Diagnostic Agents for Distributed Systems

Peter Fröhlich[1], Iara de Almeida Móra[2], Wolfgang Nejdl[1]
and Michael Schroeder[1]
{froehlich, nejdl, schroeder}@kbs.uni–hannover.de, idm@fct.unl.pt

October 5, 1996

Abstract

In this paper we introduce an agent–based framework for the diagnosis of spatially distributed technical systems, based on a suitable distributed diagnosis architecture. We implement the framework using the concepts of vivid agents and extended logic programming. To demonstrate the power of our approach, we solve a diagnosis example from the domain of unreliable datagram protocols.

Submission Information: This paper has not been submitted to another conference or workshop.

Keywords: Multi Agent Systems, Logic Programming, Diagnosis

Post-Proceedings: We also submit this paper for the post–proceedings.

1 Introduction

The advent of large distributed technical systems like computer and telecommunication networks has been one of the most striking developments of our time. Research in model–based diagnosis as documented in several AI conferences and a series of workshops [Pro94, Nej95] has up to now not tackled the question how to support such systems by a suitable diagnosis architecture.

We introduce an agent–based framework for the diagnosis of spatially distributed systems. The motivation for such a framework is the unnecessary complexity and communication overhead of centralized solutions. Consider a distributed system with n nodes, e.g. a computer network consisting of n machines. When using a centralized diagnosis system the size of the system description (i.e. number of ground formulas) is linear in n. Diagnosis time will usually be worse than linear in n [MH93]. Also all observations have to be transmitted to the central diagnosis machine, causing a large communication overhead.

Our agent–based approach decomposes a system into a set of subsystems. Each subsystem is diagnosed by an agent which has detailed knowledge over his

[1]Institut für Rechnergestützte Wissensverarbeitung, Lange Laube 3, 30159 Hannover, Germany, Tel.: +49 511 762 9714, Fax: +49 511 762 9712

[2]Departamento de Informática, Universidade Nova de Lisboa, 2825 Monte de Caparica, Portugal

subsystem and an abstract view of the neighboring subsystems. Most failures can be diagnosed locally within one subsystem. This decreases diagnosis time dramatically in large systems. In the case of the computer network most machines in a subnet can usually fail without affecting machines in other subnets. Only those computers in other subnets can be affected which have sent messages to the faulty machine. Moreover, the local computation of diagnoses avoids the communication overhead which would be needed to forward all observations to the central diagnosis engine.

Failures, which affect more than one subsystem are diagnosed by the agents cooperating with each other. The cooperation process is triggered locally by an agent, when it realizes that it can not explain the observations by a failure in his own subsystem. The cooperation process is guided by a small amount of topological information.

We have implemented spatially distributed diagnosis using extended logic programming [SdAMP96, SW96] and the vivid agents concept [Wag96a, Wag96b]. Vivid agents support both the declarative description of the domain by a flexible knowledge base component and the specification of the reactive behavior of agents by a set of rules, which are activated by communication events.

To demonstrate the power of our approach we formalize the domain of an unreliable protocol (like UDP) in a computer network and diagnose an example scenario.

2 Spatially Distributed Diagnosis

In their survey article in [BG88], Bond and Gasser discuss several measures of conceptual distance among agents, which define different types of distribution. In [FN96] semantical and spatial distribution are identified as the relevant distribution concepts for diagnosis. Semantical distribution refers to a situation where the knowledge is distributed among the agents. Each agent is an expert for a certain problem domain. Diagnostic concepts for semantical distribution must rely on external criteria rather than cooperation among the agents because the knowledge bases of the diagnostic agents are not compatible.

In this paper we describe spatially distributed diagnosis. Distributed technical systems often consist of subsystems which have the same structure. So we can describe the subsystems by a common set of axioms. The particular properties of the concrete subsystem are defined by logical facts. As we will see, the description of the subsystems by a common vocabulary allows us to resolve conflicts using cooperation among the agents.

After giving a short overview of the necessary concepts from model–based diagnosis, we will describe our view of spatially distributed diagnosis in more detail. Then we will define the diagnostic conflicts between the subsystems as well as the distributed diagnosis concept formally.

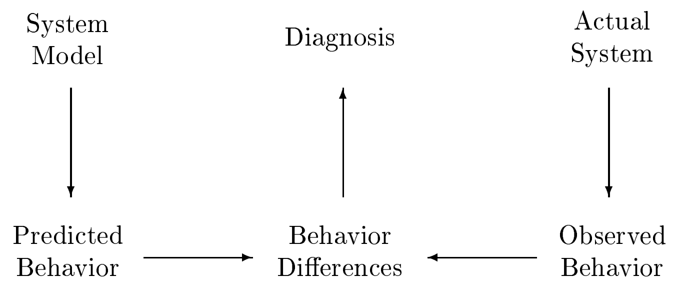

Figure 1: Model–Artifact Difference

2.1 Model–based Diagnosis

Heuristic rule–based expert systems were the first approach to automated diagnosis. The knowledge bases of such systems could not be easily modified or verified to be correct and complete. These difficulties have been overcome by the introduction of model–based diagnosis [Rei87], where a simulation model of the device under consideration is used to predict its normal behavior, given the observed input parameters. Diagnoses are computed by comparison of predicted vs. actual behavior (Fig. 1).

This approach uses an extendible logical model of the device, called the system description (SD), usually formalized as a set of formulas expressed in first–order logic. The system description consists of a set of axioms characterizing the behavior of system components of certain types. The topology is modeled separately by a set of facts.

We will now define the diagnostic concept mathematically. The diagnostic problem is described by system description SD, a set $COMP$ of components and a set OBS of observations (logical facts). With each component we associate a behavioral mode: $Mode(c, Ok)$ means that component c is behaving correctly, while $Mode(c, Ab)$ (abbreviated by $Ab(c)$) denotes that c is faulty.

In *Consistency–Based Diagnosis*, the concept we are using throughout this paper, a *Diagnosis* D is a set of faulty components, such that the observed behavior is consistent with the assumption, that exactly the components in D are behaving abnormally. If a diagnosis contains no proper subset which is itself a diagnosis, we call it a *Minimal Diagnosis*.

Definition 2.1 *(Reiter 87) A* Diagnosis *of* $(SD, COMP, OBS)$ *is a set* $\Delta \subseteq COMP$*, such that* $SD \cup OBS \cup \{Mode(c,Ab) | c \in COMP\} \cup \{\neg Mode(c,Ab) | c \in COMP - \Delta\}$ *is consistent.* Δ *is called a* Minimal Diagnosis*, iff it is the minimal set (wrt.* $\subseteq$*) with this property.*

Minimal Diagnoses are a natural concept, because we do not want to assume that a component is faulty, unless this is necessary to explain the observed behavior. Since the set of minimal diagnoses can be still quite large and the ultimate goal is to identify a single diagnosis, stronger minimality criteria are used which allow stronger discrimination among the diagnoses.

The most frequently used concepts are *Minimal Cardinality Diagnosis* and *Most Probable Diagnosis*. In addition to these stronger definitions of diagnosis the agents can use measurements to discriminate among competing diagnoses.

For our distributed diagnosis framework we assume that every agent has identified a single diagnosis for its subsystem.

2.2 Properties of Spatial Distribution

Spatial distribution is a natural organization scheme for the distributed diagnosis of large technical systems like communication networks. With each agent we associate a certain area of the system, for which it is responsible.

Consider a large distributed system, e.g. a communication network, which is divided into a set of spatially distributed subsystems (subnets), as shown in figure 2. Each square in the grid is a subsystem and has a diagnostic agent associated with it.

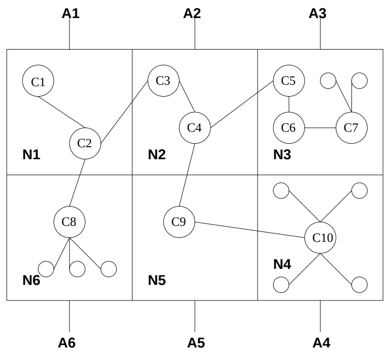

Figure 2: A communication network

What could be the system view of agent A_1? Of course, it has detailed knowledge about its own subsystem (provided by the control component). For components in its own subsystem the agent himself is responsible and its diagnoses are reliable. Since it does not share its local observations and measurements with other agents (except for specialized information used during cooperation) it is the only agent, which can compute detailed diagnosis of its subsystem. In the decentralized structure of this network, the machine C_2 must have at least some routing information. It has to know that there are two adjacent subnets N_2 and N_6, to which it can send information. More generally we assume that each agent has some information on the neighboring subsystems, i.e. the subsystems directly connected to its own in the system structure.

Now we will describe this view by means of abstractions and simplifications: An agent A_i knows only the name of each neighboring subnet N_j (and perhaps a name of a server within N_j) but not N_j's internal structure. When A_i diagnoses an error involving subnet N_j (e.g. a lost message routed via N_j), then the diagnosis will contain $Mode(N_j, Ab)$. The abstract literal $Mode(N_j, Ab)$ implicitly implies that some particular component within N_j is faulty. In general, an agent A_i has an abstract model of the neighboring subsystems. Furthermore, A_i only knows that N_j is the first subnet on the route to the destination of the lost message. It is a simplifying assumption, that N_j is the only subnet involved in the transmission. Stated more generally, an agent A_i initially uses the simplifying assumption that all errors it cannot explain are caused by its immediate neighbors. We will see, how he can get more detailed information during the cooperation process.

2.3 Formalization

The subsystems and also the components within each subsystem have standard names. A predicate *Area_Component* denotes that component c is situated within area a of the system. We call the extension of this predicate for a given system the *Component Hierarchy*.

Definition 2.2 *Component Hierarchy. The* Component Hierarchy *CH for a distributed system is a set of facts for the predicate Area_Component.*

Example 2.3 *For our communication network we have*

$$CH: \begin{array}{l} Area\_Component(N_1, C_1) \\ Area\_Component(N_1, C_2) \\ Area\_Component(N_2, C_3) \\ \ldots \end{array}$$

Using the predicate *Area_Component*, we can formulate a consistency condition between the abstract subsystem–level and the detailed component level. We define the consistency of abstractions axiom:

Definition 2.4 *Consistency of Abstractions. The axiom*

$$\begin{array}{ll} CA: \forall c. & ((Mode(c, Ab) \land \exists d. Area\_Component(c, d)) \\ & \rightarrow \exists e. (Area\_Component(c, e) \land Mode(e, Ab))) \end{array}$$

requires, that each abnormality of an abstract component is caused by an abnormality of one of its subcomponents.

The axiom *Disjointness of Modes* states that a component can only be in one behavioral mode at a given time point and is expressed by the following axiom:

Definition 2.5 *Disjointness of Modes.*

$$DM: \forall c. \forall m_1. \forall m_2. (Mode(c, m_1) \land Mode(c, m_2)) \rightarrow m_1 = m_2$$

2.4 Diagnosis by Cooperation

Each diagnostic agent knows only a small part of the entire system. It can compute diagnoses independently, because it maintains a set of assumptions concerning the other parts of the system. In this paper, we will assume that all locally computed diagnoses are considered as reliable. The bargain from distributed diagnosis is that a lot of problems can be solved locally so that the simplifying assumptions hold. The cooperation process is necessary when an agent cannot detect a faulty component within its subsystem. In this case, it starts a cooperation process:

Definition 2.6 *Need for Cooperation*
Given observations OBS, a component hierarchy CH, the axiom of consistency of abstractions CA, and a system description SD such that CH,CA ∈ SD. If A_i knows that it is not abnormal, i.e.

$$SD_{N_i} \cup OBS \models \neg Mode(N_i, Ab) \text{ and } SD_{N_i} \cup OBS \cup \{Mode(N_j, Ab)\} \not\models -$$

then there is a need for cooperation *to determine a global diagnosis and N_j is a possible partner for cooperation.*

Example 1 *In the example of the communication network the observation of a lost message (let us assume an unreliable protocol such as UDP) means that it was lost somewhere on the way from sender to recipient. But of course the agents know only their own subnet in detail and have an abstract view of the neighboring subnets. Reported loss of a datagram is formalized as an observation using the predicate Message_Lost. $Message\_Lost(N_1, N_5)$ means that a lost message has been reported which was sent from network N_1 to network N_5. When agent A_i transmits a message via a neighboring subnet N_j and the message is lost, A_i will assume that it was lost in N_j since this is the only point on the route it knows. We can formalize this simplifying assumption explicitly by introducing a predicate On_Route.*

$$\begin{aligned} CLM: \quad & Message\_Lost(Sender, Recipient) \\ & \rightarrow \exists c.(On\_Route(Sender, Recipient, n) \wedge Mode(n, Ab)) \end{aligned}$$

is called Existence of a Cause for a Lost Message.

Initially, each agent A_i knows the following facts about On_Route, if N_{r_k} is the routing table entry of $Recipient_k$:

$$\begin{aligned} RT: \quad & On\_Route(N_i, Recipient_1, N_i) \\ & On\_Route(N_i, Recipient_1, N_{r_1}) \\ & On\_Route(N_i, Recipient_2, N_i) \\ & On\_Route(N_i, Recipient_2, N_{r_2}) \\ & \ldots \end{aligned}$$

Now assume a message gets lost from N_1 to C_7, i.e. $Message\_Lost(N_1, C_7)$ and the agent A_1 determines that it is not its fault, i.e. $\neg Mode(N_1, Ab)$ holds. Then A_1 computes a local diagnosis $Mode(N_2, Ab)$ and thus there is a need for cooperation in order to obtain a global solution.

A cooperation process is started by sending/receiving an observation. With the new observation the agent computes diagnoses which can lead to three different situations. First, it might turn out that it is abnormal itself. Then other solutions can be neglected since we assume that the agents have certain knowledge about their own state. Second, there are no diagnoses at all which means that the initial fault is intermittent. Third, there is a need for cooperation. Then the agent refines the received observation and sends it to the neighbor waiting for its reply. In any of the cases the requesting agent is informed of the final result.

Definition 2.7 *Diagnosis by Cooperation*
Given an agent A_1 which receives a message from agent A_2 with an observation OBS such that $SD_{A_1} \cup OBS \models -$ then there are three cases:

1. *$SD_{A_1} \cup OBS \cup \{Mode(N_1, Ab)\} \not\models -$, i.e. the agent's own subsystem is faulty*
2. *there are no D such that $SD_{A_1} \cup OBS \cup D \not\models -$, then there must have been an intermittent failure*
3. *there is a need for cooperation (see definition 2.6) and the observation is refined and sent to another agent which is then in charge of providing a diagnosis result*

The diagnosis result is sent to A_2.

Example 2 *Assume A_1 receives by a subcomponent the message that a message is lost from N_1 to C_7 and N_1 not being abnormal. A diagnosis of A_1 is that A_2 is abnormal and thus there is a need for cooperation. The initial observation $Message\_Lost(N_1, C_7)$ is refined as follows:*

$$\begin{array}{ll} RO: & Message\_Lost(Sender, Recipient) \wedge \neg Mode(Sender, Ab) \wedge \\ & On\_Route(Sender, Recipient, Next\_Sender) \rightarrow \\ & New\_Message\_Lost(Next\_Sender, Recipient) \end{array}$$

The new observation is sent to A_2. Since a_2 is not abnormal this agent asks A_3 for help. A_3 is faulty and replies that it is responsible. A_2 passes this result to A_1. The union of all system descriptions involved is consistent with the final diagnosis of A_3.

Now we can define distributed diagnosis. A diagnosis for the union of all system descriptions is called distributed diagnosis:

Definition 2.8 *Distributed Diagnosis*
A Distributed Diagnosis *of $(\{SD_{A_1}, \ldots SD_{A_n}\}, COMP, OBS)$ is a set $\Delta \subseteq COMP$, such that $SD_{A_1} \cup \ldots \cup SD_{A_n} \cup OBS \cup \{Mode(c,Ab) | c \in COMP\} \cup \{\neg Mode(c,Ab) | c \in COMP - \Delta\}$ is consistent.*

Example 3 *$\{Mode(N_3, Ab), Mode(C_7, Ab)\}$ is a distributed diagnosis for the system description and observations in the above example.*

In order to implement the scenario above we need separate diagnostic agents for each area. The agents need a knowledge base containing the description of their area and they have to be capable of reactive behavior in order to solve a problem in cooperation with other agents.

The theoretical basis of the implementation is the concept of vivid agents [Wag96b] and a prototype developed for fault-tolerant diagnosis [SdAMP96, SW96]. Below we briefly review the vivid agents and extended logic programming. We proceed by showing how the axioms can be expressed as extended logic program and how the agents' reactive behavior is coded in terms of reaction rules. We round out the picture with a trace of the agents' communication after a message is lost.

3 Vivid Agents

A *vivid agent* is a software-controlled system whose state is represented by a knowledge base, and whose behavior is represented by means of *action* and *reaction rules*. Following [Sho93], the state of an agent is described in terms of mental qualities, such as beliefs and intentions. The basic functionality of a vivid agent comprises a knowledge system (including an update and an inference operation), and the capability to represent and perform actions in order to be able to generate and execute plans. Since a vivid agent is 'situated' in an environment with which it has to be able to communicate, it also needs the ability to react in response to perception events, and in response to communication events created by the communication acts of other agents.

Notice that the concept of vivid agents is based on the important distinction between action and reaction: actions are first planned and then executed in order to solve a task or to achieve a goal, while reactions are triggered by perception and communication events. Reactions may be immediate and independent from the current knowledge state of the agent but they may also depend on the result of deliberation. In any case, they are triggered by events which are not controlled by the agent. A vivid agent without the capability to accept explicit tasks and to solve them by means of planning and plan execution is called *reagent*. The tasks of reagents cannot be assigned in the form of explicit ('see to it that') goals at run time, but have to be encoded in the specification of their reactive behavior at design time.

We do not assume a fixed formal language and a fixed logical system for the knowledge-base of an agent. Rather, we believe that it is more appropriate to choose a suitable knowledge system for each agent individually according to its domain and its tasks. In the case of diagnosis agents, extended logic programs proved to be an appropriate form of the knowledge base of an agent because it is essential for model-based diagnosis to be able to represent negative facts, default rules and constraints.

3.1 Specification and Execution of Reagents

Simple vivid agents whose mental state comprises only beliefs, and whose behavior is purely reactive, i.e. not based on any form of planning and plan execution, are called *reagents*. A reagent $\mathcal{A} = \langle X, EQ, RR \rangle$, on the basis of a knowledge system $\boldsymbol{K}$ consists of

1. a knowledge base $X \in L_{\text{KB}}$,
2. an event queue EQ being a list of instantiated event expressions, and
3. a set RR of *reaction rules*, consisting of epistemic and physical reaction and interaction rules which code the reactive and communicative behavior of the agent.

A multi-reagent system is a tuple of reagents:

$$\mathcal{S} = \langle \boldsymbol{A}_1, \ldots, \boldsymbol{A}_n \rangle$$

3.1.1 Operational Semantics of Reaction Rules

Reaction rules encode the behavior of vivid agents in response to perception events created by the agent's perception subsystems, and to communication events created by communication acts of other agents. We distinguish between epistemic, physical and communicative reaction rules, and call the latter *interaction rules*. We use L_{PEvt} and L_{CEvt} to denote the perception and communication event languages, and $L_{\text{Evt}} = L_{\text{PEvt}} \cup L_{\text{CEvt}}$. The following table describes the different formats of epistemic, physical and communicative reaction rules:

$$\begin{array}{rcl} \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon(U), S],\ \mathit{Cond} \\ \mathsf{do}(\alpha(V)),\ \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon(U), S],\ \mathit{Cond} \\ \mathsf{sendMsg}[\eta(V), R],\ \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon(U), S],\ \mathit{Cond} \end{array}$$

The event condition $\mathsf{recvMsg}[\varepsilon(U), S]$ is a test whether the event queue of the agent contains a message of the form $\varepsilon(U)$ sent by some perception subsystem of the agent or by another agent identified by S, where $\varepsilon \in L_{\text{Evt}}$ represents a perception or a communication event type, and U is a suitable list of parameters. The epistemic condition $\mathit{Cond} \in L_{\text{Query}}$ refers to the current knowledge state, and the epistemic effect $\mathit{Eff} \in L_{\text{Input}}$ specifies an update of the current knowledge state.

Physical Reaction: $\mathsf{do}(\alpha(V))$ calls a procedure realizing the action α with parameters V.

Communicative Reaction: $\mathsf{sendMsg}[\eta(V), R]$ sends the message $\eta \in L_{\text{CEvt}}$ with parameters V to the receiver R.

Both perception and communication events are represented by incoming messages. In general, reactions are based both on perception and on knowledge. Immediate reactions do not allow for deliberation. They are represented by rules with an empty epistemic premise, i.e. $Cond = \mathsf{true}$. Timely reactions can be achieved by guaranteeing fast response times for checking the precondition of a reaction rule. This will be the case, for instance, if the precondition can be checked by simple table look-up (such as in relational databases or fact bases).

Reaction rules are triggered by events. The agent interpreter continually checks the event queue of the agent. If there is a new event message, it is matched with the event condition of all reaction rules, and the epistemic conditions of those rules matching the event are evaluated. If they are satisfiable in the current knowledge base, all free variables in the rules are instantiated accordingly resulting in a set of triggered actions with associated epistemic effects. All these actions are then executed, leading to physical actions and to sending messages to other agents, and their epistemic effects are assimilated into the current knowledge base.

4 Extended Logic Programming and Model-based Diagnosis

In order to represent diagnosis problems we briefly review extended logic programming and its application to model-based diagnosis.

Since Prolog became a standard in logic programming much research has been devoted to the semantics of logic programs. In particular, Prolog's unsatisfactory treatment of negation as finite failure led to many innovations. Well-founded semantics [GRS88] turned out to be a promising approach to cope with negation by default. Subsequent work extended well-founded semantics with a form of explicit negation and constraints [PA92] and showed that the richer language, called WFSX, is appropriate for a spate of knowledge representation and reasoning forms [PAA93, PDA93, ADP95]. In particular, the technique of contradiction removal of extended logic programs [PAA91] opens up many avenues in model-based diagnosis [PDA93, DNP94, DNPS95, DPS96, SdAMP96, SW96].

Definition 4.1 *Extended Logic Program*
An extended logic program is a (possibly infinite) set of rules of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not L_{m+1}, \ldots, not L_n \ (0 \leq m \leq n)$$

where each L_i is an objective literal $(0 \leq i \leq n)$. An objective literal is either an atom A or its explicit negation $\neg A$.[3] *Literals of the form $not L$ are called default literals. Literals are either objective or default ones.*

To capture that it is contradictory for the predicted behavior to differ from the actual observations, we introduce integrity constraints:

Definition 4.2 *Constraint*
An integrity constraint has the form

$$\bot \leftarrow L_1, \ldots, L_m, not L_{m+1}, \ldots, not L_n \ (0 \leq m \leq n)$$

[3] Note that explicit and implicit negation are related: $\neg L$ implies *not* L.

where each L_i is an objective literal $(0 \leq i \leq n)$, and $-$ stands for false.

In order to avoid a contradiction we change the beliefs that support the contradiction. The only beliefs that are subject to change concern the closed world assumption ones. From these we can define a set of revisable default literals, whose truth values may be changed to remove contradictions.

Definition 4.3 *Revisable*
The revisables R of a program P are a subset of the default negated literals which do not have rules in P.

In general, we might remove a contradiction by partially dropping the closed world assumption about some revisable. To declaratively define the contradiction removal, we consider all subsets R' of the revisables, change the truth value of the literals in R' from false to true and check whether the revised program is still contradictory. Among those revisions that remove the contradiction we are interested in the minimal ones:

Definition 4.4 *Revision*
Let R be the revisables of the program P. The set $R' \subseteq R$ is called a revision if it is a minimal set such that $P \cup R'$ is free of contradiction.

The revision of contradictory extended logic programs is a suitable technique to compute diagnoses for model-based diagnosis.

4.1 The Agents Knowledge Base

Using the extended logic programming formalism, the agents knowledge base contains the following logic sentences:

Routing tables The routing information comprises facts stating to which neighbor node a message addressed to a component has to be sent. The knowledge is local since each agent only knows its neighbors. In order to keep the facts in a single knowledge base which is the same for all agents the facts hold only for the respective agent (i_am). For example, for n_1 and n_2 we get the following routing tables:

$$
\begin{array}{lrclrcl}
RT: & & & & & & \\
 & on\_route(n_1, c_3, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_1, n_1) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_4, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_2, n_1) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_5, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_5, n_3) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_6, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_6, n_3) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_7, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_7, n_3) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_8, n_6) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_8, n_1) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_9, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_9, n_5) & \leftarrow & i\_am(n_2). \\
 & on\_route(n_1, c_{10}, n_2) & \leftarrow & i\_am(n_1). & on\_route(n_2, c_{10}, n_5) & \leftarrow & i\_am(n_2). \\
 & \ldots & & & \ldots & &
\end{array}
$$

Component Hierarchy Additionally, each agent knows its components. Since this knowledge is local it is only derivable for the respective agent (i_am):

$$
\begin{array}{llllll}
CH: & & & & & \\
area\_component(n_1, c_1) & \leftarrow & i\_am(n_1). & area\_component(n_3, c_6) & \leftarrow & i\_am(n_3). \\
area\_component(n_1, c_2) & \leftarrow & i\_am(n_1). & area\_component(n_3, c_7) & \leftarrow & i\_am(n_3). \\
area\_component(n_2, c_3) & \leftarrow & i\_am(n_2). & area\_component(n_6, c_8) & \leftarrow & i\_am(n_6). \\
area\_component(n_2, c_4) & \leftarrow & i\_am(n_2). & area\_component(n_5, c_9) & \leftarrow & i\_am(n_5). \\
area\_component(n_3, c_5) & \leftarrow & i\_am(n_3). & area\_component(n_4, c_{10}) & \leftarrow & i\_am(n_4).
\end{array}
$$

Disjointness of Modes In the implementation we model only two modes, abnormality (ab) and being ok ($not\ ab$). Therefore disjointness of modes is satisfied. The predicate ab is revisable. The default truth value of the predicate ab is false, which means that by default we assume components to be working fine. Possible contradictions to these assumption are caused by violation of consistency of abstraction and existence of a cause for a lost message.

Consistency of Abstraction An abnormal area contains at least one abnormal component. A contradiction arises if the area is detected to be abnormal but no faulty component is abduced. This constraint has only local character (i_am), since an agent cannot detect abnormal components of other areas.

$$
\begin{array}{ll}
CA: & \bot \leftarrow i\_am(N), ab(N), not\ has\_ab\_component(N). \\
& has\_ab\_component(N) \quad \leftarrow \quad area\_component(N, C), ab(C).
\end{array}
$$

Existence of a cause for a lost message The basic integrity constraint to start the diagnostic process states that it is contradictory to observe a lost message from node N to component C and not to have lost it on the route from N to C. The message is lost somewhere on this route route if at least one the involved nodes is abnormal:

$$
\begin{array}{llll}
CLM: & \multicolumn{3}{l}{\bot \leftarrow message\_lost(N, C), not\ lost\_on\_route(N, C).} \\
& lost\_on\_route(N, C) & \leftarrow & ab(N). \\
& lost\_on\_route(N, C) & \leftarrow & on\_route(N, C, M), ab(M).
\end{array}
$$

The following constraint allows us to abduce new observations. If a message is lost from N to C and M is a neighbor of N which is assumed to be abnormal by N, then N abduces the new observation that the message was lost on the way from M to C:

$$
\begin{array}{llll}
RO: & \bot & \leftarrow & message\_lost(N, C), on\_route(N, C, M), \\
& & & ab(M), not\ new\_message\_lost(M, C).
\end{array}
$$

4.2 The Agents' Reaction Rules

The reaction rules specify how the agents behave. Since the behavior depends on their diagnostic findings they need meta predicates to revise their knowledge base in the light of new observations. Based on the revisions three results are interesting

1. There is no diagnosis to explain the observation (*no_diags*).
2. There is a diagnosis that the agent itself is abnormal. In this case, since an agent knows its own state, other diagnoses are not of interest.
3. There are diagnoses which do not involve the agent itself (*next*). In this case the agent abduces a new, refined observation.

With the two meta predicates $no\_diags/1$ and $next/2$ we encode the agents' reaction rules:

If an agent receives an observation and has no explanation for it, the fault must be intermittent, since neither the agent itself is faulty nor are any neighbors to accuse. This is reported to the requesting agent:

$$\begin{array}{lcl} \mathsf{sendMsg}(intermittent\_failure(B), A) & \longleftarrow & \mathsf{recvMsg}(message\_lost(N, C), A), \\ & & no\_diags(message\_lost(N, C)), \\ \textit{Def.}\ 2.7.2 & & i\_am(B). \end{array}$$

If an agent receives an observation and is himself the cause of the problems it reports this fact back to the requesting agent:

$$\begin{array}{lcl} \mathsf{sendMsg}(responsible(B), A) & \longleftarrow & \mathsf{recvMsg}(message\_lost(N, C), A), \\ \textit{Def.}\ 2.7.1 & & i\_am(B), obs(down, B). \end{array}$$

If the agents area is not abnormal and there are diagnoses suspecting the agents neighbors, the newly abduced observation is sent to the suspected neighbor:

$$\begin{array}{lcl} \mathsf{sendMsg}(message\_lost(M, C), M) & \longleftarrow & \mathsf{recvMsg}(message\_lost(N, C), A), \\ & & i\_am(B), not\ obs(down, B), \\ \textit{Def.}\ 2.7.3 & & next(M, message\_lost(N, C)). \end{array}$$

In this case the agent has to remember to forward the final diagnosis result to the requesting agent:

$$\begin{array}{lcl} remember\_to\_reply\_to(A) & \longleftarrow & \mathsf{recvMsg}(message\_lost(N, C), A), \\ & & N \neq A, i\_am(B), not\ obs(down, B), \\ & & not\ no\_diags(message\_lost(N, C)). \end{array}$$

If an agent receives a diagnosis result from one of its neighbors and has to report the result to another neighbor, it forwards it:

$$\begin{array}{rcl} \mathsf{sendMsg}(intermittent\_failure(A), C) & \longleftarrow & \mathsf{recvMsg}(intermittent\_failure(A), B), \\ & & remember\_to\_reply\_to(C). \\ \mathsf{sendMsg}(responsible(A), C) & \longleftarrow & \mathsf{recvMsg}(responsible(A), B), \\ & & remember\_to\_reply\_to(C). \end{array}$$

After forwarding a diagnosis result, the "bookmark" to reply is removed from the agent's knowledge base:

$$
\begin{aligned}
neg(remember\_to\_reply\_to(C)) &\longleftarrow \mathsf{recvMsg}(intermittent\_failure(A), B),\\
&\qquad remember\_to\_reply\_to(C).\\
neg(remember\_to\_reply\_to(C)) &\longleftarrow \mathsf{recvMsg}(responsible(A), B),\\
&\qquad remember\_to\_reply\_to(C).
\end{aligned}
$$

4.3 Traces

To make the diagnosis process using the described knowledge base clearer, we consider the following scenario. Node n_1 sends a message to c_7, but the messages gets lost. Since n_1 does not receive an acknowledgment, a timeout mechanism informs n_1 that the message is lost and the diagnosis process starts. In the first scenario n_3 looses the message, whereas in the second one an intermittent failure occured.

Initially the creator process sends a start message to all nodes (1,2,3,4,9,14). The timeout mechanism informs n_1 of the lost message (5,6). Node n_1 knows that it is working fine and suspects the neighbor in charge of sending messages to c_7, namely n_2. Subsequently n_1 sends the refined observation that the message is lost from n_2 to c_7 to n_2 (8,10). Similarly n_2 informs n_3 (11,12,15). Additionally it remembers that it has to report the final result to n_1 (13). Finally, n_3 turns out to be the cause of the fault and the result is sent from n_3 to n_2 (16,17) and from n_2 to n_1 (18,19). n_2 removes the fact that it has to respond to n_1 (20).

1	n_2	$\longleftarrow$	$creator$	$start$
2	n_4	$\longleftarrow$	$creator$	$start$
3	n_1	$\longleftarrow$	$creator$	$start$
4	n_5	$\longleftarrow$	$creator$	$start$
5	n_1	$\longrightarrow$	n_1	$message\_lost(n_1, c_7)$
6	n_1	$\longleftarrow$	n_1	$message\_lost(n_1, c_7)$
7	n_1			$diag[ab(n_2), new\_message\_lost(n_2, c_7)]$
8	n_1	$\longrightarrow$	n_2	$message\_lost(n_2, c_7)$
9	n_6	$\longleftarrow$	$creator$	$start$
10	n_2	$\longleftarrow$	n_1	$message\_lost(n_2, c_7)$
11	n_2			$diag[ab(n_3), new\_message\_lost(n_3, c_7)]$
12	n_2	$\longrightarrow$	n_3	$message\_lost(n_3, c_7)$
13	n_2			$assimilates\, remember\_to\_reply\_to(n_1)$
14	n_3	$\longleftarrow$	$creator$	$start$
15	n_3	$\longleftarrow$	n_2	$message\_lost(n_3, c_7)$
16	n_3	$\longrightarrow$	n_2	$responsible(n_3)$
17	n_2	$\longleftarrow$	n_3	$responsible(n_3)$
18	n_2	$\longrightarrow$	n_1	$responsible(n_3)$
19	n_1	$\longleftarrow$	n_2	$responsible(n_3)$
20	n_2			$assimilates\, neg\, remember\_to\_reply\_to(n_1)$

In the second trace all nodes are ok at diagnosis time so the fault is intermittent. The initial phase is similar to the first trace. Only when n_3 comes up with no diagnoses (16), message of an intermittent failure is sent back.

1	n_2	$\longleftarrow$	$creator$	$start$
2	n_1	$\longleftarrow$	$creator$	$start$
3	n_5	$\longleftarrow$	$creator$	$start$
4	n_1	$\longrightarrow$	n_1	$message\_lost(n_1, c_7)$
5	n_1	$\longleftarrow$	n_1	$message\_lost(n_1, c_7)$
6	n_1			$diag[ab(n_2), new\_message\_lost(n_2, c_7)]$
7	n_1	$\longrightarrow$	n_2	$message\_lost(n_2, c_7)$
8	n_2	$\longleftarrow$	n_1	$message\_lost(n_2, c_7)$
9	n_6	$\longleftarrow$	$creator$	$start$
10	n_2			$diag[ab(n_3), new\_message\_lost(n_3, c_7)]$
11	n_2	$\longrightarrow$	n_3	$message\_lost(n_3, c_7)$
12	n_2			$assimilates\, remember\_to\_reply\_to(n_1)$
13	n_4	$\longleftarrow$	$creator$	$start$
14	n_3	$\longleftarrow$	$creator$	$start$
15	n_3	$\longleftarrow$	n_2	$message\_lost(n_3, c_7)$
16	n_3			$nodiagnoses$
17	n_3	$\longrightarrow$	n_2	$intermittent\_failure(n_3)$
18	n_2	$\longleftarrow$	n_3	$intermittent\_failure(n_3)$
19	n_2	$\longrightarrow$	n_1	$intermittent\_failure(n_3)$
20	n_1	$\longleftarrow$	n_2	$intermittent\_failure(n_3)$
21	n_2			$assimilates\, neg\, remember\_to\_reply\_to(n_1)$

5 Conclusion

We have defined an agent–based framework for the diagnosis of large spatially distributed technical systems. In this framework we assign an agent to every subsystem. This agent has detailed knowledge over its own subsystem and abstract knowledge over its neighbors. Using its declarative system description it can usually diagnose its own subsystem independently. Whenever it cannot detect a cause for an observed fault, it accuses a suitable neighboring subnet and starts cooperation with the responsible agent. This distributed framework leads to attractive algorithm complexity compared to a centralized solution, both concerning communication overhead and computational complexity.

Our implementation is based on the concepts of vivid agents and extended logic programming. The system description as well as the axioms needed for distributed diagnosis are formulated as extended logic programs. Reaction rules allow the flexible implementation of the communication among the agents, so that the cooperation can be tailored to all kinds of applications.

We have implemented the diagnosis of lost messages in a computer network as an example application and are planning to use the system for more complex engineering applications in the future.

6 Acknowledgements

We would like to thank Gerd Wagner, who worked on the implementation framework during a joint project and contributed most of the work presented in section 3. Furthermore we would like to thank Luís Moniz Pereira for his support. Financially, the work was partially supported by ESPRIT 8319 ModelAge and the German BMBF as well as the Portuguese JNICT.

References

[ADP95] J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.

[BG88] Alan H. Bond and Les Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1988.

[DNP94] Carlos Viegas Damásio, Wolfgang Nejdl, and Luís Moniz Pereira. REVISE: An extended logic programming system for revising knowledge bases. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Knowledge Representation and Reasoning*, pages 607–618, Bonn, Germany, May 1994. Morgan Kaufmann.

[DNPS95] C. V. Damásio, W. Nejdl, L. Pereira, and M. Schroeder. Model-based diagnosis preferences and strategies representation with meta logic programming. In Krzysztof R. Apt and Franco Turini, editors, *Meta-logics and Logic Programming*, chapter 11, pages 269–311. The MIT Press, 1995.

[DPS96] C. V. Damásio, L. Pereira, and M. Schroeder. Revise progress report. In *Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*. University of Sussex, Brighton, 1996.

[FN96] Peter Fröhlich and Wolfgang Nejdl. Resolving conflicts in distributed diagnosis. In *ECAI Workshop on Modelling Conflicts in AI*, 1996. To appear.

[GRS88] Allen Van Gelder, Kenneth Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceeding of the 7th ACM Symposium on Principles of Databse Systems*, pages 221–230. Austin, Texas, 1988.

[MH93] Igor Mozetič and Christian Holzbauer. Controlling the complexity in model–based diagnosis. *Annals of Mathematics and Artificial Intelligence*, 1993.

[Nej95] W. Nejdl, editor. *Proceedings of the 6th International Workshop on Principles of Diagnosis*, Goslar, Germany, October 1995.

[PA92] Luís Moniz Pereira and José Júlio Alferes. Well founded semantics for logic programs with explicit negation. In *B. Neumann (Ed.), European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, 1992.

[PAA91] Luís Moniz Pereira, José Júlio Alferes, and Joaquim Aparício. Contradiction Removal within Well Founded Semantics. In A. Nerode, W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 105–119, Washington, USA, June 1991. MIT Press.

[PAA93] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning*, 17(2, 3 & 4), 1993.

[PDA93] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 334–348, Lisboa, Portugal, June 1993. MIT Press.

[Pro94] G. Provan, editor. *Proceedings of the 5th International Workshop on Principles of Diagnosis*, New Paltz, NJ, October 1994.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[SdAMP96] Michael Schroeder, Iara de Almeida Móra, and Luís Moniz Pereira. A deliberative and reactive diagnosis agent based on logic programming. In *ATAL96 - Agent Theories, Architectures and Languages*. Springer–Verlag, 1996. To appear.

[Sho93] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[SW96] Michael Schroeder and Gerd Wagner. Distributed diagnosis by vivid agents. Technical report, Institut für Rechnergestützte Wissensverarbeitung, Universität Hannover, July 1996.

[Wag96a] Gerd Wagner. A logical and operational model of scalable knowledge-and perception-based agents. Proceedings of ModelAge96, Sesimbra, Portugal, 1996.

[Wag96b] Gerd Wagner. A logical and operational model of scalable knowledge-and perception-based agents. In *Proceedings of MAAMAW96, LNAI 1038*. Springer-Verlag, 1996.