Supporting the Design of Manageable Network Applications: a Proposal

Stephen Fickas

Oregon_CIS-TR-92-08
April 8, 1992

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

1.0 Introduction

We are interested in Software Development Environments (SDE) for building networked applications. We have built and experimented with several versions of an SDE based on interactive, incremental design. The results of this work, for the most part, have been encouraging. However, a major deficiency emerged as we began to tackle more complex network applications: we were not addressing the manageability of the systems we were constructing. In particular, modern network applications must be designed for a high degree of robustness and adaptability. One way to provide these traits is to remotely monitor, control, and manage each application using network management resources such as SNMP [Rose, 1991]. The current practice, and the one employed by us until now, is to build management components into applications as an afterthought. In this proposal, we argue for a more principled approach - consider management needs from the earliest points in design.

Our approach will be to use a knowledge-based design tool called Critter to assist in the design of manageable network applications. Critter is an outgrowth of our research in the design of *composite systems* [Feather, 1987]. Starting from a statement of a problem, Critter helps a designer derive a distributed architecture of users, processes, and communication protocols to solve the problem. Critter also helps determine what network management support the architecture needs, *during* the design process. We thus make management a first-class concern during network application design.

To date, we have tested Critter by reconstructing existing designs in network protocols and related areas. In [Fickas and Helm, 1992a] we describe the results of two such reconstructions. The results suggest that Critter could help network application designers create designs which are more robust and maintainable. We propose to apply Critter to the design of two realistic network applications that have a key management component: The applications are taken from two broad areas:

1. Distributed document control systems. This is a broad class of applications that involve two or more users collaborating on the completion of a common artifact. We will concentrate on a distributed assignment control system for a new undergraduate electronic teaching lab that we are setting up; the artifact in this case is a collection of completed homeworks that meet some assignment specification. Related applications that are a natural follow on to this work include version control systems (for both code and documents), CSCW applications such as groupware editors, and distributed CAD systems.
2. Distributed decision systems. The application that we have chosen as a first problem in this area is a distributed group meeting system. Here the decision is when to meet. Other areas that we consider related are distributed decision support systems for grad school applications (who to accept), equipment purchases (what to buy), and course scheduling (what to teach and when).

We propose designing manageable versions of both applications, implementing our designs, and testing the implementations we produce in our department networking environment. Results will be disseminated in the networking community.

In the remainder of this proposal, we describe Critter in more detail, and our program for evaluating it. We will use the distributed assignment control system as an example.

2.0 Background

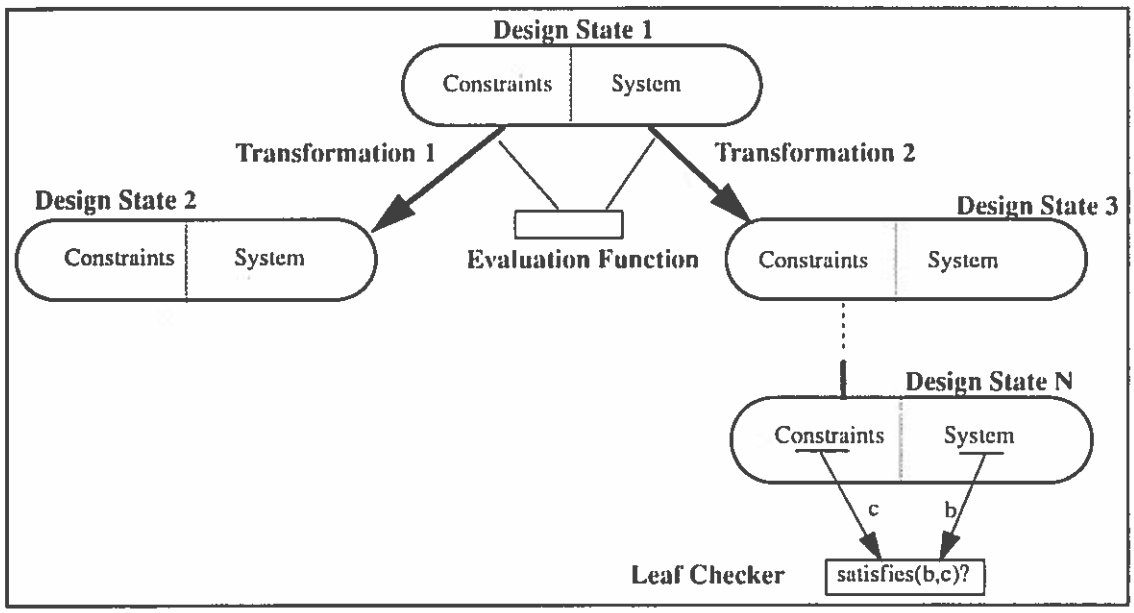
In this section, we describe Critter, our model for designing composite systems. We follow Critter through a simple example, the design of an information server for student assignments. We present Critter informally in this proposal. The interested reader can find additional details in [Fickas and Helm, 1992a] and [Fickas and Helm, 1992b].

2.1 The design model

Critter helps a human designer design network applications using the state-space search paradigm. State-space search was initially applied to puzzle-solving and game-playing. A search algorithm for solving chess problems, for instance, starts from an initial statement of the board configuration and of the problem, such as "checkmate for white in four moves". The algorithm generates new possible board configurations or *states* by trying legal chess moves.The moves are modelled as *search operators* or *transformations*. The algorithm uses a *solution state* or *leaf-node* checker to see if any of the states generated satisfy the problem statement. In chess, this amounts to looking for checkmates. Most practical chess playing programs also apply a *heuristic evaluation function* to each state. This function returns a score indicating how promising that state is for future play, and is used to make decisions on which transformation (move) to apply next.

Critter uses the state-space search paradigm to produce high-level network application designs (Figure 1).

Figure 1. Critter state-based search model of composite system design



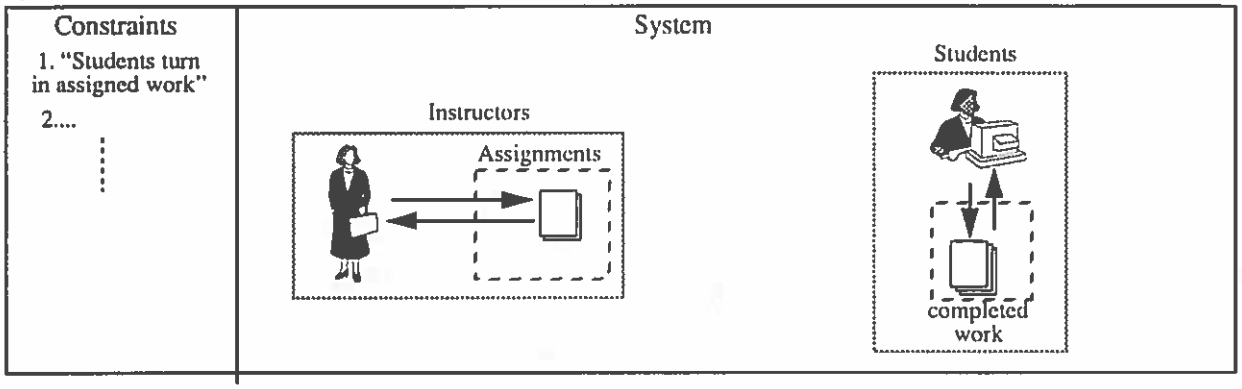
The components of state-space search are used as follows:

- *States*

  Each state in Critter's search contains 1) a high-level network application design, and 2) a statement of constraints on the application.

  Figure 2 shows the initial state of the example we use to illustrate our approach, the design of a student assignment server. The *constraints* part of the state indicates the goals of the design. The *system* part of the state is an incomplete initial design, divided into *agents*. An agent represents an active component of the design, be it a human user, a hardware sensor, or a software process. Here, the system portion includes two agent classes: instructor agents, which can generate assignments for a class, and student agents, which can generate and hand in work. Note that in the initial system design, instructors can generate any kind of assignment and students can hand in any kind of work, right or wrong.

Figure 2. Initial design state for the assignment server problem



- *Leaf-node checkers*

  The goal of Critter's search is to reach a design state which is *consistent*: all of the behavior generated by the system part satisfies the constraints. Critter's automated specification analysis tools help the designer determine whether a design state is consistent. They include a planner [Anderson and Fickas, 1989], a reachability set generator [Huber et al., 1986], and a design simulator [Fickas and Nagarajan, 1988].

  The initial design state shown in Figure 2 is not consistent. Among other things, the system allows students to turn in assignments which are different from those assigned, or to never turn in any assignment at all. Both of these problems are detected by running Critter's analysis tools on the design state. Both must be corrected by employing further search.

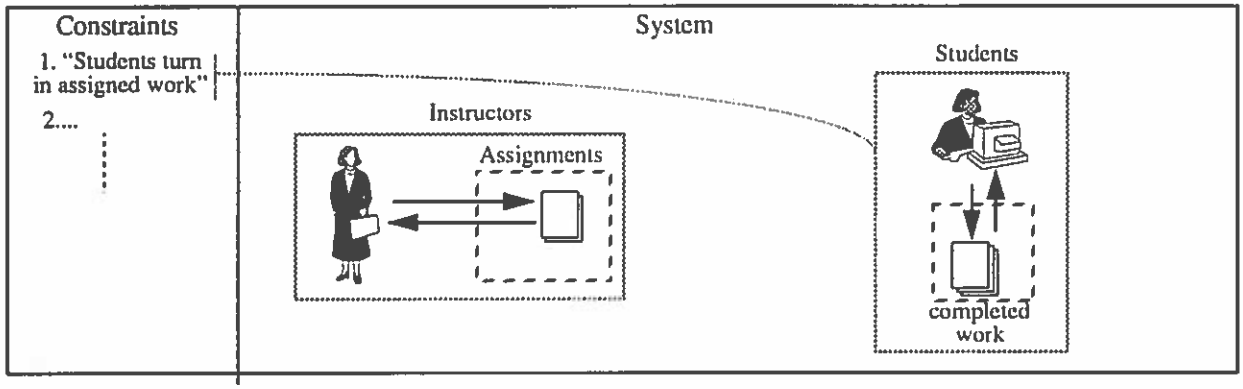
- *Transformations*

  Critter includes a library of design transformations that drive search. The analyst applies transformations to a design state and generates new design states that may be closer to consistency. Critter keeps track of the transformations applied and the design states generated. Each transformation in the library also includes a list of drawbacks or potential failures associated with its use. The latter will allow us to reason about management issues.

  In our example (Figure 2), the designer applies a simple transformation which makes student agents responsible for turning in only the assignments given. This generates a new design state (Figure 2); the gray line indicates that an agent class (students) has been made responsible for a constraint (constraint 1).

The transformation also raises a design issue: how will the responsible agent (the student) know which assignments have been given? The designer will apply additional transformations to resolve this issue.

Figure 3. New design state with students responsible for constraint.



- *Heuristic Evaluation Function*

  Evaluating high-level network application designs requires expertise we have not tried to capture in Critter. We rely on the human designer to decide which design states are most promising for continuing design.

With the exception of heuristic evaluation, all of the components of Critter described above use formal notations as described in [Fickas and Helm, 1992a]. We will employ informal diagrams here to simplify presentation.

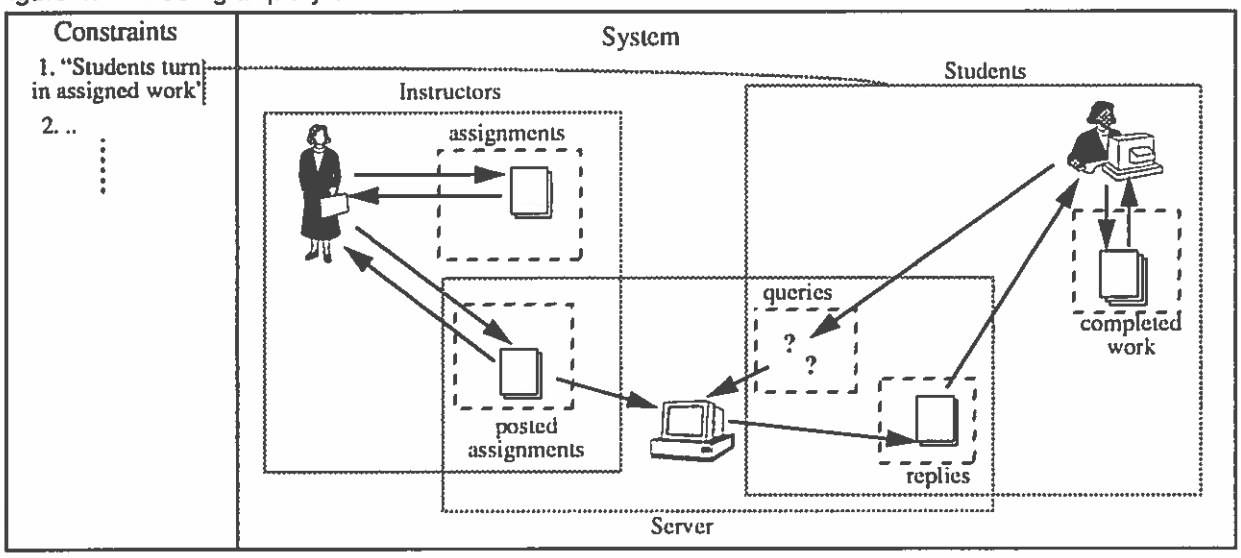
2.2 Designing in management: recovering from failures.

We believe that network applications can be made more robust and maintainable by choosing a strategy for managing an application during the early design of the application itself. We next illustrate how Critter supports this process.

In the student assignment problem we discussed above, we applied a transformation which required students to turn assignments given by the instructor, i.e., it "assigned responsibility" constraint 1 to the agent class Student. This raised a design issue: how will students know which assignments have been given?

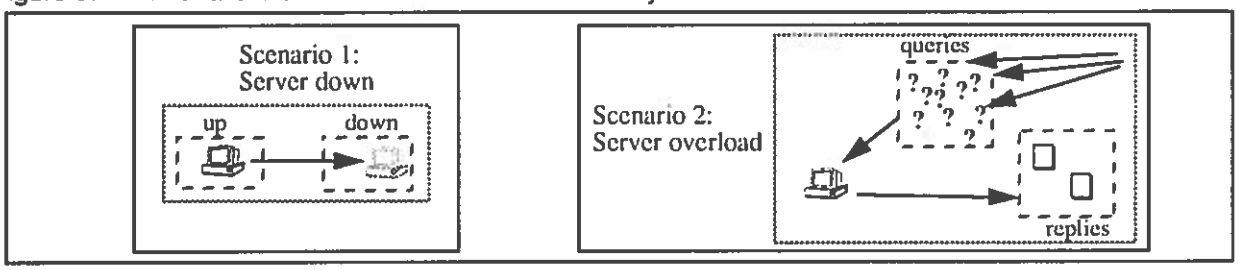
The designer applies a *Query Server* transformation to resolve this issue. The query server transformation generates a new design state represented by Figure 4. This transformation introduces a new agent (a server) into the design. The server will 1) accept assignments from the instructors, and 2) relay those assignments to students who query it.

Figure 4. Adding a query server



The Query Server transformation also has attached *failure scenarios*, representing known problems or errors that crop up in systems with query servers. Figure 5 shows these scenarios. Each scenario can be fed into one of Critter's analysis tools. The analysis tools will then try to run the scenario against the design. If successful, they will report the problem to the designer. We will look at Scenario 1 in this section, and Scenario 2 in the next section.

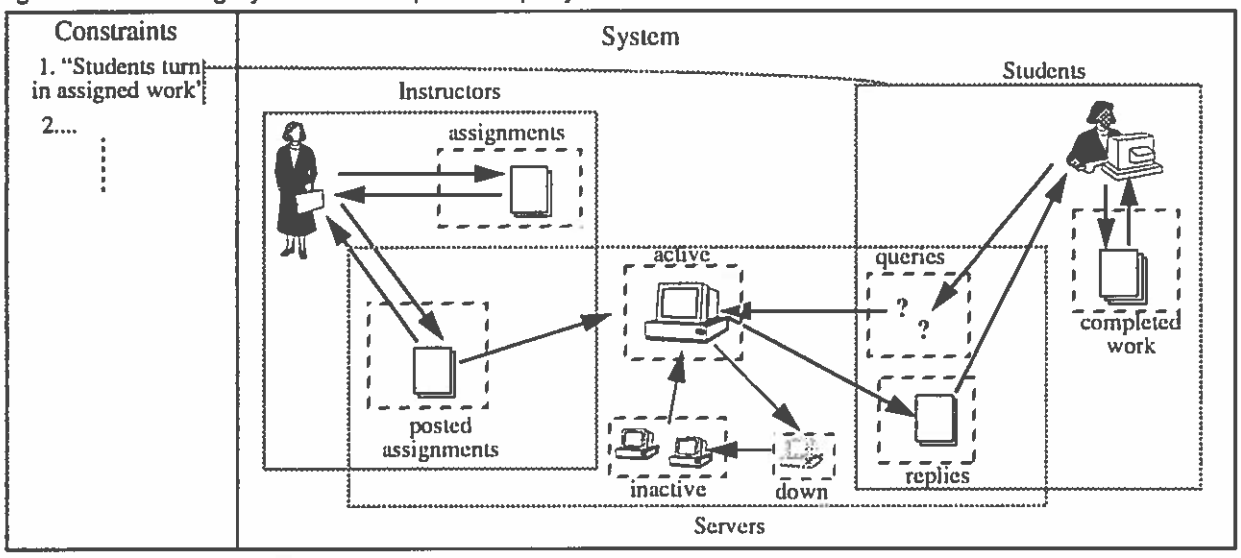
Figure 5. Two failure scenarios attached to the Query Server transformation.



Scenario 1 shows one failure scenario Critter reports on the query server. If the server crashes, students will not be able to get their assignments. Fixing this problem leads the designer to insert network management components into the application.

First, the designer applies a *Dynamic Backup* transformation to the query server agent. The result is shown in Figure 6. There will now be multiple server agents, with only one active at any one time. When the active server goes down, a new server will become active and take its place.

Figure 6. Adding dynamic backup to the query server.

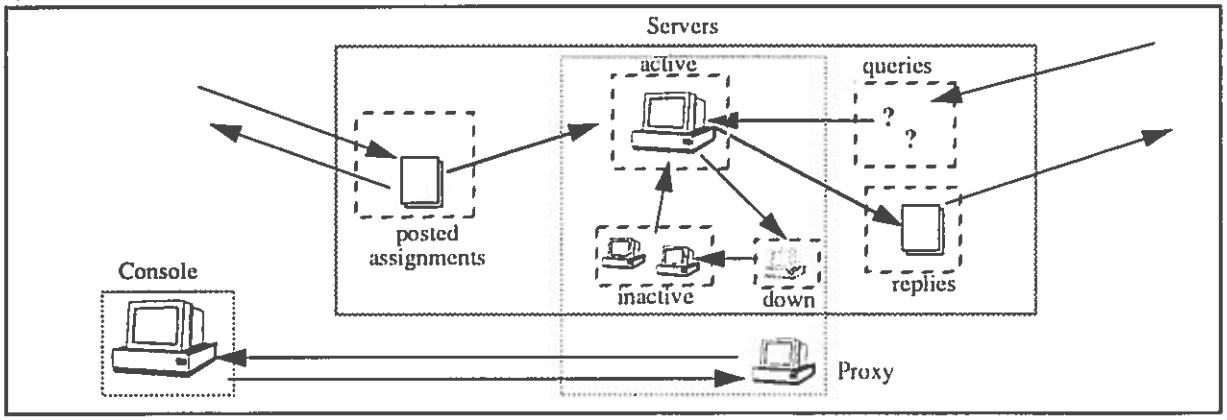


This strategy has its own design issues: How will we know when the active server goes down, and how will we select the backup to take its place? The designer resolves these issues by adding a network management function to the application. Specifically, the designer applies transformations which:

- Introduce a proxy agent which will monitor the status of the active server.
- Introduce a console, which will receive reports from the proxy agent on the status of the active server and awaken a new server if the active one has gone down.

Part of the resulting design state is shown in Figure 7. To implement this design state using SNMP, for example, we need to create a proxy agent along with the server application. The proxy agent will maintain a management database giving the status of all of the assignment servers (Active, Inactive, Down). We will also have to program an event report request for the management console. The request will require the proxy agent to report when the current server is down. Finally, the console will also be able to generate a set request to the proxy, to bring up a new server if none is currently active.

Figure 7. Management strategy for server recovery.



2.3 Designing in management: monitoring performance

The preceding example shows how Critter can help design network management strategies to recover from application failures. The next example shows how Critter can introduce network management to help resolve uncertainties about the performance of an application's design.

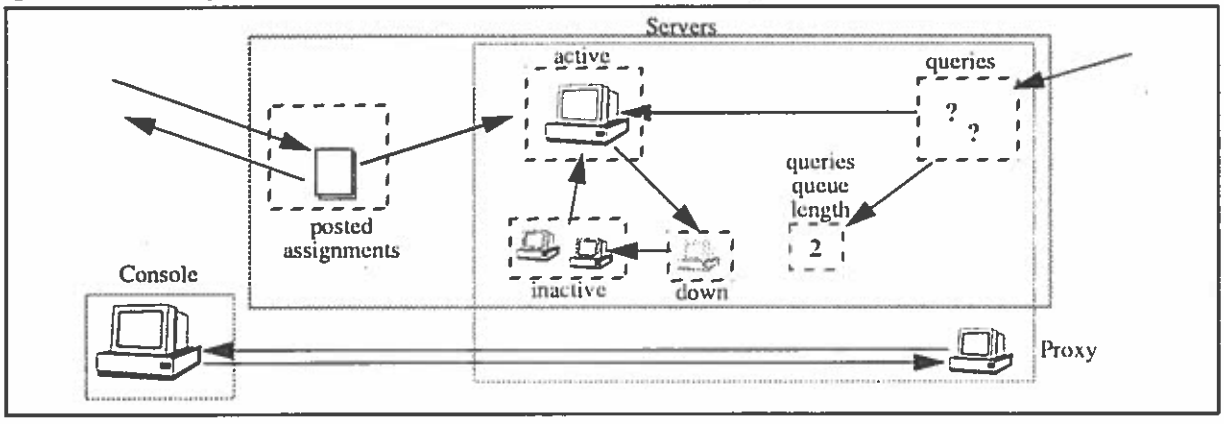
Scenario 2 in Figure 5 gives another failure associated with Query Server transformation. The server may become overloaded if too many students query it at a time. This may cause it to postpone answering a given query beyond any reasonable time bound. It may also cause the server to use too many resources on its host system. The overload scenario is run in the design state in Figure 4 and generates a problem report.

The designer could choose to ignore this report, arguing that the load on the server will be too low to cause problems in the server's implementation environment. Or the designer could back up to the design state in Figure 4, and try a different design alternative. Instead of designing a system with a single server with backups, the designer could create a system with multiple running servers, more similar to the Internet Domain Name System [Mockapetris, 1987].

The designer chooses a third option. There will be a single server, but network management components will monitor usage statistics, to help detect performance problems. The designer applies a transformation called *Instrument Load* to the design. This generates a new design which will keep a running count of the number of outstanding queries in the server. The designer then assigns the proxy agent for the server to report this count to the management console on request.

Figure 8 shows part of the design state which results from these transformations. To implement this new design state, we add a variable to the proxy management database. This variable will contain the current length of the query queue. The management console can issue a data report request to the proxy to compute load statistics.

Figure 8. Adding load monitoring to the management strategy.



3.0 Proposed Research

The examples in the previous section suggest several potential benefits of the Critter design model for network application design:

- *Reuse.* The transformations of Critter give the designer a set of building blocks for generating application designs. The Query Server transformation used in the homework assignment server problem can also be used to generate other kinds of servers, such as an e-mail based archive server.
- *Analysis.* Using failure scenarios and analysis tools, Critter can help the designer identify potential weak points in a design. The server failure scenario, for instance, reminded the designer to consider the consequences of a crash of the assignment server.
- *Recording and Rationalization.* Critter keeps track of all of the design states generated. This serves as a record of the designs considered, including alternatives which were rejected. The designer in our example could back up to the state in Figure 4 and try a different redundancy scheme if the single-server+backups design seemed untenable. The design record might also be useful for explaining the features of the assignment server to other designers, or to implementers, who need to know what decisions lay behind its design.
- *First-class Management.* Components for managing an application are designed along with the application. The makes it easier to ensure that the application is not under-managed (managers have too little control over the application) or over-managed (too many unused management hooks are provided). For instance, we can trace the variable for queue length in the proxy management database to a specific problem (the overloading scenario) reported during design, and to the designer's decision to monitor this potential problem.

We propose research to test whether Critter can actually provide these benefits to designers of network applications. We will look at two problems initially. First, we will fully design and implement the example started in section 2. The goal is an "assignment server" that we can test in our new beginning computer science lab, funded by the National Science Foundation Instructional Lab Instrumentation program, or ILI for short. As described earlier, the assignment server will be the first component of what we see as a distributed classroom. It provides an ideal field setting in which to test our designs.

The second problem we will tackle is that of distributed group scheduling. We have used Critter to design a non-managed version of a group scheduler. During this design, it became clear that we 1) needed to add in management capability, and 2) it was best to start from scratch as opposed to tacking on management pieces after the fact. We plan to design, implement and field test the new, managed version, within our department. We discuss the group scheduling problem in more detail in [Fickas and Helm, 1992b].

Both of these applications will initially be designed to run over the TCP/IP suite of protocols, and will be implemented with these protocols. However, we will also begin work on generating comparable designs over the OSI application layer. We believe this will be a useful test of the generality of our approach.

In summary, we will use the Critter model to design network applications for use within our department research and instructional environment. As in the example problem of section 2, we will design these applications to be remotely managed, using standard network management protocols. We will produce implementations of our designs, and use these implementations to help evaluate the Critter model and the designs it produces.

3.1 Disseminating the results

We will deliver the following:

- *Standard description of design.* We will document our designs in a recognized network specification language. Currently we use a form of Numerical Petri Nets, but we have begun to seriously consider others, e.g., SDL [Belina and Hogrefe, 1988].
- *The design history of our systems.* We will produce our design history in a form that can be disseminated in both hardcopy (e.g., RFCs) and electronically (e.g., it-IBIS [Yakemovic and Conklin, 1990]).
- *An implementation.* We will implement our designs. Each implementation will run under systems supporting TCP/IP (via BSD sockets), X Window output devices, and SNMP. Our distribution will include source code for user agents, scheduler agents, or other agents our protocol requires. It will also include ASN.1 data definitions and associated C code for information exchanged between agents. Finally, it will include operating and maintenance documentation. The code of the implementation will be produced manually, or generated by the tools in the ISODE (ISO Development Environment) [Rose, 1989].
- *Evaluation of results.* We will test our implementations both in our lab, and in the field. The results of our testing will be published.

We have a particular interest in generating reproducible results. Hence our interest in using standard implementation components, and producing well-documented designs.

4.0 Acknowledgments

Our early work on SDEs for networked applications was sponsored by a National Science Foundation grant #CCR-880485. This support allowed us to 1) build a theoretical foundation for the more applied work we propose here, 2) experiment with different prototypes of the Critter tool. We thank the NSF for this key and timely support.

We also thank Keicho, Ltd in Tokyo and NEC in Tokyo for preliminary funding of our work on application of Critter to distributed group scheduling.

5.0 References

[Anderson and Fickas, 1989] J. S. Anderson and S. Fickas, "A proposed perspective shift: viewing specification design as a planning problem," in *Proc. 5th International Workshop on Software Specification and Design* (Pittsburgh, PA, USA). Los Alamitos, CA: IEEE Computer Society, 1989, p. p. 177-184.

[Belina and Hogrefe, 1988] Belina, F., Hogrefe, D., "The CCITT specification and description language SDL," *Computer Networks and ISDN Systems* Vol. 16, 1988/89, p. p. 311-341.

[Feather, 1987] M. S. Feather, "Language support for the specification and development of composite systems," *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 2, p. p. 198-234, November 1987.

[Fickas and Helm, 1992a] Fickas, S. Helm, B. R., "Knowledge representation and reasoning in the design of composite systems," *IEEE Transactions on Software Engineering*, Special Issue on Knowledge Representation and Reasoning, June, 1992. Also available as Technical Report 92-07, Department of Computer and Information Science, University of Oregon, Eugene, OR, 97403.

[Fickas and Helm, 1992b] Fickas, S., Helm, B. R., "Automating Specification of Network Applications," In *Proceedings of IFIP International Conference on Upper Layer Protocols, Architectures and Applications* (Vancouver, B.C., May, 1992). North-Holland, 1992.

[Fickas and Nagarajan, 1988] S. Fickas and P. Nagarajan, "Critiquing software specifications: a knowledge based approach," *IEEE Software*, November 1988.

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

[Mockapetris, 1987] Mockapetris, P. V., "Domain names: concepts and facilities," Request For Comments 1033, November 1987.

[Rose, 1989] Rose, M. T., *The Open Book: A Practical Perspective on OSI*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1989.

[Rose, 1991] Rose, M. T., *The Simple Book: An Introduction to Management of TCP/IP-based Internets*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991.

[Yakemovic and Conklin, 1990] Burgess Yakemovic, K. C., Conklin, J., "Report on a development project use of an issue-based information system," *CSCW 90 Proceedings*, October 1990.