ICON: A System for Implementing Constraints in Object-Based Networks

Shravan K. Goli

Dept. of Comp. Sc., and ISR, Univ. of Maryland, College Park. Currently at Microsoft Corporation.

Jayant Haritsa

IISC, Bangalore, India, and ISR, University of Maryland, College Park.

Nick Roussopoulos

Dept. of Computer Science, ISR, and UMIACS, University of Maryland, College Park.

Abstract

A vitally important step in network configuration management is to check the validity of updates made to data elements in the Management Information Base (MIB). For example, if an operator mistakenly configures a ninth port on an eight port card, the MIB should both detect and prevent this error. In this paper, we focus on the problem of checking MIB update validity and introduce the design of ICON (Implementing Constraints in Object-Based Networks), a proposed network constraint management system. In ICON, constraints are expressed through rules, which are based on the Event-Condition-Action paradigm. Rules and events are integrated cleanly into the object model by treating them also as objects.

1 Introduction

In enterprise communication networks, the network operator's interface to the network is through a Management Information Base (MIB). The MIB stores all management-related data such as network and system configurations, accounting information, and trouble logs. A vitally important step in network *configuration* management is to check the validity of updates made to MIB data elements. For example, if an operator mistakenly configures a ninth port on an eight port card, the MIB should both detect and prevent this error. In this paper, we focus on the problem of checking MIB update validity, which can be viewed as a specific instance of the general problem of *constraint management* in database systems. In particular, we introduce the design of ICON (Implementing Constraints in Object-Based Networks), a proposed network constraint management system intended for use in the *object-based* MIB of the PES (Personal Earth Station) network, a proprietary product of

Hughes Network Systems, Inc., Germantown, Maryland, U.S.A. We also discuss here the integration of ICON with the PES data model. A simplified ICON system prototype has been developed and integrated with a graphical user interface.

2 Examples of Constraints

A sample set of typical network management constraints is shown in Figure 1.

The constraint that attempting to configure more than 8 ports on an 8-port card exceeds the physical limitations of the card is expressed in Figure 1(a). Another type of constraint is shown in Figure 1(b) – here, the LAN type between communicating HUB and REMOTE LANs should be the same, that is, they should both be ethernet or both be token ring. In Figure 1(c), it is mandated that the only legal values for a modem's baud rate attribute are 2400, 4800 and 9600. Finally, Figure 1(d) states that only certain operators are allowed to make updates to parameters of network switches.

From the above examples, we observe that network management constraints have a variety of dimensions:

1. Constraints may be *physical* as in Figure 1(a), or *logical* as in Figure 1b.
2. Constraints may refer to a *single* object as in Figure 1(a) or span *multiple* objects as in Figure 1(b).
3. Constraints may be checked *immediately*, that is, as soon as the update is made, as in Figure 1(c) or *deferred* to a later time (e.g. completion of a related set of updates), as in Figure 1(b).
4. Constraints may apply *universally* to all applications accessing an object, as in Figure 1(c), or be *selectively* enforced based on the application accessing the object, as in Figure 1(d).

3 Incorporating Constraints in OO DBMSs

Constraint maintenance in object oriented databases differs from that of relational databases in many aspects, as discussed in [1, 2, 9, 11]. A detailed discussion of the differences between relational and object-oriented database management systems with respect to constraint management is also given in [15].

Virtually all research into constraint management in object-oriented database systems has assumed that the constraints are expressed in the form of *rules*. Several OO systems that support such rules are described in the literature. For example, in [9, 11], rule management for Ode, a product of AT&T, has been described. In Ode, constraints are associated with class definitions. Ode has extended C++ [17] to O++ which provides

Figure 1: Constraint Examples

facilities for associating constraints with an object. The specified constraints are checked every time an instance of that class is updated, a new instance is created, or an old instance is removed. Each constraint is expressed as a two-tuple <condition, action>. Whenever the constraint condition is violated, the action code is executed and the constraint is again tested. Detailed examples of how to express network management constraints in Ode are given in [15].

A different approach to constraint management, called ADAM, is described in [7]. Unlike Ode, where constraints are specified as a part of the object definition, the rules (or constraints) in ADAM are also treated as objects similar to the other objects in the system. Relationships can be established between the monitored objects and their associated rules. Each rule object maintains a list of monitored objects, and at the same time, the monitored objects also maintain a list of rules on them thus forming a two-way-relationship. In [15], a few examples of how to express network management constraints in ADAM are provided.

Yet another approach to constraint management, called Sentinel, is described in [1, 2]. The Sentinel approach captures the advantages of both Ode and ADAM, and extends them to provide significantly new features. It supports both constraints specified along with class definitions (as in Ode) as well as constraints specified as separate objects (as in ADAM). This has features to build rules spanning multiple objects which is difficult to do in both Ode and ADAM. In the following section, we discuss how several features of Sentinel were used in building the ICON system.

4 The Design of ICON

In this section, we present the design of the ICON network constraint management system, designed for use in the object-based MIB of the PES network (a satellite based network), a proprietary product of Hughes Network Systems, Germantown, Maryland, USA. The ICON design has taken most of its features from Sentinel (discussed in the previous section) and adapted them for the special requirements of the network management domain. The integration of ICON with the PES data model is described in Section 5, while the implementation details of the ICON prototype are presented in Section 7.

4.1 PES

In this section we give a brief description of the PES network. This network is composed of a hub, the systems control center and multiple remotes as shown in Figure 2.

- The hub provides centralized communication management for the remotes. All traffic between the remotes must pass through the hub; traffic cannot be passed from one remote to another directly over the satellite link.

Figure 2: PES network

- The systems control center (SCC) controls the network, i.e. all management of this network occurs from the SCC which is thus conceptually centralized but may be distributed in practice. The SCC and Hub is usually co-located. Management is done through operator consoles through which operators configure, monitor and control the network.
- The remotes are geographically dispersed sites that contain *remote node equipment*. The remote equipment is typically attached to customer equipment such as LANs, computers and workstations. Customer equipment is connected to the network via remote ports.

Information is exchanged between the remote sites over a satellite link through the hub. Remote to hub transmissions travel over *inroutes*, while hub to remote transmissions travel over *outroutes* as shown in Figure 2. Thus, remote to remote travels from the remote to hub over an inroute, then from the hub to the other remote on an outroute. Of course, all transmission must be relayed through the satellite.

4.2 Design Details

We consider first the problems of how constraints are specified, how they are stored, and how they are evaluated and enforced. In ICON, constraints are expressed through *rules*. Each rule is composed of an *event*, a *condition*, and an *action* (this is also known as the E-C-A paradigm [4]). Occurrence of the event triggers the rule, the condition is a boolean check, and the action is executed if the condition is satisfied. This rule definition is illustrated in the following example, which is used during configuration of the HUB module of the PES network to check for uniqueness of data port card (DPC) names:

```
class DPC{

private:  char dpcname[20];
public:   Set_Name(char *name);

rule dpcname_uniq;
when Set_Name(name)                /* event */
    if not_unique(dpcname)          /* condition */
        then highlight(dpcname_field); /* action */
```

In the above example, a class DPC is defined for data port cards. The rule `dpcname_uniq` monitors the configuration of DPC objects, and is triggered whenever a DPC object invokes the method `Set_Name`. A check is then made as to whether or not the 'dpcname' is unique. If the name is not unique then the action routine 'highlight()' is called to indicate the error to the operator.

It is important to note that the term event used here does not refer to network events, but refers to *database events*. In our object-oriented framework, database events consist primarily of *object method invocations*. With respect to configuration management, database events would mainly be initiated by operator actions. More generally, however, we expect that network event messages received by the MIB during network operation could lead to the generation of one or more database events.

4.3 Rule Specification

In ICON, rules are implemented as *first-class objects*, following the Sentinel approach. With this approach, rules can be created, modified, and deleted in the same manner as other objects, thus providing a uniform view of rules in an OO context. Second, rules are now separate entities that exist independently of other objects in the system. This allows for rule definitions to exist even when the object classes on which the constraints operate do not exist. Third, each rule has an object identity, thereby allowing rules to be associated with other objects. Finally, an extensible system is provided due to the ease of introducing new rule attributes or operations on rules.

4.4 Event Specification

In ICON, events are implemented as *first-class objects*, as in Sentinel. This implementation is chosen because events exhibit the properties of objects in terms of having state, structure and behavior. The state information associated with each event includes the occurrence of the event and the parameters computed when the event is raised. The structure of an event consists of the events it represents, while the behavior consists of specifying when to signal the event. By making events to be objects, events can be created, deleted, modified, and designated as persistent similar to other types of objects. The introduction of new event types and attributes is easily incorporated. Also, events spanning distinct classes can be expressed in a clean fashion. Complex events can be constructed using a hierarchy of event operators such as conjunction, disjunction, etc.

4.5 Event Generation

In ICON, each object is allowed to declare some subset of its public method interface to be *reactive*. This means that an event message is generated whenever a method in this reactive subset is invoked by the object. These event messages are propagated to other objects by a mechanism described in the following subsection. Event messages have the following structure:

Event Message = *Oid* + *Class* + *Parameters*

Here, *Oid* denotes the object identifier of the object generating the message, *Class* denotes the class of this object, and *Parameters* denotes the set of parameters with which the

method is invoked.

4.6 Rule – Event Association

Rules are associated with events through a *subscription* mechanism. This mechanism allows rules to dynamically subscribe to the events generated by reactive objects. After the subscription takes place, the rule is informed whenever an object it subscribes to issues an event. Each reactive object maintains a list of its subscribed rules.

The above subscription mechanism results in some advantages: First, the runtime rule checking overhead is reduced since only those rules which have subscribed to an event are checked when that event is generated, that is, rule checking is localized (or distributed). Second, a rule is defined only once and it can be associated with any number of reactive objects. This is more efficient than defining the same rule multiple times and applying each rule to one type of object. Finally, rules triggered by events spanning distinct classes can be expressed. This is accomplished by a rule subscribing to the events generated by instances of different classes.

4.7 Object Classification

In ICON, objects are classified into the three categories described in Sentinel: passive, reactive, and notifiable. These categories and their relationship to events are described below.

Passive objects: These are regular C++ objects. They can invoke methods but do not generate events. Objects which do not need to be monitored fall into this category.

Reactive objects: Objects on which rules may be defined are made reactive objects. Once a method is declared as an event generator, its invocation will be propagated to other objects. Thus, reactive objects communicate with other objects via event generators.

Notifiable objects: Notifiable objects are those objects capable of being informed of the events generated by reactive objects. Therefore, notifiable objects become aware of a reactive object's state changes and can perform operations as a result of these changes. All rules are notifiable objects. There is an m:n relationship between reactive objects and notifiable objects, that is, a reactive object instance can propagate events to any number of notifiable object instances and a notifiable object instance can receive events from several reactive object instances.

4.8 ICON Example

The basic paradigm in ICON is that events are produced by reactive objects (producers) and they are consumed by notifiable objects (consumers). An example of Producer/Consumer paradigm is shown in Figure 3, taken from [1, 2]. Here, object P generates (produces) a primitive event `event1` and sends it to a rule R1. The rule passes

Figure 3: Event Producer/Consumer Analogy

Figure 4: Integration with PES data model

(consumes) the event to the event detector for storage and event detection, and if the event is detected, the rule checks the condition and takes appropriate actions. In this example, P is a reactive object, event1 is a primitive event, and R1 is a notifiable object. With reference to the earlier example in section 4.2, P is of type DPC and R1 is 'dpcname_uniq'.

4.9 Summary

The above design of ICON provides for: (i) rule definitions to be independent from the objects which they monitor, (ii) rules to be triggered by events spanning sets of objects, possibly from different classes, and (iii) objects to dynamically determine which object state changes they should react to and associate a rule object for reacting to those changes. Essentially, this separates the object and rule definitions from the event specification and detection process. This aids in building a modular and extensible system.

5 Integration with Data Model

We now describe how the constraint management in ICON could be integrated with the PES data model. The details of the PES data model are given in [6]. For the purpose of integration, we have subdivided the top-level managed object class of the PES data

model into a reactive class, a passive class, and a notifiable class, as shown in Figure 4. All the elements of the PES data model which need to generate database events fall into the reactive class. Similarly, all the constraint management objects are in the notifiable subclass. The remaining objects, which do not generate any database events or which do not have any rules imposed on them, fall into the category of passive objects.

6 Implementation details

As mentioned earlier, a prototype of a simplified version of ICON has been developed. The prototype implementation is discussed in detail in [16]. The [16] describes in detail the implementation of Reactive, Notifiable, Event and Rule classes. It also describes a simple algorithm for ICON which we discuss in the section below. In addition, it also describes MOTIF/Galaxy [13] versions of graphical user interface developed for ICON. Some implementation examples are also described. The prototype was developed on the object-oriented database platform provided by ObjectStore, a commercial OO-DBMS [18, 14, 5].

6.1 ICON Algorithm

We show here, in pseudo-code, a simple algorithm which describes the mechanism of event generation and rule checking in ICON:

```
algorithm_ICON()
{
    Whenever a Reactive method is accessed, at some point in its
    processing, a method called Notify() is used to send a message
    to all Rules subscribed to that reactive object;

    For each rule subscribed {
        if (the rule is enabled) {
            pass the event to the rule's event detector;
            if (event detected) {
                check the condition of the rule;
                if (condition is satisfied)
                    perform the action; } }
    }
    return the appropriate value;
}
```

6.2 Graphical User Interface

As described in earlier sections, rules exist as independent objects and they subscribe to reactive objects. Each reactive object maintains a list of currently subscribed rules, and a list of rules which are valid but currently unsubscribed. It is possible to dynamically change these lists and, thereby, the behaviour of the monitored objects themselves. In our implementation, we have developed a Motif user interface to help the user of the system decide the behaviour of the monitored objects. The main advantage of this is that it helps in customization of the product to match different users requirements. Note that if the rules were implemented in the traditional way of writing code wherever required, this kind of dynamic behaviour would not have been possible. For any change, the code would have to be not only re-written but also recompiled. This would create several problems in network MIBs since recompilation requires database access

References

- [1] Anwar, E., Maugis, L., and Chakravarthy, S. (May 1993) A New Perspective on Rule Support for Object-Oriented Databases. *ACM SIGMOD*, 99-108.
- [2] Anwar, E. (1992) Supporting Complex Events and Rules in an OODBMS: A Seamless Approach. *Master's Thesis, Univ. of Florida*.
- [3] Bauzer Medeiros, C. and Pfeffer, P. (1990) A mechanism for Managing Rules in an Object-oriented Database. Altair Technical Report.
- [4] Chakravarthy, S. et al. (July 1989) HiPAC: A Research Project in Active, Time Constrained Database Management. *TR XAIT-89-02*, Xerox Advanced Information Technology, Cambridge, MA.
- [5] Class Notes, ObjectStore, Object Design, Inc.
- [6] Datta, A. and Ball, M. (1993) MOON :- A Data Model for Object Oriented Network management. To be published, ISR, University of Maryland at College Park.
- [7] Diaz, O., Paton, N. and Gray, P. (Sept. 1991) Rule Management in Object-Oriented Databases : A Unified Approach. *Proc. of VLDB, Barcelona*, 317-26.
- [8] Dupuy, A. et al (March 1991) NETMATE: A Network Management Environment. *IEEE Network Magazine*.
- [9] Gehani, N. and Jagadish, H. (Sept. 1991) Ode as an Active Database: Constraints and Triggers. *Proc. of VLDB, Barcelona*, 327-36.

- [10] Haritsa, J. et al (1993) Design of the MANDATE MIB. *Integrated Network Management, III*, Elsevier Science Publishers, 85-96.
- [11] Jagadish, H. and Qian, X. (Aug. 1992) Integrity Maintenance in an Object-Oriented Database. *Proc. of VLDB, Vanouwer*, 469-80.
- [12] Klerer, S. (March 1988) The OSI Management Architecture: an Overview. *IEEE Network*, 2(2).
- [13] Plaisant, C., Kumar, H., Teittinen, M. and Shneiderman, B. (1994) Visual Information Management for Network Configuration. *TR 94-48*, ISR, University of Maryland at College Park.
- [14] Reference Manual, ObjectStore Release 2.0, Object Design, Inc.
- [15] Shravan, G., Jayant, H. and Nick, R. (1994) Integrity Constraints in Configuration Management. *TR 94-62*, ISR, University of Maryland at College Park.
- [16] Shravan, G., Jayant, H. and Nick, R. (1994) A System for Implementing Constraints in Object-based Networks. *TR-xxx under preparation*, ISR, University of Maryland at College Park.
- [17] Stroustrup, B. (1986) The C++ Programming Language. Addison-Wesley.
- [18] User Guide, ObjectStore Release 2.0, Object Design, Inc.
- [19] Yemini, Y. (May 1993) The OSI Network Management Model. *IEEE Communications Magazine*, 20-29
- [20] Zdonik, S. and Maier, D. (1990) Object Oriented Fundamentals. *Readings in Object-Oriented Database Systems*, 1-32.

Shravan K. Goli received the B.E. degree in Computer Science and Engineering from the Osmania University, Hyderabad, India, in 1992, and the M.S. degree in Computer Science from the University of Maryland, College Park in 1994. During 1992-1994, he was a Graduate Fellow at the Institute for Systems Research, University of Maryland, College Park. He is currently working in Microsoft Corporation, Redmond, WA. He previously worked with Hughes Network Systems, Germantown, MD. His research interests include distributed systems, network protocols, network management and object oriented database systems.

Jayant R. Haritsa received the B.S. degree in Electronics and Communications Engineering from the Indian Institute of Technology, Madras, India, in 1985, and the M.S. and Ph.D. degrees in Computer Science from the University of Wisconsin, Madison in 1987 and 1991, respectively. During 1991-1993, he was a Post Doctoral Fellow at the Institute for Systems Research, University of Maryland, College Park. He is currently an Assistant

Professor in the Supercomputer Education and Research Centre and in the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore, India. During 1988 and 1990, he spent summers at the Microelectronics and Computer Technology Consortium and at the IBM T.J. Watson Research Center, respectively. Dr. Haritsa's research interests include database systems, real-time systems, network management and performance modeling. He is a member of IEEE and ACM.

Nick Roussopoulos received the B.A. degree from the National University of Athens, Greece, and the M.S. and Ph.D. degrees from the University of Toronto. He has worked as a Research Scientist at IBM Research at San Jose, and as faculty with the Department of Computer Science at the University of Texas at Austin. Since 1981 he has been with the University of Maryland, where he is a Professor of the Computer Science Department and the Institute of Advanced Computer Studies. His research area is in database systems, multi-databases and interoperability, engineering information systems, geographic information systems, expert database systems, and software engineering.