Process Modelling with Cooperative Agents

George T. Heineman*
Columbia University, New York NY 10027, USA

Abstract. Concurrency Control is the ability to allow concurrent access of multiple independent agents while still maintaining the overall consistency of the database. We discuss the notion of Cooperation Control, which gives a DBMS, the ability to allow cooperation of multiple cooperating agents, without corrupting the consistency of the database. Specifically, there is the need for allowing cooperating agents to cooperate while preventing independent agents from interfering with each other. In this paper, we use the MARVEL system to construct and investigate cooperative scenarios.

1 Introduction

Concurrency Control in database management systems allows multiple independent agents to concurrently access the database while maintaining its consistency. Cooperation Control extends this concept by considering situations with cooperating agents. To realize cooperation we need to have semantic information about how the agents will act. Our research on Process Centered Environments (PCEs) has shown that these systems have a rich body of semantic information available. In such environments, a process is formally specified in a Process Modeling Language (PML). As part of this specification, the cooperation between agents needs to be defined.

There are several reasons why multiple agents might need to cooperate:

1. Uniqueness of agents – There might be certain tasks which can only be carried out by a particular agent; consider a task which can only be performed by a database administrator.

2. Encapsulation of tasks – The process might be designed such that there are clusters of tasks which are separated from other tasks. This hierarchical organization of tasks becomes necessary as the size and number of tasks grows.

3. Group tasks – There are tasks which need multiple agents to work in concert with each other; consider a conference phone call between three parties.

The MARVEL project is an example of a PCE applied to software development. In this PCE, the process of software development is formally encoded in terms of rules, and the concurrency control of the database is tailored to provide specific

* Heineman is supported in part by IBM Canada, Ltd.
behavior. In this paper we explore how to use MARVEL to produce a cooperative environment. We start with a simple example of cooperating agents in a “Blocks World” environment, and then apply our results to a fragment of the ISPW-7 [4] sample problem. We conclude with a discussion of the limitations and benefits of this approach.

2 Example problem

Consider the “Blocks World” example, as shown in Fig. 1. Blocks can either sit on the table or on top of another block (the table is large enough to accommodate all blocks). A block $X$ is *clear* if no block is sitting on top of $X$. Only clear blocks may be moved and a block cannot have two blocks sitting directly on it. To move $A$ on top of $E$, for example, $C$ must first be moved to the table; then both $A$ and $E$ are clear and the move can take place.

The PROLOG program in Fig. 3 is a goal-directed process which solves the problem of putting block $X$ on top of $Y$ by first making sure that both $X$ and $Y$ are clear, thus allowing the move to take place. Note that the *Table* may not be moved but blocks may be moved onto it. This particular process achieves the put_on($X$, $Y$) goal by first achieving two sub-goals clear_space($X$) and clear_space($Y$). Figure 2 shows the solution for the request put_on(d, a). Note how put_on and clear_space are recursively defined to invoke each other.

We now introduce multiple agents to this example problem. Assume, in the blocks world, that there are two agents, *Placer* and *Clearer*. These agents cooperate in the following way:

1. When Placer moves block $X$ to sit on object $Y$, Clearer is invoked to clear both $X$ and $Y$. Note that $Y$ may be a block or the *Table*.
2. When Clearer clears block $X$, Placer is invoked to move block $Y$, sitting on $X$, onto the *Table*.