Acting Responsible: Reasoning about Agents in a Multi-agent System

Stephen Fickas
Rob Helm

CIS-TR-91-02
January 31, 1991

Abstract

Our general interest is in the design of multi-agent (composite) systems. We focus in this paper on the design of joint problem-solving strategies for such systems. In particular, we are interested in a) the creation of agents, b) their assignment of roles and c) their communication or problem-solving protocol.

We make two claims in the paper: 1) there is a general, formalizable set of multi-agent protocols that account for an interesting class of composite systems, and 2) one can evaluate a given protocol on the *ability* and *reliability* of its constituent agents, and on the amount of *interference* agents have in carrying out their roles. We describe a design model based on these two claims, and then rationally reconstruct an existing composite system using our model. From this we make sufficiency and necessity arguments.

Department of Computer and Information Science
University of Oregon
Eugene, OR 97403

1. Introduction

Our general interest is in the design of multi-agent systems, or what Feather has referred to as composite systems [Feather 1988]. Composite systems encompass multiple agents involved in ongoing, interactive activities. We focus in this paper on the design of joint problem-solving strategies for such systems. In particular, we are interested in a) the creation of agents, b) their assignment of tasks and c) their communication or problem-solving protocol as part of a composite solution to a problem.

We have studied representative composite systems from different domains. From this study, we make two claims:

Tractable set. A general, formalizable set of multi-agent protocols account for an interesting class of composite system solutions.

Evaluation criteria. The choice among competing protocols is founded on three criteria:

1. Is an agent *capable* of carrying out an assignment (duty, role)?
2. Given that an agent is capable, will it be *reliable* in carrying out an assignment?
3. Given that an agent is capable, will an assignment *interfere* with its other responsibilities, either in the current system or some other system where it plays a role?

The two claims, together, form a theory of design founded on state-based search: there is a general set of operators for producing members of composite systems; there are criteria for pruning the search space and judging/comparing the value of proposed solutions in a particular application domain.

We have defined a model of design for composite systems that is based on this theory. At the heart of the model is an operator we call *assignment of responsibility.* It is through this operator that problem-solving protocols are built. In typical composite design problems, there is more than one way of applying the assignment of responsibility operator. It is here that the 3 evaluation criteria come into play.

We have used our model to both synthesize new composite systems and rationally reconstruct the design of existing composite systems. In this paper, we will use the latter approach to provide evidence for our claims: we will use our model to rationally reconstruct the design of an existing (circa 1880) composite system, that of a multi-agent train-scheduling system. In doing so, we will make two arguments: 1) our model is *sufficient* to generate an interesting class of composite systems, and 2) our model provides, in some cases, the only means that certain composite design components can be generated/rationalized, i.e., the model is *necessary* (at least in these cases).

2. Our design model

Our design model is based on a transformational approach. We start with a naive, incomplete representation of a system and gradually transform it into a composite solution. Our representation of a system comprises two parts:

1. a *generative* part denoting the possible behavior produced by agents in the system. In our model, this takes the form of a discrete event language that can be viewed, alternatively, as a subset of Gist [London and Feather 1982] or a superset of

Numerical Petri Nets [Wilbur-Ham, 1985].We will use the latter view in this paper for presentation purposes.

2. a *constraining* part consisting of a set of *goals*, i.e., constraints on behavior. Goals are expressed in terms of system-wide properties, regardless of how responsibility is decomposed among agents. In our model, goals are represented in a style of temporal logic roughly similar to that of that of the ERAE language [Dubois and Hagelstein, 1988].

Transformations are defined such that they map one system state to another. This mapping may be correctness-preserving or not. In particular, new agents may be introduced, the specification of existing agents may be changed, new problem-solving protocols may be introduced, and even the system-wide goals may be modified.

The control of our design model is patterned after the Glitter system [Fickas, 1985]: an open task is selected; methods are found that address the task; a choice is made among the methods; the selected method is applied, leading to a new design state. As with Glitter, we define a design state to be the combination of system state and task agenda (represented in our model as a problem reduction tree). An overview task agenda for the design in this paper appears in Appendix A.

The further details of both specification and development state representations are found elsewhere [Fickas and Helm, 1990]. We will present a simplified representation in this paper to allow us to concentrate on our true focus, that of agent responsibility and general multi-agent problem-solving protocols.

3. An example

We will demonstrate and evaluate our design model with an example of composite system design. The example we have chosen is the multi-agent station-train signalling protocol described in McGean [McGean 1976], henceforth simply the "McGean design"[1]. The problem-solving protocol attempts to prevent collisions of trains by allocating regions of track of 5 to 15 mile length (called blocks) to a single train at a time. Each block has a single station associated with it. When a train T enters a block B (by passing or stopping at the station for B) the operator of that station sets a signal that block B is now occupied. No other trains may enter B until the signal is cleared. The operator clears the signal when the next operator down the line sends a signal back that the train T has been spotted (and hence, has vacated block B).

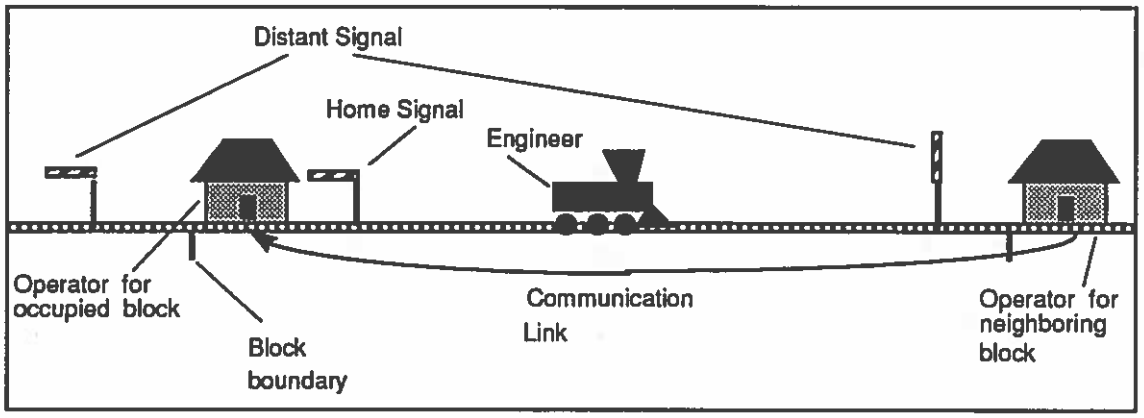
The occupied/clear signal is actually broken into two components, a "distant" signal and a "home" signal. Both are always in the same state. The distant signal is set out far enough from the station (in the preceding block) to guarantee that a train can stop before reaching the station. The home signal is in front of the station (Figure 1). In essence, the distant signal tells a moving train when to start stopping, and the home signal tells a stopped train when to start moving. If a train sees a clear distant signal then it will see a clear home signal (see the constraint of one train per block at a time).

Looking at the solution from a composite system view, the protocol makes engineer

1. This is clearly a misnomer, for McGean is only the chronicler of the 1880 design. However, the identity of the original designer is unknown to us, and hence, McGean becomes the stand-in.

agents responsible for entering a block of track between stations only if the signal for that block reads clear (vertical). Station operator agents are given the responsibility of 1) setting the distant/home signal to occupied (horizontal), 2) resetting the signal to clear, and 3) passing information among themselves. Finally, dispatcher agents (not shown in Figure 1) are responsible for making sure trains enter a system in a safe fashion.

Figure 1 Simplified McGean design.,



We will rationally reconstruct the McGean design to make the following points:

- The design uses a standard constraint satisfaction technique, which we call *brinkmanship*, for keeping trains from colliding. Two agents, a dispatcher and an engineer, are made jointly responsible for not moving beyond the brink.
- The design uses a standard, multi-agent problem solving protocol, which we call *sequential split*, to break collision avoidance into two temporally-disjoint pieces. A separate agent is assigned to each piece, with responsibility shifting from one agent to the next.
- The design uses a standard, multi-agent problem solving protocol, which we call *set/reset*, to allow station operators to communicate with engineers.
- The design uses a standard, multi-agent problem solving protocol, which we call *remote/note send*, to allow station operators to communicate with each other.
- The design addresses some, but not all of the *reliability* problems resulting from responsibility assignments. For instance, it provides an active warning (distant) signal of a train ahead (as opposed to just relying on line-of-sight or a painted warning sign). It also implements communication between operators that is partially fail-safe - a train may not proceed if communication is lost between operators. On the other hand, the design does not provide for recovery or redundancy if a dispatcher, engineer or operator became incapacitated at a critical time. We will assume that such agent reliability was considered and dismissed in the original design.
- The design addresses some, but not all, of the *interference* problems resulting from responsibility assignments. For instance, the design overloads operator agents, assigning them three separate responsibilities: 1) set a signal when their block of track is occupied, 2) reset the signal when it is clear, and 3) send a signal to the previous opera-

tor O when it is safe for O to reset her signal. In our model, overloading is one type of interference issue: how many different responsibilities can a single agent take on before the cumulative effect begins to interfere with its ability to perform?
The McGean design does not explicitly address interference arising from *motivation*, i.e., does an assigned responsibility interfere/conflict with other responsibilities or local goals of an agent. However, it is interesting to speculate whether the human operators were playing an implicit motivational role for engineers. In particular, there is no report of a problem with engineers "running the distant signal" (which would have been caught by a human station-operator), but McGean does report a problem in modern day systems (with hardware versus human operators) of engineers "riding the yellow", i.e., not completely stopping when signaled to do so.

The remainder of this section follows the development of the McGean system using our design model. Before starting the reconstruction, a note about the presentation is in order. The designer (D) sometimes calls on a critic (C) to find a counter-example, in terms of system behavior, that refutes a goal. This critic is implemented. Its initial version is described in [Anderson and Fickas 1990]. In its current state, it can use reachability graphs to produce behaviors (paths) that involve transitions *without a* not-arc. While the McGean system can be modeled with such transitions, it is unbearably messy. Hence, we will trade lucidity for automation in the following sections: transitions with not-arcs will be used and scenarios that reference them are manually produced.

We also apply transformations during the design. We have constructed a composite system editing tool that allows primitive composite-system modifications. All transformations seen in the example were carried out by manual calls to the editing tool.

3.1 The initial system

We will start our design with the system shown in figure 2[1]. To paraphrase, there are objects called trains (t) that are created and deleted. There are objects called blocks (b) that represent locations and destinations. Some blocks are adjacent to one another. Trains have locations and destinations, (i.e., there are relations Location(t,b) and Destination(t,b)), which can be created (asserted), deleted (retracted) or modified. The deletion of a train must be accompanied by the deletion of its associated relations. Trains cannot be in more than one place at the same time. Trains have a unique destination.

There are two system goals: two trains should never be at the same location (safety); trains should eventually get to their destination (progress).

The system of figure 2 represents one that captures both correct and incorrect behavior. The designer will now try to develop a new system which satisfies all of the system goals. We start by looking at "safety" goals [Alpern and Schneider, 1985], in which some state must be maintained indefinitely:

(D): Verify the ProtectTrains goal in SYS0 given Block(b1)

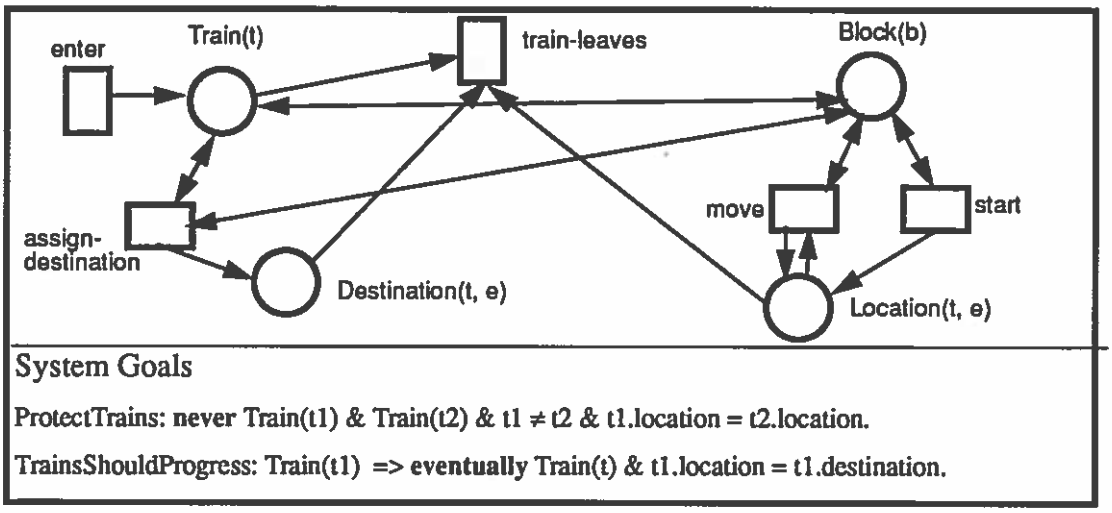
The critic generates a scenario in which two trains enter the system at the same location, a violation of the ProtectTrains goal.

---

1. Our approach is to *generate* an initial system that is consistent with the types appearing in the goal statements. This process does *not* rely on domain knowledge. Alternative approaches to initial systems construction that rely on a more domain-specific approach include *retrieving from* an abstract library [Reubenstein and Waters, 1989], or using a domain theory to *acquire* the initial system [van Lamsweerde et. al., 1991].

(C): The goal ProtectTrains is violated by scenario S1 in state SYS0:
1. given Block(b1),
2. train-enters => t1!Train,
3. train-enters => t2!Train,
4. start (t1, b1) => Location (t1, b1),
5. start (t2, b1) => Location (t2, b1),
Violation: ProtectTrains in SYS0

Figure 2 System state SYS0.



The designer has three alternative (but potentially complementary) methods to address the negative scenario S1:

1. Modify the system. The designer could change the system so that the scenario no longer occurs. For instance, the designer may decide to provide separate arrival points for each train, making it impossible to generate the "crash on arrival" scenario of S1.
2. Modify the goal. The designer could adopt a less stringent goal, such as, "No more than N trains are ever at the same location", or "99 % of all trains never encounter another train at the same location[1]". This would lead to a new task of verifying that the occurrence of S1 is in some acceptable tolerance.
3. Assign responsibility for the goal to a class of *agents*.

An *agent is* a component of a composite system which can sense a portion of the system's state, make decisions, and perform or prevent actions of the system which the agent controls. *Assigning responsibility for* a goal to an class of agents requires that all agents in that class limit their actions so the goal is achieved. Informally, only those agents responsible for a goal are expected to limit their own behavior to ensure satisfaction of that goal. For example, if a train engineer/agent is alone responsible for keeping her train from colliding with another, then she must limit her actions accordingly, and in particular, cannot rely on

1. Unfortunately, the latter goal would be impossible to express in our requirements language, or any other *formal* requirements language that we are aware of. However, it is a common type of requirement statement that clearly needs more attention.

other agents to constrain their actions to avoid collision.

The designer chooses the third method, and introduces a new agent class called "dispatcher", which controls the arrival of trains into the system. The designer assigns instances of the dispatcher agent (henceforth, "dispatchers") to control trains so that they do not collide. But now the designer must define how dispatchers will protect the goal. Using scenario S1 as a guide, the designer decides that a dispatcher should control the choice of starting location for a train. This is done by instantiating a standard control transformation, called brinkmanship, that maintains a constraint by manipulating transitions so that all of the constraint conjuncts never become true at the same time. Stated another way, the agent prevents transitions which are the last step in breaking the constraint[1]. In our example, this means never allowing a train to start at the same location as another train.

The brinkmanship transformation is shown in Figure 3. The top portion is the applicability condition: The agent A will control a transition T whose firing allows C1&C2 to become true, and hence, violates a constraint. The right side replaces the transition T with two separate transitions: Tc, a version of T that is under the control of agent A; Tu, a version of T that is uncontrolled, acting as a type of reality check - few control regimes are perfect. We will return later to the designer's handling of Tu in the McGean design.

Finally note that the grey arcs in Figure 3 represent *virtual* communication lines: they must be established (a separate design task). This is done by *verifying* that the controlling agent, the dispatcher in this case, can access whether another train is at the location of a train about to enter. In this case, the designer gives the dispatcher direct access to this information (i.e., the designer changes the grey arcs to black ones).

Figure 3 Brinkmanship transformation.

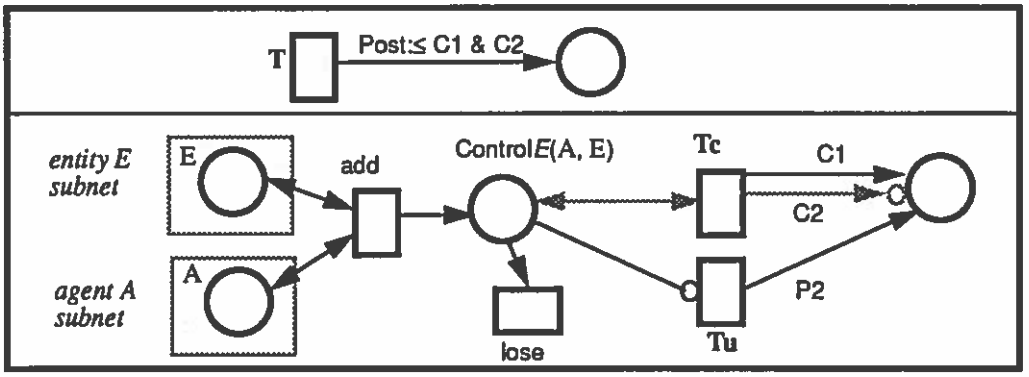


Figure 3 shows the state of the system after application of the brinkmanship transformation. We can paraphrase Figure 3 as follows:

There are sub-nets for creating and destroying trains and dispatchers (shown as dashed boxes). Trains normally arrive (start-c) under the control of a dispatcher (Dispatch-con-

1. Other control strategies for maintaining constraints include 1) never allowing a specific condition in a constraint to ever be true (e.g., never allowing trains in the system), and 2) ensuring that two conditions in a constraint are mutually exclusive (never allowing more than one train in the system).

trol). Using start-c, a train is not allowed to enter at the same location as an existing train. If no dispatcher is in control then it is possible for unrestricted entrance of trains (start-u) to take place.

3.2 The use of certification

Have we now foiled the S1 scenario? Only if we can count on start-c as always being chosen (i.e., that control is always reliably created and never lost). But SYS1 leaves a loophole: if the critic can remove all Dispatch-control(d,t) relations then it can again produce something like S1 using start-u. In at least one view, this can be seen as a reliability question: how likely is it that the controlling agent (dispatcher) will lose control (i.e., that lose-dispatch-control will fire)? If the designer decides that this is a problem, then at least two modifications are possible: Weaken the ProtectTrains goal to allow an infrequent loss of control; lessen the odds of disaster by firing the add-dispatch-control several times, generating more than one dispatch-control relation for each train, i.e., create back-up dispatchers for each train.

Figure 4 System state SYS1: Application of brinkmanship transformation.

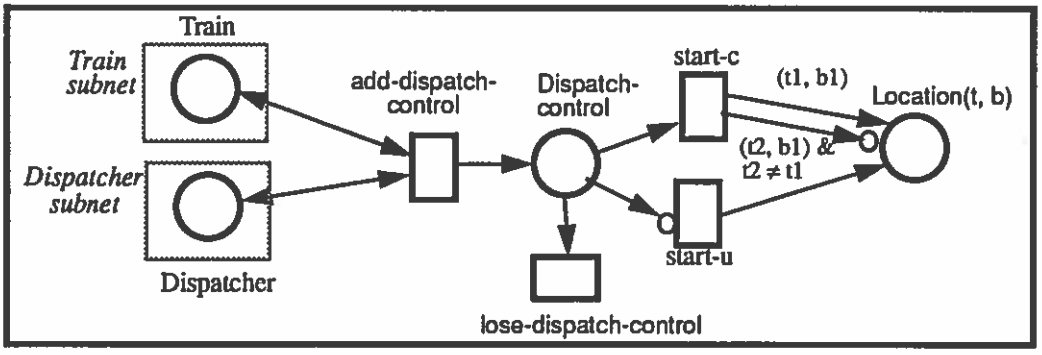
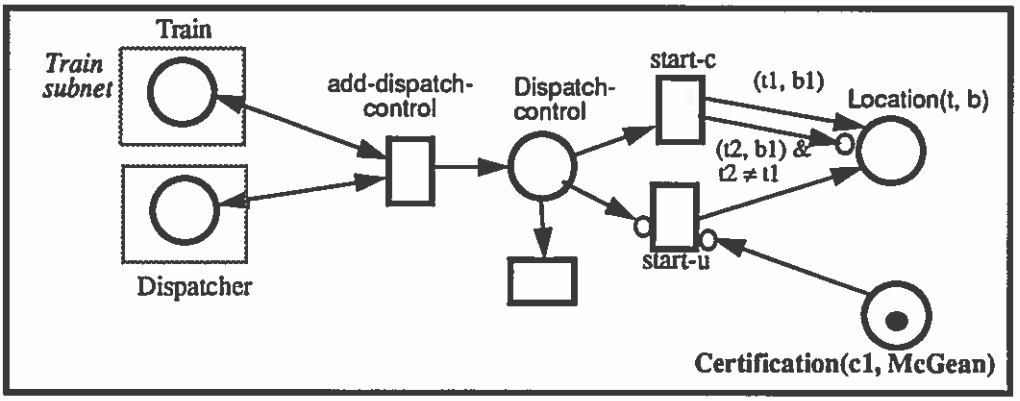


Figure 5 System state SYS2: Application of certification transformation.



There is yet another reliability problem with the current specification state as seen in Figure 3: there is no guarantee that control will be created "in time". That is, there is nothing preventing the sequence train-enters => start-u => add-dispatch-control. i.e., a train enters before a dispatcher can be assigned.

The McGean design does not 1) address either of these reliability problems, or 2) suggest that the ProtectTrains goal was weakened. Hence, we will allow the designer to certify that uncontrolled entry (start-u) is impossible. Our model provides a transformation for such certification, one that adds an explicit record of the designer's claim. The result is shown in Figure 3. The certification token, c1, can be paraphrased as "all scenarios involving uncontrolled train arrival have been considered and dismissed as either implausible or not worth handling".

3.3 A split of responsibility

The designer again asks the critic if the ProtectTrains goal is satisfied:

(D): Verify ProtectTrains in SYS2 given Block(b1) and Block(B2)

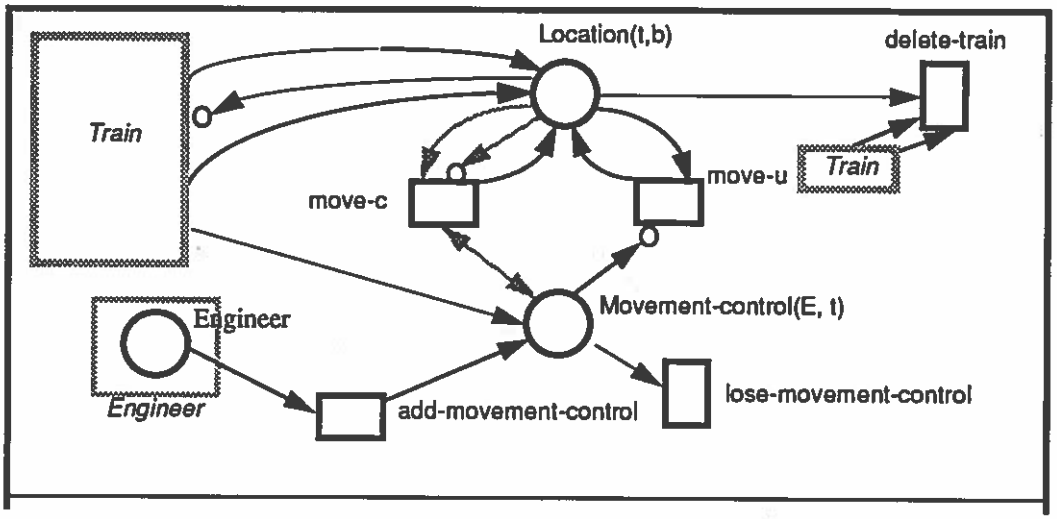
The planner generates a scenario in which two trains enter the system safely, but still end up at the same location.

(C): The goal ProtectTrains is violated by scenario S2:
1. given Block (b1)
2. given Block (b2)
3. train-enters => t1
4. create-dispatcher => d1
5. add-dispatch-control(d1, t1) => Dispatch-control(d1, t1)
6. start-c(t1, b1) => Location(t1, b1)
7. move(Location(t1, b1), b2) => Location(t1, b2)
8. train-enters => t2
9. create-dispatcher => d2
10. add-dispatch-control(d2, t2) => Dispatch-control(d2, t2)
11. start-c(t2, b1) => Location(t2, b1)
12. move(Location(t2, b1), b2) => Location(t2, b2)
Violation: ProtectTrains in SYS2

Of course the problem is that we fixed the dispatch problem of scenario S1, but not the movement problem of S2. The step the designer takes to counter the S2 scenario is similar to that taken to counter S1: the brinkmanship transformation is applied. Figure 6 shows the result of this development. To paraphrase, the brinkmanship transformation introduces a new move-c transition which can only occur when (1) an engineer is controlling the train, and (2) the conditions of brinkmanship are met. As before, the new system retains a move-u transition, which can fire for uncontrolled trains. The combination of the brinkmanship applications to the dispatcher and the engineer gives us a *sequential split* of responsibility, a common joint problem solving approach: break the problem into pieces (temporal in this case) and assign separate agents to each piece.

As with dispatch, we have reliability problems to address in Figure 6. Can a train move before an engineer is assigned? Perhaps more realistically in the transportation domain, can an engineer (driver, pilot) be lost/incapacitated? If either question is a concern then we may need to introduce a fail-safe mechanism to *disable* uncontrolled movement (see the "dead man's switch" on power lawnmowers, trains), or an understudy mechanism to *recover* from it (see "co-pilots" on large ships, planes). Adding more sophistication, we may decide to consider an auto-pilot mechanism that allows temporary but anticipated loss of control.

Figure 6 SYS3: new model of train protection behavioral



Since no discernible design effort was made to handle these type of reliability issues in the McGean design, we will again allow the designer to certify that it is not a concern by applying the certification transformation to block move-u (not shown in Figure 6).

3.4 Deriving interface requirements.

As can be seen in Figure 6, the brinkmanship transformation has introduced virtual (grey) arcs between the move-c transition and the Location place, and between the move-c transition and the Movement-control place. When faced with such arcs in the earlier dispatcher case, the designer simply blackened them, stating that it was expected that a dispatcher had direct access to the vacancy of entry blocks. To blacken the grey arcs here requires answers to the following questions:

1. Can the engineer directly access the move-c transition?
2. Does the engineer know the train's current location?
3. Does the engineer know if the track adjacent is empty (see the not-arc in figure 6)?

The designer decides that engineer agents will have direct access to both the location of the train they control, and to the "throttle" itself, and blackens the appropriate arcs. While this seems like common sense knowledge if one uses on-board humans as engineers, one can imagine vehicles that are controlled remotely by an engineer (e.g., unmanned spacecraft), that require sophisticated two-way communication devices to bring about control.

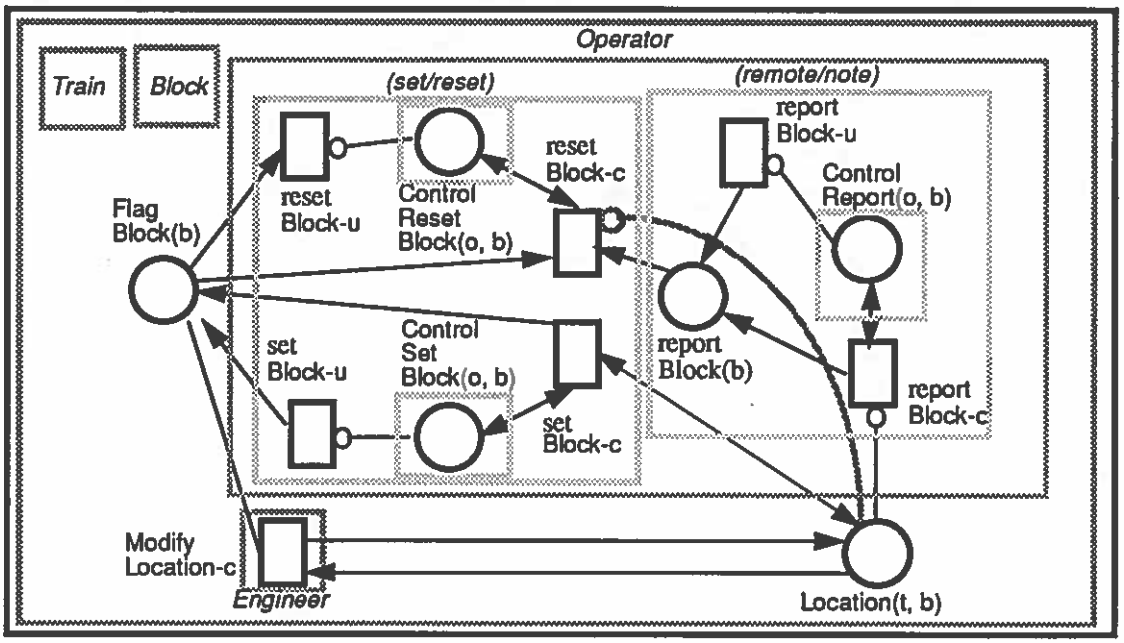
The last question asks if engineers can "see" into adjacent blocks? This was not realistic in McGean's domain: blocks were 5 to 15 miles long and engineers could not directly sense the entire length of blocks adjacent to them. Fortunately, a standard solution is available: ask another agent to act as middleman, and gather the vacancy information for the engineer and pass it along to her. To select this joint problem-solving protocol, the designer will apply a transformation we call set/reset that is defined to handle just such problems[1].

Figure 7 shows the effects of this decision on the development state, with the result of the set/reset transformation in the left portion of the figure. As with other transformations we have seen, the application of set/reset generates further design questions:

1. Can the designer guarantee that each block will be controlled at all times, or at least, whenever it is occupied? What agents will control the various actions?

The designer certifies the first question, but deliberates more carefully on the second. The set-reset transformation called for *two* disjoint sets of agents (A1 to control "reset" and A2 to control "set") to cover blocks, but the designer decides that the same agent which manages "set" (sets the home signal) can also manage "reset" (clear the signal), and applies a merge-agent transformation to agent classes. While this is clearly more economical (and follows the McGean design), it also introduces a risk. In a particular implementation of the agent class (now called "Operator"), an agent may not have sufficient time to carry out both of its responsibilities (set and reset) for all of the blocks it is tracking -- it may be *overloaded*. In the McGean design, this is partially addressed by assigning only one block to each Operator. Our designer will likewise add this capacity constraint here.

Figure 7 System SYS4 with set/reset and remote/note transformation applications



In addition to reliability and interference questions, installing the set/reset transformation raises access issues:

2. Can each agent sense whether a block it controls is occupied? Can it sense when the block is clear?

The designer decides that the operator can directly sense that a train is in its block, simply by seeing the train pass by the station. The second access issue is represented by the virtual arc entering the "reset-c" transition in Figure 7. As with engineers, an operator would have to see into the adjacent block (to see a train leaving) in order to determine whether

1. The set/reset transformation is modeled after the communication and synchronization primitives typically found in concurrent programming languages and operating systems, distributed and otherwise [Mackawa et. al.,1987].

the block it controls is, in fact, empty. This is not practical given the length of blocks. The solution adopted here is to replace the virtual arc by application of a transformation called "remote/note" (Figure 7, right). This transformation assigns an agent to monitor an entity, and report when the entity changes from one state to another. The recipient consumes ("notes") the report when it acts on it. In this context, the reporting agent (A3) generates a report when a train enters the block it is monitoring (and hence, leaves the adjacent block, making it vacant). An operator receiving a clear event resets the signal for its block.

As with set/reset, the report/note transformation raises issues of control, reliability, and interference. The designer dismisses all except the interference issue: as with set/reset, the designer merges the functions of the reporting agent A3 with those of the operator (merged from A1 and A2). In addition to monitoring their "own" blocks, operators now will monitor the blocks from which trains arrive. When a train enters a block, that block's operator will notify the previous operator that her block is now clear. The previous operator will receive the report and reset its signal.

3.5 Making progress

The designer next takes up the goal TrainsShouldProgress (see Figure 2). The designer asks the critic to verify the goal. To do so, a behavior has to be found that produces either a state from which a train can never reach its destination (possible if block adjacency is not set up correctly), or an infinite behavior/plan which maintains the negation of the goal[1]. As a special case of the latter, there is a counterexample in which a train "oscillates" between two locations, neither of which is its destination. The scenario generated moves a train back and forth between two adjacent blocks.

After studying the scenario, the designer decides to assign responsibility for the progress goal to the engineer. While there are interesting details of how the final McGean design falls out of this, at a high level it is more of the same: the engineer is made responsible for controlling her actions so that the train progresses. In particular, no new inter-agent protocols are added to the system.

Before leaving the example, there is one last note to make. The progress goal in real train systems is a case where goals might be traded-off. For instance, in automated descendants of the McGean design, engineers have been observed to deliberately ignore mechanically actuated "early warning" signals (such as yellow lights). Effectively, engineers behave as if they value the TrainsShouldProgress goal higher than ProtectTrains. In one view, this is a motivational problem: we should "make" engineers value safety first. But the larger problem is one of interference: the safety and progress goals have points of conflict in the McGean design (and in every other resource management problem that is of any interest). Another member of our group, Bill Robinson, has begun to look at ways such conflict might be detected and resolved [Robinson, 1990].

4. Summary

The outcome of our reconstruction is not the McGean design as shown in Figure 1. That design consists of physical implementations of the abstract components of our final system, e.g., mechanical signals, human agents, throttles, telegraph lines, etc. What we *have*

1. Our automated critic is capable of either type of analysis, using omega-substitution for the second case [Huber et. al., 1986]. However, as noted earlier, the critic is limited to transitions without not-arcs.

produced is a design of the agents that will participate in the composite solution, and the joint problem-solving roles they will play. Furthermore, we have considered how those roles interact with issues of ability (and access), reliability and interference. The actual choice of human versus mechanical implementations of our agents is not of interest to us - any implementation that preserves the abilities we have specified is allowable.

It is now fair to ask whether we have substantiated our claims. We have shown that our design model, based on responsibility assignment, is sufficient to generate the McGean design. During the design reconstruction, we pointed out the generality of the concepts we used by showing how different choices would have lead to different transportation systems. We have, indeed, applied the same techniques to reconstruct a traffic control system, an elevator system [Fickas and Helm, 1990], and even the user interface of a simple text editing system.

Given sufficiency, we turn to necessity: is our model the sparest one for designing composite systems? It is too early to answer this question in general - the formal study of multi-agent systems is in its infancy. However, we argue that certain pieces of the McGean design require a rationale such as produced by our model. Perhaps most key, our notion of agent ability (perhaps better stated as lack of agent ability) is tied directly to the semi-byzantine problem solving protocol of McGean. For example, if asked to rationalize the communication (telegraph) line between operators in figure 1, one would be forced to eventually explain both the roles of engineers and operators *and* their limitations as agents in a joint problem-solving system.

5. Related work.

Bond and Gasser note in [Bond and Gasser , 1986] that

> There has been remarkably little work on DAI that addresses automated problem formulation and decomposition. Correspondingly greater effort has been put into flexible task-allocation mechanisms that are used after a problem has been described and decomposed into subproblems (p. 11)

Our work fills a portion of this gap. Our design model can be viewed as an (semi-)automatic programming system for distributed problem solvers. It defines classes of agents or actors [Hewitt, 1991], allocates tasks (responsibility assignments) for those agent classes, and identifies the abilities each agent class needs in order to achieve some overall system goals. Ideally, our model will eventually include standard protocols for changing the abilities of agents [Hudlicka and Lesser, 1987], or dynamically reallocating tasks [Davis and Smith, 1983]. In addition to DAI techniques, we are also investigating theories of cooperative work in institutions, such as "coordination theory" [Malone, 1990].

The output of our model is a formal specification of the agents in a system, and their interaction. Our work is thus complementary to research such as that reported in [de Bakker et. al., 1989], which tries to develop correct distributed programs by correctness-preserving refinement of specifications; our output is a formal specification which might serve as a starting point for further development. Our approach similarly seems to complement research in formal methods in human-computer interaction, such as that described in [Harrison and Thimbleby, 1990]. That work focuses on development of abstract models of human-computer interaction which formally define properties such as consistency and

learnability, and developing interface programs from specifications guided by such models [Runciman, 1990].

Our use of a transformational approach to composite system design extends and formalizes that of [Feather , 1987], which in particular introduced the notion of responsibility assignment. [Dubois, 1990] has provided a formal semantics for responsibility assignment in terms of deontic logic constructs in his ERAE requirements language. The specification language we use to describe "specification states" was strongly influenced by both ERAE and Gist. Our goal formalism also appears largely equivalent to that developed independently by Castro [Castro, 1990] to describe multiagent specifications.

Gasser [1991] has argued that any theory of DAI must explain the role of agent knowledge and communication in coordinated action. The agents and transformations of our model amount to such a theory, although a limited one. Agents in our model cannot introspect [Morgenstern, 1986] about their abilities, nor reason about the knowledge or goals of other agents [Georgeff, 1983], nor about the goals or laws of the system as a whole. Moreover, due to our choice of Petri nets as a formalism, our agents can only react to the current state of the system, and take action according to a pre-enumerated set of rules which do not allow any inference. If a signal fails to set, for instance, an operator in our model cannot infer that another action is necessary, formulate a new plan, and take corrective action. There are undoubtedly design problems which require more powerful design models and representations than ours, but we believe that we can synthesize, analyze, and rationalize interesting composite system designs by using simpler theories expressed in tractable formalisms.

6. References

[Alpern and Schneider, 1985] Alpern, B. and Schneider, F. B., Defining liveness, *Information Processing Letters* **21**, 1985.

[[Anderson & Fickas 1989] Anderson, J., Fickas, S., 1989, A proposed perspective shift: viewing specification design as a planning problem, 5th International Workshop on Software Specification and Design

[de Bakker, et. al., 1989] de Bakker, J. W., de Roever, W.-P., Rozenberg, G. (eds.) *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness* (Lecture Notes in Computer Science 430). Springer-Verlag, 1989.

[Balzer, R., Goldman, N., and Wile, D] Operational specification as the basis for rapid prototyping, *ACM Sigsoft Software Engineering Notes* 7(5) (December 1982).

[Bond and Gasser, 1990] Bond, A. and Gasser, L., An analysis of problems and research in AI, in Bond, A. and Gasser, L. (eds.) *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kauffman, 1988.

[Castro, 1990] Castro, J., Distributed System Specification using a temporal-causal framework (Ph. D. thesis), Imperial College of Science and Technology and Medicine, University of London, Department of Computing, 1990.

[Davis and Smith, 1983] Davis, R. and Smith, R, Negotiation as a metaphor for distributed problem solving, *Artificial Intelligence 20*:63-109, 1983 (reprinted in Bond, A. and Gasser, L., An analysis of problems and research in AI, in Bond, A. and Gasser, L. (eds.) *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kauffman, 1988).

[Dubois and Hagelstein, 1988] Dubois, E., Hagelstein, J., A logic of action for goal-oriented elaboration of requirements, in *Proceedings: 5th International Workshop on Software Specification and Design* (Pittsburgh, Pennsylvania, May 19-20, 1989) In *ACM SIGSOFT Engineering Notes* **14**(3) (May 1989).

[Dubois, 1990] Dubois, E. Supporting an incremental elaboration of requirements for multi-agent systems, *Proceedings of the International Conference on Cooperating Knowledge-Based Systems*, University of Keele (UK), October 3-5, 1990.

[Feather, 1987] Feather, M. S. (1987). Language Support for the Specification and Development of Composite Systems. *ACM Transactions on Programming Languages and Systems*, 9(2), 198-234.

[Fickas, 1985] Fickas S., Automating the transformational development of software, In *IEEE Transactions on Software Engineering*, Vol. 11, No. 11 Nov. 1985

[Fickas and Helm, 1990] Fickas, S., Helm, R., 1990, A transformational approach to composite system specification, TR-90-19, CS Dept., U of Oregon, Eugene, Or., 97403

[Gasser, 1991] Gasser, L., Social conceptions and action: DAI foundations and open system semantics, *Artificial Intelligence Journal* Special Issue on Foundations of Aritificial Intelligence (to appear, January 1991). Also: Research Note 54, University of Southern California Distributed Artificial Intelligence Group. Also: Working notes of the 5th DAI conference, Bandera, Texas, October 1990.

[Georgeff, 1983] Georgeff, M. P., Communication and Interaction in multiagent planning, *Proceedings of the Fourth National Conference on Aritificial Intelligence*, Washington, D. C., 1983.

[Harrison and Thimbleby, 1990] Harrison, M., Thimbleby, H. (eds.). *Formal Methods in Human-Computer Interaction.* Cambridge, Great Britain: University Press, Cambridge, 1990.

[Hewitt, 1991] Hewitt, C., Open information systems semantics for distributed artificial intelligence, *Artificial Intelligence Journal* Special Issue on Foundations of Aritificial Intelligence (to appear, January 1991). Also: Working notes of the 5th DAI conference, Bandera, Texas, October 1990.

[Huber et. al., 1986] Huber, P., Jensen, A., Jepsen, L., Jensen, K., Reachability trees for high-level Petri nets, *Theoretical Computer Science* **45** (1986) 262-292.

[Hudlicka and Lesser, 1987] Hudlicka, E., Lesser, V., Modelling and Diagnosing Problem Solving System Behavior, *IEEE Transactions on Systems, Man, and Cybernetics 17*(3):407-419, 1987. (reprinted in Bond, A. and Gasser, L. (eds.) *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kauffman, 1988).

[van Lamsweerde et. al., 1991] van Lamsweerde, A., Dardenne, A., Delcourt, B., Dubisy, F., The KAOS project: knowledge acquisition in automated specification of software, In Working notes of the 1991 AAAI Spring Symposium on Composite Systems, Stanford University, March 26-28, 1991 (to appear).

[London and Feather, 1982] London, P. and Feather, M., Implementing specification freedoms, *Science of Computer Programming* 2 (1982).

[Maekawa et. al., 1987] Maekawa, M., Oldehoeft, A., Oldehoeft, R. *Operating Systems: Advanced Concepts*. Menlo Park, California: The Benjamin/Cummings Publishing Co., 1987.

[McGean , 1976] McGean, T. *Urban transportation technology*. Lexington, MA: D. C. Heath and Company, 1976.

[Morgenstern, 1986] Morgenstern, L., A first-order theory of planning, knowledge, and action, in Halpern, J. (ed.) *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference* (Monterey, California, March 19-22, 1986). Los Altos, California: Morgan Kauffman Publishers, Inc., 1986.

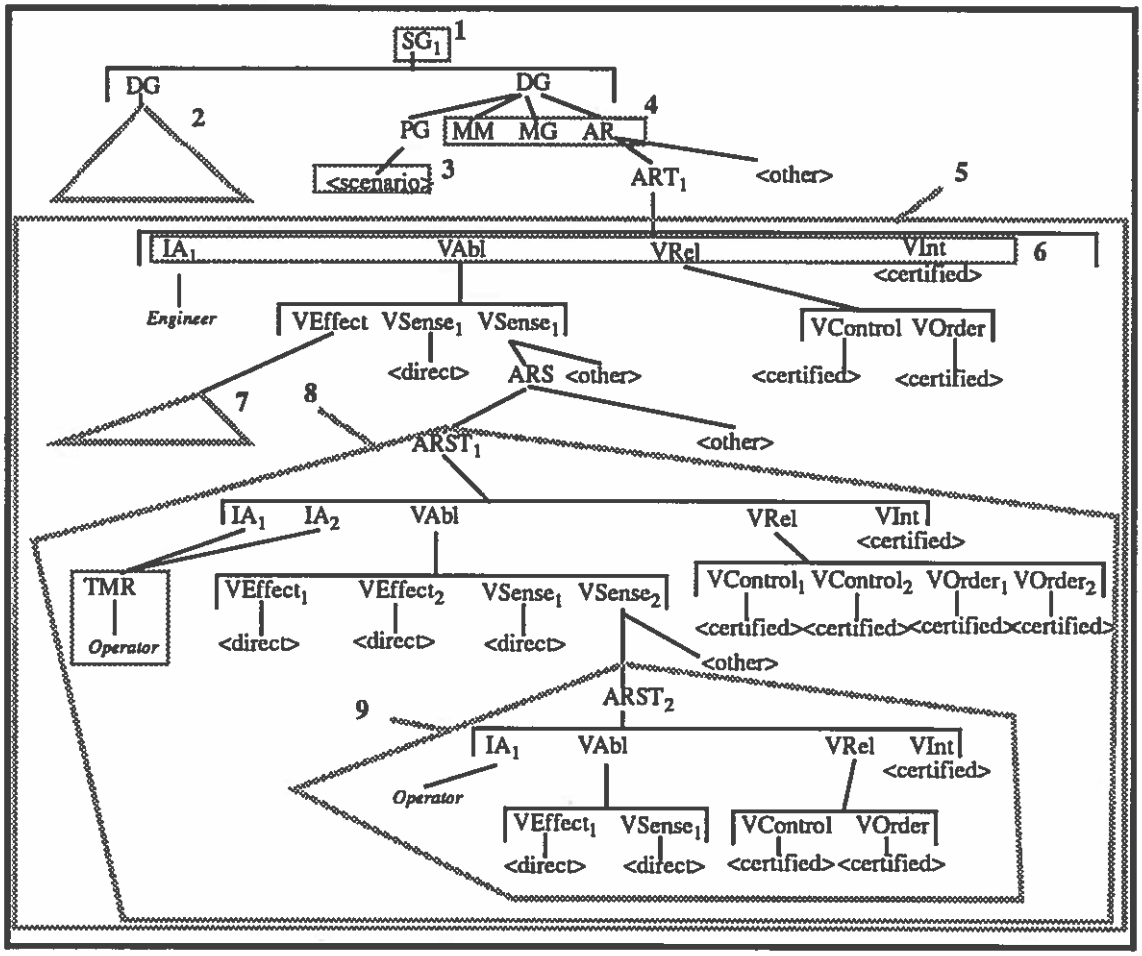
[Reubenstein and Waters, 1989] Reubenstein, H. B.and Waters, R. C., The requirements apprentice: an initial scenario, *Proceedings: 5th International Workshop on Software Specification and Design* (Pittsburgh, Pennsylvania, May 19-20, 1989) In *ACM SIGSOFT Engineering Notes* **14**(3) (May 1989).

[Robinson, 1990] Robinson, W., ,A multi-agent view of requirements, *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, 1990.

[Runciman, 1990] Runciman, C., From abstract models to functional prototypes, in Harrison, M., Thimbleby, H. (eds.). *Formal Methods in Human-Computer Interaction*. Cambridge, Great Britain: University Press, Cambridge, 1990.

[Wilbur-Ham, 1985] Wilbur-Ham, M. C., Numerical petri nets -- a guide. Telecom Australia Research Laboratories, Report 7791.

Appendix A. Partial development state generated by "protect trains" sequence .



1. Satisfy Goal "Protect Trains" (Figure 2).
2. Develop Dispatcher subnet (Figure 3).
3. Generate scenario 2 violating "Protect Trains" goal (section 3.3 on page 9).
4. Standard transformations for addressing a goal: Modify Model, Modify Goals, Assign Responsibility (section 3.1 on page 5).
5. Brinkmanship Transformation (Figure 3).
6. Standard Brinkmanship tasks: Identify agent, verify ability, verify reliability, verify interference (section 3.1 on page 5).
7. Development of the distant signal (section 3. on page 3).
8. Set/reset transformation (Figure 7).
9. Remote/note transformation (Figure 7).