Coordination Principles

8/12/07

These principles concern how autonomous entities work together toward a common result.

A coordination system is a set of agents interacting within a finite or infinite game toward a common objective.

A. Agents can be humans or computational processes.

B. Interaction means that agent behaviors are influenced by information agents exchange.

C. A protocol is an algorithmic pattern of information exchange among a set of agents.

D. Coordinated interaction means that individual interactions are aligned toward the system's objective.

E. Coordination implies feedback in the interactions so that agents can tell that they are moving toward the system objective. Feedback can be direct (such as one agent acknowledging a request from another) or indirect (such as a merchant adjusting inventory depending on what customers buy).

F. A game is a framework specifying objectives, players, resources, rules, and strategies.

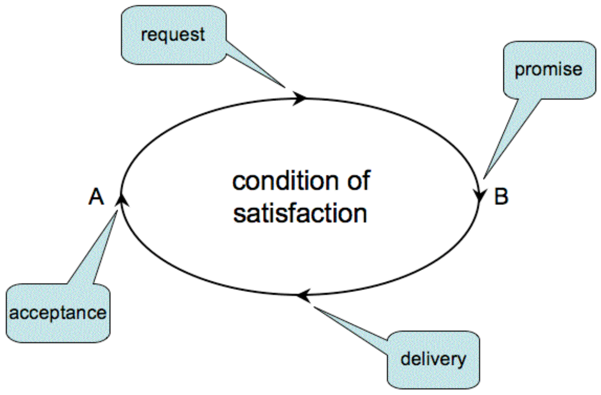
G. A finite game is one that aims to terminate with someone declared a winner.

H. An infinite game is one that aims to continue the play indefinitely.

© Copyright 2007 by Peter J. Denning, with Craig Martell

Action loop is the foundational element of all coordination protocols.

A. Most individual human interactions are modeled by an action loop in which a performer delivers a condition satisfying a customer. The loop has a requester (A) and performer (B) and four time segments culminating in request, promise, delivery, and acceptance. Before the loop starts, the condition is not true; when it completes, the condition is true.



A	B	
human	human	CSCW
human	computer	HCI
computer	computer	CC

B. Each of the four segments can be linked to further action loops that respond to requests from components of the segment. The resulting network of action loops is called an action process or workflow.

Coordination tasks can be delegated to computational processes.

A. Humans delegate tasks to agents by designing computational processes to perform the tasks.

B. There are three categories of coordination systems according to the amount of task delegation:

   i. Human-human with computer assistance: all interaction is between humans, but computational processes track their joint progress states to assist them to complete tasks. (Known as Computer Supported Cooperative Work, CSCW.)

   ii. Human-computer: the performer role is delegated to a computational system. Humans interact with the system through an interaction language and interaction interface. (Known as Human Computer Interaction, HCI.)

   iii. Computer-computer: all requester and performer roles are delegated to computational processes. All interactions are carried out automatically between machines. (Known as Concurrency Control, CC)

The protocols of coordination systems manage dependencies of flow, sharing, and fit among activities.

A. Coordination is required if two (or more) activities depend on each other in some way. The nature of the dependencies follows from the game. The three basic types of dependency are flow, sharing, and fit.

B. A flow dependency exists when one activity produces a resource that is required by another activity. Message sending and flowcharting are examples.

C. A sharing dependency exists when two (or more) activities require the same resource. Accessing a database or sharing a CPU are examples.

D. A fit dependency exists when two (or more) activities contribute to the formation or state of a resource. Building a database is an example.

E. A coordination process is a mechanism implementing these dependencies. Many alternatives are available for each dependency.

It is impossible to select one of several simultaneous or equally attractive alternatives within a preset deadline (Choice Uncertainty Principle)

A. Coordination protocols are constantly faced with the need to choose one of several concurrent alternatives, and defer the others for later action.

B. When the choice is made by a physical system (such as a hardware flipflop, a brain, or even a social system) the concurrent arrival of signals denoting alternatives can send the system into a metastable state, from which the exit time is not bounded. If other entities can read that state while it is metastable, they do not see a definite value and can malfunction.

C. With request-acknowledge signaling, no entity in a metastable state can respond to a request. Except in real-time systems, which have hard deadlines, request-acknowledge signaling is a good safeguard against the delay caused by a metastable state.

D. Within a logical domain, such as an algorithm or Boolean expression of a logic circuit, all value-changes appear instantaneous: in the next state of the computation a logical formula can select one of two arriving signals. However, the assumption that signals change instantaneously does not hold for the physical system implementing the software; a software malfunction can occur if the physical system enters a metastable state (because then the variables may not have definite logical values when tested).

All coordination systems depend on solutions to the concurrency control problems of arbitration, synchronization, serialization, determinacy, and deadlock.

A. Concurrency means that tasks can be executed in parallel.

B. A concurrent system is a set of tasks, some of which are ordered and the rest concurrent (unordered). Ordered tasks can never execute at the same time; concurrent tasks can.

C. Arbitration arises when a task is required to select only one of two (or more) potentially simultaneous signals, deferring action on the unselected ones without losing them. According to the Choice Uncertainty Principle, a complete solution to the arbitration problem possible only if the tasks involved can wait an arbitrary length of time for the selection to be made. If there is a deadline, there is a probability of arbitration failure (2 or more signaling tasks, or none, may be selected, or unselected signals lost). The solutions to all the following problems depend on a satisfactory solution to arbitration in the underlying system.

D. Synchronization is a requirement that a task cannot proceed past a point until another task signals that it has passed a corresponding checkpoint. A semaphore with wait and signal operations is a model for synchronization (but not the only model).

E. Serialization is the requirement that concurrent tasks are executed in some order, but not at exactly the same time. It guarantees that all the steps of one task are done before the other task is started; there is no possibility of interleaving the steps of one task with the other. Mutual exclusion locks are a common implementation.

F. A race condition means that the outcome of concurrent tasks that read and write shared data depends on their relative timing. Some race conditions are essential in the requirements of a computation, such as a database system that assigns airplane seats. Other race conditions are accidental side effects of concurrent tasks trying to read shared data that are being updated by other tasks. Accidental races are quite difficult to locate through testing because they do not occur every time and they do not affect the share variable in a repeatable way. Accidental races can be eliminated by mutually excluding concurrent tasks that write shared data.

G. Determinacy is a requirement that a concurrent system always produces the same output for a given input, no matter what the timing of its component tasks. Determinacy is guaranteed if every pair of concurrent (unordered) tasks does not write into data read or written by the other (no race conditions). Determinacy is a stronger requirement than serialization because systems of mutually excluded concurrent tasks can be nondeterminate.

H. Deadlock occurs in a concurrent system when every member of a set of concurrent tasks has stopped to wait for a signal from another member of the set (circular wait). Waits can occur because of incompatible synchronization requirements or because some tasks request resources that are temporarily held by others. Deadlock detection is generally an intractable problem. Deadlocks can be avoided if the system imposes an order on resources and semaphores and allows tasks only to issue new requests farther up the order than anything it holds. Deadlocks can also be avoided if all tasks obtain all resources they need before commencing execution.

I. Livelock occurs in a concurrent system if a set of tasks enters a cycle of deferring to other tasks in order to avoid deadlocks. Livelocks can be minimized by requiring tasks to wait for a random amount of time, increasing with each retry.