Composite System Design: the Good News and the Bad News.

Martin Feather
Stephen Fickas
Rob Helm

Oregon CIS-TR-91-12
May 8, 1991

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

Abstract

Composite systems are ones comprised of a number of agents interacting to achieve some system-wide goals. Such systems abound in the real world. In particular, it is hard to find a complex software system that cannot be viewed as an agent in a larger composite system. When using the composite system viewpoint during requirements and specification, several benefits accrue:

- *Traceability to system-wide goals*. The system agents and communications among those agents that result from this design process can be traced to the initial system-wide goals, and the design choices made. This provides a formal rationale for the design of 1) each agent, and 2) the inter-agent communication protocol.
- *Thorough exploration of the space of design alternatives*. By beginning with the system-wide goals as the objective of the design process, we do not inadvertently pre-commit to any particular decomposition of those goals among the agents.
- *Basis for redesign*. In the event of the need to make or respond to some change (either to the system-wide goals, to the capabilities of, or communication between, agents of the system, or to the relative cost of design alternatives), the record of the design process will serve as the basis on which to do such redesign.

We have defined a search-based model of composite system design to support the above arguments. The model views design (and redesign) as a search in a space of possible alternative composite system specifications. The model requires both a component to *generate* design alternatives, and a component to *evaluate* those alternatives.

We have tested our model by rationally reconstructing a number of real-world composite systems. The good news is that the generative component appears to be implementable with interactive, domain-*independent*, transformation technology. The bad news is that there is a wealth of domain-specific knowledge that can and should be applied to evaluate composite system designs. Perhaps most ignored, evaluating the impact a new design will have on the environment and the existing agents of the system seems crucial to success. There has been little work done on incorporating the necessary evaluation knowledge into a specification or requirements tool.

Our conclusion is twofold: 1) the composite system design model brings to light important issues that must be addressed in complex software systems, issues that are missed by non-composite or stand-alone models, and 2) while the *generation* problem seems under control, the *evaluation* problem remains open. We project that even informal representations and tools can have some immediate impact on answering the latter problem

1.0 Introduction

We are attempting to develop knowledge-based assistance for requirements engineering. In particular, we wish to support the production of formal, operational specifications of multi-agent architectures from formal statements of requirements. We would like automated tools to help the analyst to analyze a set of requirements and identify what agents are necessary, what capabilities each agent must have to fulfill its role in the overall system, the inter-agent protocol that allows agents to cooperate, and finally, the interface each agent must have to be an active participant in the system.

Example problems we have studied from a multi-agent perspective include elevator systems, trains systems, air traffic control, libraries, power plant control, e-mail systems, and meeting management systems. In the remainder of this section, we describe an approach, called Composite System Design (CSD), that helps build specifications for such systems. We argue that CSD offers several advantages over existing approaches to requirements engineering. In sections 3 and 4 we discuss two major research questions which must be addressed before we could incorporate CSD into a knowledge-based tool for specifying and rationalizing composite systems:

1. Is CSD *sufficient* to generate designs for realistic problems?
2. Can CSD be made *tractable* for realistic problems?

We have investigated these in some detail. In particular, we have a) attempted to apply CSD to a problems in a broad variety of domains, in order to test its sufficiency, and b) looked at the type of practical knowledge necessary to evaluate and guide the process of responsibility assignment that lies at the heart of CSD. Both of these are actually preliminary steps to answering both questions 1 and 2. In particular, while we have formalized some pieces of CSD, and automated still smaller pieces, we view our work to date as a feasibility study. This paper reports on this study, and concludes with a future work section in which we briefly summarize the principal research problems we see remaining.

1.1 Composite system design

Composite systems are multi-agent systems. They comprise several agents interacting so as to achieve some system-wide goals or functional requirements. In our experience, composite systems are numerous - it is more difficult to think of systems that are not composite than those that are. The composite system design approach has been proposed by Feather as a means to capture and deal with very early stages of the design of such systems [Feather, 1987a]. The essence of his composite system design approach is to do the design (or redesign) of composite systems by beginning from a description of the properties desired of the system as a whole, and then deriving the behaviors of, and interactions between, the agents so that their combination will achieve the desired system properties. The final system may include pre-existing agents whose properties cannot be changed, and newly created agents defined just for the task at hand.

Typical systems will be a mixture of human, software, and hardware agents. Taking the standard elevator problem as an example, we might identify elevator controller agents, passenger agents, maintenance agents, etc. In essence, CSD allows us to explore the entire space of designs that might satisfy the system requirements. Choosing among alternative implementations of these designs will lead us to fully automated elevators, fully manual elevators, and many specifications in between.

The starting point for this approach is an initial specification describing system-wide goals (constraints), e.g., "move passengers to their destinations", and capabilities of the pre-existing agents of the system, e.g., "an elevator can move to an adjacent floor; a passenger at a floor can enter an open-doored elevator at that same floor". These amount to the functional requirements of the system. The design process proceeds by incrementally assigning goals as the *responsibility* of subsets of agents: only those agents responsible for a goal are expected to limit their own behavior to ensure satisfaction of that goal (e.g., if the elevator system alone is responsible for keeping passengers from falling down elevator shafts, then the elevator must keep doors closed when necessary rather than rely upon passengers to limit their choice of when to walk through an open doorway. This design process ends when all goals have been subdivided and assigned as the responsibility of individual agents, at which point those agents can be independently implemented, assured that their combination will achieve the system-wide goals of the composite system to which they belong. It is possible that they in turn could be smaller composite systems, and that this design process be recursively applied. For instance, an entire elevator composite system may be a single agent of a larger transportation composite system, e.g., within a train station or airport.

During the course of design, the interaction between agents - communication and control - is established. Such interactions may necessitate the introduction of, say, communication media, protocols for communication, and even further agents to facilitate communication.The introduction of these is motivated and rationalized in terms of the overall design process.

Figure 1 Portion of an elevator design history.

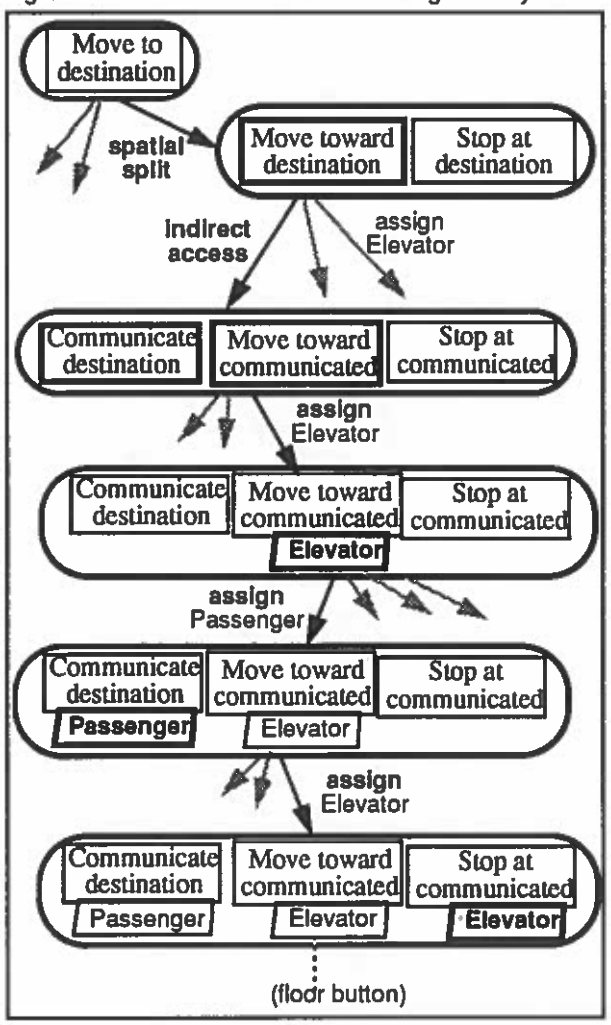


Figure 1 (adapted from [Fickas&Helm, 1990]) shows part of the end product of CSD applied to the elevator problem. Starting from a system goal ("move passengers to their destinations"), we derive the responsibility of each passenger agent to notify the elevator of its destination, while the elevator receives the responsibility to move to that destination when it is known. Along the way, we derive the need for an interface component (implemented by a button) which allows the passenger to communicate a destination floor to the elevator.

Different responsibility assignments lead to radically different systems. For instance, express elevators, scheduled elevators, reserved elevators, prison elevators, freight elevators, etc., all can be generated by exploring alternative responsibility assignments.

We see the following advantages of CSD:

- *Traceability to system-wide goals.* The system agents and communications among those agents that result from this design process can be traced to the initial system-wide goals, and the design choices made. This provides a formal rationale for the design. In contrast, a description of only the end product of composite system design, namely the behaviors of the individual agents, would be much less perspicuous from the point of view of understanding.
- *Thorough exploration of the space of design alternatives.* By beginning with the system-wide goals as the objective of the design process, we do not inadvertently pre-commit to any particular decomposition of those goals among the agents. This maximizes the likelihood that we do not overlook alternative, perhaps superior, solutions.
- *Basis for redesign.* In the event of the need to make or respond to some change (either to the system-wide goals, to the capabilities of, or communication between, agents of the system, or to the relative cost of design alternatives), the record of the design process will serve as the basis on which to do such redesign. Without such a record, it would be hard to ascertain how to do this in a principled manner.

We believe this last point is particularly crucial. As an example, consider developing the requirements for the elevator controller of [IWSSD, 1987]. If we treat the controller in isolation from its environment, we cannot formally explain the need for any feature of the elevator, such as the presence of doors or the use of a demand-driven service protocol. Without such rationale, it is difficult to formally prove we can eliminate a given button as an economy measure, or that we should use it in a new elevator installation in another building. Finally, ignoring the high-level goals of passengers and other agents in the elevator system makes it difficult to describe or evaluate innovative designs which new information technology may make feasible, such as an elevator which predicted the arrival floor of passengers, or one which took voice reservations over a cellular phone. Because of these limitations, we argue that specification approaches which currently take a singe-agent, stand-alone view to what are, in reality, composite sys-

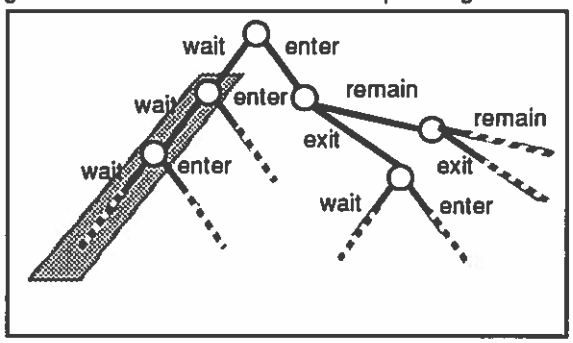
tems, can benefit from the broader perspective of CSD.

2.0 Searching the CSD space

We formalize the composite system design process by treating it as a search through the space of possible designs [Fickas&Helm, 1990]. The "states" of this space are designs or specifications; the "operators" are transformations or "methods" [Fickas, 1985] which map from one design state to the next. Figure 1 shows part of the state space for the elevator example discussed above. A key question is: can the CSD approach generate the design space for realistic problems?

We begin by presenting the components of the CSD search space. Each design state in the space has two components: a generative part that produces all alternative behaviors considered possible; a constraining part that states which of these possible behaviors are valid. As an example, figure 2 shows a portion of the behavior tree (an OR-tree) produced by the generative portion of a naive elevator design.

Figure 2 Behavior of an elevator passenger.



Assuming a stopped elevator with open doors, at any point a passenger outside that elevator can choose (nondeterministically) between waiting outside or entering; likewise, a passenger inside that elevator can choose between remaining inside or exiting. If we add a constraint that passengers eventually enter the elevator, then the shaded behavior must be pruned during design.

The formal representation of the generative and constraining portions of our model is as follows:

- The generative part, denoting the possible behavior produced by *agents* in the system, takes the form of a discrete event language that can be viewed, alternatively, as a subset of Gist [London&Feather, 1986] or a high level Petri Net [Wilbur-Ham, 1985], [Huber et. al., 1986].
- The goal/constraint language is a form of temporal logic, roughly similar to that of that of the ERAE language [Dubois&Hagelstein, 1988] and of the distributed-action logic of [Castro, 1990].

During the CSD process, we periodically examine the current design state to determine whether the generative part could produce behaviors which violate the constraining part; in other words, whether all possible interactions of agents in the system meet that system's functional requirements. This is necessary to a) recognize potential "solution states" i. e. acceptable designs and b) select design moves to make if the current state is not acceptable. To verify that a goal or constraint is met by the generative part, we have built two analysis tools[1]:

1. A planner or scenario generator called OPIE [Anderson&Fickas, 1989]. OPIE can be used in two ways: 1) to disprove a constraint by producing a disallowed behavior (i.e., counter-planning), and 2) to prove an existence goal, e.g., there exists at least one behavior that satisfies some predicate.
2. A reachability-graph (RG) tool. The tool first produces a reachability graph from a static analysis of the generative part, and then allows queries about reachable states. As with OPIE, these queries can be used to disprove a constraint or prove existence goals. However, unlike OPIE, the graph can be used, in conjunction with omega values, to disprove temporal goals such as "trains will eventually reach their destination".

As an example of a disproof that either tool could produce, but which we present in OPIE style for readability, consider the specification of figure 3, with behavior shown in petri-net form, and the ProtectTrains goal shown immediately below. Note that figure 3 is a starting model. It specifies the environment in which we will design a train system, but does not yet specify any of the agents we will need to meet the goals. In [Fickas&Helm, 1990], we follow the elaboration of this naive model into its final, complex, composite form.

1. We see a continuing need for both tools: OPIE provides efficiency through goal-directed search and abstract planning; the RG tool can be costly to run, but provides more powerful forms of analysis.

OPIE is called to disprove the specification, given certain initial conditions:

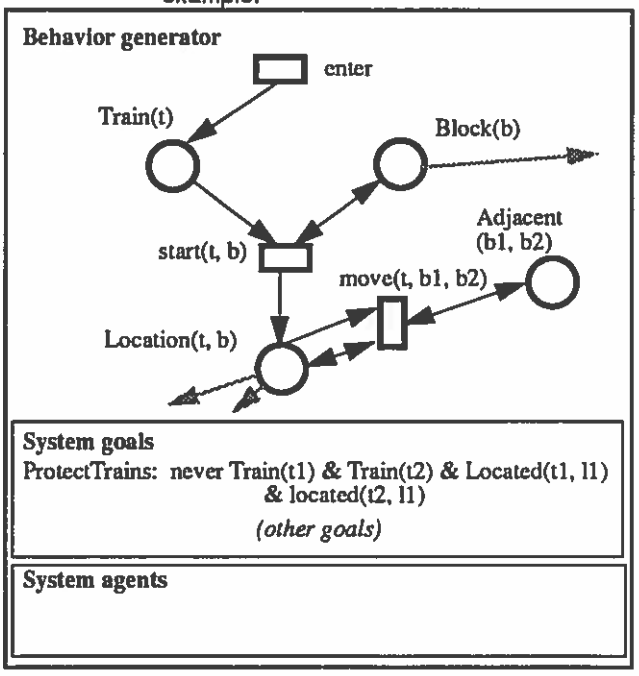
(U): **Disprove** ProtectTrains **in** SYS0
given Block(b1) & Block(b2)
& Adjacent(b1, b2)

OPIE generates a scenario (plan) in which two trains end up at the same location.

(O): The goal ProtectTrains in SYS0 is violated by scenario S1:
1. given Block (b1)
2. given Block (b2)
3. given Adjacent(b1, b2)
4. enter => Train(t1)
5. start(t1, b1) => Location(t1, b1)
6. move(Location(t1, b1), b2) & Adjacent(b1, b2) => Location(t1, b2)
7. enter => Train(t2)
8. start(t2, b1) => Location(t2, b1)
9. move(Location(t2, b1), b2) & Adjacent(b1, b2) => Location(t2, b2)

Violation: ProtectTrains in SYS0

Figure 3 Specification (SYS0) for train protection example.



We might apply several design operators or methods to address this negative scenario. We could, for example, introduce an agent (an engineer, for instance) with responsibility for the ProtectTrains goal. However, we could also modify the environment (such as the adjacency relation on blocks), or even the ProtectTrains goal itself. This highlights an important difference between our CSD approach and that of Feather's original CSD [Feather, 1987a]. In particular, Feather was concerned with specification *implementation* - given the correct set of goals and the correct set of agents, find a division of responsibility between them which maintains correctness. Thus, Feather would support only one type of design action here: the restriction of one or more existing agents' behavior to prune the crash scenario. Our use of CSD, on the other hand, is concerned with specification *design*: given an initial set of goals and an existing environment, attempt to assign responsibility. If this fails (because the initial model was incomplete, because the system is unimplementable, because it is too costly), modify the goals, the environment (including agents), or both. The major ramification of this is that we allow goal modifications (e.g., weakening a goal), environment modifications (e.g., change the existing infrastructure, create new agents) as well as responsibility assignment. Feather takes up some of these broader concerns in [Feather, 1987b].

3.0 Generating designs

We next discuss the methods which move among design states. We argue that a relatively small number of domain-independent methods can account for an interesting class of composite system design problems.

3.1 Planes, trains and automobiles

We have attempted to build a design generator based on the CSD model. The approach we have taken is to use transformations as design operators, i.e., as the actions that produce new states/specifications in our state-based search model. Our goal was to define a tractable set of transformations that take goals and agents as input, and produce restricted agent actions as output (the essence of responsibility assignment). We expected these transformations to be interactive, relying on the human specifier to do the complex reasoning sometimes necessary to determine agent action and control. Our success criteria involves a comparison with other interactive, assistant-based design systems (e.g., [Fickas, 1985], [Reubenstein&Waters, 1989]): if we could obtain the same mixture of human/machine interaction as these systems, we would judge our results as a success.

To test our ideas, we looked at several domains. First, we attempted to redesign Feather's elevator system (as

reported in [Feather, 1987a]) using a transformational approach. The result, as discussed in [Fickas&Helm, 1990], was that we were able to rederive the elevator with a relatively small number of transformations.

Our other major effort has been to work our way towards a CSD model of Air Traffic Control (ATC). Preliminary to this, we have studied several simpler problems: traffic intersections as a CSD problem; rail-highway crossings as a CSD problem; train/subway control as a CSD problem. Our general approach has been to rationally reconstruct problems from the real world, i.e., we attempted reconstruct systems that exist now or existed in the past.

We have chosen these example problems for several reasons: 1) each has some of the same multi-agent protocol problems as the more complex ATC composite system, but without the attendant jargon and plethora of details, 2) train failures (like ATC failures) are well documented [Shaw, 1961], 3) evaluation functions have the same complex nature, and 4) transportation systems, in general, share a property that is endemic to many other real world composite systems - any new system design must work its way into the existing infrastructure (i.e., a revolutionary approach[2] is not a practical specification development philosophy in most composite system domains). The last two points, evaluation functions and the need to fit a new design into an elaborate existing infrastructure, is taken up in section 4.

3.2 Transformation examples

We could best illustrate our results from these design studies by following several designs through in detail, pointing out how a small number of transformations were used over and over[3]. However, the limited space in this paper will not allow us to do that. Instead, we will briefly describe several of the key transformations used repeatedly in the designs we have produced.

Brinkmanship. This is a standard constraint satisfaction technique. It matches on a conjunctive system constraint, and identifies the brink, i.e., the actions that, if allowed to happen, will push the constraint over the edge. As an example, suppose we have the following constraint taken from figure 3:

never: `train(T1) & train(T2) & located(T1, L1) & located(T2, L1)`

The effect of the brinkmanship transformation here would be 1) to identify actions that change a train's location (e.g., a move action), 2) to add a control component (e.g., make train movement a controlled action), and 3) to set up a sub-task to assign an agent to be the controller[4]. Thus, brinkmanship acts as a "jittering" transformation, one that sets up a goal for subsequent responsibility assignment. Looking at Feather's original development of the elevator, one can see that much of the work is involved in exactly this type of goal-jittering process. For example, Feather introduces a goal/constraint that a passenger must not be in an elevator moving the 'wrong way'. Application of brinkmanship transforms this to a control problem: prevent the passenger's entry into the elevator. Once this goal-jittering step is carried out, control (responsibility) can be assigned to an agent.

Spatial-split. This introduces a standard, multi-agent problem solving protocol. It breaks goal responsibility into two spatially-disjoint pieces[5]. A separate agent is assigned to each piece, with responsibility shifting from one agent to the next. Looking back at the brinkmanship problem in the train example, there are actually two actions that we must worry about: trains already under control of the rail line system moving into the same location, and trains entering control of the rail line system (say, from a holding yard). This eventually leads to two actions to control. The spatial-split transformation would suggest that responsibility be split with one agent (i.e., a dispatcher) assigned to entry or "train-in-yard" and a second agent (i.e., an engineer) assigned to movement or "train on line"[6]. As a side-note, there is a drawback that goes along with most of the split-responsibility transformations - a clear handoff protocol must be agreed upon among agents, and inter-agent

2. *Revolutionary* = design from scratch (also known as greenfield). *Evolutionary* = work a design in to an existing infrastructure, e.g., existing physical structures, existing laws, existing standards, existing work practices, etc. (also known as brownfield).
3. This is exactly the style used in [Fickas&Helm, 1990] - a lengthy composite system specification design is presented along with the transformations employed.
4. Clearly there are other strategies/transformations that are possible, e.g., disallow two trains in the system at the same time, make the set of locations of two trains mutually exclusive (e.g., provide two sets of disjoint tracks). These, along with the brinkmanship strategy, are encoded in domain-*independent* terms.
5. Non-spatial splits are also possible, e.g., split by property, split by time, etc.
6. We also have a *single-assignment* transformation that would make a single agent responsible for the entire constraint, i.e., control of both actions.

communication must be reliable. Looking at documentation of train crashes, one finds an alarming rate of train accidents caused by handoff failures [Shaw,1961].

Indirect access. This introduces a standard, multi-agent problem solving protocol, which calls on an agent A to signal another agent B as to the state S of the system. Typically, S is something that B must know to act responsibly, but B does not have direct access to S. In these case, A is asked (given the responsibility) to sense S[7] and pass the information along to B. Such inter-agent communication is ubiquitous in transportation systems, e.g., elevators letting passengers know what direction they are heading, station operators signalling trains when the track ahead is clear, air traffic controllers warning a plane that its landing gear has not properly deployed.

Responsibility accumulation. We may assign multiple responsibilities to the same agent. The transformation action is to merge in new responsibilities to those already existing in an agent. The motivation for this transformation is obvious when looking at more complex systems: agents typically play multiple-roles. For instance, train engineers share responsibility for both train progress and train safety goals. Of course, that there is the potential for conflict here is well documented in case studies of train crashes [Shaw, 1961]. In general, responsibility overloading achieves cost savings at the expense of decreased reliability.

The results of our experiments were positive - we were able to identify a set of domain-independent transformations that sufficed to generate a number of real-world designs in each of several transportation domains. At the heart of this set lay a small core of transformations that account for the vast majority of design steps - this core comprises the transformations that we described above. Beyond this core set, we identified transformations that are needed to add components to the design that address agent reliability and motivation issues. We briefly discuss these issues further in section 4.3; the interested reader is referred to [Fickas&Helm, 1990] for a more thorough discussion.

7. Variations exist where A simply "knows" S - there is no need for a separate sensing operation. For example, a passenger A knows its destination; most elevator systems give A the responsibility of providing that information to the elevator B so it can carry out its responsibility of getting the passenger to their destinations.

4.0 Evaluating composite designs

We argued in the last section that a relatively small number of domain-independent transformations can account for an interesting class of composite system design problems. Thus, if a problem from any domain can be viewed in a composite system light, our domain-independent transformations could produce the space of designs which covers that problem. It does not follow, however, that we have no need of domain knowledge in designing real systems. Viewing CSD as a search process, we need a form of "heuristic function" which can evaluate the designs in the search space and select those which will lead to implementable, reliable, safe, cost-effective systems.

As an example of the kind of evaluation problem which arises in CSD, reconsider the ProtectTrains constraint in figure 3. We could (and did) use our transformations to produce the following two alternative responsibility assignments[8]:

1. Assign ProtectTrains to dispatchers and engineers.
2. Assign ProtectTrains to dispatchers, engineers, and station operators (who controlled track-clear signals).

Both of these showed up in actual train management systems. The first describes pre-1850 train management. It relied on clever scheduling and line-of-sight by engineers to avoid crashes. The second describes most post-1850 train management systems, which employ the notion of protected blocks of track. This is the same basic design that is used today, with computers replacing or augmenting the human agents of the pre-computer era.

Given these two choices 140 years ago, which would we have made? The first leads to a large number of accidents. When the second was introduced, a dramatic drop in accidents ensued. Hence, the second seems the best choice. However, the first remained in place long after 1850 on some lines - it appears that the expense of erecting stations, signals and telegraph lines along with paying the station operators outweighed the cost of accidents. In summary, domain-dependent measures of cost and safety are used to choose among the alternatives generated by domain-independent transformations

8. Of course, we could also generate all other combinations possible.

In this section we look at the form evaluation knowledge takes in the transportation problems that we have studied. In reviewing the literature on our example domains, we have identified *metrics* or criteria to evaluate specification alternatives, and *models* to compute values for those metrics for a given specification. While many metrics and models we have identified are domain-specific and even problem-specific, we also believe we have also located a set which address issues across multiple domains.

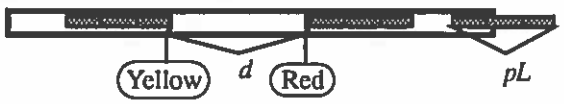
4.1 Revolutionary (greenfield) models

Each domain has a collection of specific metrics and models to evaluate a specified component *in isolation*, i.e., as a greenfield system. Figure 4 gives a collection of these for the evaluation of passenger rail system protocols and layouts from [McGean, 1976] [Anderson, 1978]. They include:

- Analytic models for estimating capacity.
- Statistical models to predict delays due to component failures.
- Guidelines for selecting station arrangements.

Figure 4 Evaluating rail and crossing designs.

Capacity of two-block permissive signalling protocol



capacity = 1/h
h = Kv0/d + pL/v0

Accident prediction factors (excerpt)

Category	Constant.	Exposure
Walkway	0.002268	$((c * t * 0.2) / 0.2)^{0.3334}$
Flashing	0.003646	$((c * t * 0.2) / 0.2)^{0.2953}$
Gates	0.001088	$((c * t * 0.2) / 0.2)^{0.3116}$

Pedestrian conveyor performance

System	Cap.	Width	Speed
Walkway	600	(per ft.)	250
	1500	(per ft.)	crush flow
Escalator	3750	2	90
	5025	2	120
Moving walk	3600	2	120
Stairs	1000-1200	(per ft.)	crush flow

Similar metrics and models can be found in textbooks on a given domain, such as for rail-highway crossing signals [Taggart et. al., 1987]. [Tong, 1989] and [Kant, 1985] discuss integration of similar "nonfunctional requirements" into the design of VLSI and algorithms, respectively. We could apply some of the more formal models to the specification directly. For the less formal models, or those which have highly specialized inputs, it will be necessary to rely on a domain expert to map from the artifact's abstract specification into the terms required by a given model. In addition, many domain-specific evaluation models make assumptions which constrain the implementation of specification constructs. A model for the operating cost of a train signal, for instance, would necessarily identify the technology used to implement that signal. Thus, it remains an open question how and when to apply a given evaluation model in the CSD process.

4.2 Evolutionary (brownfield) models

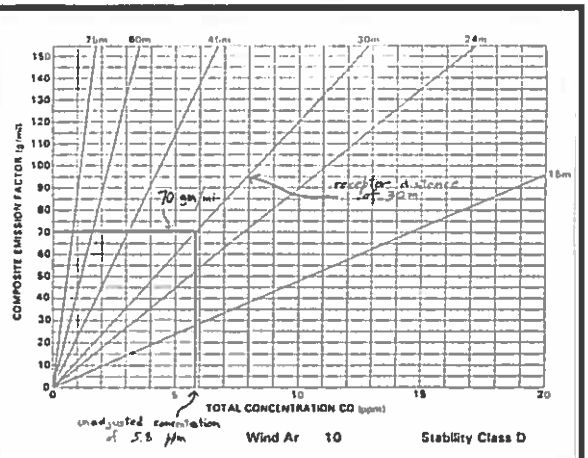
In the domains we have studied, brownfield constraints prune out many otherwise plausible designs. A principle brownfield constraint is the *operating impact* the artifact may have on its environment. Suppose that in specifying an elevator system we apply a "split" transformation which divides responsibility among passengers and elevators according to the distance they wish to travel. Passengers will be solely responsible for reaching their destination (via stairs) when they want to go up or down only one floor, while elevators and passengers will be jointly responsible when the distance to travel is greater. Greenfield metrics for the elevator controller specification, such as cost, performance, and maintainability, might make this a superior design in most contexts. This protocol, however, is ruled out for new elevators by local, state, and federal requirements for handicapped access [ANSI, 1987].

Figure 5 shows a selection of models for evaluating the impact of rail-highway crossing alternatives on their environment. These include graphical models for computing carbon monoxide emissions, guidelines for evaluating aesthetic impacts, and checklists of special concerns such as emergency vehicle and school bus movements. In general, to genuinely evaluate specification alternatives, we have to consider the written codes and guidelines which have developed in the domain to limit the disruption a system creates in its environment.

In addition, the specification process of large-scale composite systems must consider the *transition impact* from standards which may exist in the given domain. Agents,

whether software, hardware, or human, must be programmed, designed, or trained to carry out a given set of responsibilities. In a system where a large group of agents have similar responsibilities, it is typically costly to alter these responsibilities. In the rail-highway crossing, for instance, we cannot easily change the protocol by which cars cross railroad tracks "at-grade" (on the same level as the track). It is clearly not feasible to train all drivers to carry out their part of a radically different protocol, (such as not using the crossing when trains are scheduled), even if the protocol were a major improvement. Even altering the format or behavior of the lights, gates, and gongs at crossings would probably entail unreasonable expense in the short term, since it would require selecting custom hardware and software over the cheaper, mass-produced equipment available to support the standard design.

Figure 5 Operating impact models.



Aesthetic impacts
Insignificant: The proposed improvement would create visual characteristics similar to those that currently exist in the landscape.
Low: The improvement would introduce additional visual characteristics into the landscape that would be evident but would not necessarily attract attention.
Moderate: The improvement would introduce visual characteristics that would be noticeably different from existing visual elements.
High: The improvement would visually dominate the landscape and would cause substantial change in the visual character of the landscape.

School bus safety indicators
1. # of school bus crossings
2. # of students on the bus during crossing
3. # Train operations during crossing times
4. % of crossings by buses at-grade
5. # Historical Bridge/Bus Incidents
6. Alternative route delay
7. # High hazard locations on alternative routes

4.3 Irresponsible behavior: a recurring issue

One key issue that recurs across domains is that of *reliability*. In many cases, an agent in a system can be assigned a responsibility and engineered or trained to carry it out. In the systems we have examined, however, this is no guarantee that the agent will act "responsibly" in all cases. In essence, irresponsible behavior by an agent A is simply behavior that does not meet the responsibility assigned to A. This can be failing to act when necessary, acting when unnecessary, or acting when necessary but in an incorrect fashion. Each of these can lead to problems in achieving system-wide goals.

Typically, designs incorporate features which can only be explained as "fault-tolerant" design components to account for irresponsible agents. For instance, one sometimes sees lowerable barriers along with flashing lights and gongs at highway-rail crossings. These barriers are superfluous if driver agents act responsibly, i.e., stop when signalled to do so. However, drivers do not always carry out their responsibility, and barriers act as a back-up mechanism (although clearly not a fool-proof one). Similarly, modern train systems have retained flagmen who are required to walk back behind a train which makes an unscheduled stop. This is required even though current "block signalling" protocols, *when properly executed*, ensure that no more than a single train can be on the blocked track. It is not possible to explain the existence of either flagmen or barriers if agents are either totally capable or totally incapable of carrying out a given responsibility. Clearly, some mechanisms are needed in our model to predict the failure modes of agents, to evaluate the reliability of a specification given those modes, and possibly to mitigate failures by designing in redundancy or recovery mechanisms.

Similarly, a performance issue we term *interference* seems to explain many features we observe in real composite system designs. An agent may well be responsible for a given goal, but other commitments, either within the system (see *responsibility accumulation* from section 3.2), or outside of it, can interfere with execution. In manual train switching systems, for instance, operators of warning signals have caused collisions when their other assigned duties -- reporting to neighboring stations, transferring train orders, recording train passages -- left them with insufficient time to perform signalling. In general, agents in a realistic system may have to play multiple roles[9], some which may conflict with one another. Evaluation of composite system specifica-

tions requires that such conflicts be detected and minimized, by methods such as prioritization, further splitting of responsibility, or degraded service modes.

4.4 Compromised designs

Composite system designs are typically compromises among multiple, conflicting criteria. Rail scheduling protocols, for example, are typically not maximally safe, nor do they provide maximal capacity at given cost. Rather, they typically aim for an adequate compromise among these concerns. Further, the judgement of what is "adequate" may shift over time. The rail scheduling protocols of this century stress safety over capacity to a far greater degree than was common in the 1800's, resulting in a significantly better safety record [Shaw, 1961].

Clearly there is a technical problem in combining metrics which measure very different attributes. For example, metrics which influenced train signalling protocols include the cost of instantaneous communication and automated sensing (presumably in dollars), potential for collisions (in dollars and lives per vehicle-mile), and the on-time performance of trains (in minutes). Analytic methods for decision-making on multiple criteria, such as the multi-criteria simplex method [Zeleny, 1982] typically require that such "incommensurate criteria" be normalized to a compatible scale, a task which at present requires the expertise of a domain analyst.

Specifications of composite systems complicate selection further by introducing negotiation issues. The people involved in a large-scale composite system, whether as designers or as participants, will rarely agree on a common weighting of the metrics involved in selecting among alternative specifications. Representatives of rail passengers, for example, placed pressure on the railways of the time to incorporate signals and telegraph into operating procedures, despite company resistance to untried technology and capital outlay [Shaw, 1961]. These passengers placed different weights on the profit and safety metrics than the railroad operating companies. Similarly, the parties to rail-highway crossing specification listed in [Taggart et. al., 1987] include the designing agency itself (typically the state highway authority), railroad companies, local planning authorities, neighboring property owners, emergency service providers, and local school districts. The authors of the rail-highway text apparently despair at arriving at a compromise through analytic methods; instead, they discuss strategies for presenting evaluation data to the parties in such a way as to promote agreement.

5.0 Summary and future work

We return to our state-based search perspective to discuss the results of our study of CSD. There are two search components to consider: 1) the design operators that generate new states, and 2), the evaluation heuristics that guide the search to acceptable solutions (composite system specifications). Our design operators take the form of transformations on goals and agents. We have been able to produce a small number of powerful transformations that apply across the transportation problems that we have studied, and which could serve as the basis of an interactive design assistant. These transformations are interactive because we lack the formal analytic models (e.g., theorem provers) to guarantee the responsibility-assignment operation of CSD. We have partially plugged this analysis gap with tools like OPIE and the RG tool. We also continue to work on automating our transformations, gradually making them less dependent on human intervention.

We believe we have had mixed results on evaluation heuristics. On the positive side, the composite system view has forced us to address difficult issues, ones that we and others have missed or ignored by taking a standalone view of specification. In particular, by studying small but *realistic* composite system problems, the issues of revolutionary and evolutionary models is raised. Further, the need for models of compromise and negotiation during system design is made clear by CSD.

On the negative side, there has been little work in the requirements and specification field to address these issues. One immediate gain would be to integrate existing evaluation models into a tool based on CSD. This is a task we have taken on in a tool we call Critter [Fickas&Helm, 1990]. Our initial goal is to *informally* catalog the type of models discussed in section 4. Of course, the actual integration of these models into an *automated* search-based design tool is a difficult task, indeed - it requires mapping between multiple ontologies at various levels of formalism. While Farley&Liu have shown that this is possible in non-composite domains [Farley&Liu, 1990], we believe much hard work lies ahead to scale their results up.

9. As a further complication, composite systems are often intertwined in complex ways, leading to the same agent playing a role in two or more separate systems with *differing* responsibilities in each.

We are more encouraged by a small but growing field of researchers interested in conflict and compromise as part of the software engineering process. One can see a sample of this work, and a comprehensive survey, in [Robinson, 1990].

Finally, AI researchers have also been concerned with reasoning in difficult domains, ones that lack any tractable first-order models. We have investigated two AI techniques as possible formalisms for search heuristics in CSD. The first uses a case-based approach to represent "experience" in a domain. As an example, we built a case-based critic that incorporated a set of standard library problems that crop up in a university library [Fickas&Nagarajan, 1988]. In some sense, these cases replace or compile a first order theory of the human behavior of the patrons of a library. The transportation field also seems to rely at least partially on a case-based approach to system failure [Shaw, 1961].

The second technique we have studied is qualitative reasoning (QR). Here, we attempt to model a system at a gross level of detail, either because we don't know the lower level details or they are not of interest. We have constructed and tested a qualitative reasoning tool for the library domain [Downing&Fickas, 1991]. The transportation domain also seems well suited to a QR approach [McGean, 1976].

6.0 Acknowledgments

Feather has been supported in part by Defense Advanced Research Projects Agency grant No. NCC-2-520, and in part by Rome Air Development Center contract No. F30602-85-C-0221 and F30602-89-C-0103. All three authors have been supported by NSF grant No. CCR-8804085. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official opinion or policy of DARPA, RADC, NSF, the U.S. Government, or any other person or agency connected with them.

7.0 References

[Anderson, 1978] Anderson, J. E., *Transit Systems Theory*. Lexington, MA: D. C. Heath and Company, 1978.

[Anderson&Fickas, 1989] Anderson, J. S. & Fickas, S., A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem. *Proceedings: Fifth International Workshop on Software Specification and Design* (Pittsburgh, PA). ACM SIGSOFT Engineering Notes, Volume 14, Number 3 (May 1989).

[ANSI, 1987] ANSI national standard safety code for elevators and escalators, ANSI/ASME A17.1-1987.

[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.

[Downing&Fickas, 1991] Downing, K., Fickas, S., Specification criticism using a qualitative reasoning model, *Proceedings of the Sixth International Workshop on Software Specification and Design* (Lake Como, Italy), 1991. Also available as TR-90-12, CS Dept., U of Oregon, Eugene, Or., 97403

[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).

[Farley&Liu, 1990] Farley, A. M., Liu, Z. Y, Shifting Ontological Perspectives in Reasoning about Physical Systems, *Proceedings of the 1990 AAAI Conference*, Boston.

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

[Feather, 1987b] Feather, M.S. The evolution of composite system specifications, *Proceedings of the 4th International Workshop on Software Specification and Design*, Monterey, California (USA), April 3-4, 1987.

[Fickas, 1985] Fickas, S., Automating the Transformational Development of Software. *IEEE Transactions of Software Engineering*, 11(11), 1268-1277.

[Fickas&Nagarajan, 1988] Fickas, S., Nagarajan, P., Being suspicious: critiquing problem specifications, In *Proceedings of the 1988 AAAI Conference*, Minneapolis.

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

[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.

[IWSSD, 1987] *Proceedings of the 4th International Workshop on Software Specification and Design*, Monterey, California, April 3-4, 1987, IEEE Computer Society Press.

[Kant, 1985] Kant, E., On the efficient synthesis of efficient programs, in Rich, C. and Waters, R. (eds.), *Readings in Artificial Intelligence and Software Engineering*, 285-305, Morgan Kaufmann, 1986. Originally in *Artificial Intelligence* **20**, 1983, 253-306.

[London&Feather, 1986] London, P.E., and Feather,M.S., Implementing specification freedoms, In Rich, C. and Waters, R. (eds.), *Readings in Artificial Intelligence and Software Engineering*, 285-305, Morgan Kaufmann, 1986. Originally in *Science of Computer Programming* 2 (1982) 91-131.

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

[Reubenstein&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.

[Shaw, 1961] Shaw, Robert B. *Down Brakes: a History of Railroad Accidents, Safety Precautions, and Operating Practices in the United States of America*. London, United Kingdom: P. R. Macmillan Limited, 1961.

[Taggart et. al., 1987] Taggart, R. C., Lauria, P., Groat, G., Rees, C., Brick-Turin, A. (1987), "Evaluating grade-separated rail and highway crossing alternatives", National Cooperative Highway Research Program Report 288, Transportation Research Board, National Research Council, June 1987.

[Tong, 1989] Tong, C., Makin routine design possible and robust: research directions, Technical report WP-TR-140, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903.

[Wilbur-Ham, 1985] Wilbur-Ham, M. C., Numerical Petri Nets - A Guide. Telecom Australia Research Laboratories, Report 7791.

[Zeleny, 1982] Zeleny, Milan, *Multiple Criteria Decision Making*. New York: McGraw Hill, 1982.