Understanding Issues in Multi-Agent System Design Through the Blackboard System Model.

Zachary B. Rubinstein
University of New Hampshire
zbr@cs.cmu.edu
Type of problems

- Collaborative systems made up of multiple providers.
- There is an intermediate number of providers (roughly > 3 and < 1000)
- Providers work together to solve a complex, broadly defined problem with many grades of solutions.
- No monolithic solution known a priori,
- Problem is highly interdependent and often situation-dependent
Providers

- A provider corresponds to a class of service offered.
- Providers don’t necessarily correspond to the number of agents.
- Same functionality but different state means different providers.
Example Domains

- Sensor Fusion
- Intelligence Gathering and Assessment
- Emergency Response
- Military Operations
Focus of the talk

- To understand some of the issues in these systems through a discussion of Blackboard systems.
- Blackboard systems share many of the same issues with these types of multi-agent systems.
- The insight is to use the more straightforward form of these issues as they are presented by the Blackboard model to better understand how they are manifested in the MAS systems.
Challenges in Blackboard Systems (Dan Corkill)

- **Representation**
  - getting software modules to understand one another

- **Awareness**
  - making modules aware when something relevant to them occurs

- **Investigation**
  - helping modules to quickly find information related to their current activities

- **Interaction**
  - creating modules that are able to use the concurrent work of others while working on a shared task

- **Integration**
  - combining results produced by other modules

- **Coordination**
  - getting modules to focus their activities on the right things at the right time
Introduction to Blackboard Systems
Blackboard Systems

- Based on a “brainstorming experts” analogy
  - Experts work as a team to “brainstorm” a solution to a problem, using a large blackboard as the workplace for cooperatively developing the solution
  - Problem specifications are written onto the blackboard
  - Experts all watch the blackboard, contributing their expertise (on the blackboard) when possible
What is a Blackboard System?

- A simple problem-solving concept
  - Knowledge modules interacting via a shared database
- Extremely subtle and open-ended in realization
  - Will detail these issues

E=mc²

```
(first ,b)
```

Module A  Module B  Module C

Module D  Module E

Eat at Joe's
Blackboard System Components

- Knowledge Sources (KSs)
  - software specialists; each providing expertise needed by the application
- The Blackboard
  - shared repository of problems, partial solutions, suggestions, recommendations, and contributed information
- Control Shell
  - controls the flow of problem-solving activity in the application
Blackboard-System Operation

Blackboard is a repository for common data and a trigger mechanism for KS activation. KS activations are executed, one at a time. Control decisions are made by the control shell.
Basic Blackboard-System Control Cycle

- Blackboard
- Control Shell
- Pending KSAs
- Executing KSA
- KS
- Best KSA

Events flow from the Blackboard to the Control Shell, which then interacts with the Pending KSAs. The best KSA is selected from the Pending KSAs and executed. New KSA KSs are added to the Pending KSAs.
Characteristics of a Blackboard System

- Large-grained cooperating knowledge source (KS) problem-solving model
  - KSs can use diverse internal problem-solving representations and implementations
- KSs interact anonymously using shared global database called the blackboard
  - Blackboard serves as communication medium and buffer
  - Blackboard serves as community memory of data, results, and control information
  - Blackboard serves as KS trigger mechanism
- Opportunistic problem solving directed by explicit control component
  - Separate from individual “domain” KSs
  - Large-grained control of KS executions
Additional Characteristics

- No consensus, but often present!
  - Solution is generated incrementally
  - Multiple levels of abstraction
  - Structured blackboard
    - beyond level partitioning
  - Competing hypotheses problem-solving representation
  - Blackboard used for control information
  - Reflexive control implemented using blackboard system
Communicating Modules

- Data-flow-systems design
  - identify functional modules
  - connect them according to communication paths
- Advantages
  - simple, predictable organization
- Disadvantages
  - static processing paths
  - direct interaction (changes in functional modules requires redesign)
  - leads to “private” interaction protocols that make interoperability difficult
Blackboard Systems

- Blackboard-systems design
  - identify functional modules, blackboard structure, and objects
  - add control strategies as needed

Advantages
- dynamic processing paths
  - adapt to situation
- indirect interaction
  - allows transparent reorganization
- “public” representation
  - allows other modules, development/monitoring tools, and control components to access communications
- blackboard serves as repository

Disadvantages
- more complex system infrastructure is needed
Advantages of Blackboard Systems

- **Modularity**
  - KSs can be developed independently
  - KSs can have been developed long before the blackboard-system application itself

- **Integration**
  - KSs can use widely differing approaches, representations, programming languages
  - KSs can use diverse hardware--locally or remotely

- **Extensibility**
  - New KSs can be added easily
  - Existing KSs can be upgraded to new versions
Advantages of Blackboard Systems

- **Reusability**
  - KSs that provide expertise to one application can be redeployed in new applications

- **Strategic control**
  - Determines where the application expends its resources
  - Important when
    - The number of KSs grows
    - KSs have overlapping capabilities
    - The best approach to solving today’s problem differs from that used in yesterday’s problem
Why Use a Blackboard System?

- When multilevel reasoning or flexible, dynamic control is required
- When heterogeneous problem-solving representations and expertise must be integrated
  - Including integration of legacy applications
- When many developers are involved
  - Large-grained anonymous modularity is important for design, implementation and maintenance
    - “metaprogramming”
Blackboard-System Application Areas

- Sensory interpretation
- Design and layout
- Process control
- Planning and scheduling
- Computer vision
- Knowledge-based simulation

- Command and control
- Symbolic learning
- Case-based reasoning
- Data fusion
- Knowledge-based instruction
Issues in Blackboard Design

- Develop common representation - must be lingua franca among KSs.
- Design blackboard structure - dimensionality tailored for investigation.
- Determining triggers, preconditions.
- Assigning Ratings - determine relative importance of KSAs. Often developing rating scheme without knowing all the contexts in which a KS may be invoked.
- Queue Latency - triggered KSs’ KSAs stay on queue
  - Triggering state may no longer apply. Need to manage queue (revalidate and obviate conditions).
  - Very similar rated KSAs may result in the marginally lower-rated KSAs not being serviced (best-first behavior).
- Frequency of new data vs. processing time
  - Worry about how to balance being reactive to new data while still making progress towards a good solution.
  - Since iterative control, need to control the time any one KS takes or introduce safe points when KSs can be interrupted.
Blackboard-System Challenges (strengths & weaknesses)

- **Representation**
  - Public on the blackboard; design decisions important to success

- **Awareness**
  - Simple, disjunctive triggers; fast & easy

- **Investigation**
  - BB search crucial to performance

- **Interaction (iterative only)**
  - Direct interaction prohibited; single executing KSA

- **Integration**
  - Principled (e.g., Bayesian net) and Ad hoc

- **Coordination**
  - Single thread, highly reactive, instantaneous; leads to continuity-of-purpose problems
Now let’s look how these issues play out in a cooperative multi-agent system...
COORDINATORS

- DARPA funded program for developing distributed software systems to help fielded units adapt to changes (dynamics) in the situation.
- Examples of changes are modified mission plans, unexpected execution results, unexpected environments, and unexpected level of adversity.
- Each unit is responsible for evaluating the current situation and, if necessary, modify its schedule by either changing the timings of tasks to complete or to select from preplanned contingencies.
COORDINATORs

- Problems of Interest
  - Collaborative, distributed management of joint plans/schedules
- Assumptions
  - No agent has complete global view (penalty for centralizing parts of solutions)
  - Uncertain execution environment
  - Objective is to maximize quality
CTAEMS Problem Representation

- HTN-style representation of problem constraints
- Methods (leafs) are primitive process steps
  - Executable by a specific designated agent
  - Have associated duration and outcome (quality) distributions
- Tasks model abstract processes and encapsulate sets of more detailed tasks
  - QAF type - determines how quality is accumulated
- Tasks/methods can have start and end time constraints
- Tasks/methods can have non-local effects (NLEs)
  - Hard constraints must be enforced - A enables B
  - Soft constraints boost (or degrade) quality - A facilitates B
cTAEMS Structure
Visibility

- Based on “one-step removed” rule.
- Agents know about
  - methods they own and the path of tasks back to the root
  - remote nodes that are connected to a local nodes via NLEs.
  - owners of the remote nodes.
  - Agents that can see a known node, i.e. if an agent knows about a node, it knows the names of the agents that can also see that node.
Example Coordination Problem

Suppose at time 2, the blue agent receives a message that M3 failed. How do the agents coordinate to drop M1 and schedule M2 and M4?
Representation

- Program dictates cTAEMS, which includes:
  - Model representation
  - Simulator command language (start, complete, quality accumulated, change events, etc.)

- What about internal representation? Coordination protocols? Agent-to-agent instruction language?
  - More about this later in coordination.
Awareness

- Messages trigger attention
- Execution-time messages from simulator may report an anomalous state, e.g.,
  - Method overrun
  - Method underrun
  - Failed or lower quality than expected
  - Changed problem structure, including new tasks, deleted tasks, changes in outcomes, etc.

- Agent-to-agent Coordination Messages, e.g.,
  - Updating states of dependent remote agents state
  - Coordination, including hypothetical (“what if”) reasoning.
    - Essentially search directives.
Note that protocols must be able to convey domain information. There is no escaping domain specific protocol even when it is wrapped in general speech acts protocols like KQML (the content slot still has to have domain-specific language).
Digression on Problem of Timescales and Messages

- Major issues of keeping up if the reaction speed is calculated in terms of milliseconds.
- Can’t do extensive local reasoning and keep up with new messages, each potentially having a significant impact on the current behavior (schedule).
Investigation

- Two types of investigation: local and distributed.
- Local investigation looks at the local view and sees if there is an opportunity to do better
  - May be the result of a direct conflict (negative situation) or a genuine opportunity (positive situation).
  - For example, an STN may be used to help detect these situations.
- Distributed investigation is looking to see if there are better opportunities by coordinating.
  - For example, a good local option may be discovered but requires certain commitments from remote agents.
    - Those commitments have to be explored and their impact has to be evaluated.
These two types of investigation begin to merge when abstractions of other agents states and the worth of their tasks to the global quality are passed around.

- The idea is to keep the local state informed so its local reasoner always has enough information to make good local decisions.
- This tended to be what happened in the first-year of COORDINATORs due to the time scale being too small for extensive search and the evaluation problems not requiring extensive coordination, i.e., a good strategy is to stay fairly close to the initial schedule.
By definition of the program, agents can interact with the simulator and other agents included in the local model by the “one-step removed” rule.

Static organization structure could specify communication channels, authority relationships, and what can be exchanged.

- For example, owners could be assigned to all tasks and scheduling would be a top-down search with each owner being responsible for its subtree.

Depending on the coordination protocols, agents may learn about other agents via coordination dynamic partial centralization.

- E.g., Asynchronous Partial Overlay
King of Dynamic Partial Centralization
Integration

- In COORDINATORs, integration is assimilating other agents actions or potential actions into the local view.

- The goal is to make local decisions that maximize global quality. Both actual behavior and hypothetical behavior has to be integrated in order to evaluate
Coordination

- Huge distributed search space with great opportunity for better solutions
- Exploring hypothetical scenarios has problem of knowing when hypotheticals are no longer relevant.
  - Major question of knowing when to make a commitment
  - Almost a bigger question is knowing when to drop a commitment.
  - Would like the answer to those question to lead to a strategy that is admissible, i.e., constantly leads to a solution without overestimating.
  - If not, then thrashing or looping is possible (cycle of do, undo, do, undo, and so on....).
Coordination (cont.)

- Issues in COORDINATORs
  - Greedy approaches can’t succeed in context of execution-time dynamics
  - Explicit coordination can’t succeed in highly dynamic environments that have no time for thinking.
  - Need to understand what the limits the problem and make reasonable goals.
What to take away…

- When you first go to design a collaborative system, spec out answers to the six challenges (representation, awareness, investigation, interaction, integration, coordination). These answers will give you the critical vision you will need to have a cohesive, consistent system rather than one designed by incremental evolution.

- Do not forget to address whether it should be concurrent, distributed, multi-agent?

- Loosely structured collaboration can exist in a centralized form, i.e., a blackboard system.
Questions or Discussion?

Thanks