# BEHAVIOURAL SPECIFICATION AND NOTIFICATION ENHANCEMENTS TO GDMO *

Michael J. Katchabaw and Michael A. Bauer
Department of Computer Science
University of Western Ontario
London, Ontario, Canada

James W. Hong
Department of Computer Science and Engineering
Pohang University of Science and Technology
Pohang, Korea

**Abstract**

Capabilities currently provided by GDMO for specifying the behaviours of notifications is very weak – while the standard allows a text string for defining the circumstances generating a notification, no formal mechanism is currently provided for this specification. For more powerful and useful notifications and event reports, GDMO must be enhanced. This paper describes the enhancements made to GDMO to allow more flexible and powerful specification of notifications and the events triggering them in defining various managed objects. This was accomplished through utilizing the extensibility of behaviour definitions in GDMO. As a proof of concept, we have extended the GDMO compiler provided with the OSIMIS system. This paper presents our prototype implementation effort on extending the OSIMIS GDMO compiler and its agents to handle such enhancements.

## 1 Introduction

Most network and systems management systems are comprised of two main components, namely *agents* which interact with the resources being managed, and *managers* which send various management requests to agents. Communication in this kind of management framework usually takes the form of requests from managers to agents and agents to managed resources (and subsequent replies), and notifications about the occurrence of some events pertaining to a managed resource, which are sent from agents to alert managers in the form of event reports. Managed objects, which are abstract representations of real managed resources, can be defined using several possible specification languages, one of which is GDMO (Guidelines for the Definition of Managed Objects) [5]. These components of a general management architecture are illustrated in Figure 1.

Currently, GDMO only allows trivial specification of the behaviour of notifications, with no formal method of specifying the events resulting in the triggering of a notification. This must be done separately, and hand-coded by the programmer into the agent. This is both inflexible and requires extra work when this process can be automated. Automation is particularly favourable in allowing users unfamiliar with the internal workings of the management system to create and develop their own management agents. To deal with this, enhancements to GDMO are required to allow the specification of conditions resulting in the triggering of notifications. This can be accomplished through utilizing the extensibility of the behaviour template. Furthermore, GDMO compilers must be extended to take advantage of these modifications, allowing the compilers to generate code for capturing events and triggering appropriate notifications in agents automatically.

Some earlier work also identified this problem with the current GDMO standard [1, 3]. Our work goes further and proposes an extension to GDMO that will allow the behavioural specification of notification conditions.

To date, we have been using OSIMIS, an OSI management development environment developed at UCL [7], as the environment for testing out our ideas and developing management tools [4, 6, 10]. OSIMIS uses GDMO for the specification of managed objects [2]. Through our experience working with this system, it has become evident that the trivial notification specification capabilities provided by GDMO are not sufficient for practical applications.

This paper describes our work on extending GDMO to allow a more powerful and flexible specification of events and notifications in defining various managed objects. This includes a formal method of specifying the circumstances triggering a

*This research work is supported by the IBM Center for Advanced Studies and the Natural Sciences and Engineering Research Council of Canada.



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

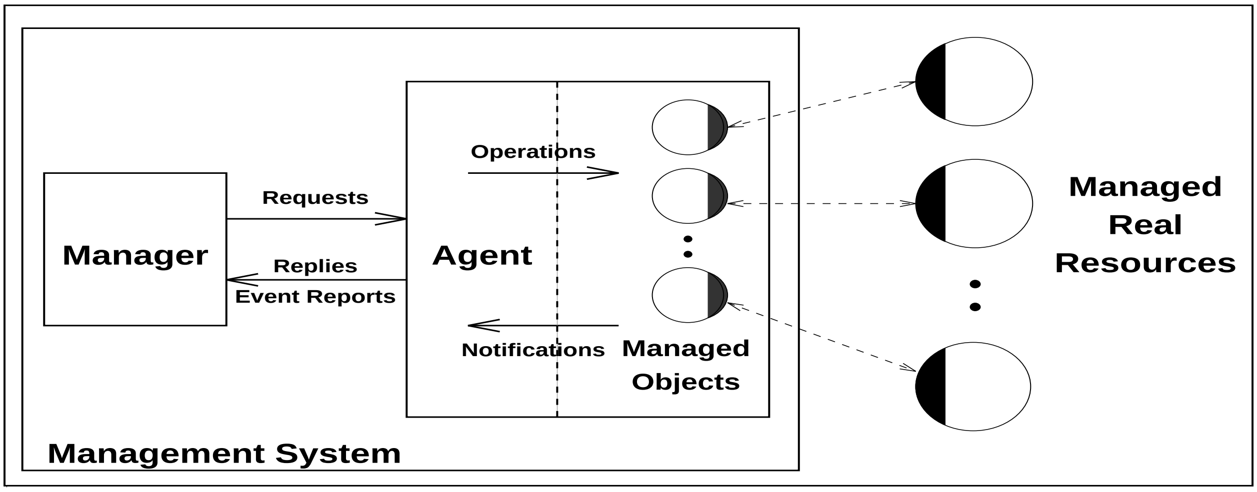


Figure 1: General Architecture of a Management System

notification using a behavioural definition. As a proof of concept, we have extended the OSIMIS GDMO compiler to generate appropriate C++ source code needed in agents for capturing events, triggering notifications automatically and generating the corresponding event reports to managers. We also describe the necessary changes that had to be made to the OSIMIS agent infrastructure to achieve the intended goal.

The remainder of this paper is organized as follows. Section 2 presents the the proposed enhancements to GDMO. Section 3 examines and describes the internal architecture of the OSIMIS GDMO compiler. Section 4 describes the necessary modifications made to the OSIMIS GDMO compiler to support the specification of enhanced notification conditions and to generate appropriate source code. Section 5 describes the modifications required in the OSIMIS agents to handle the enhanced notification functions. Section 6 presents our implementation effort to date. Section 7 summarizes our work and proposes some future work in this area.

## 2 Enhancements to GDMO

The GDMO standard is presented in the CCITT Recommendation X.722 [5]. Notifications are discussed in section 9.11 of the recommendation, defining the structure and semantics of a notification. This section discusses the enhancements made to the GDMO standard to allow the specification of notification conditions. For a frame of reference, the existing portions of the standard pertaining to notifications are examined first. The following template for a notification in GDMO is proposed by the recommendation:

```
<notification-label> NOTIFICATION

   [BEHAVIOUR     <behaviour-definition-label> [, <behaviour-definition-label>]* ;
   ]
   [PARAMETERS    <parameter-label> [, <parameter-label>]* ;
   ]
   [WITH INFORMATION SYNTAX   type-reference
       [AND ATTRIBUTE IDS        <field-name> <attribute-label>
                              [, <field-name> <attribute-label>]*
       ];
   ]
   [WITH REPLY SYNTAX         type-reference ;
   ]

REGISTERED AS object-identifier ;
```

Most of this template is used either to identify a notification in one way or another or to specify information passed in the notification process and the appropriate syntax of this information. The behaviour field, however, is of great interest.



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

According to the standards document, this field "shall specify the circumstances under which a notification of the type is generated". This is the field that requires examination. To be able to determine how these circumstances are specified, we must examine the behaviour template:

```
<behaviour-definition-label> BEHAVIOUR

   DEFINED AS    delimited-string;
```

A behaviour definition is simply a natural language text string describing the behaviour of some other entity. While this does not seem very useful, it does allow us to use a behaviour to specify the conditions leading to the generation of a notification. We could simply embed these conditions in a text string in the "DEFINED AS" field of the behaviour definition. This allows the flexibility of specifying these conditions in almost any manner. Furthermore, flexibility is achieved because notifications can specify several behaviours, and hence can be triggered by several, possibly different, conditions. Using the behaviour to store these conditions has the additional advantage of remaining within the current GDMO standards and not introducing any new notation. The key to the enhancements lies with how the conditions for notifications can be represented in a text string. We describe this in detail below.

Since the only restriction placed by the GDMO standards for specifying conditions for notifications is that they must be represented in a text string, there is great flexibility available. Our approach is to satisfy the basic requirements of providing a simple condition verification mechanism, but allow the room for possible extensions to permit more complex conditions. The bare minimum requirements for a condition checking mechanism would involve the following:

- Relational expressions – expressions used to compare two values for equality, inequality, *etc.*
- Arithmetic operations and other manipulations of the values.
- Set operations (*e.g.*, $x$ is an element of the set $\{1, 3, 5, 7, 9\}$).
- Range checking (*e.g.*, $x$ is in the range from 0 to 10).
- Boolean combinations of expressions – using AND, OR, and NOT to form more complicated expressions based on the above expressions.
- Values being compared should at least include attribute values and constant values.

Using the above minimum guidelines, we can define the expressions to be dealt with more formally using the following simple grammar:

```
<behaviour-definition-label> BEHAVIOUR
   DEFINED AS    <condition-expression>;

<condition-expression> := '(' <atomic-expression> '&' <condition-expression> ')' |
                          '(' <atomic-expression> '|' <condition-expression> ')' |
                          <atomic-expression>
<atomic-expression> := <expression> | '!' '(' <expression> ')'
<expression> := <exp> <rel-op> <exp> |
                <exp> 'IN' <range> |
                <exp> 'IN' <set>
<exp> := <arithmetic-exp> | <term>
<rel-op> := '=' | '>' | '>=' | '<' | '<=' | '!='
<range> := '[' <constant> '..' <constant> ']'
<set> := '{' <set-body> '}'
<set-body> := <term> ',' <set-body> | <term>
<arithmetic-exp> := '(' <arithmetic-exp> '+' <arithmetic-exp> ')' |
                    '(' <arithmetic-exp> '-' <arithmetic-exp> ')' |
                    '(' <arithmetic-exp> '*' <arithmetic-exp> ')' |
                    '(' <arithmetic-exp> '/' <arithmetic-exp> ')' |
                    <term>
<term> := <attribute-label> | <constant>
<constant> := <text-string>
```



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

In the above grammar, all terms not defined by the grammar are addressed in the GDMO standards document. It is also important to note that all constants, including numerics, are given as string representations of the appropriate value (for example the number five hundred and eleven is given by the text string 511, not the actual encoding of the number). Type checking and conversions will be done as necessary to convert constant values into the appropriate internal representations.

Because we are working with text strings for storing these expressions in a specification, the possibilities are nearly endless. As a result, this simple methodology can be extended even more, as necessary. With this in mind, we have designed the above simple grammar to be as extensible as possible.

To illustrate how these extensions can be used, consider the managing of a file system resource. The managed object class modelling this resource would contain attributes such as name, mount, point, and free space. Using the methodology described above, we could define a notification for this class to be triggered when there is no more available space on the file system. A partial GDMO specification for this could be:

```
fsObjPackage  PACKAGE
    ATTRIBUTES
        fsObjId                 GET,
        fsName                  GET,
        fsMountPt               GET,
        fsBlocksAvail           GET;

    NOTIFICATIONS
        fsNoSpaceAvail;

    REGISTERED AS               { uclPackage 45 };

fsNoSpaceAvail NOTIFICATION
    BEHAVIOUR                   fsNoSpaceAvailBehav;
    WITH INFORMATION SYNTAX     Notification-ASN1Module.ObjectInfo
    AND ATTRIBUTE IDS
        sourceIndicator         sourceIndicator,
        attributeList           attributeList;
    REGISTERED AS               { smi2Notification 6 };

fsNoSpaceAvailBehav BEHAVIOUR
    DEFINED AS                  fsBlocksAvail = 0;
```

When compiled, and the appropriate code generated, the above specification would result in the triggering of the notification fsNoSpaceAvail whenever the management system detects that the file system being managed has no free blocks available.

# 3 Overview of OSIMIS GDMO Compiler

We are extending the GDMO compiler provided in OSIMIS to implement the enhancements made to GDMO. Before we describe the extensions the OSIMIS GDMO compiler [2], we present an overview of the existing structure of the compiler.

The OSIMIS GDMO compiler works in exactly the same way that an ordinary compiler (*e.g*, C or Pascal) does. It inputs code in a source form, uses lexical analysis to break the code into tokens, parses the tokens using syntactic analysis into an internal intermediate representation, and finally generates the appropriate output code. For the OSIMIS GDMO compiler, the source is a GDMO specification for a MIB, and the output generated is C++ code implementing the given MIB. This code is then compiled and linked with hand-written code (for interaction with the real resources from the managed objects in the MIB), the OSIMIS GMS [8] (Generic Management System – generic functionality common to all agents), and agent initialization routines. The end result of this process is a fully functional agent. This process of compiling GDMO definitions by the GDMO compiler and generating appropriate C++ source code required for constructing agents is illustrated in Figure 2.

As can be seen from Figure 2, additional information in the form of attribute, managed object, and syntax databases must also be read in by the compiler to assist in the syntactic analysis and code generation stages. Code generation scripts are also used to provide templates defining the structure of the output C++ code.

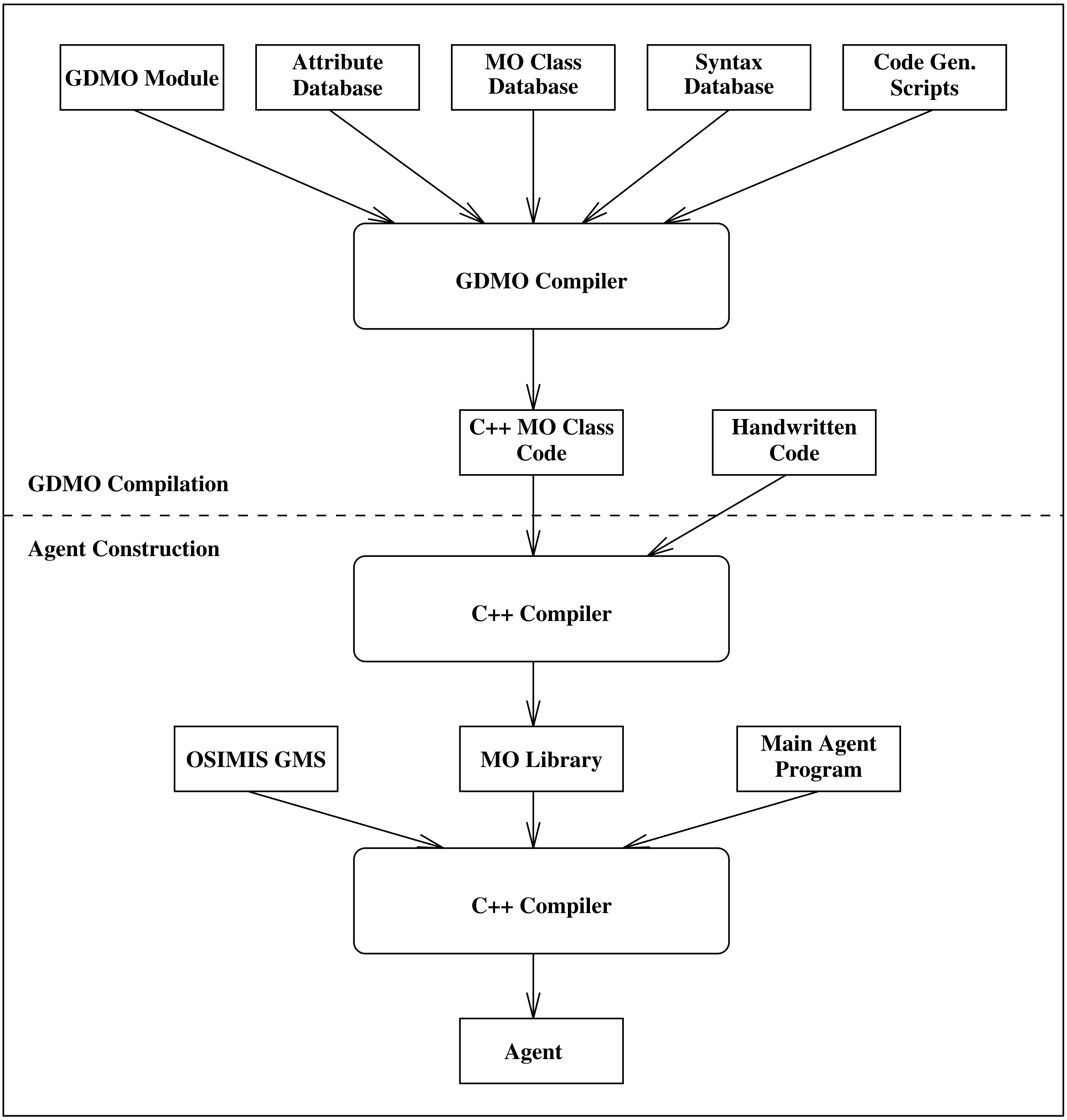


Figure 2: Flow of Information Through the GDMO Compiler and Agent Construction



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

## 4 Enhancements to the OSIMIS GDMO Compiler

Having introduced the compiler we are working with, we now describe where the compiler must be modified in order to allow the enhanced notification behavioural specifications we proposed in GDMO. The modifications required for parsing the new specifications are described first and then those for generating appropriate source code to trigger notifications automatically are described.

### 4.1 Parsing GDMO Enhancements

To be able to handle the enhancements to GDMO, the compiler must be able to parse the enhancements and store them in the internal representation in data objects. For the most part, the existing compiler handles this well. The enhancement made to GDMO involved using the "DEFINED AS" attribute of a behaviour associated with a notification to record the conditions under which the notification is triggered. Both notifications and behaviours are completely parsed and stored by the compiler – including the behavioural definition text string. However, the critical missing element is the association between a notification and its corresponding behaviours. As a result, new code was required to make this association.

With this done, the compiler could successfully read in an input GDMO specification and associate notifications with their behaviours indicating the conditions that result in their triggering. At this point, however, the text string defining the behaviour has not yet been parsed according to our grammar. It would be possible to perform this parsing either during the parsing phase, or the code generation phase, just before the required code is output. The code generation phase was chosen because it would be simpler to create a separate parser to handle parsing at this stage, rather than integrating our grammar with the existing grammar used during the parser phase. Furthermore, during the parse phase, we cannot easily determine whether a given parsed behaviour corresponds to a notification or some other GDMO structure, so we cannot parse it as a notification condition.

### 4.2 Generating Code to Automatically Trigger Notifications

Now that the GDMO compiler can parse our refined GDMO, the code generation phase can now take advantage of these enhancements. To be able to generate this new code, changes are required to both the script files that provide a template for code generation and the script interpreter itself. Since script files instruct the interpreter as to what code should be generated, any changes to the code to be output would require changes in these files. For our purposes, a new method would be added to check the conditions for each notification for the managed object; triggering the notification if the conditions are met. The generated code has the following general structure:

```
Class <MOClassName>:
      Method CheckNotificationConditions() {
         FOR EACH notification IN managed object's notification list DO:
             FOR EACH behaviour IN notification's behaviour list DO:
                 IF (behavioural condition met) THEN
                    TriggerNotification(notification);
      }
```

Where exactly this code is placed in the agent and how it is invoked is discussed in the next section.

The above structure for output code is capable of triggering notifications automatically in our system. Since notifications, managed objects, and behaviours can be accessed by simple script language constructs, the only part to deal with is the "behavioural condition met" code. This is where the processing of behavioural definitions, discussed in the previous section, takes place. Unfortunately, the script language is not sophisticated enough to allow the processing to be done in the script files itself. Instead, a new function must be introduced to do the processing. This function would appear as a built-in function of the script language, as user-defined functions are not supported. As a result, the following function is used in the scripts:

```
GET_NOTIFICATION_CONDITION(behaviour: GDMOBehaviour): String
```

This function takes a behaviour as input and parses the definition string in the behaviour to produce another string containing a boolean expression in the output language. If this boolean expression evaluates to True when executed, then the condition is met and the notification is triggered; otherwise, nothing is done. This expression string is then returned by the function.



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

The above modifications to the OSIMIS GDMO compiler make use of the enhancements to GDMO to produce code in agents that automatically triggers notifications, as opposed to having the programmer do so manually. The next section examines agent applications in more detail. Specifically, the section describes where the above generated code is placed in the agent, and how this code is invoked to automatically trigger notifications.

# 5 Enhancements to OSIMIS Agent

The previous sections discussed how GDMO and the OSIMIS GDMO compiler need to be modified to allow the specification of notification triggering conditions in GDMO specifications. The GDMO compiler outputs code containing automatic notification triggering in an agent, but this code must be placed and invoked properly within an agent in order for it to work. This section resolves this issue. The existing subsystem decomposition of a generic management agent (including the OSIMIS agent) were investigated thoroughly in [9]. Figure 3 illustrates the decomposition of a generic management agent, which we use here as a reference.

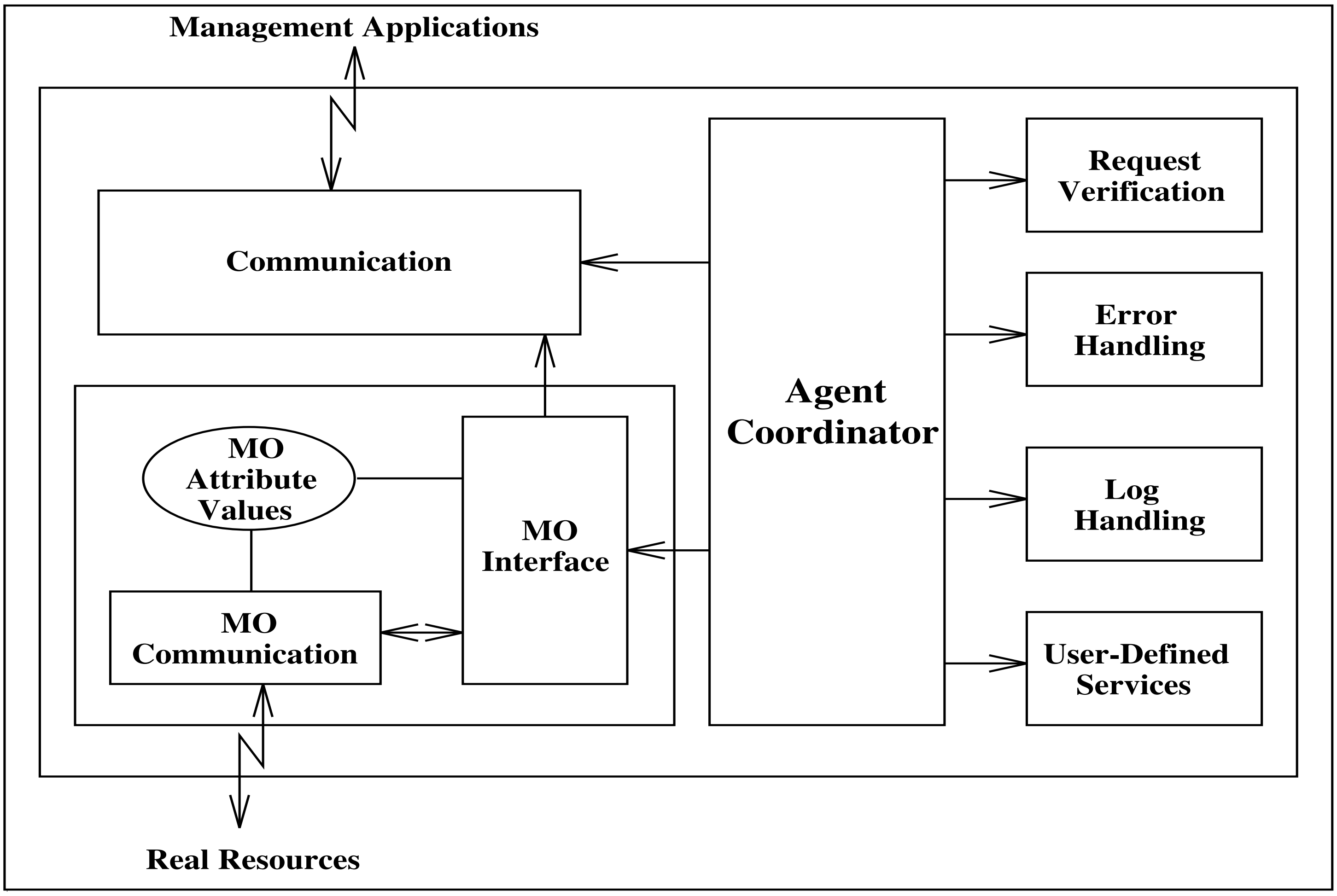


Figure 3: Generic Agent Subsystem Decomposition

## 5.1 Placement of the New GDMO Compiler-Generated Code

The previous section outlined the new code that the enhanced GDMO compiler generates to allow automated condition checking for notifications. Since this code is associated with the notifications for a managed object, it should be placed with the other in-line methods for the managed object generated by the compiler. That is, the new method clearly belongs in the managed object interface component of the agent.



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

While addressing the issue of the placement of notification-triggering code in the agent, we must also consider code already in existence. This code was hand-written and inserted throughout the code for methods of managed objects. However, with the enhancements to the GDMO compiler, this code is no longer necessary, and can be removed from the agent. It is important to note that this need not be the case; this design is flexible enough to work with existing agent code without conflicts.

## 5.2 Invoking New Methods

Just having the code for automatically triggered notifications from within managed objects in place is not sufficient for the code to be useful. The code must first be invoked. At this point, it must be determined where this code is invoked from and how it is invoked. There are two possible components – the agent coordinator, and the managed object interface. Because of their positions and roles in the agent, other components are not suitable choices.

### 5.2.1 Agent Coordinator Invocation

After the coordinator forwards a request to a managed object's interface, times out on a wake-up (when a managed object periodically checks the state of its corresponding real resource), or receives data on a communication endpoint (when a real resource periodically sends messages to its managed object updating its state), it is possible for conditions to have been met that would lead to the triggering of a notification. At this point, the coordinator could then make another request to the managed object interface to check managed objects for conditions leading to notification triggering. However, this is where this method runs into problems.

At this point, if a management request was received, the coordinator does not know which managed objects were affected by the request. The coordinator, because of the CMIP protocol, only has a base object in the management information tree (MIT), a scope, and a filter. These parameters are used by the managed object interface when it receives a request from the coordinator to determine which managed objects the interface should invoke the request on. As a result, if this method was used, either all managed objects would have to be checked for notification conditions, or scoping and filtering performed in the original request would have to be repeated. Either method would result in additional unnecessary overhead with every request that agents receive from management applications. As a result, this method of invocation alone is not attractive.

### 5.2.2 Managed Object Interface Invocation

When the managed object interface receives a request forwarded by the coordinator, it receives a base object, scope, and filter that determine which managed objects the request applies to. The scoping and filtering process produces a list of managed objects. The method corresponding to the requested operation is then invoked in each managed object by the managed object interface to complete its task. After the requested operation is completed, the managed object interface could then invoke the method to check the conditions for notifications in the managed object. Because this invocation is performed here, only the managed objects affected by the original request are checked, and no additional scoping or filtering is required.

To give an idea how this is accomplished, consider the following within the managed object interface component:

```
Operation_X(base, scope, filter, ...):
     mo_list = Scope(base, scope);
     mo_list = Filter(mo_list, filter);
     FOR EACH mo IN mo_list DO:
         mo->Operation_X(...);
         mo->CheckNotificationConditions();
```

This simple algorithm will check for notification conditions only in managed objects affected by the given operation. The automatically generated CheckNotificationConditions() method previously discussed would then check each condition for a notification for the managed object, triggering a notification if necessary. All of this code is handled behind the scenes, without any work required by managed object or agent programmers.

Alternatively, if a wake-up occurs or a message arrives at some communication endpoint, similar code to that above could handle invoking the CheckNotificationConditions() method in the appropriate managed objects. Because of the way the OSIMIS agent coordinator handles wake-ups and communication endpoints, the coordinator could easily handle invoking the methods in these cases as well. This leads to the possibility of a hybrid approach to automatically check for notification



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations
and Management, Ottawa, Canada, October 1995

conditions. In such a hybrid, the managed object interface would handle invoking the method when external requests are received and the agent coordinator would handle the invocation when wake-ups occur or messages are received at a communication endpoint. Because of its efficiency, simplicity, and its clean integration into existing code, this hybrid design was used.

## 6 Prototype Implementation

Having designed the appropriate enhancements to GDMO and to the OSIMIS GDMO compiler and agents, we decided that it may be best to first implement the enhancements using a subset of the simple condition grammar outlined in Section 2. Doing so would allow us to quickly determine if the proposed modifications were feasible and performed their required tasks properly and efficiently. After confirmation that our enhancements were suitable, the complete grammar could then be implemented fairly easily by simply replacing the subset used in our original prototype.

For our first prototype, we chose to test equality comparisons only between combinations of attributes or constant values. While the resulting expressions are quite simple, and not nearly powerful or flexible enough to cover all possible cases, they do however provide a solid foundation for experimentation and building upon in the future when we implement the full grammar. This also provides a basic formalism not present in the GDMO standard. Using this small grammar then, the necessary modifications were made to the OSIMIS GDMO compiler and agents. The actual implementation proceeded according to the design outlined in previous sections.

Using our enhanced system, we then modified existing agents that we had developed for Unix system management and distributed application management to make use of notifications whose behaviours and triggering conditions were defined only using a GDMO specification. For illustration purposes, we will focus on the file system managed object discussed previously that was part of our Unix system management agent.

When the enhanced GDMO compiler was used on the specifications, the appropriate managed object code was generated to handle checking the triggering conditions for notifications, and was linked into the resulting agent seamlessly. For the file system managed object, this code appeared as:

```
void fsObj::checkNotificationConditions() {
   if (*((int *) _attrs[I_fsBlocksAvail]->get()) == 0) {
      triggerEvent(I_fsNoSpaceAvail , _level);
   }
}
```

When the agent is executed, this method is invoked whenever this managed object is referenced, such as when an operation is invoked upon it. Only when this happens, can the state of the managed object change to cause new conditions that would result in the triggering of the notification. For example, with our file system managed object, if we were monitoring the file system (through a sequence of Get operations), the above method would be invoked after each monitoring poll. When the available space on the file system reaches 0, the condition in the `if` clause would be satisfied, and the notification would be triggered. This process was confirmed through a series of tests on this managed object, and several others that we enhanced. Similarly, when wake-ups occur or data is received on a communication endpoint, tests have shown that this process works in these cases as well. Consequently, the appropriate notifications are triggered as they should be, using only the compiler generated code from our enhancements to GDMO and the appropriate OSIMIS components.

Having validated that our methodologies and enhancements were sound, we are now prepared to implement the full grammar for further experimentation. Since this now only requires modifications to the grammar, and no other changes, this should be a smooth process.

## 7 Concluding Remarks

We have proposed enhancements to GDMO for supporting more powerful notification and event reporting functions in OSI management systems. The extensions to GDMO allow more complex events to be captured and automatically notified by agents to managers. Since these extensions have been made using behaviour definitions, they conform well to GDMO standards. We proceeded to extend the OSIMIS GDMO compiler to verify our GDMO enhancements and to generate necessary



Appeared in the Proceedings of the 1995 International Workshop on Distributed Systems Operations and Management, Ottawa, Canada, October 1995

code to capture the specified events and notify appropriate managers by the OSIMIS agents. The methodology used in enhancing the compiler and agents in OSIMIS could be easily applied to other implementations for similar results.

With the extensions we made to the OSIMIS GDMO compiler and agents, notifications could be triggered in agents automatically, but neither the manager application included in OSIMIS or the OSI DAM Manager [6] we developed earlier can handle event reports that agents generate in response to notifications. We plan to extend our OSI DAM Manager to handle event reports from agents. In doing so, we will be able to take full advantage of the new notification mechanism.

There are still many unanswered questions needing to be addressed. One leading question is whether or not the simple grammar for specifying conditions resulting in the triggering of notifications is sufficient. The answer, most likely, is no. We can already see that notifications for the creation or deletion of managed objects cannot be specified using the grammar proposed, as well as other conditions. The question then becomes, what kind of a specification is needed to define all of the conditions for triggering a notification (if one exists). We hope that providing a simple grammar that can handle many cases is a sufficient foundation for further experimentation to answer this question, as we gain more experience working with notifications in this way.

Other interesting future work would be to examine the use of behavioural specifications for actions, attributes, packages, *etc.* The work that we have discussed in this paper, and our experience with working with behavioural specifications for notifications would be very useful in completing this task. We have already demonstrated that such specifications for notifications can be very useful and can assist in the development of managed objects, agents, and management systems in general. Similar behavioural definitions for other structures of GDMO should be equally as valuable.

## References


[1] A. Clemm and O. Festor. Behaviour, Documentation, and Knowledge: An Approach for the Treatment of OSI-Behaviour. *4th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1993.

[2] J. Cowan. *OSIMIS GDMO compiler user manual*. Department of Computer Science, University College London, UK, August 1993.

[3] O. Festor and G. Zorntlein. Formal Description of Managed Object Behaviour – A Rule Based Approach. *Proc. of the Third International Symposium on Integrated Network Management*, pages 45–58, San Francisco CA, April 1993.

[4] J. W. Hong, M. J. Katchabaw, M. A. Bauer, and H. Lutfiyya. *Distributed Applications Management Using the OSI Management Framework*. Technical Report #448, Dept. of Computer Science, University of Western Ontario, London, Canada, January 1995.

[5] ISO. *Information Technology - Open Systems Interconnection - Management Information Services - Structure of Management Information Part 4: Guidelines for the Definition of Managed Objects*. International Organization for Standardization, International Standard 10165-4, 1991.

[6] M. J. Katchabaw, J. W. Hong, and M. A. Bauer. *OSI DAM Manager User's Guide Version 1.0*. Technical Report, *in preparation* Dept. of Computer Science, University of Western Ontario, London, Canada, 1995.

[7] G. Pavlou, S. N. Bhatti, and G. Knight. *The OSI Management Information Service User's Manual*. Version 1.0, University College London, UK, February 1993.

[8] G. Pavlou, G. Knight, and S. Walton. Experience of Implementing OSI Management Facilities. *Proc. of the Second International Symposium on Intergrated Network Management*, Washington DC, April 1991.

[9] G. S. Perrow. The Abstraction and Modelling of Management Agents. *MSc. Thesis*, Dept. of Computer Science, University of Western Ontario, London, Ontario, Canada, September 1994.

[10] G. S. Perrow, J. W. Hong, M. A. Bauer, and H. Lutfiyya. The Abstraction and Modelling of Management Agents. *Proc. of the Fourth International Symposium on Integrated Network Management*, Santa Barbara CA, May 1995.