Communicating Process Architectures 2004
Ian East, Jeremy Martin, Peter Welch, David Duce, and Mark Green (Eds.)
IOS Press, 2004

Focusing on Traces to Link $\mathcal{VCR}$ and CSP

Marc L. SMITH
Department of Computer Science, Colby College, Waterville, Maine 04901-8858, USA

Abstract. View-Centric Reasoning ($\mathcal{VCR}$) replaces CSP's [1] *perfect observer* with multiple, possibly imperfect observers. To employ view-centric reasoning within existing CSP models [2] requires a bookkeeping change. Specifically, $\mathcal{VCR}$ [3] introduces parallel events as a new primitive for constructing traces, and distinguishes two types of traces: histories and views. Previously, we gave the operational semantics of $\mathcal{VCR}$ [4], and demonstrated the utility of parallel traces to reason for the first time unambiguously about the meaning of the Linda predicate operations `rdp()` and `inp()`. The choice of using an operational semantics to describe $\mathcal{VCR}$ makes direct comparison with CSP difficult; therefore, work is ongoing to recast $\mathcal{VCR}$ denotationally, then link $\mathcal{VCR}$ with the other CSP models within Hoare and He's *Unifying Theories of Programming* [5]. Initial efforts in this direction [6] led to a comparison of $\mathcal{VCR}$ with Lawrence's $\mathcal{HCSP}$ [7]. In this paper, we present some recent insights and abstractions – inspired by modern quantum physics – that have emerged whilst contemplating parallel traces in light of the unifying theories. These insights lead to a more natural expression of $\mathcal{VCR}$ traces, in the sense that they more closely resemble CSP traces, thus forming a basis for linking $\mathcal{VCR}$ and CSP.

1 Introduction

According to Hoare and He [5], a programming theory consists of elements from three orthogonal dimensions. First, there is a set of primitive concepts, or *alphabet*, at some desired level of abstraction. Elements of an alphabet are those variables and constants that may be used in the specification of, or observed during the execution of, a program. Second, the *signature* of a programming theory is the set of primitive statements, and rules for statement composition, that may be used to specify programs. Third, a theory has a mathematical foundation, with a corresponding set of provable equations, or *laws*, that aid in the design of programs with desired properties. To compare two or more programming theories, one compares elements of their respective alphabets, signatures, and laws. Programming theories are unified by their shared elements and differentiated by their unique (relative to each other) elements. To link one theory of programming to another is to relate the two somehow, e.g., a subset or refinement relationship.

The main thrust of Hoare and He's Unifying Theories of Programming (UToP) [5] is to develop links between different theories of programming, within the discipline of computing science. Establishing these links between all theories of programming is the basis for Hoare and He's grand challenge of unification (in much the same way that Physics seeks to discover a Grand Unified Theory that accounts for all four known fundamental forces in the universe). UToP is thus not a completed body of work, but rather, a thorough and important starting point for ongoing research. View-centric reasoning ($\mathcal{VCR}$) [3] wasn't initially developed with UToP in mind; instead, $\mathcal{VCR}$ was based on CSP [1]. $\mathcal{VCR}$ replaced CSP's interleaved trace of a single observer with the *views* (traces of non-interleaved, parallel events) of multiple, possibly imperfect observers. In support of this grand challenge of unification, the current goal is to both define $\mathcal{VCR}$ as its own programming theory and link it to existing CSP models

within UToP. Indeed, CSP is one of the theories of programming illuminated in UToP, and it is linked as a subtheory of other theories of programming.

The approach of this paper departs from Smith et al. [6], where the authors began to establish links between $\mathcal{VCR}$ and Lawrence's $\mathcal{HCSP}$ [7]. $\mathcal{HCSP}$ has merged events and bags, similar in spirit to $\mathcal{VCR}$'s parallel events. Our justification for this former approach was that it made sense to link to a CSP model that has $\mathcal{VCR}$'s abstraction of parallel events. However, linking to $\mathcal{HCSP}$ represented more work than linking to the Traces model of CSP, $\mathcal{T}$, and $\mathcal{VCR}$'s abstractions were originally meant to extend $\mathcal{T}$. The work to link $\mathcal{VCR}$ to $\mathcal{T}$ is ongoing, and not presented in this paper, but new insights into traces have emerged, consistent with the approach of UToP and the theories of modern quantum physics UToP embraces, as cited in UToP [5]. Ultimately, linking $\mathcal{VCR}$ to $\mathcal{T}$ will satisfy the desire to characterize $\mathcal{VCR}$ in a way that is most accessible to the CSP community, thus permitting a more direct basis for comparison.

The remainder of this paper is organized as follows. Section 2 gives background information concerning UToP's perspectives on observation of computation within an environment, as well as the history and major tenets of $\mathcal{VCR}$. Section 3 contains the heart of this paper, describing a new characterization of traces. We conclude and discuss future work in Section 5.

2 Background

Before we present a new characterization of recording traces, some background information is in order. Beyond the information presented in this section, the reader is encouraged to consult the complete treatments of Hoare and He's Unifying Theories of Programming (UToP) [5] and Smith et al.'s View-Centric Reasoning ($\mathcal{VCR}$) [8, 4]. Section 2.1 emphasizes those aspects of UToP impacted by the inclusion of View-Centric Reasoning ($\mathcal{VCR}$) into the unifying theories. Pertinent aspects of $\mathcal{VCR}$ are introduced in Section 2.2.

2.1 UToP Perspectives on Environment and Observation

Hoare and He present theories of reactive processes in their Unifying Theories of Programming [5]. In terms of linking theories of programming, the theory of reactive processes is a subtheory of the imperative theory of designs; CSP is a further subtheory of the theory of reactive processes. The notion of environment is elucidated early in this presentation, as environment is essential to theories of reactive processes, examples of which include CSP and its derivative models. Essentially, the environment is the medium within which processes compute. Equivalently, the environment is the medium within which processes may be observed. The behavior of a sequential process may be sufficiently described by making observations only of its inputs and corresponding outputs. In contrast, the behavior of a reactive process may require additional intermediate observations.

Regarding these observations, Hoare and He borrow insight from modern quantum physics. Namely, they view the act of observation to be an interaction between a process and one or more observers in the environment. Furthermore, the roles of observers in the environment may be (and often are) played by the processes themselves! As one would expect, an interaction between such processes often affects the behavior of the processes involved.

A process, in its role as observer, may sequentially record the interactions in which it participates. Recall participation includes the act of observation. Naturally, in an environment of multiple reactive processes, simultaneous interactions may be observed. Hoare and He define a *trace* as "the sequence of interactions recorded up to some given moment in time." Thus a trace represents an ongoing chronological record of observable events that occur within the

environment of a program's execution. Traditionally, simultaneous events are recorded in some arbitrary order, since traces are considered partially, not totally, ordered. An event that represents the synchronization of multiple processes is recorded once. It is sometimes useful to consider the occurrence of an individual event to be synchronization with the environment. Likewise, an observer could be the environment, or a process within the environment.

2.2 Origins and Evolution of $\mathcal{VCR}$

View-Centric Reasoning was inspired by Hoare's CSP [1]. In particular, the central role traces play in reasoning about the behavior of processes, and the metaphor of a computation's history being recorded by an idealized observer were the basis for $\mathcal{VCR}$, before learning of Hoare and He's UToP. In its original form, view-centric reasoning was a parameterized operational semantics for reasoning about properties of concurrency. That is, $\mathcal{VCR}$ was a meta model, capable of individual instantiation via parameter specification: by passing in different sets of parameter values to $\mathcal{VCR}$'s operational semantics, the $\mathcal{VCR}$ semantics could assume the characteristics of other existing models of concurrency. The idea of developing a meta model was inspired by the author's desire to discover abstractions common to all models of parallel and distributed computation. The author does not claim to have discovered all common abstractions for concurrency, but indeed, many abstractions and parameters emerged, and $\mathcal{VCR}$ was successfully instantiated for two very different models of concurrency: Agha's Actors [9] and Gelernter's Linda [10]. For more information about early work on $\mathcal{VCR}$, see Smith [3] and more recently Smith et al. [4, 8].

From the beginning, the inspiration for $\mathcal{VCR}$ has been Hoare's CSP [1], and the elegance of using traces to reason about properties of computation. Still, $\mathcal{VCR}$ departed from CSP in several noticeable ways: the operational semantics; multiple, possibly imperfect observers (in contrast to CSP's perfect observer, an imperfect observer is capable of occasionally "blinking," and thus nondeterministically miss recording zero or more events); parallel events to obviate the need for arbitrary interleaving; and the distinction of a computation's history and views. In $\mathcal{VCR}$'s operational semantics, CSP's rich process algebra was abstracted away; processes were represented at a higher level of abstraction by their continuations. These process continuations were interpreted by meaning functions, as part of $\mathcal{VCR}$'s transition relation; that is, upon invocation, given a process's continuation, a meaning function simulates computational progress, returning the process's new continuation and any resulting observable events of interprocess communication. $\mathcal{VCR}$, not surprisingly, focused solely on constructing a computation's history and views to support view-centric reasoning. Using an operational semantics for $\mathcal{VCR}$ was an excellent choice for reasoning at this level of abstraction.

Since the environment as well as the processes within the environment could all be observers of a program's computation, $\mathcal{VCR}$ permits these multiple observers to each record their own trace. Furthermore, to account for the multiple, possibly imperfect perspectives, $\mathcal{VCR}$ introduces some bookkeeping changes. Recognizing the possibility of event simultaneity in the absence of synchronization, and the possibility for different perspectives among observers, unordered and ordered parallel events become the new primitives for recording traces in $\mathcal{VCR}$. A parallel event is represented as a multiset of events. Furthermore, two types of traces are distinguished: histories and views. A computation's history is a trace that consists of a sequence of unordered parallel events. Multiple views may be constructed from a given history, in the form of traces that consist of sequences of correspondingly ordered parallel events.

Around the time of Smith et al. [4] several points became clear. First, Hoare and He's Unifying Theories of Programming (UToP) [5] shared the goals of $\mathcal{VCR}$, but encompassed

a far broader range of concerns. Next, UToP was more mature than $\mathcal{VCR}$, and already well established. Finally, there were important benefits to linking $\mathcal{VCR}$ with the other CSP theories of programming within UToP, and UToP explicitly addresses linking together theories of programming. This last point revealed a notable disadvantage to continuing to develop $\mathcal{VCR}$ solely as an operational semantics. To draw $\mathcal{VCR}$ within the unifying theories, it would have to be expressed in the alphabetized relational calculus, and so began the authors' efforts to do so in Smith et al. [6].

3 Revisiting the CSP Trace

The original CSP metaphor concerning traces involves an infallible observer who watches a process performing a computation. Each time an observable event occurs, the observer faithfully records the event's name in a notebook. The events are recorded in the order they occur; thus the trace is chronologically ordered... almost. We are instructed to disregard the possibility that two or more events may occur simultaneously, since the observer will merely record all such events in some arbitrary sequence. Thus, to be more precise, CSP traces are partially ordered, chronologically. Consider the following example of a CSP trace:

$$tr = \langle a, b, c, d, e, f, g \rangle$$

The interpretation of trace tr is pronounced, "a then b then c...". But when reasoning about tr it is not clear which consecutive events occurred sequentially, and which consecutive events (if any) appeared to occur simultaneously from the olympian observer's frame of reference (true simultaneity, not just the perception of simultaneity by a single observer, is problematic in light of relativity, and the inspiration for views in $\mathcal{VCR}$). Since tr is partially ordered, rather than delimit the events of tr with commas, it might be clearer to delimit tr's events with the partial order relation, $\leq$, thus representing both possibilities. For further clarity, we could decorate this relation with a subscripted *c*, $\leq_c$, to reflect the chronological nature of the partial order. Once again, the same CSP trace, newly delimited:

$$tr' = \langle a \leq_c b \leq_c c \leq_c d \leq_c e \leq_c f \leq_c g \rangle$$

The interpretation of trace tr' is pronounced, "a before-or-with b before-or-with c...". It is reasonable to wonder what is accomplished by substituting the event delimiters of the trace. It would seem nothing, beyond the acknowledgment of two chronological possibilities between consecutive events in the original tr. If only we could look more closely at each individual $\leq_c$ relation in tr', and discern which of the two possibilities holds. Indeed, there is a way. Recall the CSP observer: as each event occurs, the observer *knows* in that instant whether it occurred in sequence or simultaneously with other events. In other words, total order knowledge existed even though it isn't preserved in CSP's partial order traces.

To overcome this obstacle, we borrow an abstraction from modern quantum physics: superposition. Consider $\leq_c$ to be a *quantum relation*, and instances of $\leq_c$ in tr' to be in a state of superposition. That is, for each $\leq_c$ in tr', *both* "$<_c$" and "$=_c$" remain possible states, until one relation or the other is observed upon reasoning about tr'. The observed underlying state for each $\leq_c$ relation would correspond to the CSP observer's knowledge at the time its related events were recorded. Once each $\leq_c$ relation has been observed in tr', the partially ordered trace becomes a total ordering (a "strict" interleaving) and might for example look like this:

$$tr'' = \langle a <_c b =_c c <_c d =_c e =_c f <_c g \rangle$$

The interpretation of trace tr'' is pronounced, "a before b-with-c before d-with-e-with-f before g." It is straightforward to see that trace tr'' would be equivalently expressed in a $\mathcal{VCR}$ history trace like this:

$$tr'' = \langle \{a\}, \{b, c\}, \{d, e, f\}, \{g\} \rangle$$

where event multisets represent parallel events. From either of the above two forms of tr'', $\mathcal{VCR}$ views of the other, possibly imperfect, observers (represented as lists of lists, or lists of traces) could be generated. The inner traces in $\mathcal{VCR}$ are called ROPEs (Randomly Ordered Parallel Events). Here are some of the many possible views of tr'':

$$\langle \langle a \rangle, \langle b, c \rangle, \langle d, e, f \rangle, \langle g \rangle \rangle$$

$$\langle \langle a \rangle, \langle b, c \rangle, \langle d, f, e \rangle, \langle g \rangle \rangle$$
$$\langle \langle a \rangle, \langle b, c \rangle, \langle f, e, d \rangle, \langle g \rangle \rangle$$
$$\langle \langle a \rangle, \langle b, c \rangle, \langle e, d, f \rangle, \langle g \rangle \rangle$$
$$\langle \langle a \rangle, \langle c, b \rangle, \langle d, e, f \rangle, \langle g \rangle \rangle$$
$$\langle \langle a \rangle, \langle c, b \rangle, \langle d, f \rangle, \langle \rangle \rangle$$

Furthermore, the views of tr'' could also be expressed in the strict interleaving form of tr'', thus avoiding the additional syntax required by a list of ROPEs:

$$\langle a <_c b =_c c <_c d =_c e =_c f <_c g \rangle$$

$$\langle a <_c b =_c c <_c d =_c f =_c e <_c g \rangle$$
$$\langle a <_c b =_c c <_c f =_c e =_c d <_c g \rangle$$
$$\langle a <_c b =_c c <_c e =_c d =_c f <_c g \rangle$$
$$\langle a <_c c =_c b <_c d =_c e =_c f <_c g \rangle$$
$$\langle a <_c c =_c b <_c d =_c f \rangle$$

There is at least one disadvantage to this new form of views, namely, the ability to reason about imperfect observation is limited in the case of empty ROPEs (compare the corresponding last views, where event *g* is missing).

Before we leave this new approach to recording traces, some overall characterization is desirable. There is more than one way to proceed from here. Briefly, we describe two.

For the first characterization, let the Olympian CSP observer record the trace of a computation, introducing no changes to existing CSP models. Thus, for some computing process *P*, let

$$tr \in traces(P)$$

where tr's partially ordered elements are as before. Since tr is partially ordered, chronologically, we can immediately represent the trace of events, tr', delimited by $\leq_c$ relations rather than as a comma-separated list:

$$tr' = \langle a \leq_c b \leq_c c \leq_c d \leq_c e \leq_c f \leq_c g \rangle$$

Next, we could consider a focus oracle, $focus()$, whose domain is the set of partially ordered CSP traces, and whose range is the set of totally ordered (strictly interleaved) traces,

as before. The focus oracle magically reveals the state of each $\leq_c$ relation that corresponds to what the CSP observer witnessed when recording tr. Thus,

$$focus(tr') = \langle a <_c b =_c c <_c d =_c e =_c f <_c g \rangle$$

If the thought of utilizing the quantum physics abstraction of superposition seems troublesome, we offer one additional interpretation, from Computer Science: There is a sense in which the tr' trace is a lazy data structure, whose $\leq$'s are not yet elaborated. Under this premise, the focus oracle merely elaborates those parts of the trace not yet elaborated. The oracle is still magic, and correctly focuses on the total ordering known to the CSP observer.

For the second characterization, let the *environment* of *P* be a distinguished observer who records traces directly as a total order with "$<$" and "$=$" to delimit recorded events, rather than the comma-separated trace of the CSP observer. That is, the environment is implicitly able to observe each quantum "$\leq$" and record the observed operator.

Conveniently, both approaches characterized are equally backward-compatible with CSP simply by replacing "$<_c$"s and "$=_c$"s with commas in the totally ordered trace. Thus any trace analyzed using view-centric reasoning is also analyzable with the other existing CSP models.

4 Motivating Parallel Events

Given the unqualified success of CSP's approach to representing concurrency via sequentially interleaved traces, it is reasonable to ask, why introduce parallel events at all? One reason is that it is not always possible to determine from an interleaved trace, or even the set of all such possible traces, of a computation, whether two or more events occurred simultaneously (that is, simultaneously from the CSP observer's frame of reference). The best way to illustrate this point is with a degenerate example. Consider the interleaved trace:

$$\langle a,\ a,\ a,\ a,\ a \rangle$$

This is a degenerate example because the set of all possible traces of this computation would be a singleton containing just the above trace. Now suppose what we wished to reason about is the degree of parallelism that occurred during this computation. Using our new approach to focusing on this trace, we quickly realize many actual possibilities exist. For example:

$$\langle a <_c a <_c a <_c a <_c a \rangle$$

$$\langle a =_c a <_c a <_c a <_c a \rangle$$

$$\langle a =_c a =_c a <_c a <_c a \rangle$$

$$\langle a <_c a =_c a =_c a <_c a \rangle$$

$$\langle a =_c a <_c a =_c a <_c a \rangle$$

Clearly, these are not all the possibilities. Yet not one of these possibilities could be identified – with certainty – from the given comma-delimited, interleaved trace as the actual corresponding computation recorded by the CSP observer.

Perhaps a more practical example deserves the attention of $\mathcal{VCR}$'s parallel event semantics. I/O-PAR (and I/O-SEQ) are design patterns described by Welch, Martin and others in [11, 12, 13, 14]. The reason these design patterns are appealing is because *arbitrary topology* networks of I/O-PAR processes are guaranteed to be deadlock/livelock free, and thus they are desirable components for building systems (or parts of systems).

Informally, a process *P* is considered I/O-PAR if it operates deterministically and cyclically, such that, once per cycle, it synchronizes in parallel on all the events in its alphabet. For example, processes *P* and *Q*, given by the following CSP equations, are I/O-PAR:

$$P = (a \rightarrow SKIP \ ||| \ b \rightarrow SKIP);\ P$$

$$Q = (b \rightarrow SKIP \ ||| \ c \rightarrow SKIP);\ Q$$

Using the focused trace notation presented in this paper, the traces of *P* and *Q* are, respectively, all prefixes of tr_P and tr_Q:

$$tr_P = \langle a =_c b <_c a =_c b <_c a =_c b <_c \ \ldots \rangle$$

$$tr_Q = \langle b =_c c <_c b =_c c <_c b =_c c <_c \ \ldots \rangle$$

Notice how elegantly these focused traces capture the essence of the behavior of processes *P* and *Q*. If one were to attempt to represent the behavior of *P* and *Q* using traditional comma-separated traces, the effort would be more tedious and cumbersome.

5 Conclusions and Future Work

This paper presented recent insights into the nature of CSP's traces that emerged while studying Hoare and He's Unified Theories of Programming. The goal of expressing $\mathcal{VCR}$ in the alphabetized relational calculus and linking a $\mathcal{VCR}$ theory of programming to existing CSP models in the unifying theories remains. However, we are much closer to our goal now that we've identified ways to map from classic CSP traces to the parallel traces of $\mathcal{VCR}$, via the $focus()$ oracle. Furthermore, we can map back to CSP traces from $\mathcal{VCR}$ traces and views. Finally, we are encouraged by the relationship and resemblance of CSP's traces to those of $\mathcal{VCR}$, and remain optimistic that $\mathcal{VCR}$ will soon be linked within the unifying theories.

While the basis for linking $\mathcal{VCR}$ to CSP is established, work remains to define $\mathcal{VCR}$ as a theory of programming given in the alphabetized relational calculus. Along this path, a new goal has emerged, to extend parallel events from traces alone to the specification of processes that engage directly in parallel events. For example, in Section 4, how would one express process *P* in terms of parallel events? Furthermore, how would one represent the process that results from pipelining *P* into *Q*, and so on? In general, the semantic description of I/O-PAR should benefit from such a desired algebra, one that is capable of manipulating parallel events. Such an abstraction could ultimately help enhance our insights into the design and verification of concurrent systems.

Acknowledgments

Professor Charles E. Hughes and Dr. Rebecca J. Parsons were original collaborators during the development of the $\mathcal{VCR}$ model as a parameterized operational semantics. Professor Jim Woodcock provided valuable feedback for an earlier CPA conference paper [4], as well as the current roadmap toward linking $\mathcal{VCR}$ within the Unifying Theories of Programming. In particular, Professor Peter Welch suggested the I/O-PAR design pattern to motivate and illustrate the value of parallel event semantics, which helped to focus this paper's presentation and point the way toward future work. Finally, the author wishes to thank the anonymous referees, whose remarks and insights helped improve the *focus* and content of this paper.

References

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

[2] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall Europe, 1998.

[3] Marc L. Smith. *View-centric Reasoning about Parallel and Distributed Computation*. PhD thesis, University of Central Florida, Orlando, Florida 32816-2362, December 2000.

[4] Marc L. Smith, Rebecca J. Parsons, and Charles E. Hughes. View-centric reasoning for linda and tuple space computation. In J. S. Pascoe, P. H. Welch, R. J. Loader, and V. S. Sunderam, editors, *Communicating Process Architectures 2002*, volume 60 of *Concurrent Systems Engineering Series*, pages 223–254, Amsterdam, 2002. IOS Press.

[5] C.A.R. Hoare and Jifeng He. *Unifying Theories of Programming*. Prentice Hall Series in Computer Science. Prentice Hall Europe, 1998.

[6] Marc L. Smith, Charles E. Hughes, and Kyle W. Burke. The denotational semantics of view-centric reasoning. In J.F. Broenink and G.H. Hilderink, editors, *Communicating Process Architectures 2003*, volume 61 of *Concurrent Systems Engineering Series*, pages 91–96, Amsterdam, 2003. IOS Press.

[7] Adrian E. Lawrence. Hcsp: Imperative state and true concurrency. In J. S. Pascoe, P. H. Welch, R. J. Loader, and V. S. Sunderam, editors, *Communicating Process Architectures – 2002*, Concurrent Systems Engineering, pages 39–55, Amsterdam, 2002. IOS Press.

[8] Marc L. Smith, Rebecca J. Parsons, and Charles E. Hughes. View-centric reasoning for linda and tuple space computation. *IEE Proceedings–Software*, 150(2):71–84, apr 2003.

[9] Gul A. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. The MIT Press Series in Artificial Intelligence. The MIT Press, Cambridge, Massachusetts, 1986.

[10] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1), January 1985.

[11] P.H. Welch. Emulating Digital Logic using Transputer Networks (Very High Parallelism = Simplicity = Performance). *International Journal of Parallel Computing*, 9, January 1989. North-Holland.

[12] P.H. Welch, G.R.R. Justo, and C.J. Willcock. Higher-Level Paradigms for Deadlock-Free High-Performance Systems. In R. Grebe, J. Hektor, S.C. Hilton, M.R. Jane, and P.H. Welch, editors, *Transputer Applications and Systems '93, Proceedings of the 1993 World Transputer Congress*, volume 2, pages 981–1004, Aachen, Germany, September 1993. IOS Press, Netherlands. ISBN 90-5199-140-1.

[13] J.M.R. Martin, I. East, and S. Jassim. Design Rules for Deadlock Freedom. *Transputer Communications*, 3(2):121–133, September 1994. John Wiley and Sons. 1070-454X.

[14] J.M.R. Martin and P.H. Welch. A Design Strategy for Deadlock-Free Concurrent Systems. *Transputer Communications*, 3(4):215–232, October 1996. John Wiley and Sons. 1070-454X.