CBR Support for CSP Modeling of InterOperability Testing

Mohammed H. Sqalli and Eugene C. Freuder
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
msqalli,ecf@cs.unh.edu

Abstract

In this paper, we suggest diagnosing InterOperability Testing problems by integrating Constraint-Based and Case-Based Reasoning. We model the problem as a *Constraint Satisfaction Problem* (CSP), then *Case-Based Reasoning* (CBR) is used to compensate for what is missing in this model. CBR supports the process of learning by supplying the case-base with new cases that can be used to solve future similar problems. CBR is also used to update the CSP model, and make it more robust for solving more problems. The domain we are using is *InterOperability Testing* of protocols in ATM (Asynchronous Transfer Mode) networks.

Introduction

In this work, we suggest diagnosing interoperability testing problems by integrating two modes of reasoning: constraint-based and case-based. The first step is modeling our system as a *Constraint Satisfaction Problem* (CSP). CSP has proven very useful in many applications including diagnosis of protocols, and interoperability testing. The model of any system may be insufficient for solving all the problems, because the model may be incomplete or incorrect. We propose to deal with incompleteness and incorrectness by using a case-base of the specific system. Previous versions of this paper appeared as (Sqalli & Freuder 1998a) and (Sqalli & Freuder 1998b).

It was shown in (Sqalli & Freuder 1998a) that modeling can be categorized using two parameters: the complexity of the system to be modeled and the status of the knowledge in these systems (complete vs. incomplete).

One example of simple systems with incomplete and/or incorrect knowledge is interoperability testing of protocols. Protocol specifications are written by experts but may have flaws and be incomplete, which may lead to non-interoperability of devices. In addition, if many protocols are running at the same time between two devices, they may cause the wrong behavior of one protocol due to the external interactions with the other.

A model is *incomplete* if it is missing some knowledge about the system's behavior. A model is *incorrect* if it represents wrong knowledge. An example of an incomplete model is a CSP problem where a constraint or a variable is missing. An example of an incorrect model is a CSP problem where a constraint is incorrect.

Models can be incomplete because they represent the behavior of a specific system and may not include all the interactions with the external world. In addition, models can be incorrect because they may represent systems that are not well-defined or contain flaws (mistakes, bugs, errors, ...), because these systems are the outcome of imperfect human thinking.

The problem domain presented in this paper is motivated by the work we are doing with the InterOperability Lab (IOL) at the University of New Hampshire. One mission of IOL is to provide testing services for vendors of computer communications devices.

Interoperability testing is testing whether two devices connected to each other and implementing the same protocol are operational. This is done by monitoring the data between the two devices using an analyzer, and then comparing the data observed with what is expected (what is stated in the specifications of the protocol tested).

One of the main challenges at IOL is how to debug and diagnose interoperability problems in a timely manner. At the present, these tasks are done by the experts who work at IOL, and sometimes it becomes exhausting to check the data manually and try to determine what the problem is.

Some of the difficulties that the IOL is having are:
1- checking a large amount of data to find out where there is a mismatch between what is expected and what is observed.
2- Wasting time in solving problems that the IOL has solved before or in solving problems that are very similar to old problems solved.

Test suites have been written to help in diagnosing the interoperability problems. But, using a test suite manually does not solve the above mentioned problems.

This shows that there is a need to make the process of debugging interoperability problems easier, quicker and more efficient. The work we propose in this paper is tuned toward solving some of these problems by automating the process of running test suites and diag-

nosing interoperability problems.

Related Work

(Karamouzis & Feyock 1992) show that the integration of CBR and Model-Based Reasoning (MBR) enhances CBR by the addition of a model that aids the processes of matching, and adaptation; and it enhances MBR by the CBR capacity to contribute new links into the causality model.

In (Purvis & Pu 1995), case adaptation process in assembly planning problems was formalized as a CSP. Each case is represented as a primitive CSP, and then a CSP algorithm is applied to combine these primitive CSPs into a globally consistent solution for the new problem. CBR is used to fill the values of the problem, then CSP is used to make the problem consistent.

In (Portinale & Torasso 1995), it is stated that approaches combining MBR and CBR can be roughly classified into two categories: approaches considering CBR as a speed-up and/or heuristic component for MBR, and approaches viewing CBR as a way to recall past experience in order to account for potential errors in the device model. Their proposal was in the first category by means of the development of ADAPtER, a diagnostic system integrating the model-based inference engine to AID (a pure model-based diagnostic system), with a case-based component intended to provide a guide to the abductive reasoning performed by AID.

(Lee *et al.* 1997) developed a case and constraint based project planning expert system for apartment domain. This large scale, case based and mixed initiative planning system integrated with intensive constraint-based adaptation utilizes semantic level meta-constraints and human decisions for compensating incomplete cases embedding specific planning knowledge. The case and constraint based architecture inherently supports cross-checking cases with constraints during the system development and maintenance.

(Hastings, Branting, & Lockwood 1995) describe a technique for integrating CBR and MBR to predict the behavior of biological systems characterized both by incomplete models and insufficient empirical data for accurate induction. They suggest the exploitation of multiple, individually incomplete, knowledge sources to get an accurate prediction of the behavior of such systems. They state that precise models exist for the behavior of many simple physical systems. However, models of biological, ecological, and other natural systems are often incomplete, either because a complete state description for such systems cannot be determined or because the number and type of interactions between system elements are poorly understood. In this paper, MBR is mainly used to determine values for variables in cases, and compute new values from old cases' values. MBR is used for the adaptation of cases (MBR is used within the CBR formalism).

Our focus in this paper is to deal with interoperability testing and show how we can get better results by enhancing the CSP model with the case-base reasoner. First, CSP is used to solve the problem. If the CSP model is insufficient, then CBR is used. This way CBR will not be used unless CSP fails. The result obtained from the CBR is then used to update the model. This is similar to what has been done in integrating CBR and MBR to update causality models. The difference is that we are using CSP models, taking advantage of the CSP representation and applying that to the interoperability testing domain.

InterOperability

InterOperability Testing of the PNNI Protocol in ATM Networks

Asynchronous Transfer Mode (ATM) has emerged as a networking technology capable of supporting all classes of traffic (e.g. voice, video, data). ATM is a connection-oriented technology that uses fixed-size cells, and can guarantee certain quality of service (QoS) requested by the user.

PNNI (Private Network Network Interface) protocol provides dynamic routing, supports QoS, hierarchical routing, and scales to very large networks (ATM 1996). Two switches running PNNI are able to send data to each other either via direct link or by using a route. The PNNI protocol is composed of PNNI routing that includes discovery of the network topology to become ready to route to different points, and PNNI signaling which is responsible for dynamically establishing, maintaining and clearing ATM connections between two ATM networks or nodes (ATM 1996). The PNNI routing protocol starts when the link is up, and every switch should send HELLO packets (information about itself) during the Hello Protocol phase.

InterOperability Testing in networks is used to ensure that a device does what it is intended for. It is meant to supplement conformance testing by verifying that the end-to-end behavior of devices is compatible with the protocol specifications. This work is focused on testing protocols that run over ATM networks, and the examples used are taken from the PNNI protocol.

For our purposes, interoperability testing of PNNI allows us to detect any problems that arise when two switches supporting the PNNI protocol are connected. The network can be large with many switches connected. But, for simplicity we propose to work on a two-switch network and perform interoperability testing on them. We suppose that the two switches have passed conformance testing. We base our work on the BTD-TEST-pnni-iop.000.000 document which provides the test suite for performing PNNI interoperability testing (ATM 1998).

The monitor gets all the data (observations) necessary to test the interoperability of the devices hooked to it. An observation is the data representing an event that occurred. After we get the results of monitoring all the traffic between the two switches, we want to analyze the data obtained and determine if both devices

are interoperable.

Representation

We propose to use the CSP formalism to analyze the data and test the interoperability of the device. CSP provides more flexibility in the representation of the events and constraints that must be satisfied.

Modeling the entire protocol may be a costly way of approaching this particular problem (Sqalli & Freuder 1996), since the model must include all the information found in the protocol specifications. In addition, interoperability testing is usually presented as a test suite (ATM 1998). A test suite is a collection of tests. Each test provides the mechanisms for testing a particular phase or component of the protocol. We propose a simple way of modeling the protocol by using sub-models where each sub-model represents one test. Tests are written in an incremental way, and some are subsets of others. Running all tests then would be the same as testing the whole protocol. The advantage of this is to simplify the representation and be able to pinpoint problems at a smaller scale.

CBR support for the CSP model

CSP is a powerful and extensively used artificial intelligence paradigm (Freuder & Mackworth 1992). CSPs involve finding values for problem variables subject to restrictions on which combinations of values are acceptable. A constraint graph is a representation of the CSP where the vertices are variables of the problem, and the edges are constraints between variables. Each variable has labels which are the potential values it can be assigned. CSPs are solved using search (e.g. backtrack) and inference (e.g. arc consistency) methods. CSP representations and methods will be used for modeling our interoperability problem since they provide a powerful tool in this case.

CBR is very useful when there is enough empirical data for accurate induction. It is composed of four main steps: case matching, case retrieval, case adaptation, and case storage. CBR uses a case-base where it stores learned cases. A case is usually composed of a description of a problem, and a solution to it. Whenever there is a new problem, it is matched to what is already in the case-base using similarity metrics. Then, the useful cases are retrieved and adapted to the new problem to provide a solution. The new case (problem and its solution) will be stored in the case-base if it provides new information.

In this paper, the main focus is on the CBR/CSP interface and how CBR can be used to deal with incompleteness and incorrectness in the CSP model.

Approach

In figure 1, we show how CBR and CSP are combined to deal with incompleteness and incorrectness.

The first step consists of modeling the protocol specification into a CSP representation. The protocol specification is taken from the test suite that represents it. Each test in the test suite is modeled and tested separately.

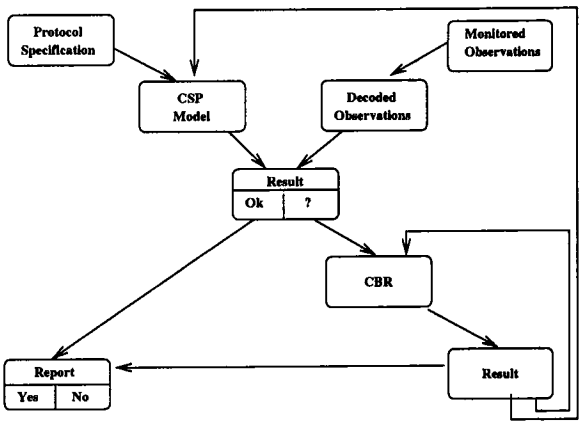

Figure 1. Integration of CSP Model and CBR for InterOperability Testing.

Each event has many parameters. For example, the Hello event has Node_ID, PG_ID, ..., Source, Time, Status as parameters, and some of these parameters are shown in figure 2. The CSP graph shown in figure 2 is the model of a test case taken from the BTD-TEST-pnni-iop.000.000 document (ATM 1998).

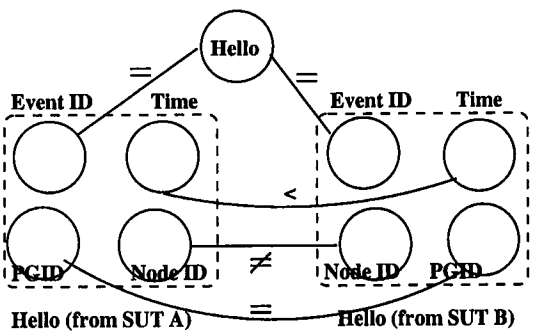

Figure 2. CSP Model of Test Case ID: V4201H_001

Each of these parameters represents a vertex (variable) in the CSP graph. The labels in each vertex (values inside the vertex) are the values a variable may take, called the variable's domain. When no labels are present, the variable can be assigned any value. The constraints may be either unary, binary, ternary, etc. The unary constraints are the restrictions on the variable's domain. For example, in the Hello(A) event, the variable Status can be assigned the value '1' only, meaning this event is mandatory; in other optional events, the associated variable may take either value '0' or '1'. In the same way, the Source can be either (A) or (B) representing the device which sent this Hello event. The binary constraints are restrictions on the relation between two variables' domains. For example, there is a $<$ constraint between the Time variable of the Hello(A) event and the Time variable of the Hello(B) event.

The following steps are taken from(Sqalli & Freuder 1996) and they show how the CSP modeling of protocol testing is done:

1. Identify uniquely each event using the event name, the address, the time-stamp, etc. In the case where we have more than one event with the same name, the other parameters (e.g. address, time-stamp, ...) can be used to identify each of them.

2. Represent the events into a constraint graph where the variables are parameters (e.g. Node_ID, Status, Time, ...) of the events (e.g. Hello, ...), or other extra variables (e.g. Event_ID, ...) used for checking some constraints (e.g. Event_ID = Hello). The variable labels are the values that may be assigned to these variables (e.g. (A) or (B) for variable Source), and the edges are the constraints we need to satisfy such as the order of receiving events, or the value a parameter may take.

3. Get the input data by monitoring the traffic between the devices tested (e.g. (A) and (B)). These are called Observations.

4. Use the event identifier (e.g. Hello(A)) to map the events' parameters into variables, and assign values to them. We may wait until all the graph is instantiated (i.e. all events have occurred), then check whether all the constraints are satisfied or there is no violation. Alternatively, we may assign values to variables as events occur, and test if the LPCG is satisfied. A LPCG *(Logical Partial Constraint Graph)* is the portion of the constraint graph that has been so far instantiated. If there is a violation of a constraint, the process is stopped and a failure is reported. This allows us to detect errors earlier.

5. Test if all the constraints are satisfied after instantiating all the variables.

6. Report the results (OK or Error).

More details on how to model the protocol as a CSP and how to check the consistency of this CSP model is provided in (Sqalli & Freuder 1996).

Thus, the actual behavior of the system is checked against this model. If we can get the answer as to whether the actual behavior matches what is expected by the model, then the result is reported. If the model does not provide enough information to give the answer, then CBR is used. This work focuses more on the CBR/CSP interface and how CBR is used to deal with the incompleteness and incorrectness of a CSP model. CBR components are still under development and we present here a proposal for future implementation.

Each case is represented as a CSP. In the example presented later we show how this representation is done for CASE X. CBR checks if there is a similar case in the case-base. We propose to use syntactic similarity. Two cases are similar if they have a similar structural CSP representation. This involves the number of events represented and the parameters of these events in addition to the constraints between different parameters. Because of the way test suite is written, many tests will have similar representations. However, further empirical study is needed to prove the effectiveness of this similarity. If one or many similar cases are found, then they are retrieved and used to solve the new problem. The adaptation process is simple in many cases because of the tests similarity within the same test suite. To simplify further this process, some rules expressed by the expert are used. Finally, the user will check interactively whether the adaptation is appropriate, and whether this new case can be used to update the model.

The new case, consisting of the problem and solution, is eventually stored in the case-base. The new solution can also be used to update the CSP model, and make it more adaptable to new situations. The process of updating the model is done manually. Ultimately, the goal is to automate this process so that the user interaction with the system can be minimized. A set of general rules are used to update the model from a case. Some examples of these rules are:

1. Add the constraints from the case's solution to the model.

2. Add or remove the necessary constraints to make the CSP graph consistent.

In the example section, more explanation will be given on how to combine these two modes of reasoning in a practical problem.

Advantages

The advantages of our approach can be summarized as follows:

- The modeling of the protocol specifications as a CSP is easier to start with than gathering a set of cases. If we use only CBR then we will need to store many cases. Instead, we choose to reduce the number of cases by using the CSP model. The CSP model represents the core of the system, and CBR adds the missing elements in this model.
- There is no need for CBR use at first but only after CSP fails. The CSP model is easier to use at first because of its generalization.
- CSP is enhanced by the CBR results. The effectiveness of the model increases as more problems are solved, because the CSP model gets updated as needed.
- The representation of cases is done using CSP. This assures uniformity of representation.
- The system is open to new expertise and easily updated. The expert can add cases as needed by the system.

- One case can be used to update different parts of the model. This will assure that the expert is only consulted when CBR fails.

Example

We are interested in the PNNI routing protocol to demonstrate the advantages of integrating CBR and CSP to interoperability testing. In this example we will show how we can deal with an incorrect model.

The following is an example of a test (Test Case ID: V4202H_004) from the BTD-TEST-pnni-iop.000.000 document (ATM 1998):

```
Test Case ID: V4202H__004
Test Purpose: Verify that the first Hello
  sent from both sides contains Remote node
  ID and Remote port ID set to zero.
Pre-requisite: Both SUTs are in
  different lowest level peer groups.
Test Configuration: The two SUTs (e.g., ATM
  switches (PNNI capable)) are connected.
Test Set-up: Connect the two SUTs with one
  physical link.
Test Procedure:
  1- Monitor the PNNI (VPI/VCI=0/18)
     between SUT A and SUT B.
Verdict Criteria: The first Hello
  packet observed from each SUT will have
  the Remote node ID field and Remote
  port ID field set to zero.
Consequence of Failure: The old PNNI
  information was retained causing the
  protocol not to operate.
```

The following is part of the CSP model of this test, representing the time variables and constraints:

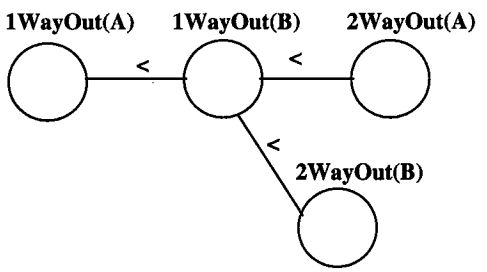

Figure 3. Initial CSP Model of Test Case ID: V4202H_004

These are some of the results we may observe:

Observation 1	**Observation 2**	**Observation 3**
(Bad)	*(Good)*	*(Bad?)*
Nothing	1WayOut(A)	1WayOut(A)
	1WayOut(B)	2WayOut(B)
	2WayOut(A)	2WayOut(A)
	2WayOut(B)	

A Hello packet is a 1WayOut if the Remote node ID field and Remote port ID field are set to zero. Otherwise, it is a 2WayOut.

This is the case stated by the expert when we encountered an earlier problem: *If a device receives 1WayIn before sending one, then it can skip sending 1WayIn and send 2WayIn.*

In the adaptation process, this rule stated by the expert is used:
"1WayOut (respectively 2WayOut) generate similar behavior as 1WayIn (respectively 2WayIn)."

This case is then retrieved and reused in this example. The new case we obtain is: *If a device receives 1WayOut before sending one, then it can skip sending 1WayOut and send 2WayOut.*

The CSP representation of this case is the following:

CASE X
Description: time[1WayOut(A)] < time[2WayOut(B)], and time[2WayOut(B)] < time[2WayOut(A)]
Problem: 1WayOut(B) is missing. That is: Status[1WayOut(B)]={0}
Solution: 1WayOut(B) is optional. That is: Status[1WayOut(B)]={0,1}

In the original model: Status[1WayOut(B)]={1}, which means that the event 1WayOut(B) is mandatory.

To update the model, we use the general rules we described earlier in this paper. These rules applied in this example become as follows:

1. Make the variable 1WayOut(B) optional.

2. For each variable Z connected to the optional variable Y (1WayOut(B) in this case), add transitivity constraints between Z and the other variables connected to Y (e.g. if Z < Y and Y < P then we add the constraint Z < P).

Using the above case (CASE X), the model becomes as follows:

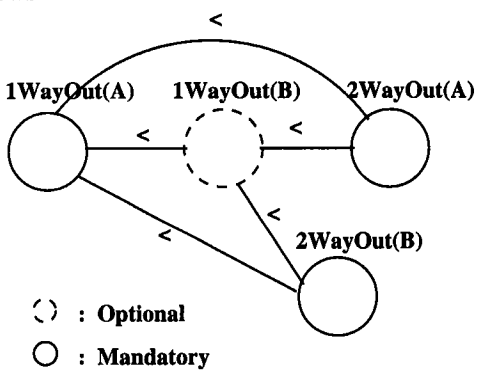

Figure 4. Corrected CSP Model of Test Case ID: V4202H_004

This problem happened because of misinterpretation of the specifications that caused an incorrectness in test V4202H_004 of the test suite. When the model was corrected by taking out one inequality constraint, the two observations 2 and 3 are shown to be correct.

Conclusion

The model for interoperability testing can be incomplete or incorrect because it may not represent all the interactions with the outside world and because such systems are not always well-defined. We propose to solve this problem by integrating Constraint-Based and Case-Based Reasoning. Models are constraint-based supported by a case-base where special problems with their solutions are stored for future use. The model is used first, and if it is not sufficient then similar cases are retrieved and CBR is used. The solution obtained can be used to update the model and compensate for its incompleteness and/or incorrectness.

Acknowledgments

This material is based on work supported by the InterOperability Laboratory (IOL) at the University of New Hampshire, and by the National Science Foundation under Grant No. IRI-9504316. The first author is a Fulbright grantee sponsored by the Moroccan-American Commission for Education and Cultural Exchange. Special thanks to Robert Blais and Scott Valcourt from IOL, and David Cypher from NIST for discussions, comments, and ideas on InterOperability Testing and the PNNI protocol. This work has benefited from discussions with members of the Constraint Computation Center at UNH, and from the reviewers' comments.

References

The ATM Forum, Technical Committee. 1996. *Private Network-Network Interface Specification Version 1.0 (PNNI 1.0).* af-pnni-0055.000.

The ATM Forum, Technical Committee. 1998. *Interoperability Tests for PNNI v1.0.* BTD-TEST-pnni-iop.000.000.

Freuder, E., and Mackworth, A. 1992. Constraint-Based Reasoning, Special Volume. *Artificial Intelligence* 58.

Hastings, J. D.; Branting, L. K.; and Lockwood, J. A. 1995. Case Adaptation Using an Incomplete Causal Model. In Veloso, M., and Aamodt, A., eds., *Topics in Case Based Reasoning, Proceedings of the First International Conference on Case Based Reasoning, LNAI Series*, 181–192. Springer Verlag.

Karamouzis, S. T., and Feyock, S. 1992. An Integration of Case-Based and Model-Based Reasoning and its Application to Physical System Faults. In Belli, F., and (Eds.), F. R., eds., *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Lecture Notes in Artificial Intelligence 604.* Springer-Verlag.

Lee, K. J.; Kim, H. W.; Lee, J. K.; Kim, T. H.; Kim, C. G.; Yoon, M. K.; Hwang, E. J.; and Park, H. J. 1997. Case and Constraint Based Apartment Construction Project Planning System: FASTrak-APT. In *Proceedings of IAAI-97.*

Portinale, L., and Torasso, P. 1995. ADAPtER: An Integrated Diagnostic System Combining Case-Based and Abductive Reasoning. In Veloso, M., and Aamodt, A., eds., *Topics in Case Based Reasoning, Proceedings of the First International Conference on Case Based Reasoning, LNAI Series*, 277–288. Springer Verlag.

Purvis, L., and Pu, P. 1995. Adaptation Using Constraint Satisfaction Techniques. In Veloso, M., and Aamodt, A., eds., *Topics in Case Based Reasoning, Proceedings of the First International Conference on Case Based Reasoning, LNAI Series*, 289–300. Springer Verlag.

Sqalli, M., and Freuder, E. 1996. A Constraint Satisfaction Model for Testing Emulated LANs in ATM Networks. In *Proceedings of the 7^{th} International Workshop on Principles of Diagnosis (DX-96)*, 206–213.

Sqalli, M., and Freuder, E. 1998a. Diagnosing Inter-Operability Problems by Enhancing Constraint Satisfaction with Case-Based Reasoning. In *Proceedings of the 9^{th} International Workshop on Principles of Diagnosis (DX-98)*, To appear.

Sqalli, M., and Freuder, E. 1998b. Integration of CSP and CBR to Compensate for Incompleteness and Incorrectness of Models. In *Multimodal Reasoning*, AAAI 1998 Spring Symposium, AAAI Technical Report, 74–79.

Appendix

1. **Integration name/category**: Constraint Satisfaction/CBR
2. **Performance Task**: Interoperability testing of networking protocols
3. **Integration Objective**: Compensate for incompleteness and incorrectness of models
4. **Reasoning Components**: Constraint Satisfaction and CBR
5. **Control Architecture**: CBR as slave, sequential: CSP is used then CBR
6. **CBR Cycle Step(s) Supported**: CBR supports and updates the CSP model
7. **Representations**: Constraint Satisfaction for models and cases
8. **Additional Components**: User validation
9. **Integration Status**: Proposed
10. **Priority future work**: Automate the process of updating the model, and address more specific issues in CBR (similarity, adaptation, ...)