Using Constraint Technology to Diagnose Configuration Errors in Networks Managed with SPECTRUM

Mihaela Sabin, Robert Russell, Eugene Freuder, and Ioan Miftode

Computer Science Department
University of New Hampshire
Durham, NH 03824, USA
{mcs,rdr,ecf,imiftode}@cs.unh.edu
(603) 862-3778

Abstract

The model-based reasoning approach to network management supports problem-solving capabilities, such as finding root causes of signaled faults. A distinct modeling technique that has recently emerged in the field of network management is the utilization of *constraints*. In industry, a prime example of the use of model-based reasoning in enterprise management is SPECTRUM[1]. The platform supports modeling the enterprise components, with their attributes, relations among components, and component behavior.

The main goal of our research project is to enhance SPECTRUM's management capabilities with constraint-based reasoning techniques. To demonstrate these techniques, we designed and implemented a prototype constraint-based architecture for SPECTRUM that focuses on configuration management. This work is a direct extension of previous results from diagnosing problems with Internet network services, such as FTP and DNS. The example considered in this project is a simple local area network monitored with SPECTRUM management tools. The network manager provides sets of specifications which are used to synthesize a constraint representation of the network configuration. This is used to automatically build a diagnostician which in turn uses constraint algorithms and actual data obtained from the running network to detect configuration faults.

Keywords: fault management, configuration management, model-based diagnosis, constraint satisfaction, diagnostic tools.

1 Introduction

Network service configuration is the process of assigning values to configuration variables in such a way as to enable a network service to operate according to the wishes of its managers. However, it is invariably the case that a choice for one variable has an effect on the subsequent choice that can be made for a different variable – the variables are seldom completely independent of one another. In effect, the values chosen for one variable are constrained by values assigned to other related variables. Configuration errors arise when the choices are inconsistent. Managers of large networks are confronted with hundreds if not thousands of configuration choices, and the constraints between them may not be obvious. Likewise, the inconsistencies produced by making wrong choices may not be immediately obvious, and may not manifest themselves in terms of operational faults until days or weeks after the erroneous choice was made.

Constraint-based network management tools do address this problem. Based on a model-based approach [Hamscher, Console, & de Kleer, 1992], *constraint technology* has been used to diagnose communication protocols [Riese, 1993] and special classes of randomly generated circuits [El Fattah & Dechter, 1995]. More recently, constraints have been applied to routing to solve bandwidth allocation planning [Frei & Faltings, 1999] and QoS-based multi-domain routing [Calisti & Faltings, 1999]. In our work, we have extended the problem domain to encompass fault management of Internet network services [Sabin, Russell, & Freuder, 1997], [Sabin *et al.*, 1999].

At the center of constraint technology lies the concept of the *constraint satisfaction problem (CSP)* [Freuder & Mackworth, 1994]. A CSP is defined by a set of variables, a set of value domains, and a set of constraints. Each variable has an associated domain of values that can be assigned to that variable. A constraint is defined on a subset of variables and expresses the combinations of values the variables are allowed to take. To solve a CSP means to find value assignments for all variables such that all constraints are satisfied.

It is natural to cast network service configuration problems in the CSP framework by defining a CSP variable for each network service variable such that the domain of the CSP variable is the set of values that a manager is allowed to assign to that network service variable. The constraints are

[1]Trademark of Aprisma

then the relationships between the various choices for the variables. A configuration error or fault occurs when one or more of the constraints is violated.

In industry, a prime example of the model-based reasoning approach applied to network management is SPECTRUM[Lewis, 1999]. SPECTRUM's auto-discovery tool constructs the enterprise model that can then be used in real-time to perform event correlation or other management tasks. In this paper we show how a constraint-based architecture can be automatically constructed and then operated to diagnose configuration errors. To construct and operate the constraint-based tool we access and query SPECTRUM's database.

The main goal of this project is to enhance SPECTRUM's management capabilities with constraint-based reasoning techniques. This work is a direct extension of previous results from diagnosing problems with Internet network services, such as FTP and DNS [Sabin, Russell, & Freuder, 1997], [Sabin *et al.*, 1999]. The example considered in this paper is a simple local area network monitored with SPECTRUM management tools. The network manager provides sets of specifications which are used to synthesize a constraint representation of the network configuration. This is used to automatically build a diagnostician which in turn uses constraint algorithms and actual data obtained from the running network to detect configuration faults.

The paper is organized as follows. In the next section we present a sample management problem dealing with DNS name service configuration. We use this example as a running example throughout the paper to describe our approach to fault management. In Section 3 we introduce the CSP paradigm and discuss its applicability to configuration error diagnosis. We then elaborate on the construction and operation phases that support our constraint-based solution. Section 4 describes the constraint-based architecture and its integration with SPECTRUM. Section 5 summarizes the contribution of our work and outlines directions for future work.

2 Sample Problem: Name Service Configuration

As a simple but illustrative example, consider the situation encountered by every network administrator who must manage a local area network (LAN). Among other tasks, he is responsible for the assignment of IP addresses and DNS names to every node in the LAN, including server nodes, client nodes, router nodes, etc. In order for the network to function and be managed effectively, these assignments are made known through the use of various software tools. Consequently, the information defining these assignments is often replicated in several different locations and formats in the network.

There is a very real potential for these various repositories to contain contradictory information as the LAN's configuration changes. For example, binding an IP address to a DNS name is usually enforced by a DNS name server whose database stores that binding. An identical binding is replicated, stored and used by integrated management platforms, such as SPECTRUM. SPECTRUM's database maintains updated configuration information about all the network devices and services currently running in the network. If a change occurs in the network and is not consistently reflected in all the databases, then the LAN configuration information is contradictory. This can easily happen in relatively large LANs where different administrators are responsible for configuration settings made by the different network and management tools.

The problem can also manifest itself at many levels that are often difficult to trace. For example, as discussed in an earlier work [Sabin, Russell, & Freuder, 1997], DNS name resolution can be done using files local to the host requesting the resolution, or using files residing on a name server that is accessed remotely from that host. If these files contain contradictory information, it can often be hidden until, for example, the type of resolution is switched from global to local during infrequently occurring situations.

Our prototype system is designed to solve the following generalization of this simple configuration example (see Figure 1):

Given

- a set of IP addresses for which the network administrator wants to check the name configuration, and
- any number of different sources that define the bindings between IP addresses and DNS names,

Detect any and all configuration inconsistencies before they lead to operational faults.

3 Diagnosis with Constraint Technology

In this section we present our constraint-based solution according to the following organization: Section 3.1 starts with a brief introduction to the constraint satisfaction problem (CSP) paradigm. Section 3.2 gives details about the two steps in the *Construction Phase*:

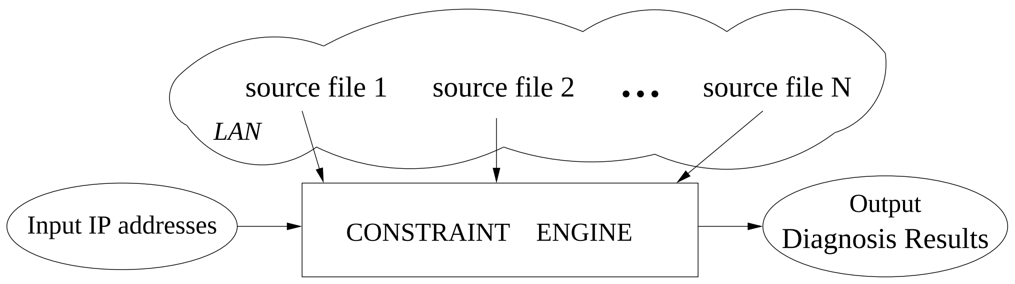


Figure 1: Constraint-based diagnosis of configuration errors with a LAN name service

1. the automatic synthesis of a constraint representation or a *constraint source* for the above sample problem, and

2. the compilation of a *constraint engine* by applying the approach we presented in [Sabin, Russell, & Freuder, 1997].

Section 3.3 presents the *Operational Phase*, in which we show the constraint engine "in action": how constraint algorithms and probing tools work to detect and report configuration inconsistencies. Section 3.4 shows some results corresponding to the example introduced in section 2 to illustrate the two phases. Section 3.5 concludes with a more general solution in which we consider parameterized probing and more than two sources to check configuration validity.

3.1 Constraint Satisfaction Problem Paradigm

CSP framework is model-based. To diagnose a system within the CSP framework one must first represent the model of the system structure and correct behavior. The structural elements of the system being diagnosed are modeled as CSP variables, and the behavioral relationships among the constituent components are expressed as constraints.

Applied to fault and configuration management of network services, the building blocks of the network service model are the management object descriptions associated with network devices and modeled as CSP *variables*. The diagnostically meaningful dependencies among the network devices are modeled as *constraints*. The diagnosis scheme employs a CSP diagnostic engine which outputs the expected diagnoses based on the discrepancies between the predictions of the CSP representation and the observations from the running network service. The diagnosed network service is found faulty if some constraints can not be satisfied, in which case the violated constraints are precise indicators of the cause of the fault.

In our work in the area of diagnosis we use two extensions to the classical paradigm. First, if a fault exists, the CSP algorithm does not settle for "no solution found". Instead, it records the partial solutions as valid diagnoses for which some constraints are violated [Freuder & Wallace, 1992]. Second, the distributed nature of network services and applications, as well as the dynamic configuration changes cannot impose a predefined set of participating network elements. The relevant or active portions of the model that include observed symptoms and actual configuration data can be isolated at run-time. This is the mechanism of activity control proposed in [Mittal & Falkenhainer, 1990]. Third, we embed network probing tools in the CSP representation to get dynamic values for the CSP variables. Details about how the CSP paradigm is applied to network management can be found in [Sabin, Russell, & Freuder, 1997] and [Sabin *et al.*, 1999].

The CSP paradigm has proven its applicability in many application domains. Besides diagnosis, other examples are: scheduling, planning, resource allocation, and product configuration. This illustrates one essential strength of the CSP paradigm: representing problems in terms of constraints is the natural language of discourse for expressing many real-world problems. Moreover, the declarative character of the CSP language has the advantage of totally decoupling the CSP representation phase from the solving phase. Thus, the user of constraint-based diagnostic tools is not concerned with implementing CSP algorithms, but focuses on the representation of the diagnosis problem. Another advantage of the CSP paradigm is the wealth of thoroughly examined CSP algorithms, from which the most effective for the problem at hand can be chosen.

3.2 Construction Phase

The following scenario exemplifies a diagnostic task for the sample problem introduced in Section 2:

1. The network administrator wants to diagnose the configuration of the name service for the IP address `132.177.12.157`.

2. He examines two source files which have information about the names configured for the given IP address. The two source files are `SPECTRUM_source` and `DNS_source`, and are obtained from querying the SPECTRUMplatform and DNS server databases.

3. Access to the source files is provided by two probing functions: `resolveIP_SPECTRUM` and `resolveIP_DNS`. Each probing function looks up the IP address in the corresponding source file and returns the configured name, if any.

4. If both probes return the same name, then name configuration specified in the two sources is consistent. Otherwise, there are four error cases:

   case I: neither `SPECTRUM_source`, nor `DNS_source` has a configured name for the given IP,

   case II and III: either `SPECTRUM_source` or `DNS_source` has the name, but not both sources, and

   case IV: the names found in the two sources are different.

The CSP formulation of the above scenario defines the same diagnostic task (Figure 2) in CSP terms, i.e. variables, values, and constraints. Thus, we identify three CSP variables: `V0` corresponds to the IP address, and `V1` and `V2` represent the results of querying the `DNS_source` and `SPECTRUM_source` for the name assigned to the IP address. The value for `V0` is predefined to "132.177.12.157". The values for `V1` and `V2` are dynamically acquired from the two source files by "asking" the two probing functions to get those values. The CSP formalism uses `DEF` directives to associate predefined values to variables, and `ASK` directives to get on-line values from the running network.

The dynamic values are not the only dynamic aspect in our example. Variable `V1` and `V2` are activated through `ARV` (always require variable) activity constraints. The role of the activity constraints is to extend or reduce the CSP representation (variables and constraints) in a dynamic fashion, based on on-line changes observed in the running network. In our example, the presence of the `V0` variable stands for the fact that there is an IP address whose name configuration the network administrator needs to check. Thus, `V0` "triggers", or activates, the addition of variables `V1` and `V2`. The CSP formalism uses two `ARV` constraints to represent the dynamic addition of the two variables. Finally, once they are active, a compatibility check of their values can be performed. CSP uses `CON` statements to express constraints that restrict the value assignments the "constrained" variables can simultaneously take.

Figure 2 also shows that the network administrator is warned when either an `ASK` directive (value cannot be obtained dynamically), or a `CON` directive (compatibility check failed) is violated. That is why we associate diagnostic messages with the `ASK` and `CON` statements in the CSP code.

How do we use the CSP code to perform automatic diagnosis? In our previous work we showed that constraint technology can be used to generate diagnostic tools for network services. Figure 3 illustrates the architecture of the prototype system ADNET presented in [Sabin, Russell, & Freuder, 1997]. A constraint compiler translates the CSP code into a C++ source. The resulting C++ source encodes the diagnosis procedure for solving the original CSP problem. The C++ source is then compiled and linked to produce a constraint engine capable of probing and CSP solving. Probing capabilities are specified in the CSP code through `ASK` directives, and made available to ADNET in a probing library. Solving capabilities are made available in a constraint library.

In this paper we take the automation process of generating diagnostic tools a step further. The challenge we address is to assist the network administrator with writing the constraint code. This way we achieve two objectives: we shield the network administrator from

1. the implementation details of the ADNET constraint algorithms, and

2. the syntax details of the ADNET constraint language.

The sample problem in Section 2 can be formulated generically as a constraint template (Figure 4). `V0` deals with *IP input values* the network administrator is interested in verifying. `V1` and `V2` probe the files *source_file_1* and *source_file_2* using the probes *probe_1* and *probe_2*. The rest of the template resembles the generated constraint code, and shows the activation of `V1` and `V2` and the equality constraint between them.

We observe that the constraint template is parameterized in: the input from the network administrator, the probe function prototypes, and the source file names. This information is sufficient to synthesize the actual constraint code from the constraint template. The input and output information for the representation synthesis prepass is shown in Figure 5. The representation synthesis prepass module and the ADNET module form the construction phase.

```
VAR V0       DEF ''132.177.12.157''
VAR V1       ASK resolveIP_DNS(''DNS_source'', $V0)
    ''*** $V0: nonexistent IP in the DNS database''
VAR V2       ASK resolveIP_SPECTRUM(''SPECTRUM_source'', $V0)
    ''*** $V0: nonexistent IP in the SPECTRUM database''
START V0
ARV V0 => V1
ARV V0 => V2
CON $V1 EQUAL $V2
    ''*** Inconsistent names for IP=$V0, $V1 different from $V2''
```

Figure 2: Generated constraint code for checking name configuration consistency in the DNS and SPECTRUM databases

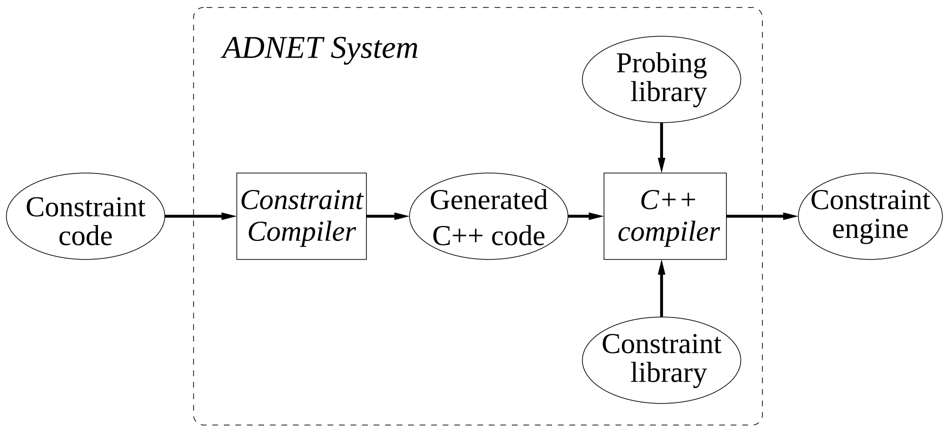
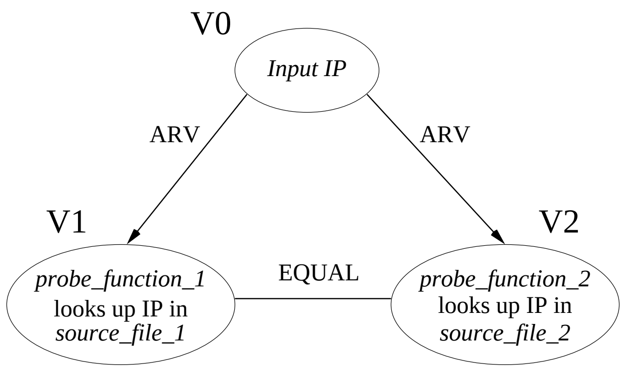


Figure 3: Architecture of the Automatic Diagnosis for Network Services (ADNET) prototype system


V0
Input IP
ARV
ARV
V1
V2
EQUAL
probe_function_1 looks up IP in source_file_1
probe_function_2 looks up IP in source_file_2

```
VAR V0 IP input value
VAR V1 ASK probe_1(source_file_1, V0)
VAR V2 ASK probe_2(source_file_2, V0)

START V0

ARV V0 => V1
ARV V0 => V2

CON V1 EQUAL V2
```

Figure 4: Graphical and text representation of the constraint template

3.3 Operational Phase

The result of the construction phase is a constraint engine which operates on a LAN to diagnose problems with the name configuration. We call the *Operational Phase* the process by which the network administrator actually uses the constraint engine to find the diagnosis results.

The diagram in Figure 6 gives an overall view of the operational phase when two databases are used. It also highlights the principle of model-based diagnosis employed by the constraint technology. Probing the source files and prompting the network administrator to get management information results in a set of *observations*. Using the constraint representation to encode the diagnostic procedure results in a set of *predictions*. The task of the constraint algorithm is

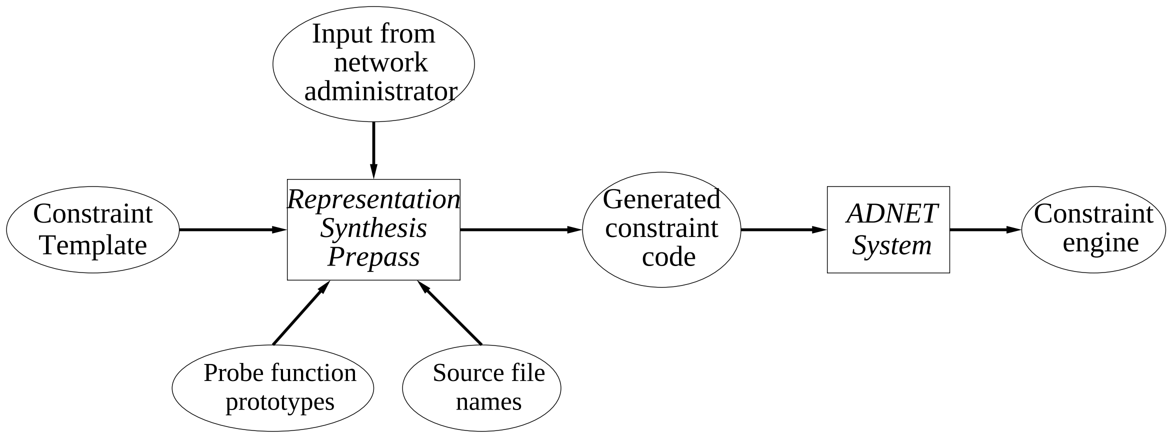

Input from network administrator
Constraint Template
Representation Synthesis Prepass
Generated constraint code
ADNET System
Constraint engine
Probe function prototypes
Source file names

Figure 5: Construction Phase: generate a constraint engine from a constraint template of the diagnosis problem

to find discrepancies between observations and predictions. These discrepancies correspond to the constraint violations the algorithm records during the solving process. The output diagnosis results contain diagnostic messages from the generated constraint code, which are paired with the violated constraints.

3.4 Diagnosis Results

Figure 7 shows the IP and name bindings contained in the two source files obtained from the DNS and SPECTRUM databases respectively. Figure 8 shows the text of the generated constraint code in which the input IP addresses become predefined values for the `V10` variable. The probing functions dynamically "ask" for name values from the source files whose names are shown. Figure 9 shows the execution trace using the input data from Figure 7. The diagnoses generated are indicated by lines starting with "***", and are summarized in Figure 10.

3.5 Generalization

In this section we briefly consider the straight-forward generalization of the previous example to an arbitrary number of databases, where consistency is required among all of them.

The diagram in Figure 11 a) gives an overall view of the generalized operational phase. There is one probe function corresponding to each database involved in the constraints. Figure 11 b) shows the representation and constraints for a typical network node, with a given IP address, in which equality is required for the corresponding name entry in each database. During the construction phase, this representation is replicated for each IP address.

Two additional phases are envisioned but not yet implemented. One would allow dynamic alteration of the CSP representation, and one would allow not only reporting of configuration errors but also their automatic correction.

4 Design of the Constraint-Based Architecture

The design and development of a constraint-based architecture and its integration with SPECTRUM includes the following steps:

1. Identify a sample configuration problem using a local area network which has SPECTRUM installed and in operation.
2. Develop the CSP representation for the configuration problem of interest.
3. Develop the auxilliary probing functions needed by the CSP engine to check constraints. These functions must interact with SPECTRUM to obtain network data dynamically.
4. Diagnose the configuration problem with a constraint engine that finds minimal sets of violated constraints.

What distinguishes this architecture from the other constraint-based management tools we developed previously is the delegation of probing functions to the SPECTRUM management platform. There are two approaches to allow for exchange of data between SPECTRUM and the constraint-based architecture:

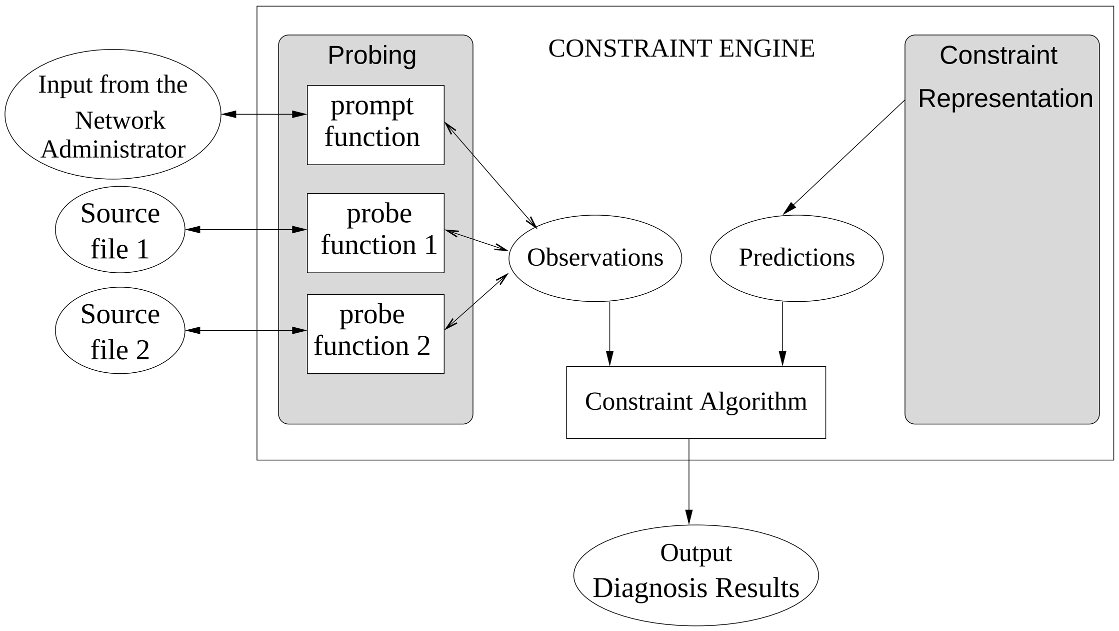


Figure 6: Operational phase uses two source files and input from the network administrator to produce the diagnosis results. Probing and the constraint representation feed the constraint algorithm with observations and predictions.

```
       DNS_source file                        SPECTRUM_source file

132.177.4.157 birch.cs.unh.edu        132.177.4.157 birch.cs.unh.edu
132.177.4.159 mint.cs.unh.edu         132.177.4.158 garnet.cs.unh.edu
132.177.4.4 csrouter.cs.unh.edu       132.177.4.159 lint.cs.unh.edu
132.177.4.30 lava.cs.unh.edu          132.177.4.4 csunhrouter.cs.unh.edu
                                      132.177.4.26 phub1.cs.unh.edu
```

Figure 7: IP and name bindings in the two source files obtained from the DNS and SPECTRUM databases

```
VAR V10
    DEF {"132.177.4.158", "132.177.4.159", "132.177.4.30", "128.177.4.30"}
VAR V11
    ASK resolveIP_DNS( "DNS_source", $V10 )
VAR V12
    ASK resolveIP_SPECTRUM( "SPECTRUM_source", $V10 )
```

Figure 8: Text of the generated constraint code.

- through the Platform External Interfaces (PEIs), by using interfacing tools to transfer data to/from files;

- through the Application Programming Interfaces (APIs), by defining new C++ objects that are compiled into SPECTRUM.

While the API approach allows integration of an external application at the source level, the PEI approach uses a set of procedures and interfaces available both in SPECTRUM and the external application. A viable method of integration combines the best of both approaches: rapid prototyping with PEIs, and then a better performance solution using the API approach. We therefore decided to start with the PEI approach for our prototype. Use of the API approach is discussed in section 5 on future work.

```
V10 <- "132.177.4.158"
  V11 <- "unknown"
    V12 <- "garnet.cs.unh.edu"
V10 <- "132.177.4.159"
  V11 <- "mint.cs.unh.edu"
    V12 <- "lint.cs.unh.edu"
V10 <- "132.177.4.30"
  V11 <- "lava.cs.unh.edu"
    V12 <- "unknown"
V10 <- "128.177.4.30"
  V11 <- "unknown"
Found 3 minimal diagnose(s):
  *** - "132.177.4.158": nonexistent IP in the DNS database
    V10 -- "132.177.4.158"
    V11 -- "unknown"
    V12 -- "garnet.cs.unh.edu"
  *** - Inconsistent names for IP="132.177.4.159",
        "mint.cs.unh.edu" different from "lint.cs.unh.edu"
    V10 -- "132.177.4.159"
    V11 -- "mint.cs.unh.edu"
    V12 -- "lint.cs.unh.edu"
  *** - "132.177.4.30": nonexistent IP in the SPECTRUM database
    V10 -- "132.177.4.30"
    V11 -- "lava.cs.unh.edu"
    V12 -- "unknown"
```

Figure 9: Execution trace for the input data in Figure 7

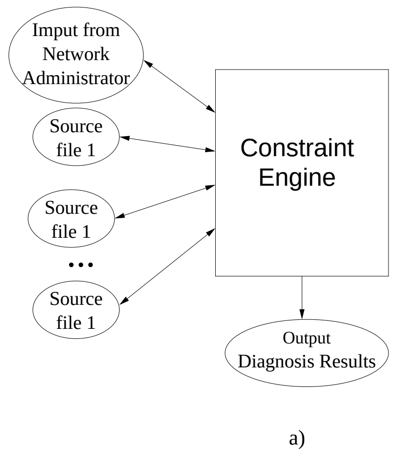
```
Found 3 minimal diagnose(s):
  *** - "132.177.4.158": nonexistent IP in the DNS database
  *** - Inconsistent names for IP="132.177.4.159",
        "mint.cs.unh.edu" different from "lint.cs.unh.edu"
  *** - "132.177.4.30": nonexistent IP in the SPECTRUM database
```

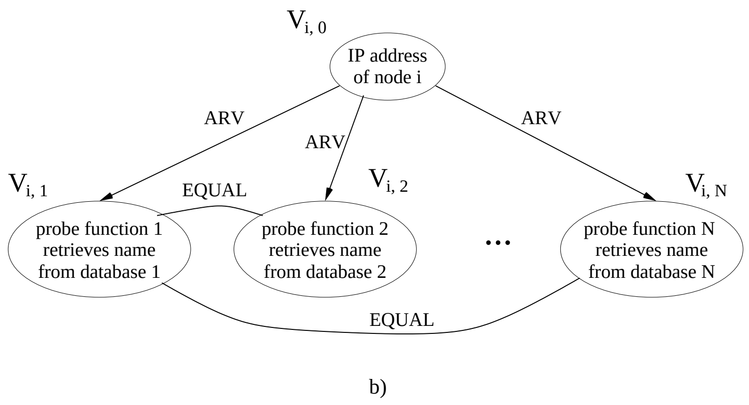
Figure 10: Results for the input data in Figure 7

The integration of our constraint-based application for managing network configuration requires retrieval of SPECTRUM data and synthesis of a constraint model of configuration management. The PEI interfacing tool that allows access to and retrieval of network topology data is the SPECTRUM *Topology Export* tool. The tool is implemented by using the SPECTRUM *Command Line Interface (CLI)*, a database query language that enables retrieval of information from the SPECTRUM database. This database collects the results of the SPECTRUM's automatic topology mapping facility, known as the AutoDiscovery mechanism. AutoDiscovery finds devices on the network and creates corresponding models in the database. Using the Topology Export tool information from the database can be stored in a file for use by the constraint-based configuration diagnostician.

5 Conclusion and Future Work

This paper describes the design and implementation of a prototype constraint-based architecture to enhance SPECTRUM's configuration management capabilites with constraint-reasoning technology. This work is a direct extension of previous results diagnosing problems with Internet network services, such as FTP and DNS [Sabin, Russell, & Freuder, 1997], [Sabin *et al.*, 1999]. The example considered is a simple local area network monitored with SPECTRUM management tools. We use constraints to synthesize a constraint representation of the network configuration, and then solve possible configuration problems with constraint algorithms. The constraint-based architecture ac-





V i, 0
IP address of node i
ARV
ARV
ARV
V i, 1
V i, 2
V i, N
EQUAL
probe function 1 retrieves name from database 1
probe function 2 retrieves name from database 2
...
probe function N retrieves name from database N
EQUAL
b)

Figure 11: Operational phase and constraint template for an arbitrary number of source files

cesses the SPECTRUM management database to obtain the current values for configuration parameters in order to detect faults.

There are a number of extensions to be made to our prototype system. The first would be to improve the matching of DNS names. Currently our equality constraints for DNS names utilize exact character for character string matching. However, it is common for host names to be used in some places and fully qualified domain names in others. Therefore, it would be convenient for “lava”, “lava.cs”, “lava.cs.unh” and “lava.cs.unh.edu” to all be accepted as equal, but “lav”, “lav.cs.u”, etc. to be rejected. This can be easily handled since the test for constraint satisfaction is an arbitrary boolean function that can be defined by the user.

The second improvement to the prototype would be to define probe functions that perform real-time interrogation of active processes, such as a DNS server, a running SPECTRUM manager, etc. The current probe functions simply search files that have been prepared off-line, but the probe mechanism is completely general, and our earlier work [Sabin, Russell, & Freuder, 1997] demonstrated the utility of real-time probe functions. This will require use of the SPECTRUM *Developer’s Tools*, which provide C++ interfaces that extend SPECTRUM’s functionality and enable developers to integrate new C++ objects and programs with SPECTRUM.

A third improvement would be to revise the construction phase so that it is performed as part of running the DCSP engine, rather than as a prepass occurring before compilation. This will require extensions to the DCSP language to permit dynamic instantiation of an arbitrary number of components as determined by the number of values in a given domain. This research will require an extension along the lines being investigated for Composite CSP.

Another area of future work is to extend the problem domain beyond simple DNS name to IP address mapping in order to encompass other areas of configuration management, such as connectivity, topology, routing, quality of service provisioning, etc.

Acknowledgements

This work was supported in part by Aprisma, Inc.

References

[Calisti & Faltings, 1999] Calisti, M., and Faltings, B. 1999. A distributed approach for QoS-based multi-domain routing. In *Proceedings of the AAAI’99 Workshop on Artificial Intelligence in Distributed Information Networks (AiDIN’99)*.

[El Fattah & Dechter, 1995] El Fattah, Y., and Dechter, R. 1995. Diagnosing tree-decomposable circuits. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1742–1748.

[Frei & Faltings, 1999] Frei, C., and Faltings, B. 1999. Bandwidth allocation planning in communication networks. In *Globecom’99*.

[Freuder & Mackworth, 1994] Freuder, E., and Mackworth, A., eds. 1994. *Constraint-Based Reasoning*. MIT Press.

[Freuder & Wallace, 1992] Freuder, E., and Wallace, R. 1992. Partial constraint satisfaction. In *Artificial Intelligence*, 21–71.

[Hamscher, Console, & de Kleer, 1992] Hamscher, W.; Console, L.; and de Kleer, J., eds. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann Publishers.

[Lewis, 1999] Lewis, L. 1999. Event correlation in SPECTRUM and other commercial products. Technical Report ctron-Imp-99-05, Cabletron.

[Mittal & Falkenhainer, 1990] Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 25–32.

[Riese, 1993] Riese, M. 1993. *Model-based diagnosis of Communication Protocols*. Ph.D. Dissertation, Swiss Federal Institute of Technology, Lausanne.

[Sabin *et al.*, 1999] Sabin, M.; Bakman, A.; Freuder, E. C.; and Russell, R. D. 1999. A constraint-based approach to fault management for groupware services. In Sloman, M.; Mazumdar, S.; and Lupu, E., eds., *Integrated Network Management VI*. IEEE Publishing.

[Sabin, Russell, & Freuder, 1997] Sabin, M.; Russell, R. D.; and Freuder, E. C. 1997. Generating diagnostic tools for network fault management. In Lazar, A. A.; Saracco, R.; and Stadler, R., eds., *Integrated Network Management V*. Chapman & Hall.