WEGNER83
P. Wegner and S.A. Smolka, "Processes, Tasks, and Monitors: A Comparative Study of Concurrent Programming Primitives", IEEE Transactions on Software Engineering, July 1983, Volume SE-9, Number 4, pp. 446-462.
Communicating Sequential Processes (CSP) is a programming notation based on synchronous message passing and selective communications.
Ada is a language for real-time programming and process-control systems.
Ada includes facilities for concurrent programming. Monitors were conceived as modules for handling mutually exclusive access to shared resources.
In this paper, at first "lower-level" communication, synchronization and non determinism in CSP and Ada are compared. Then, "higher-level" module interface properties of Ada tasks and monitors are examined.
The unit of concurrency is called a process in CSP and a task in Ada.
Execution causes all processes in a concurrent execution command in CSP and all tasks in an Ada procedure to be initiated concurrently and terminated before the next command or before the end statement of a procedure can be executed. Ada permits dynamic task initiation, that is the number of tasks and their time of creation cannot be predicted: there can be array and lists of tasks.
Both CSP processes and Ada tasks use synchronization primitives based on message passing. In CSP communication is accomplished by input and output commands.
It is essentially a "rendezvous" type synchronization. Both processes performing input and output must specify origin and destination of the message. Task entries in Ada have similar functions as the input/output directives of CSP. Accept statements and accept body in the task body specify the communication to be performed when synchronization has occurred. The caller must wait until execution of the called accept statement has been completed. The accept statement in they called task does not know the calling task. This is an asymmetry that follows closely subprogram calling and that does not exist in CSP. Finally, in Ada communication can be performed through shared (between tasks) non local variables and it is necessary to make sure that no simultaneous modifications of shared resources occur.
Non deterministic control features can be employed both in sequential programming when the order in which subcomputations are performed is not important, and in concurrent programming, when the order in which concurrent processes get ready is not known. Guarded commands are adopted in CSP. A guard is said to be true if a Boolean condition and an input command, which may be contained in a guard, are respectively true and
alternative and repetitive commands. Alternative commands non deterministically select one eligible action corresponding to a true guard for execution. Repetitive commands behave like alternative commands at each iteration. Repetitive commands iterate until none of the guards are true. Ada introduces the select statement, together with guarded accept statements. First the guards are evaluated, then the accept statements which are ready to accept are selected. If there is more than one ready, one is chosen out of them arbitrarily.
The CSP mechanism for non determinism is orthogonal to its mechanism for concurrency: whatever follows guards can be either sequential or concurrent. One exception to the orthogonality is that output guards are not permitted. As a result, a rendezvous can occur between a process blocked on output and a process that may be blocked on several non deterministic input alternatives. In Ada there can be non determinism at the point of call, thus giving more power to the select statement mechanism c
commands. On the other hand, the select syntax is complex and its semantics is

even more complex.
In the second part of this paper a comparative analysis of monitors, tasks and packages is performed. A monitor consists of a set of declarations of local data, a set of procedures which may be called by users and monitor initializing statements. At most one procedure of the monitor may be executing at any given time.
There is a basic difference in the control mechanism for scheduling monitor calls and task entries. On completion of a monitor procedure execution, the monitor returns to an initial state, ready for the next procedure call. In contrast, completion of an accept statement c of the accept body until task termination or another accept statement is encountered.
Furthermore, monitors have the ability to suspend and subsequently reactivate monitor procedures, whereas that cannot be easily simulated in Ada. Tasks have no internal mechanism for initiating another activity when a primary activity has been interrupted.
In a way a monitor may be viewed as a concurrent Ada package. And in fact the author of this article shows how using packages in Ada a monitor-like behaviour can be implemented for a job scheduler problem. It is suggested that Ada packages may be more appropriate as a user interface for concurrent computation.
This paper does not try to be an exhaustive study on all concurrent aspects of CSP,Ada and monitors, but attempts to address major shortcomings and relevant features of each notation compared to the other. It is not an article that contains original ideas or proposes new solutions, but it summarizes the state-of-the-art in concurrent programming and outlines future directions in the development of concurrent programming.