Towards a Theoretical Foundation for Cooperating Knowledge Based Systems

S M Deen and C A Johnson

Computer Science Department, University of Keele, Staffs, ST5 5BG, England

Abstract. A theoretical grounding is provided for a cooperating knowledge based systems architecture, which is based upon cooperation blocks and cooperation block hierarchies. The architecture describes the requirements for task decomposition, negotiation, cooperation and coordination, fault tolerance and recoverability. A description of behaviour is given in terms of state transition diagrams.

1 Introduction

In recent years agent technology has become an important area of research and development in distributed applications. Cooperating Knowledge Based Systems (CKBS) is a subarea in which each autonomous knowledge based system, known as an agent, cooperates (or even competes) to solve tasks in a distributed environment. As such, a CKBS is close to a multi-agent system (MAS) of distributed AI (DAI) [5], except that whereas MAS/DAI draws its concepts from AI, CKBS uses a more open approach with ideas drawn from the experience and requirements of real world distributed applications, where effectiveness, performance, reliability and usability are of paramount importance.

In [1,2] we described an architectural framework for CKBS applications based upon the concept of a cooperation block. This framework, although developed independently, is very close to the IMS/HMS reference model [4,6] for holonic manufacturing systems, provided that an atomic holon is interpreted as a CKBS agent, and a cooperation domain as a CKBS cooperation block. Both of these frameworks are somewhat informal, and hence the purpose of the current paper is to provide them with a precise and rigorous grounding, thus yielding a firm theoretical foundation for the implementation and study of CKBS and HMS systems.

We first give an overview of our architecture. In order to process a request for task execution, a coordinator agent forms a cooperation block, this being a temporary and dynamic alliance between agents allied to the processing of the given task request. Within the block, negotiation firstly involves the decomposition of the task into (possibly interdependent) subtasks and the allocation of these subtasks to agents (cohorts) with the suitable skill to carry out (or coordinate) subtask execution. Following this allocation, a further negotiation phase attempts to generate an execution plan which determines how the cohorts will coordinate their individual executions in order to ensure that the relevant interdependencies are satisfied.
Any cohort $A$ may in turn form a lower-level cooperation block with other agents for the execution of a subtask. In this case agent $A$ acts as a cohort in the upper-level block and as the coordinator of the lower-level block. Cooperation blocks thus form hierarchies which parallel the hierarchy of task decompositions.

The operation of cooperation blocks and their hierarchies should be tolerant to failure, with failures being handled locally as far as is possible, and with recovery mechanisms that ensure that such failures do not leave the system in an unacceptable state. In this respect, the behaviour of a cooperation block hierarchy is similar to that of a holon [4,6].

2 Agents

As in [1], we assume that an agent is a large grain entity having a compulsory software component and an optional hardware component. We assume that agents are cooperative, in the sense that they participate in joint task processing, but that they are autonomous, in the sense that agents decide independently which joint tasks to participate in, and negotiate their own involvement in such task processing.

Agents possess skills, these skills being represented by their membership in skill classes. For each task class there are two skill classes: one for processing, and one for coordinating tasks in the task class. Each skill class may have many agents (called twins) offering that same skill. In particular, if one twin breaks down, then another twin can replace it. Twins in a given class may be supervised by a class agent called the (skill-class's) minder [3].

There is a very wide literature surrounding agents, a reasonable starting point being [5].

3 Tasks

3.1 Task schemata and requests

We assume that a finite number of task classes are known to the system, and that the general description of a task class is given by a task schema describing the task procedure and the task parameters.

The task procedure itself does not play a major role in what follows. We assume that agents with the appropriate skill know how to process (or coordinate) the task procedure, and moreover are able to report whether the procedure was carried out successfully.

A (task) parameter is any value associated with the execution of the task procedure which is externally visible (i.e. public). Any parameter value can in theory be output from one task, and input into another. Two obvious parameters are start-time and end-time. A pre/post condition is a Boolean condition over the task parameters.

A task request (for a particular schema) consists of the task schema, values for some, all or none of the task parameters, and a set of pre/post conditions. Notice