MASCOT: An Agent-Based Architecture for Coordinated Mixed-Initiative Supply Chain Planning and Scheduling

Norman M. Sadeh, David W. Hildum, Dag Kjenstad,† and Allen Tseng

Intelligent Coordination and Logistics Laboratory
The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3890
412.268.{8827,2357,6676}
fax: 412.268.5569
{SADEH,HILDUM,YHTSENG}@CS.CMU.EDU

† SINTEF Applied Mathematics
Department of Optimization
PO Box 124 Blindern
N-0314 Oslo
Norway
(47) 22 06 75 33
fax: (47) 22 06 73 50
DAG.KJENSTAD@MATH.SINTEF.NO

Abstract

A key to agility in today's dynamic high-mix production environments is the ability to (1) effectively coordinate production across multiple facilities, whether internal or external to the company, and (2) quickly and accurately evaluate new product/subcomponent designs and strategic business decisions (e.g., make-or-buy or supplier selection decisions) with regard to capacity and material requirements across the supply chain. This paper provides an overview of MASCOT ("Multi-Agent Supply Chain cOordination Tool"), a reconfigurable, multilevel, agent-based architecture for coordinated supply chain planning and scheduling aimed at supporting these functionalities. It reviews key innovative elements of the MASCOT architecture with a special emphasis on its support of real-time mixed-initiative "what-if" functionalities, enabling end-users at different levels within the architecture to rapidly evaluate alternative tradeoffs and their respective impact across the supply chain. The paper also discusses new coordination protocols aimed at better exploiting the power of finite capacity scheduling functionalities across the supply chain. Empirical results are presented quantifying the benefits afforded by these new protocols under different loads and supply chain configurations.

1 Introduction

A key to agility in today's dynamic high-mix environments is the ability to (1) effectively coordinate production across multiple facilities, whether internal or external to the company, and (2) quickly and accurately evaluate new product/subcomponent designs and strategic business decisions (e.g., make-or-buy and supplier-selection decisions) with regard to capacity and material requirements across the supply chain. This paper provides an overview of MASCOT ("Multi-Agent Supply Chain cOordination Tool") a reconfigurable, multilevel, agent-based architecture for coordinated mixed-initiative supply chain planning and scheduling aimed at supporting these key functionalities.

The MASCOT architecture is built around a customizable mixed-initiative agent wrapper. The agent wrapper serves two functions:

This is a revised version of a paper that appeared in the workshop notes of the *Third International Conference on Autonomous Agents (Agents '99) Workshop on Agent-Based Decision Support for Managing the Internet-Enabled Supply Chain*, Seattle WA, 1 May 1999.

1. It provides an open and uniform communication and coordination interface between planning and scheduling modules operating at different levels within the architecture (e.g., facility-level planning/scheduling modules, transportation planning/scheduling modules, and Master Production Scheduling (MPS) or Available-To-Promise (ATP)-level modules responsible for one or more facilities).

2. It supports customizable mixed-initiative functionalities that enable users at different levels within the architecture to interactively manipulate and evaluate alternative planning and scheduling solutions while selectively coordinating the requirements of these solutions with those generated by other agents within the architecture (e.g., agents responsible for other facilities or agents sitting at different levels within the architecture). The architecture supports easy reconfiguration to accommodate the introduction of new products, new process flows, new suppliers and/or customers, new facilities and new transportation arrangements.

The first part of this paper provides an overview of the MASCOT architecture with a particular emphasis on its support for mixed-initiative functionalities. The second part discusses different coordination policies studied within the context of MASCOT and reports on an experimental study conducted to evaluate the benefits of these policies under different operational assumptions, different loads, and different supply chain configurations. The study suggests that new finite capacity coordination protocols can significantly improve the accuracy of ATP/Capacity-To-Promise (CTP) decisions, which in turn can translate into significant increases in profits and performance.

2 The Overall MASCOT Architecture

The MASCOT architecture aims at providing a framework for coordinated development and manipulation of planning and scheduling solutions at multiple levels of abstraction across the supply chain. Within this architecture, MASCOT agents serve as wrappers for planning and scheduling modules, each responsible for supporting the development and revision of planning/scheduling solutions for a particular facility or group of facilities at a particular level of abstraction. Lower-level agents are typically wrappers for planning and scheduling modules that support single facilities over short to medium-term horizons. Higher-level agents are coordination wrappers for tactical or strategic planning and scheduling modules that generally require looking over longer horizons and across multiple facilities (whether within a single company or across multiple organizations). An overview of the architecture illustrating the interaction between various types of multilevel MASCOT agents is presented in Figure 1.

MASCOT agents are shells that support the following key functionalities:

1. *Coordination:* MASCOT agents act as coordination wrappers for planning and scheduling modules distributed across the supply chain, supporting both lateral and vertical coordination protocols. Lateral coordination protocols support interactions between peer-level agents, while vertical coordination protocols support interactions between high-level agents and those lower-level agents directly responsible for internal facilities. We further distinguish between (1) high-level lateral coordination protocols that involve, for example, requests for bids on subcomponents and exchange of ATP information, and (2) low-level lateral coordination protocols to dynamically establish and revise delivery dates across feeder and consumer facilities for individual batches.

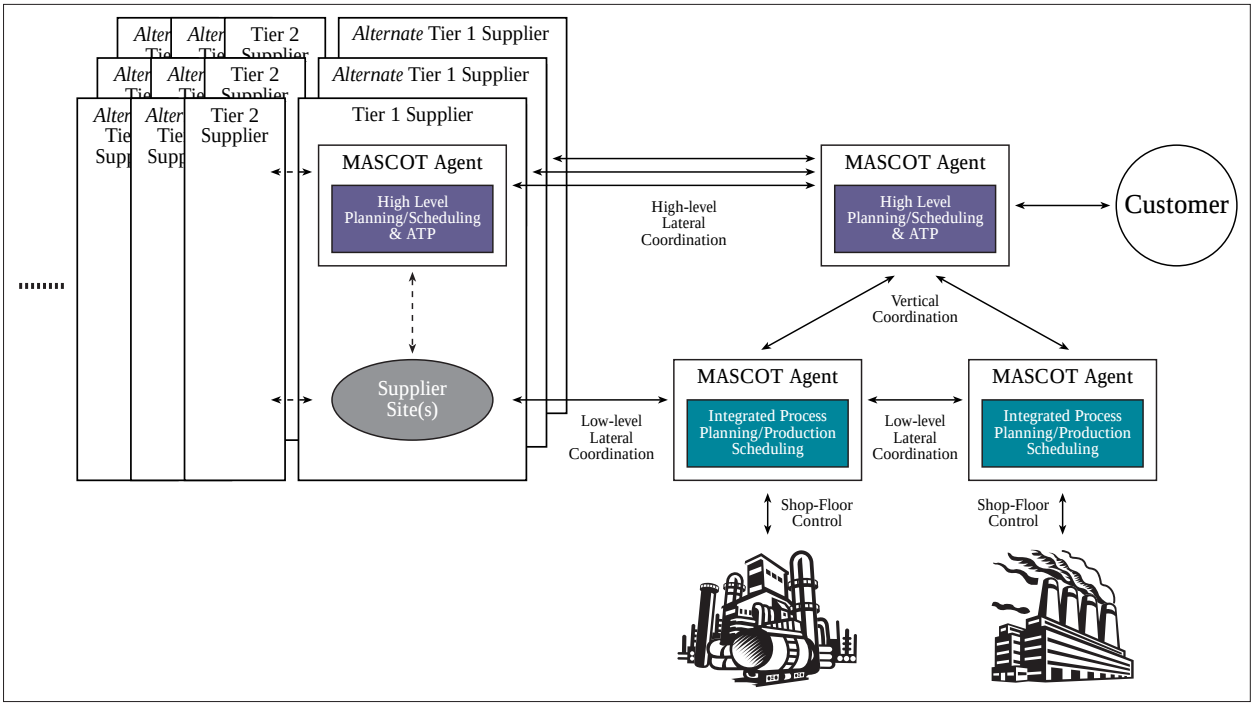

Figure 1: Overall MASCOT Architecture

Vertical coordination protocols, on the other hand, help maintain consistency across multilevel capacity models and help *selectively* validate high-level planning decisions through more detailed evaluation at the level of one or more bottleneck facilities. For example, an order-promising decision for a product that needs to flow across several internal facilities can be validated by carrying out a detailed analysis of the impact of the order on the load of a particular bottleneck facility through interaction with the agent responsible for that facility.

2. *Integration with Heterogeneous Planning and Scheduling Modules:* Each MASCOT agent is configured around a blackboard architecture [Erman *et al.*, 1980, Lesser and Corkill, 1983, Sadeh *et al.*, 1998] that allows for the easy integration of multiple planning and scheduling modules (including possible legacy systems) along with analysis and coordination modules. Within a given agent, these modules can be activated to develop and revise integrated planning and scheduling solutions stored in a data structure accessible to all the agent's modules and referred to as the agent's *blackboard*. Specific modules differ from one agent to another depending on the facility(ies) for which the agent is responsible and the level at which it operates within the architecture. Typically, low-level facility-based agents might include a process-planning/process-selection module and a production scheduling module, as in the IP3S system [Sadeh *et al.*, 1998]. Higher level agents (e.g., MPS level or equivalent) will typically combine high-level scheduling and capacity-allocation modules with high-level process-flow/bill-of-material development modules to support alternative make-or-buy/outsourcing decisions.

3. *Mixed-Initiative Decision-Support Functionalities:* A MASCOT agent's blackboard is partitioned into *contexts*, each corresponding to a possibly different set of assumptions (e.g., demand as-

sumptions, capacity assumptions, internal process-flow/bill-of-material assumptions, supplier delivery date assumptions, make-or-buy assumptions,) that enable the end-user to compare different tradeoffs both locally and through coordination with other agents in the architecture. Contexts can be created either (1) manually by the (agent's) end-user (e.g., to evaluate a particular assumption), or (2) automatically by the agent (e.g., in response to a query from another agent). Within each context, solutions are developed by the end-user or the agent (or a combination thereof) through the activation of services supported by the agent's various planning, scheduling, analysis, and coordination modules.

4. *Reconfigurability:* MASCOT agents can be rapidly reconfigured to accommodate the introduction of new products, new product flows, new facilities, new suppliers, new make-or-buy arrangements, and new transportation arrangements, etc. Lateral and vertical coordination protocols rely on explicit (declarative) product, process, and facility models to determine the agents with which to coordinate. Changes in these models, such as the identification of a new supplier for a particular component or a new product flow, trigger the automatic recalculation of the relationships between agents in the architecture.

3 The MASCOT Agent Architecture

The use of blackboard architectures [Erman *et al.*, 1980, Corkill, 1991, Carver and Lesser, 1992] as an effective vehicle for integrating multiple sources of knowledge to solve complex problems has been demonstrated in a wide range of application domains. Blackboard architectures emphasize the modular encapsulation of problem-solving knowledge within independent knowledge sources (KSs) that work collectively to develop solutions to problems by communicating through a shared data structure, namely, the blackboard. By explicitly separating domain knowledge (e.g., in the case of MASCOT, planning, scheduling, analysis, coordination, and communication knowledge) and control knowledge, blackboard architectures offer several key advantages:

- *Flexibility of the control mechanism*, making it possible for the user to select from among a dynamic set of control regimes (e.g., highly interactive control regimes where most decisions are made by the end-user versus more autonomous regimes where the end-user specifies high-level tasks or "goals" and lets the system decide how to accomplish them).
- *Extensibility of the architecture*, making it particularly easy to add and enhance knowledge sources.
- *Ease of integration with legacy systems* through the encapsulation of existing problem-solving systems as knowledge sources.
- *Reusability of knowledge sources* across multiple domains.

Figure 2 presents the MASCOT wrapper-agent architecture. Each MASCOT agent includes:

- A set of planning, scheduling, analysis, coordination, and communication modules/KSs.
- A blackboard, which serves as the repository of partial and complete integrated planning and scheduling solutions organized in different contexts.

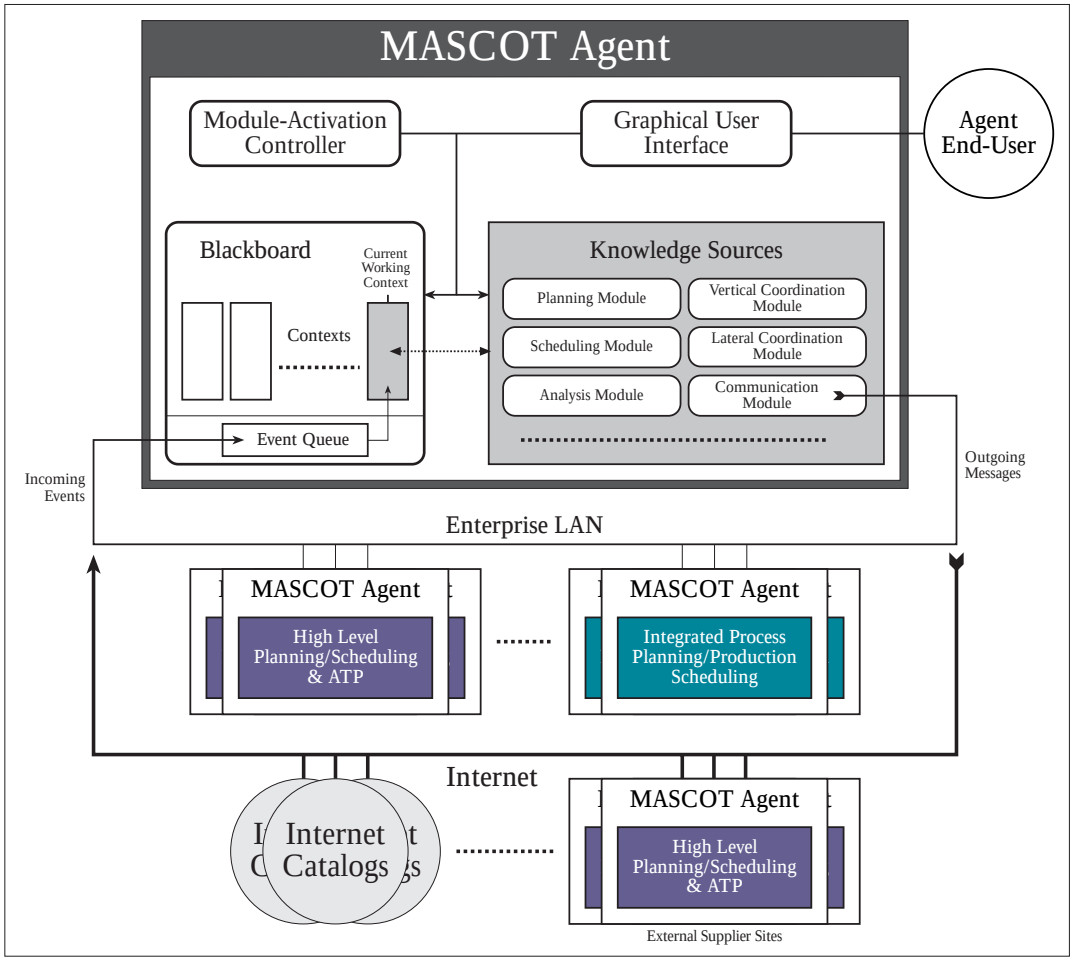

Figure 2: MASCOT Wrapper-Agent Architecture

- A module-activation controller that orchestrates the construction and revision of solutions through the activation of services supported by knowledge source modules, either based on direct input from the end-user or the agent's control heuristics (or a combination of the two).
- A graphical user interface.

3.1 Mixed-Initiative Asynchronous Problem Solving

The blackboard is a shared data structure on which KSs post solution components (e.g., new production schedules, new capacity allocations, new ATP results), analysis results (e.g., resource/capacity-utilization statistics), and coordination/communication-status information. It is partitioned into an arbitrary number of *contexts* that correspond to different sets of working assumptions (e.g., sets of orders that need to be planned/scheduled, available/allocated resource and facility capacities, supplier arrangements) and different solutions. Within each context, a summary of the current state of the solution is maintained in the form of a set of *unresolved issues*. An unresolved issue is an indication that a particular aspect of the current context solution is incomplete, inconsistent, or unsatisfactory (e.g.,

an order lacks a process plan, a resource breakdown conflicts with a reservation, a promised delivery date is violated).

Problem solving within a MASCOT agent, directed by either the agent's end-user or the agent itself (or a combination of the two), progresses in a mixed-initiative fashion through cycles during which:

1. One or more unresolved issue instances are selected to be resolved.
2. A particular method of resolution (e.g., a service supported by a KS module) is selected from among the set of methods applicable to the instance(s).
3. The selected method is executed by invoking the appropriate KS.

Instances of unresolved issues are created and deleted as a result of KS invocations, the incorporation of external events into a context, and the modification of assumptions within a context to perform "what-if" analysis.

In the remainder of this section we describe the major features of the MASCOT agent architecture, with an emphasis on the mixed-initiative problem-solving and integration capabilities it supports.

3.1.1 Alternate Problem Instances and Solutions

The mixed-initiative decision-support capabilities of MASCOT rely heavily on the use of blackboard contexts to support the representation of multiple problem instances. Each context contains all relevant problem data, including (for example), a collection of resources, tools, raw material supplies, finished-goods and work-in-process inventory quantities, and a collection of demands, orders, and bid/ATP requests and their corresponding process plans and production schedules. In addition, each context also maintains a set of unresolved issues to represent inconsistencies in any partial solution that must be removed to produce a complete and satisfactory solution. As assumptions are modified and a solution is developed within the context, the set of unresolved issues is updated to help both the agent and its end-user keep track of aspects of the current solution (within that context) that require further problem-solving attention.

Contexts can be created either by the end-user or automatically by the agent. It is through the creation of multiple contexts that "what-if" analysis is supported. By creating multiple copies of a context, changing various assumptions within the copies and producing solutions for each, alternate solution paths can be explored. Either the end-user or the agent can leave a particular context at any point in time and explore other potentially more promising alternatives in other contexts. Changes in the assumptions within one context remain local to that context and do not affect other contexts that may share the same entities. When a KS is invoked, its results are visible only within the context in which the end-user or the agent is currently working. This context is called the "current working context."

3.1.2 Selective Problem Definition

The mixed-initiative power of the context mechanism comes from the capability it provides the user to define a problem progressively and to modify the underlying assumptions about a problem. This can be done through either the modification of problem assumptions within a context (e.g., by changing various order and resource attributes such as due dates, work shifts, and supply-availability dates),

or the incorporation of events into a context (e.g., from other agents or information sources like an enterprise-level planning system, raw material suppliers, the shop floor).

Events received from other MASCOT agents and information sources (e.g., suppliers, manufacturing execution systems) are posted on the agent's *event queue* in preparation for being *incorporated* within one or more contexts by the end-user or the agent itself. These events include the notification of incoming orders, requests for bids, resource breakdowns and various shop floor updates. When an event is incorporated into a context, the blackboard translates the initial result (or implication) of the action described by the event into one or more appropriate unresolved issues. The objective for the end-user and/or the agent is to resolve each such issue through the activation and execution of one or more KSs until all events have been incorporated into a context and all unresolved issues have been resolved.

The MASCOT event-processing mechanism supports two important mixed-initiative capabilities:

1. It allows both the end-user and the agent to ignore events that are unlikely to affect the part of the solution upon which work is currently being done. For example, when revising a plan for a part that needs to be processed within the week, incoming-order events for new orders due three months downstream can often be ignored.

2. It allows both the end-user and the agent to process conditional events, such as requests for bids. For example, upon receipt of a request for bid on a possible order, a copy of the current context can be created, within which the order can be planned and scheduled. The resulting solution showing the impact of the possible order can then be evaluated to determine a realistic completion date and decide whether or not to submit a bid.

3.1.3 Unresolved Issues

As the assumptions within a particular context are modified or as new events are incorporated into a context, the set of unresolved issues within the context is updated automatically (according to declarative domain and control information). The set of unresolved issues within a context defines areas in the current partial solution where further problem-solving effort needs to be directed to produce a complete, consistent, and satisfactory solution. Unresolved issues provide a powerful *workflow management mechanism* that helps identify, plan, and control the work that remains to be done in any given context. This is similar to the *sources of uncertainty* (*SOUs*) of Carver and Lesser's RESUN control planner [Carver and Lesser, 1993].

The MASCOT architecture distinguishes between three types of unresolved issues, relating to:

- The *completeness* of the solution, such as an order lacking a process plan or an order that still needs to be incorporated in the production schedule.

- *Inconsistencies* within the solution, such as supplies being delivered past their required date.

- Potential areas for solution *improvement*, such as an order with an excessively late completion date or long leadtime.

3.2 Module-Activation Control

Each MASCOT agent is responsible for directing solution construction, revision, and analysis, either through close interaction with the end-user, or on its own with the help of a knowledge base of control

heuristics. The MASCOT agent architecture control mechanism supports two key mixed-initiative capabilities:

1. *Support for multiple control regimes*, ranging from a highly interactive mode where the user specifies each problem-solving action to an autonomous mode where the controller takes full responsibility for (1) the selection of which events to incorporate into the current context, (2) the determination of which unresolved issues to resolve, and (3) the selection of the specific methods for their resolution.

2. *Support for multilevel customizable problem-solving tasks* to provide a range of low to high-level modes of user interaction (e.g., the activation of a specific low-level KS service such as incorporating an order in an existing schedule, or the posting of high-level objectives (or "goals") such as trying to improve the completion date of an order, or the activation of a sequence of services and goals).

The MASCOT agent architecture allows an end-user to select from among different control regimes and different levels of interaction at any time. In addition, the set of high-level problem-solving tasks provided by the agent can be easily augmented to accommodate changing user-interaction patterns. Specifically, a hierarchy of high-level goals and scripts can be defined in terms of the basic set of services provided by the particular problem-solving systems encapsulated as KSs and incorporated within each MASCOT agent.

To support these mixed-initiative capabilities, the control mechanism in the MASCOT agent architecture follows an *execution profile* that records the assignment of various problem-solving tasks (e.g., the incorporation of events, the selection of unresolved issues to resolve and the methods for their resolution) to either the agent or its end-user. The assignment of tasks can be changed at any point by modifying the execution profile. To provide multiple levels of interaction with the system through the definition and activation of aggregate and goal-oriented problem-solving tasks, each MASCOT agent maintains its own declarative *control knowledge base* that links each unresolved issue to the set of appropriate problem-solving options for its resolution. The control knowledge base also contains the collection of generic and domain-specific control heuristics that are used by the agent to perform the tasks assigned to it, as recorded in the execution profile.

An *agenda* mechanism is employed by each MASCOT agent to keep track of the problem-solving tasks remaining to be executed. When a particular course of action is selected, either manually by the end-user or automatically by the agent, one or more problem-solving task items are placed on the agenda, describing an action or sequence of actions to be performed by the agent. The MASCOT control mechanism supports three types of agenda items:

1. *Service* activations, which correspond directly to the specific problem-solving services provided by the MASCOT KSs.

2. *Goal* activations, which specify high-level, objective-oriented problem-solving tasks that can be satisfied by the execution of either a service or (more likely) a *sequence* (or "script") of services and subgoals.

3. *Script* activations, which specify a predefined sequence of KS services and goals generally known to accomplish a particular problem-solving task.

3.3 Problem-Solving Flow

All problem-solving activity within the MASCOT agent architecture is triggered by either the incorporation of a new event (such as a request for bid, a request to validate a high-level schedule, or simply a shop floor status update) into the current working context, or the modification of an assumption within the current working context (e.g., "what-if" analysis to evaluate the benefits of using a different supplier, subcontracting production of a particular subcomponent, or adding work shifts, etc.), both of which can be performed by either the end-user or the agent (as specified by the execution profile). The flow of problem solving is summarized in Figure 3. It proceeds from the modification of the current working context in a clockwise direction through the following steps:

1. Updating the set of unresolved issues within the current working context to reflect the initial problem-solving action.
2. Selecting one or more unresolved issues to resolve.
3. Selecting a resolution method for the selected unresolved issue(s).
4. Activating the selected resolution method.
5. Executing the problem-solving service corresponding to the activated resolution method.

The MASCOT module-activation controller is invoked whenever there are problem-solving tasks on the agenda remaining to be executed, or, when running automatically (and depending on the execution profile), there are events to incorporate or unresolved issues to resolve.

3.4 Problem-Solving Modules

Knowledge sources serve as the primary problem solvers in MASCOT. They communicate their results by posting new information to the blackboard (e.g., new process plans and production schedules) and modifying existing information (e.g., updated process plans and reoptimized production schedules). Each domain-level KS acts primarily as a server that supports a variety of problem-solving services.

The typical MASCOT agent relies on a number of KSs, which tend to vary from one agent to another depending on, for example, the level at which the agent operates (e.g., facility-based scheduling agent versus enterprise-wide scheduling agent), and differences in legacy systems and operational policies. Examples of typical KSs are briefly discussed below:

- **Lateral Coordination KSs** manage the interaction between peer-level agents (both internal and external to the MASCOT agent hierarchy) to support lateral coordination at multiple levels. At the high level, the KS is responsible for implementing protocols to guide, for example, the exchange of requests for bids, bid submissions, and ATP information. At the low level, its protocols guide the process of dynamically establishing and revising tight yet robust delivery dates for materials and subcomponents produced by upstream facilities/agents and consumed by downstream facilities/agents.
- **Vertical Coordination KSs** implement the necessary protocols to (1) guide the maintenance of consistency across finite capacity models at different levels of abstraction within an enterprise, and (2) selectively validate high-level planning decisions across these levels through evaluation

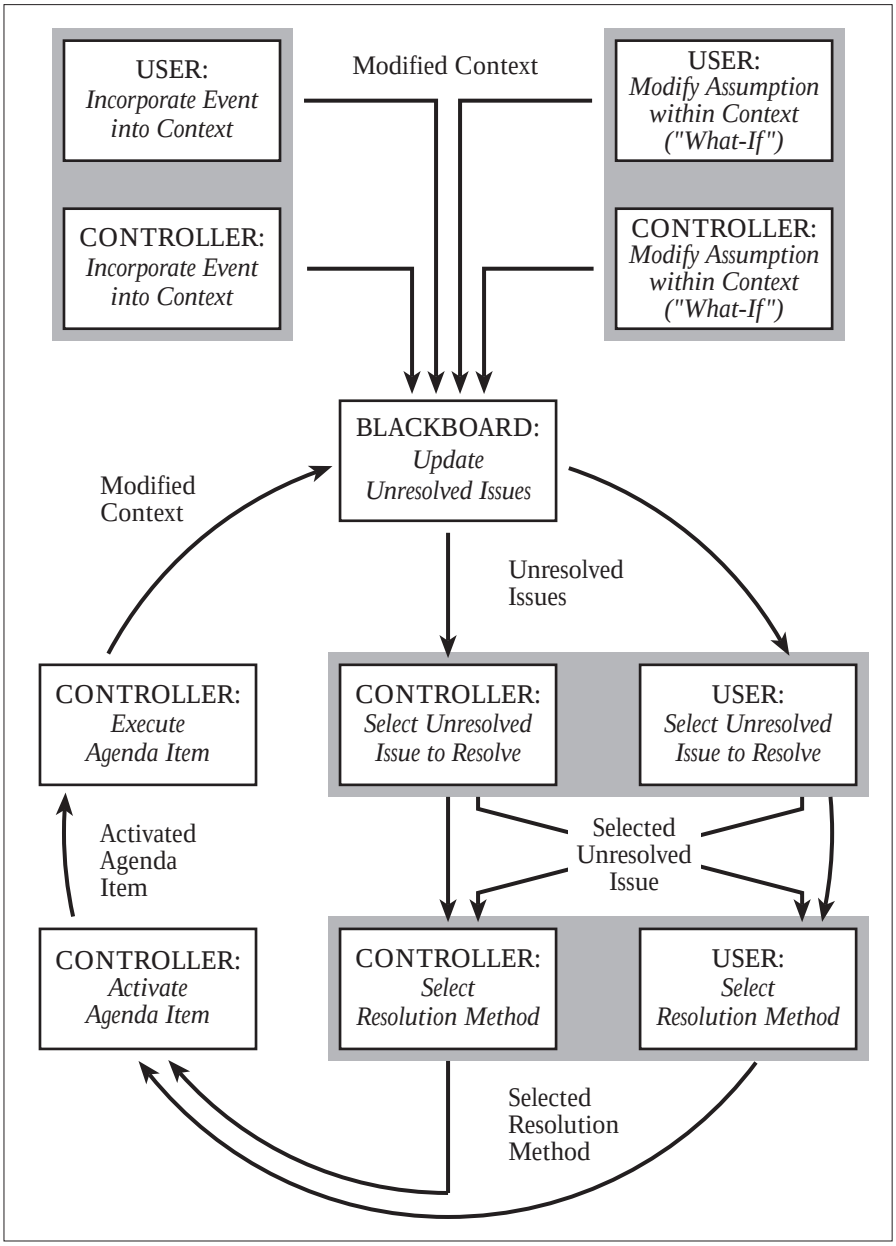

Figure 3: Problem-Solving Flow in MASCOT Agents

of the more detailed capacity models of lower-level, and often bottlenecked, facilities (via interaction with their corresponding agents).

- **Production Scheduling KSs**: in experiments reported later in this paper, we used MICRO-BOSS [Sadeh, 1994, Sadeh, 1995], a dynamic finite capacity scheduling system that has been shown to support the rapid generation and dynamic revision of high quality solutions to complex, large-scale scheduling problems at various levels within the enterprise. In other situations, this module may simply be a legacy scheduling system.

- **Planning KSs**: typically a process-flow/bill-of-material definition module or a process planning module, as in the case of the IP3S agent [Sadeh *et al.*, 1998].

- **Analysis KSs**: a typical analysis KS is a resource-utilization KS that estimates resource contention, taking into account both current reservations and projected demand from unscheduled orders. An example of such a KS has been used in the IP3S agent [Sadeh *et al.*, 1998].

- A **Communication KS** that facilitates coordination between MASCOT agents as well as with other information sources (e.g., shop floor control).

4 Empirical Evaluation of Finite Capacity Lateral Coordination Protocols

While the MASCOT architecture aims at supporting more flexible and efficient coordination across the supply chain, it does not assume any particular set of coordination protocols or policies. Instead, it is capable of supporting traditional leadtime-based coordination as well as policies aimed at taking advantage of finite capacity scheduling functionalities. This section highlights a few results of an empirical study conducted using MASCOT to compare several lateral coordination policies under different sets of business assumptions [Kjenstad, 1998].

Lateral coordination is particularly critical when it comes to generating robust yet competitive and cost-effective promise dates in response to new customer demands. Once a delivery date has been quoted to a customer, lateral coordination can help dynamically resynchronize production across the supply chain to best meet prior commitments in the face of contingencies such as delays in raw material/subcomponent deliveries, production delays, quality problems, arrival of new orders, etc. Traditionally, supply chain coordination has relied on standard leadtimes to determine synchronization dates across the supply chain. These leadtimes are insensitive to actual load conditions. As a result, depending on the situation, they will be either (1) overly conservative, yielding uncompetitive promise dates, long leadtimes, and excessive inventory, or (2) overly optimistic, resulting in low levels of customer satisfaction and late-delivery penalties. While finite capacity considerations offer the promise of more competitive and more realistic estimates, they can also be expected to yield more brittle solutions, which are easily invalidated by contingencies. A key question in this regard is whether, through dynamic coordination, it is possible to reap the benefits of finite capacity computations without suffering from the brittleness of their underlying solutions.

To answer this and related questions, our study, which is detailed in [Kjenstad, 1998], looked at a number of different supply chain configurations (e.g., different numbers of tiers, tiers with one or more suppliers, etc.) as well as different ATP/CTP scenarios (e.g., incoming orders with non-negotiable due

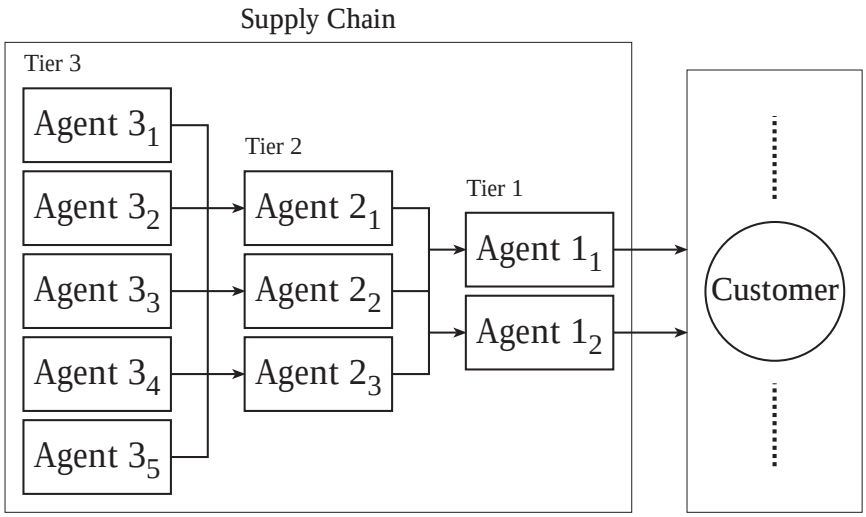

Figure 4: A Three-Tier Supply Chain Configuration

dates, incoming orders that can be turned down, incoming orders with negotiable due dates and competition with other possible suppliers). A supply chain simulation testbed was used within which MASCOT agents were responsible for coordinating their operations in response to stochastic requests for bids and stochastic production contingencies.

The testbed consisted of a problem generator and a supply chain simulator:

- The problem generator took as input a supply chain configuration (number of products, number of tiers, number of entities in each tier, and number of resources in each entity) and generated stochastic variations of supply chain models based on this data. Variations among model instances included differences in product types, process routings, and bills of materials.
- The simulator simulated operation at each of the entities across the supply chain while generating random events such as:
  - *Requests for bids* (including product types, quantities, requested delivery dates and prices).
  - *Execution contingencies* such as resource breakdowns and variations in processing times.

While a number of different scenarios were considered, the results reported below were obtained by simulating the supply chain depicted in Figure 4. This is a supply chain where multiple entities/agents are in competition at different tiers. At any point in time, each supply chain entity can have both multiple suppliers and multiple customers. In addition, process routings include assembly operations that require synchronization of subcomponent deliveries from multiple customers. This provides for situations where finite capacity lateral coordination can help alert the supplier of a subcomponent that delivery of another subcomponent from a second supplier has been delayed, allowing the first supplier to reprioritize its schedule accordingly.

Additional assumptions used in the experiments reported in this section included the following:

- The supply chain operates in a pure make-to-order lot-for-lot fashion. Facilities that are the farthest upstream are assumed to have instantaneous access to raw materials (e.g., from ample stock).

- Within each supply chain entity/agent, finite capacity schedules are regenerated on a daily basis, and jobs queuing in front of machines are executed starting with the job that is scheduled to run first on that machine. In other words, the schedule is interpreted as a priority mechanism rather than being fully enforced. All schedules were generated using the MICRO-BOSS finite capacity scheduling system [Sadeh, 1994, Sadeh, 1995].

- A request for bid can be accepted by an agent with its requested delivery date (i.e., unconditional ATP/CTP bid), it can be accepted subject to a relaxed delivery date (i.e., conditional ATP/CTP bid) or simply rejected (i.e., no bid). The chance that a bid is accepted depends on how close it is to the requested delivery date. Agents only send bids on orders that are expected to increase their profit. Profit from an order is measured as the difference between the sales revenue and costs associated with that order. Costs include fixed costs such as fixed production costs and sub-component/raw material procurement costs as well as inventory costs and late delivery penalties (see [Kjenstad, 1998] for further details).

Before regenerating a new schedule, each supply chain agent incorporates new incoming requests for bids into a copy of its working context (where unresolved issues are created accordingly). Requested delivery dates are interpreted as internal due dates. Similarly, promised delivery dates from suppliers, if available, are interpreted as earliest possible release date constraints. Each time a supply chain agent generates a new solution it communicates the resulting start time requirements to its suppliers and the expected completion dates to its customers. To avoid creating unrealistic synchronization dates, some padding is selectively added in the process, as further detailed in [Kjenstad, 1998].

Below, we report performance of each of the following three sets of lateral coordination policies:

- **Lead-Neg/C**: A reference policy relying on historical leadtime data to estimate delivery dates for prospective orders.

- **FCS-Neg/C**: This policy is similar to Lead-Neg/C, except that it relies on finite capacity schedules in its top-tier estimates (i.e., the agent that directly receives a request for bid generates a local finite capacity schedule but relies on leadtime assumptions as far as its suppliers are concerned).

- **Sync-Neg/C**: This policy relies on finite capacity calculations at each tier across the supply chain to respond to requests for bids.

The results shown in Figure 5 were obtained by averaging performance over twenty simulation runs (each simulating 150 days of execution for a particular model instance) for each policy. The vertical lines in Figure 5 show the estimated mean profits and their corresponding 95% confidence intervals. The vertical bars represent average leadtimes across the entire supply chain. Profits obtained with the finite capacity lateral coordination policy (Sync-Neg/C) are clearly higher than those obtained with the other two policies. The confidence interval is also considerably smaller, indicating that this policy is also more consistent/reliable. Leadtimes obtained with this policy are also considerably shorter. Additional results reported in [Kjenstad, 1998] show that these performance improvements can be attributed to a higher accuracy in the ATP/CTP process combined with a better ability to dynamically adjust for contingencies and satisfy prior delivery commitments.

The results in Figure 5 were obtained for an average nominal load of 80%. This is the average load of the most significant bottleneck area (across the entire supply chain) if all requests for bids were

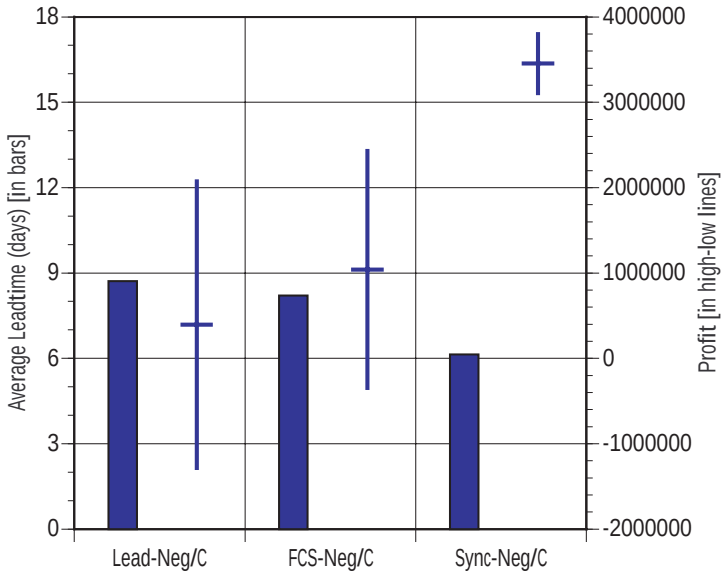

Figure 5: Sensitivity to Variations in Number of Bid Requests

to result in actual orders. Figure 6 shows similar results for nominal loads varying between 40% and 120%.

These results show that:

- All three policies result in similar performance when the nominal load is low (i.e., less than 60%). Most jobs can then be scheduled just-in-time without much resource contention. The benefits of finite capacity lateral coordination become more obvious as the nominal load increases.

- In fact the finite capacity lateral coordination policy is able to consistently maintain a high profit as the nominal load increases above 0.6, while performance of the other two policies deteriorates. This is not really a surprise: as the nominal load gets close to or greater than 100%, finite capacity coordination makes it possible to selectively turn down orders and relax requested delivery dates, which the other two policies cannot really do.

5 Concluding Remarks

A key to agility in today's global economy is the ability to dynamically coordinate planning and scheduling decisions across the supply chain. In this paper, we introduced MASCOT, an architecture that aims at providing a framework for coordinated development and manipulation of planning and scheduling solutions at multiple levels of abstraction across the supply chain. MASCOT places a particular emphasis on mixed-initiative problem solving, enabling users to flexibly select among a broad range of interaction regimes. It also supports powerful workflow management capabilities that allow users to assess complex tradeoffs while manipulating assumptions that often span multiple tiers across the supply chain.

An initial version of the MASCOT architecture was developed and validated in the context of an environment requiring coordination between a machine shop and a tool shop at Raytheon. Recently, the architecture has been further validated through experimentation with more complex supply chain

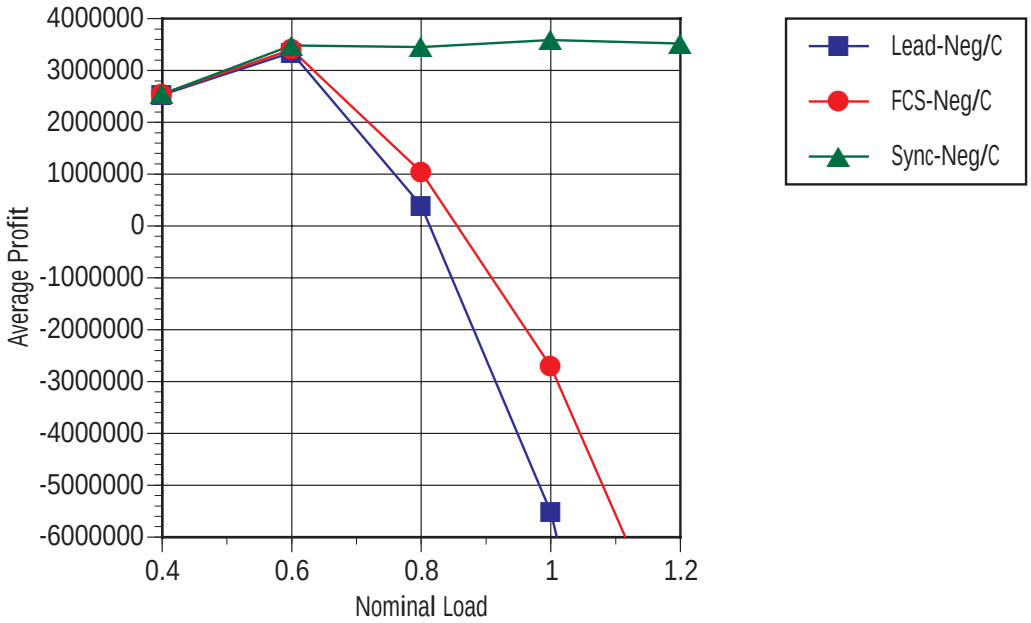

Figure 6: Sensitivity to Variations in Nominal Load

configurations and through development of coordination policies that attempt to dynamically take advantage of finite capacity considerations. Future work is expected to involve refinement of existing mixed-initiative functionalities as well as experimentation with combinations of new lateral and vertical coordination policies.

References

[Carver and Lesser, 1992] N. Carver and V.R. Lesser. The evolution of blackboard control architectures. CMPSCI Technical Report 92–71, Department of Computer Science, University of Massachusetts, Amherst, October 1992.

[Carver and Lesser, 1993] N. Carver and V.R. Lesser. A planner for the control of problem-solving systems. *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Planning, Scheduling and Control, 23(6), November 1993.

[Corkill, 1991] D.D. Corkill. Blackboard systems. *AI Expert*, 6:40–47, September 1991.

[Erman *et al.*, 1980] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy. The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[Kjenstad, 1998] D. Kjenstad. Coordinated supply chain scheduling. PhD thesis and NTNU Report 1998:24, Department of Production and Quality Engineering, Norwegian University of Science and Technology (NTNU), Trondheim, November 1998.

[Lesser and Corkill, 1983] V.R. Lesser and D.D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, Fall 1983.

[Sadeh *et al.*, 1998] N. M. Sadeh, D. W. Hildum, T. J. Laliberty, J. McA'Nulty, D. Kjenstad, and A. Tseng. A blackboard architecture for integrating process planning and production scheduling. *Concurrent Engineering: Research & Applications (CERA)*, 6(2):88–100, June 1998.

[Sadeh, 1994] N.M. Sadeh. Micro-opportunistic scheduling: The Micro-Boss factory scheduler. In M. Zweben and M.S. Fox, editors, *Intelligent Scheduling*, chapter 4, pages 99–135. Morgan Kaufmann, San Francisco CA, 1994.

[Sadeh, 1995] N.M. Sadeh. Micro-Boss: Dual-use ARPI scheduling technology helps improve manufacturing performance. *IEEE Expert*, February 1995.