bundling them will be tried.

CSP (Communicating Sequential Processes) is a theory with algebraic properties (i.e. *process algebra*) that is used for description of and reasoning about concurrent systems. One could say that CSP is a "pattern" for concurrent programming. CSP may be used both in the specification, design and implementation phases. CSP is not as well established as OO, we shall discuss later how well the "compare operator" works between the two. I am not aware of any inexpensive CSP tools (apart from the shareware occam [3] compilers) - this raises the threshold of any first usage.

In our context CSP has more got the sense of the ideas rather than the formal CSP, more using occam or "C/C++/Java plus a CSP API" than modeling with CSP tools, although the latter has also been attempted.

There is a difference between a theory and an implementation of it. Still we have taken the freedom to use "CSP" as a shorthand for both the theory and different implementations of subsets of it, even if this usage of the term is formally not correct.

Navia Maritime AS, division Autronica is a subsidiary that develops, produces and sells monitoring equipment for ship and offshore markets. The subsidiary has some 250 employees, the headquarter is located in Trondheim, Norway [4]. (Navia has recently been purchased by Kongsberg Gruppen ASA, and "Autronica" will soon appear under a different company name, continuing the "Autronica" branded products.)

This paper has been written with an external reader in mind even if the referred context is Autronica.

2. Problem

"Define your decision problem to solve the right problem"

We see several problems we would like to solve:

1. *How do we "program" "good" real-time systems?*
   Real-time programming domain.
   We want to program once and for all, not debug for years. "Program" covers analysis to design and implementation.
   Is CSP any good?

2. *Why do we choose what we choose?*
   Cognitive psychology domain.
   We want to know why we do it in a certain way, and maybe why most people don't do it that way. This is where past experience, mental rigidity vs. flexibility and intuition are issues. This is also the point where the reader's own bias and experience will influence his view of what he reads.
   Is my CSP bias & experience worth anything?

3. *What makes boxes of such a quality that they may be of any help to people?*
   Ethical domain.
   We want the result to be a trusted entity, whereas much software isn't trusted.
   Will CSP help?

According to Hammond's article, to continue asking "why" until we cannot go any further will take us from a *means* to an *end* or a *fundamental objective*. Asking "how" will take us from ends back to means, leading us towards alternatives. Then we use means to *generate* alternatives and ends to *compare* alternatives. Asking "what" should help us identify the components of our objectives. (Aside: this process could probably be modeled with CSP and analyzed for any logical flaws!)

Trying this out on the list above reveals that point (1) rephrased is a means (*We want to "program" "good" real-time systems*) and (3) rephrased is an end (*We want to make boxes of such a quality that they may be of any help to people*). Point (2) probably is a carrier for both; the results of analysis of (1) and (3) depend upon experience, intuition and our value system.

The result of our investigation also makes our living and feeds our families; much depend on us making the products correctly.

Observe that we stand in the middle of history. We want to learn from past experience and hopefully make wise decisions for the future. Reference to what we have done is therefore just as important as trying to "read" the future.

3. Objectives

"Clarify what you're really trying to achieve with your decision"

This is twofold: I would like to find out how to technically write "good" real-time programs, and I would like to find out if we could choose CSP again. With the transputer experience occam was the evident programming language, and as we know, occam is a "runnable" implementation of a subset of CSP. All thinking had already been imprinted by its designers, only after years of usage we discovered "CSP" behind it. Our entry was coincidental.

We want to find out whether CSP offers a good alternative also for programming of future concurrent systems.

We have no objective to find out about the economy involved, discussing the economics in each alternative would require more knowledge than we presently have, and it would certainly swell this report.

4. Alternatives

"Create better alternatives to choose from"

As of today there are many alternatives to CSP. We may choose to *model* the whole system, or more probable, parts of it. We could use SDL (CCITT Specification and Description Language) [5] and have the tool generate runnable programs with several languages to choose from. SDL has a long record with real-time programming, and has excellent (expensive) commercial tools. We could model in UML (Unified Modeling Language) [6] and have C++ or Java code skeletons generated. The UML designers are working on real-time UML at this moment. Other approaches could be DARTS (Design Approach for Real-Time Systems) [7] or ROOM (Real-time Object-Oriented Modeling) [8] methods. FDR2 [9] is a tool for CSP, much used and acclaimed. A simpler version of a CSP-like language is FSP (Finite State Processes), its corresponding LTSA (Labeled Transition System Analyser) tool [10] is free. The list is not exhaustive. Many commercial CASE tools would support a range of different techniques.

Multithreaded applications may also be modeled with Petri nets. Microsoft has shown an example of this [11]. They also have a tool to help detect deadlocks at runtime [12]. By doing a short search on the internet a host of Petri net tools turned up.

We could also use a general drawing tool like Visio [13] and draw class diagrams, process-dataflow diagrams, message sequence charts to name some. The older SADT (Structured Analysis and Design Technique) has some traits that fit well for a less formal approach. We could use plain English to describe the system, which we would in any case do at a certain level. From own experience we have seen that web-based documents with links, for example to the program sources, is well suited. We have also used literate programming [14] with success at the system level - with html documents as sources, containing programs as well as descriptions and figures.

When it comes to the implementation phase we can program in native C. This is the most "evident" approach, as VxWorks [15] is already running on the in-house main embedded platform based on Intel 386ex. C++ is also becoming popular, the GNU compiler that comes with VxWorks does support it. Java real-time is coming [16],[17], its initial boost surely will be over us in a year. The Java CSP (JCSP) library [18] could be a way to program concurrent systems in Java. A similar Java CSP API (CTJ) also exists [19]. In any case the target needs a run-time system that can handle processes, and a more general operating-system type API that can handle communication or any GUI needs, etc. We have used occam both with transputers and lately (presently for the second time) with Texas Instrument's TMS320C32 DSP. In the latter case the SPoC occam compiler [20] generates C from occam, bundled with a co-operative (non preemptive) scheduler. We now also have occam/SPoC code compiled under Microsoft Visual C++, with socket output/input signals connected directly to listening occam channels. We have described our experience in [21],[22],[23]. Another, recent article which tries to describe CSP from an embedded C?-programmer's perspective is [24]. We have also recently had the CCSP library [25] ported to VxWorks for Autronica, and will use it in an embedded server. CCSP should also make our application portable to a board running Linux [26].

5. Consequences

"Describe how each alternative meets your objectives"

This chapter is where cognitive psychology comes into play. I could tabulate each methodology against the initial objectives and weigh each entry - it *may* be more objective, or I may just believe it is. Experience has taught me that the final result is more subjective than I like to admit. If relying on tables only is to ditch on one side, clean belief is to ditch on the other side. As an engineer I have to choose, and it has to be *on* the road because I want to get somewhere.

In choosing between OO and non-OO I would say that both *more or less* seem to meet our objectives. For real-time systems automatic garbage collection (GC) seems to be a problem that real-time Java (an OO language) fights, so I would study any Java implementation to see if I could accept it, or how I could program around automatic GC.

I must admit that I would not be comfortable with using clean OO in a real-time system. There has to be at least a "thread" (=process) construct, my discomfort would be lessened the better the selected tool understands what a process really is.

Les Hatton has shown that OO seems to increase the number of errors [27]. He thinks the brain has difficulty grasping the concepts which *implicit and invisible* layering often inserts. Personally I like much (but not all) of the OO programs I have written or used, but the OO languages are real-time wise immature at the moment. None of them are thread safe by birth, and encapsulation is not tight enough (what do we need "public" non constant variables for?). We are often left with design rules.

I would hesitate to use full C++'s for a real-time system, simply because it is rather complicated, even if its lack of automatic GC makes it somewhat attractive. I would also hesitate to use bare C on top of, for example VxWorks, because C plus real-time API (not even Posix API) is not a "modern" real-time system programming language. It is too unsafe.

CSP is the paradigm that I like the most, and I feel I am allowed to say so. CSP makes it easy to reason about real-time processes, encapsulation is 100% water-tight, and it has proven to work. In a private communication with Bran Selic [28] he writes that:

> *"Actually, I claim that CSP is an object-based language since the dominant aspect of a CSP program is the notion of collaborating processes. Even though the word "sequential" appears in CSP, that sequentiality (i.e., representing the procedural/ algorithmic programming paradigm) is packaged into objects (processes) that are arranged into some type of communication matrix/structure. You may argue that I am stretching the definition of what OO means, but to me it is quite clear: the object paradigm is a computational model in which my program is expressed as a network of collaborating submachines. On top of this are other "object-like" things such as inheritance, polymorphism, and encapsulation - but, in my view all of those are secondary elements. I am sure that you would agree that it is not correct to think of CSP as representing a procedural programming style."*

This has by some always been the definition of OO. However, to align with the rest of the world, that OO definition is not the basis of the other OO talk in this article. The practical CSP alternatives seem for us to be clean occam, Java CSP (JCSP or CTJ) or C+CCSP + VxWorks/Linux.

The CSP paradigm seems to meet our objectives quite well. We now are able to describe how we program real-time systems: use CSP, forget the unsecure critical regions, semaphores and monitors. We seem to be able to describe why we have chosen it: experience shows that it works, and an increasing number of people seem to discover it. Doug Lea in the new revision of his Java concurrency book has now included a chapter on CSP with examples given with JCSP [29]. We also seem to be able to make good boxes for people, meaning that they should at least function.

Observe that use of CSP would not hinder some combined use of OO. Occam has better encapsulation than the OO languages I have seen. The occam CHANnel is in a way a thread-safe method interface definition. With JCSP we are free to use OO, but we must abide by the CSP rules when its *CSProcess* class is used. These rules must also be followed with CCSP. In all cases we could use from all to some of UML's diagram types, and we could use process/ data-flow diagrams. CSP does not exclude good modeling or architectural thinking, on the contrary. Also, CSP is in itself rather sufficient to describe a system.

Almost off the record I want to say something about program text editing. An editor that has folding [30] is a quite good semi-graphical environment in itself. The mature IDEs of today's development systems would be even better if they would allow also the source program part in the right window to collapse (be folded and given a comment heading) in addition to the left reference window (used for overview and navigation). Since the CSP channel input or output does *not* have a procedural style source program, even the best IDEs seem inadequate. Folding solves this quite well, and the concept is also good for other languages like C, C++, Java or Perl.

6. Tradeoffs

"Make tough compromises when you can't achieve all your objectives at once"

Have we chosen (or manipulated) our objectives so well that we do not seem to have to make any "tough compromises"?

One compromise might be that we seem to want to use occam, JCSP and C+CCSP on different platforms. Whenever we want TCP/IP or a GUI we have *not* used SPoC occam. Occam runs on Texas DSPs and will also be used on a host PC in batch mode. C+CCSP will run on top of VxWorks and probably Linux. JCSP has only so far been something we are looking at, but it seems more attractive than J Consortium's or Sun's real-time Javas, even if it does not address some of the low-level problems that those alternatives address.

7. Uncertainty

"Identify and quantify the major uncertainties affecting your decision"

There are several uncertainties with CSP. From the occam experience I would say that it was easy to understand and start to use, but from then on the learning curve of this *tiny* language has been, not steep, but long. The programmer next door may not understand what is going on. I think the same would go for any use of CSP. Even if some universities seem to have started teaching concurrency based on CSP (for example [31],[32]), without some CSP basis more traditional real-time programmers could easily feel stranded.

With occam an uncertainty may be that the language is not commercially supported any more. And neither CCSP nor JCSP are commercial products. SPoC is supported by Southampton University, CCSP and JCSP have been written by people at the University of Kent at Canterbury. In each case the need to actually have the sources available must be traded against what we could actually do with those sources provided we had to. Since we are not compiler experts, compiler sources would be for archive only.

CSP has (1) synchronous and (2) unbuffered communication over (3) unidirectional channels. From experience we have seen how difficult it is to grasp this simple behaviour. Real-time programmers are so used to asynchronous flow, with data that may be lost caused by for example buffer overflow, that whenever synchronous flow which cannot be lost is needed, it is so easy to program something that loses data anyhow. The difference between waiting for 400 ms and waiting "forever" is hard to grasp - both for inputs and outputs. Then, when CSP needs to encounter for lost data (typically at terminal points), this has to be explicitly programmed. This makes me anxious about the mental friction that CSP may cause: "the wires are so stiff"! The ALT structure of occam does allow us to introduce a *switch* in that stiff wire and no data will flow before the server closes the switch again - (`theSwitch & inputChannel ? hisData`). This is one of the pillars of CSP. Synchronous communication *must* be it.

Another uncertainty is that with these stiff channels, the programmer has to have a deep understanding of how deadlocks should be avoided: either use a CSP tool to model and analyse, use well established guidelines [33], or use design rules like buffers or overflow buffers. With infinite buffers as in SDL, deadlock feels harder to get, but it is (to me) difficult to understand why deadlock seems to be a non issue. One of the reasons for this could be SDL's infinite buffers and a design rule to always evaluate the need to timeout on input signals. An SDL tool would help finding a deadlock once it has happened. But two

CSP processes sending spontaneously and unbuffered to one another cannot time out to remedy the potential deadlock. A CSP tool will pinpoint this before run-time. (Timing out on *output* is not possible in occam's basic mechanisms.) Both with CSP and SDL or any real-time system based on concurrent processes, safety and liveness properties *are* issues, per definition. SDL is here also metaphorical for many message passing systems, just make buffers large enough is the traditional solution - and surprise is evident if the watchdog resets the shiny black box.

With the fine-grained parallelism that CSP inspires, there may be a tendency to use more memory than a corresponding (often much more complex and therefore more error prone) sequential version - for the local buffers within each process. With occam this is often counterbalanced by its static structure, each process is given its needed memory, since the compiler computes *exact* stack depth. Occam for transputers did this for all processes, stack overflow was impossible. SPoC has one stack for all processes, so there is a possibility for *one* stack overflow. Traditional real-time systems would have one stack overflow possibility per process (makes parallelism medium-grained, really), and the programmer would have to ensure enough memory - an often hard job with possibly fatal consequences. These matters have to be studied when physical memory constraints are known.

Looking at any CSP book displays a lot of mathematical formulae. This may prove to be an uncertainty when it comes to level of acceptance. I would say that it was possible to learn occam without knowing the mathematical background, so starting using for example the CCSP API would also help in a gradual understanding of the theoretical background - or what it is all about.

When it comes to support from upper management, it would not be expected to know more about CSP than the traditional real-time programmer. Since there is so much experience at Autronica to the support of CSP, we would not expect any new management to inflict more than with the required amount of counterweight.

I would like to display another of Bran Selic's comments [28]:

> *"As for CSP: it too has some major failings. Occam has not been very successful for a number of reasons, but one of them is that its model of the world is too simplistic. Hence, it does not cope well with the complexity of the real world. I once saw an attempt to implement telephone call processing with LOTOS (a derivative of CSP) and it failed miserably, for this reason. However, despite these practical limitations of CSP, my point remains that CSP is an object-based solution. Hence, it is not an alternative to OO, in my view, but a special (if somewhat constrained) realization of it.*"

CSP is "not an alternative to OO", so there is no reason to wipe the desk of either of them. The first part is a comment that I have often met, a little bewildered. At best it is an argument to language designers to develop occam further.

8. Risk Tolerance

"Account for your appetite for risk"

We would generally not have a very big appetite for risk, but the unpleasant fact is that risk is inevitable. When a compiler or a library is updated, even well established software

houses deliver more than enough risk. During the time that we have used SPoC we have altogether reported about 60 nice-to-have's, warnings, errors or critical errors. Critical errors have always been removed by the SPoC author (the exception is an error that could potentially lead to wrong code being generated according to SPoC's author, but we haven't seen this happen). Texas Instruments report about the same amount of errors in their DSP C compiler [34], but the (open) loop from us to TI (or SPoC's author for that sake) back to us with a new compiler is not controllable.

Still, both SPoC and the C-compiler have caused surprisingly few problems. Since SPoC generates C, we have an extra verification with the C compiler (and then another level with PC-lint).

With CCSP we will have access to the C sources. This is not a compiler, so trying to track errors in it (if its author has been driven over by a truck) would at least be in the same problem domain as we are used to.

We also must remember that software does have a lifespan. At Autronica we have seen that it has been possible to keep a local island of long-ago outdated hardware and software: we *still* have a Texas Instruments computer and development system for Microprocessor Pascal (based on Brinch Hansen's ideas) from 1978. So keeping functioning software on ice should be possible for a long time, especially batch jobs like compilers.

9. Linked decisions

"Plan ahead by effectively coordinating current and future decisions"

Should we decide to (continue to) go for CSP, in an optimistic scenario we would expect more developers to become interested in it, we can already see a tendency. Will the concept withstand being scaled to more projects? We think so. Following this, courses and (more) tools probably will be requested.

If we were to decide to abandon occam in new projects, one would of course have to know occam to regret it. However, if the language develops and compilers are updated, we may see an extended usage. It will be held onto as long as there is any rope available.

C+CCSP does handle CSP, but with the rather unsafe C language which has no CSP understanding. Terminating occam to start using C+CCSP only would be one step ahead and two back. Still, we believe that the CSP paradigm is important as a discipline in its own right. Also, everyone has a love/hate relationship with C and many seem to like its free fall feeling. Then, after a while, they would look further: "what was this CSP (occam?) you talked about?"

If or when Java arrives on our embedded platforms we would certainly try JCSP, CTJ or any other CSP implementation on top Java. This is the reason why we already now have had a look at JCSP.

One major linked "decision" with many non-commercial systems like SPoC or CCSP seems to be that a rather large amount of time is spent just being a more or less voluntary beta-tester, and then wait for, hope or demand updates. Our experience with actually paying hard money for updates or ports is rather good. We have then wished the results to be available to all, but the delays may approach year(s). Having said this, we must emphasise again that the systems we have used seem to be quite stable even with rather normal length error lists.

The CSP community should absolutely go beyond pure shareware. CSP is worth paying for. To get updates we need to pay, but the entry-level should be low-priced. They also *sell* at bazaars, don't they?

Also, introducing new functionality takes time. Porting SPoC to Windows for Visual C++, with blocking sockets connected to occam channels - and TIMER support; or connecting end-of-DMA interrupt on a DSP to an occam channel, are among the jobs we would have been glad to have seen in the package. Again, this may be sugar for a software engineer longing for a short break. Most projects, however, seem to have a large amount of this kind of work.

JCSP has brought CSP to a windowing environment. Instead of the Java AWT event model, buttons and dialogue boxes will be active CSProcesses and send button-press messages over channels to the other CSProcesses by the standard channel mechanism. This has also been done with the KRoC occam compiler [35] and Motif. Extrapolating from there, maybe we could try CSP also in our host GUI based programs. Microsoft Visual C++ (or Windows CE) handles threads/processes in a rather archaic way with calls like *WaitForMultipleObjects*, *SendMessage*, *EnterCriticalSection* etc. This is not CSP. We must strive higher.

From this, building *any* program as a group of communicating *parallel* processes should be viable (internally they each run a sequence of *serial* statements). This should not exclude OO usage if the code is written in an OO language.

The tendency to seek information which support our suppositions, and de-emphasise or ignore conflicting information - the *decision trap*, or overconfidence - is a common trait in human judgements. In this article we are open to it, but we must also be careful to absorb both positive and negative information. A linked "decision" therefore must be to strive to be open, both for CSP and the alternatives. There is a limit to how "loyal" one might be.

I will end with a citation by Bill Foote, Sun (who has worked in the EmbeddedJava/PersonalJava group and done some standards work for the Real Time Extensions for the Java Platform.) [36].

> *"The OO paradigm shift was a great advance in the state of the art of software engineering. IMHO the next one that's needed is a way of really understanding and modeling concurrency. The kind of encapsulation that OO offers does a lot of great things, but it doesn't seem to be adequate for concurrency issues. Concurrency breaks OO-style encapsulation, an we're reduced to coding conventions, documentation, and discipline.*
>
> *My crystal ball says that the next leap (or at least an upcoming leap) in software methodologies will be when someone comes up with a formalism that _really_ captures concurrent design, and when that formalism catches on (the way OO has).*
>
> *CSP is the most promising entrant I've seen so far."*

Others seem to enter this ship. There is a fairway. CHANnel Island is perhaps closer than Atlantis.

Acknowledgements

I would like to thank Dr. Ir. Johan P.E. Sunter of Philips TASS, NL, for reading a draft of this article and giving instructive comments.

References

[1] John S. Hammond and Ralph L. Keeney, "Making smart choices in engineering", IEEE Spectrum , Nov.1999. From the book "Smart Choices", Harvard Business School Press; ISBN: 0875848575

[2] C.A.R. Hoare, "Communicating Sequential Processes". Prentice Hall, 1985. Also see the CSP archive at archive.comlab.ox.ac.uk/csp.html

[3] "occam2 Reference Manual". INMOS Ltd, Prentice Hall, 1988 (C.A.R. Hoare is series editor) ISBN 0-13-629312-3. Also see occam archive at archive.comlab.ox.ac.uk/occam.html.

[4] Navia Maritime AS, division Autronica: www.autronica.no. R/D publications available at www.autronica.no/pub/tech/rd/index.htm

[5] Rolf Bræk, Øystein Haugen, "Engineering real time systems", Prentice-Hall, 1993, ISDN 0-13-034448-6. Also see SDL Forum Society at www.sdl-forum.org/.

[6] M. Fowler, "UML Distilled", Addison Wesley, 1997, 0-201-32563-2. Also see Object Management Group's UML page at www.omg.org/uml/.

[7] Hassan Gomaa, "Software Design methods for Concurrent and Real-Time Systems". Addison-Wesley. 1996. ISBN 0-201-52577-1. (The Artisan tool Real-time Studio supports this methodology www.artisansw.com/)

[8] Bran Selic [28], Garth Gullekson, Paul T. Ward, "Real-time Object-Oriented Modeling", Wiley. 1994. ISBN 0-471-59917-4. The ROOM methodology is described in this book.

[9] FDR (Failures-Divergence Refinement), an CSP tool from Formal Systems (Europe) Limited , see www.formal.demon.co.uk/FDR2.html

[10] Jeff Magee & Jeff Kramer, "Concurrency: State Models & Java Programs", Wiley 1999, ISBN 0-471-98710-7. Also see www-dse.doc.ic.ac.uk/concurrency/. The initial version of FSP did not have sequential processes. It now has.

[11] Ruediger R. Asche, "Detecting Deadlocks in Multithreaded Win32 Applications", Microsoft Developer Network Technology Group (Came with the CD)

[12] Ruediger R. Asche, "Putting DLDETECT to Work", Microsoft Developer Network Technology Group (Came with the CD)

[13] Visio, by Visio Corporation (a diagramming tool for Microsoft Windows)

[14] Literate programming. We have a tool that was developed by Sverre Hendseth

(sverre.hendseth@clustra.com) that extracts program sources from a hierarchy of html files. Try the The Literate Programming FAQ at shelob.ce.ttu.edu/daves/lpfaq/faq.html

[15] VxWorks by WindRiver. See www.wrs.com/

[16] Real-time Java specification from Sun: The Real Time Specification for Java Experts Group, see www.rtj.org/.

[17] Real-time Java specification from J Consortium. See www.j-consortium.org. (Autronica is an "Informational member")

[18] P.H. Welch, University of Kent at Canterbury, UK, Java Communicating Sequential Processes (JCSP), a Java CSP library, home page wotug.ukc.ac.uk/parallel/languages/java/jcsp/. Examples in [29] and [24].

[19] G.H.Hilderink, University of Twente, NL, Communicating Threads for Java (CTJ), a Java CSP library, home page www.rt.el.utwente.nl/javapp/

[20] Mark Debbage, Mark Hill, Sean Wykes, Denis Nicole, "Southampton's Portable Occam Compiler (SPOC)", In: Miles, Chalmers (ed.), "Progress in Transputer and occam Research", IOS Press, Amsterdam, 1994 (WoTUG 17 proceedings), pp.40-55. Also see gales.ecs.soton.ac.uk/software/spoc/.
`SPOC: Southampton's Portable Occam Compiler, Version 1.3b, Mon Sep 7 10:51:28 1998 M. Debbage, X. Fu, M. Hill, D. Nicole and S. Wykes University Of Southampton, ESPRIT GPMIMD P5404`

[21] Øyvind Teig, "PAR and STARTP Take the Tanks", In: P.H.Welch, A.W.P.Bakkers (ed.) "Architectures, Languages and Patterns for Parallel and Distributed Applications", pp. 1-18. 1998 (WoTUG 21 proceeding). IOS Press, Amsterdam, ISSN 1383-7575. Available at www.autronica.no/pub/tech/rd/WoTUG21_1998/PAR_and_STARTP_Take_the_Tanks.htm.

[22] Øyvind Teig, "Non-preemptive occam in DSP real-time system", Real-Time Magazine, 3Q98. Available at www.autronica.no/pub/tech/rd/RealTimeMag_3_1998/Real-Time_Magazine.htm. Also see www.dedicated-systems.com/encyc/magazine/98Q3/index983.htm

[23] Øyvind Teig, "Another side of SPoC: occam's `ALT`er ego dissected with PC-lint". WoTUG-22. Architectures, Languages and Techniques for concurrent systems, p.19-36. Edited by B.M.Cook. ISBN 90 5199 480 X, 4 274 90285 4 C3000, ISSN 1383-7575. Available at www.autronica.no/pub/tech/rd/WoTUG22_1999/Another_side_of_SPoC_occam's_ALTer_ego_dissected_with_PC-lint_7.50p.htm

[24] Øyvind Teig, "Safer multitasking with communicating sequential processes", EmbeddedSystems, Europe, June 2000, pp.57-68. Also see www.es-mag.com.

[25] J. Moores, "CCSP - A Portable CSP-Based Run-Time System supporting C and occam". WoTUG-22. Architectures, Languages and Techniques for concurrent systems, p.19-36. Edited by B.M.Cook. ISBN 90 5199 480 X, 4 274 90285 4

C3000, ISSN 1383-7575. Also see www.quickstone.com/ccsp/.

[26] Arcom SBC Media GX embedded PC board with PC/104 bus and Ethernet. We have not presently been able to install a non rotating disk Linux on it, but Arcom and we are working on it.

[27] Les Hatton, "Does OO Sync with How We Think?", IEEE Software May/June 1998

[28] Bran Selic of ObjecTime, has permitted me to publicise these two paragraphs from his e-mail of 8.Dec.1999. He is also author of [8], and active in the development of real-time UML.

[29] Doug Lea,"Concurrent Programming in Java 2.ed., Design Principles and Patterns", Addison-Wesley, 1999, ISBN 0-201-31009-0. See online supplement at g.oswego.edu/dl/cpj/index.html,

[30] We use the WinF95 folding editor from TLM Software, Bristol, see www.uk.research.att.com/~djc/files/interests/fold/tlm.htm

[31] Concurrency course at Imperial College of Science, Technology and Medicine, London, UK is based on [10].

[32] Concurrency course at Oxford University is based on "The Theory and Practice of Concurrency", A.W. Roscoe, Prentice Hall, 1998, 0-13-674409-5

[33] Jeremy Martin, Ian East, Sabah Jassim, "Design Rules for Deadlock Freedom". Transputer communications, Vol. 2(3), 121-133 (September 1994). John Wiley & Sons, ISSN 1070-454X

[34] Texas Instruments TMS320C3x/C4x Optimizing C Compiler
TMS320C3x/4x C Compiler Shell Version 5.11
Copyright (c) 1987-1999 Texas Instruments Incorporated
TI DSP C compiler error list:
www.ti.com/sc/docs/dsps/hotline/techbits/c3bug.htm

[35] Kent Retargetable occam Compiler, see www.cs.ukc.ac.uk/projects/ofa/kroc/. KROC is currently available as a full source (GPL) release for Pentium PC/Linux systems. Binary releases for Sun SPARCs, running a SunOS 4.1.3 variant or SunOS 5.5 (Solaris 2.5) or DEC Alphas running OSF1/3.0 are also downloadable wotug.ukc.ac.uk/kroc/download.shtml.

[36] Bill Foote, EmbeddedJava VM Group, Sun Microsystems. Archived at www.nist.gov/itl/div896/emaildir/rt-j/msg00312.html. Personal communication to The Requirements Working Group for Real-time Extensions for the Java™ Platform, National Institute of Standards and Technology, NIST. Also see a "thread" discussing "Higher level Real-Time" starting at www.nist.gov/itl/div896/emaildir/rt-j/msg00285.html.

Øyvind Teig is Senior Development engineer at Navia Maritime AS, division Autronica. He has worked with embedded systems more than 20 years, and is especially interested in real-time language issues. Several articles by the author are available at www.autronica.no/pub/tech/rd/.