OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) **Demande-Application**

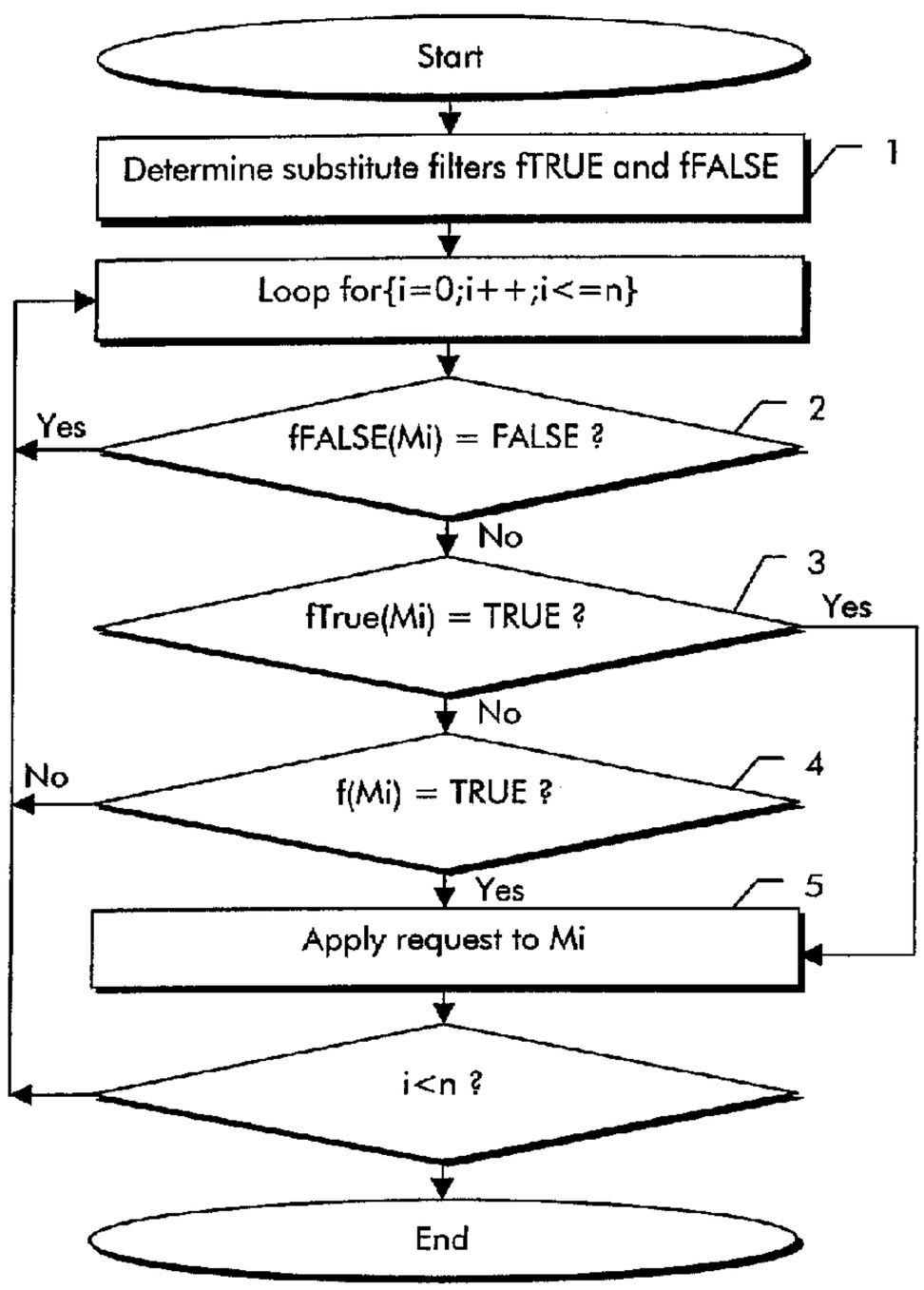
(21) (A1) **2,306,190**
(22) 2000/04/19
(43) 2000/11/19

(72) NUSCH, DIETMAR, DE
(71) ALCATEL, FR

(51) Int.Cl.[7] H04L 12/24
(30) 1999/05/19 (199 22 853.1) DE

(54) **METHODE D'EXECUTION D'UNE DEMANDE VENANT D'UNE INSTALLATION DE GESTION DE RESEAU**

(54) **METHOD OF PERFORMING A REQUEST FROM A NETWORK MANAGEMENT FACILITY**



(57) A communications network is managed according to the OSI Management Standard. In a controller (11) of a network element, requests from a network management facility are performed by being applied to managed objects (15, 16). Managed objects are mappings of physical and logical resources in accordance with an information model and are stored in a database (14) of the network element. The request contains a filter that specifies the managed object(s) to which the request is to be applied. To perform the request (17), the managed objects specified by the filter are loaded into a main memory (12) of the network element. To avoid having to load all managed objects from the database for comparison with the filter, according to the invention, at least one optimized filter is determined (1), which is applied to additional information (18) permanently available in main memory (12) and concerning the managed objects (15, 16) stored in the database (14), preferably to the naming tree, and which only contains attributes that are derivable from the additional information (18).

Abstract of the Disclosure

A communications network is managed according to the OSI Management Standard. In a controller (11) of a network element, requests from a network management facility are performed by being applied to managed objects (15, 16). Managed objects are mappings of physical and logical resources in accordance with an information model and are stored in a database (14) of the network element. The request contains a filter that specifies the managed object(s) to which the request is to be applied. To perform the request (17), the managed objects specified by the filter are loaded into a main memory (12) of the network element. To avoid having to load all managed objects from the database for comparison with the filter, according to the invention, at least one optimized filter is determined (1), which is applied to additional information (18) permanently available in main memory (12) and concerning the managed objects (15, 16) stored in the database (14), preferably to the naming tree, and which only contains attributes that are derivable from the additional information (18).

Method of Performing a Request from a Network Management Facility

This invention relates to a method of performing a request from a network management facility to a network element of a communications network as set forth in the preamble of claim 1, and to a network element as set forth in the preamble of claim 4.

Communications networks, such as synchronous digital networks, are managed according to the OSI Management Framework defined by the International Organization for Standardization (ISO). The general architecture of a network management system is based on a client-server architecture, with the server being referred to as an "agent", and the client as a "managed process".

A communications network consists of network elements that are connected with one another. Each network element has a controller, referred to as an "agent", which manages the local resources of the network element and is subordinate to a higher-ranking central network management facility. For its management tasks, the controller of a network element maintains a database, referred to as a "management information base" (MIB), in which the physical and logical resources of the network element are represented as managed objects (MO). All managed objects currently stored in the MIB are contained with a unique name in a naming tree and arranged there according to their object class membership.

Network management is provided by the central network management facility, which sends requests to the controllers of the network elements. This is done using a specific network management protocol, which is defined in ITU-T Recommendation X.710.

With the aid of these requests, data contents of the MIB of each network element can be retrieved and updated, whereby functions such as network supervision and network configuration can be performed. Such requests are aimed at retrieving or modifying individual managed objects or their parameters. As a rule, each request contains a filter that determines the managed object(s) to which the request is to be applied. In addition, each request specifies the so-called scope, which is the area or branch of the naming tree in which the managed objects in question are arranged. The scope thus relates to the fixed location of a desired managed object in the naming tree, while the filter selects managed objects based on their attributes.

Since the main memories in the network elements are limited, and because of the large number of managed objects, the latter are stored in a database on a hard disk ("persistency"). To be able to perform a request and apply it to a managed object, all managed objects that fall within the specified scope must be loaded from the database into main memory and be compared with the parameters specified in the filter. Compared with the actual evaluation of the filter, the loading of the managed object is a very time-consuming process.

It is therefore an object of the invention to provide a method of performing a request which is faster than prior-art methods. Another object of the invention is to provide a network element with which the method can be carried out.

These objects are attained by the features of claim 1 and claim 4, respectively. Further advantageous aspects of the invention are apparent from the dependent claims.

The invention provides a considerable saving of time in the performance of requests, particularly in network elements where a large number of managed objects are needed, so that many managed objects are not directly available in main memory but are swapped out to the hard disk. Therefore, the invention can be used to advantage in network elements for digital synchronous transmission networks such as SDH or SONET networks, for example in digital crossconnects and in add/drop multiplexers.

Advantageously, the invention can be implemented as a computer program that performs the steps of the method according to the invention when being executed on a computer.

The invention will become more apparent from the following description of an embodiment of the invention taken in conjunction with the accompanying drawing, in which:

Fig. 1 is a flowchart of the method according to the invention; and

Fig. 2 is a block diagram of a controller of a network element.

For the requests, a Common Management Information Protocol (CMIP) is used. The purpose of such a request is to retrieve or update one or more managed objects of a network element or of a logical layer in the management hierarchy to which the request was sent. The managed objects are defined by the parameters "scope" and "filter". The filter is a CMISE filter (CMISE = Common Management Information Service Element) and contains Boolean expressions which have to be applied to attributes of a managed object that are specified in the filter.

Thus, by evaluating the filter, all addressed managed objects must be selected by applying the Boolean expressions to all managed objects within

the specified scope. Since a large number of managed objects (several thousand) are stored in a network element, the filtering efficiency has a significant effect on the processing time of a request. However, a filter can only by applied to a managed object if the latter has been loaded into main memory. The invention eliminates the need to load at least some of those managed objects which have been swapped out to the hard disk but are not affected by the request into main memory.

The invention is predicated on recognition that even though a managed object itself may be swapped out to the hard disk, certain information about this managed object is permanently available in main memory. A basic idea of the invention is, therefore, to predict the result of the application of the filter of a request to a managed object based on such additional, permanently available information without having to load the object into main memory and compare it with the filter.

A preferred source of such information that is permanently available in main memory is the naming tree, in which all existing managed objects are registered with their unique names. Object attributes that can be derived from the naming tree without having to access the respective object itself are the *objectName* and the *objectClass*. These attributes are frequently used in CMISE filters.

If the attributes *objectClass* and *id* are queried in a CMISE filter, the following two filter expressions are possible (the queried values are given by way of example). The attribute *objectClass* is an attribute available from the naming tree, while the attribute *id* is only obtainable by direct access to the managed object.

1. (*objectClass == OID_Fabric && id == 5*); this filter will never match if the attribute *objectClass* does not have the value *OID_Fabric*, regardless of the value of the attribute *id*.

2. (*objectClass == OID_Fabric || id == 5*); this filter will always match if the attribute *objectClass* has the value *OID_Fabric*, regardless of the value of the attribute *id*.

Another basic idea of the invention is to first determine optimized filters from the CMISE filter of a request, the optimized filters serving as substitute filters and containing only attributes that can be derived from the information constantly available in main memory.

A preferred way of designing such substitute filters is to determine from a CMISE filter *f* two substitute filters *fTRUE* and fFALSE which fulfill the following three conditions:

1. *fTRUE* and f FALSE only use attributes that can be derived from the information permanently available in main memory;
2. if *fTRUE* matches for a managed object, then *f* also matches for the managed object; and
3. if *fFALSE* does not match for a managed object, then *f* does not match for the managed object, either.

Before giving an example of the design of substitute filters, the ASN.1 (abstract syntax notation one) used for CMISE filters will be considered. The ASN.1 definition of CMISE filters is as follows:

```
CHOICE CMISE Filter ::=
{
      FilterItem filteritem;
      SetOfCMISE Filter andfilter;
      SetOfCMISE Filter orfilter;
      CMISE Filter notfilter;
}

CHOICE CMISE FilterItem ::=
{
      Attribute equality;
      SeqOfSubstringChoice substring;
```

```
Attribute greaterOrEqual;
Attribute lessOrEqual;
AttributeId present;
Attribute subsetOf;
Attribute supersetOf;
Attribute nonNullSetIntersection;
}
```

The section "FilterItem" contains "simple" expressions for attributes of the CMISE filter. Such a CMISE filter is either

- a single, "simple" expression,
- a conjunction of CMISE filters (andFilters),
- a disjunction of CMISE filters (orFilters), or
- a negation of CMISE filters (notFilters).

The expressions "andFilter", "orFilter" and "notFilter" may, in turn, consist of two or more CMISE filters of any form.

To simplify the notation, the following assumptions are made in the following example of the construction of optimized filters: Let a be a filter item for an attribute of a managed object that is only obtainable by accessing the respective object; let x be a filter item for an attribute of a managed object that is derivable from the naming tree; let A be an arbitrary CMISE filter, and let f(x, a1, ..., an) be a CMISE filter that depends on the filter items x, a1, ..., an.

The substitute filter *fTRUE* for the CMISE filter *f* is determined by the following rules:

- If *f* consists of only a single filter item:
  fTRUE(present(a)) = FALSE
  fTRUE(present(x)) = FALSE
  fTRUE(a) = FALSE
  fTRUE(x) = x
- If *f* consists of the negation of a single filter item:
  fTRUE(NOT(present(a))) = FALSE

fTRUE(NOT(present(x))) = FALSE
fTRUE(NOT(a)) = FALSE
fTRUE(NOT(x)) = NOT(x)

- If *f* is a conjunction of single filter items:
  fTRUE(AND(A1, ..., An)) = AND(*fTRUE*(A1), ..., *fTRUE*(An))
- If *f* is a disjunction of single filter items:
  fTRUE(OR(A1, ..., An)) = OR(*fTRUE*(A1), ..., *fTRUE*(An))
- If *f* is the negation of arbitrary CMISE filters:
  fTRUE(NOT(AND(A1, ..., An))) = *fTRUE*(OR(NOT(A1), ..., NOT(An))
  fTRUE(NOT(OR(A1, ..., An))) = *fTRUE*(AND(NOT(A1), ..., NOT(An))
  fTRUE(NOT(NOT(A))) = *fTRUE*(A)

The substitute filter *fFALSE* for the CMISE filter *f* is determined according to the following rules:

- If *f* consists of only a single filter item:
  fFALSE(present(a)) = TRUE
  fFALSE(present(x)) = TRUE
  fFALSE(a) = TRUE
  fFALSE(x) = x
- If *f* consists of the negation of a single filter item:
  fFALSE(NOT(present(a))) = TRUE
  fFALSE(NOT(present(x))) = TRUE
  fFALSE(NOT(a)) = TRUE
  fFALSE(NOT(x)) = NOT(x)
- If *f* is a conjunction of single filter items:
  fFALSE(AND(A1, ..., An)) = AND(*fFALSE*(A1), ..., *fFALSE*(An))
- If *f* is a disjunction of single filter items:
  fFALSE(OR(A1, ..., An)) = OR(*fFALSE*(A1), ..., *fFALSE*(An))
- If *f* is the negation of arbitrary CMISE filters:
  fFALSE(NOT(AND(A1, ..., An))) = *fFALSE*(OR(NOT(A1), ..., NOT(An))
  fFALSE(NOT(OR(A1, ..., An))) = *fFALSE*(AND(NOT(A1), ..., NOT(An))
  fFALSE(NOT(NOT(A))) = *fFALSE*(A)

The expression *"AttributeID present"* plays a special role. It tests whether the managed object contains an attribute that is identified by its AttributeID, the value of the attribute being of no significance. The decision cannot be tied to the class of the object, because particular attributes may be optional. Therefore, the respective concrete object must be accessed, i.e., <u>all</u> attributes of the object are loaded and a check is made to see whether the attribute in question is contained in the object. Thus, the design rules for the substitute filters *fTRUE* and *fFALSE* must be adapted so that with *filterItem == present*, no assertion can be made only on the basis of the attributes available in main memory.

The request that contains the CMISE filter *f* is performed using the method according to the invention as shown in Fig. 1. Let *s* be the scope specified in the request; let {*M1, ..., Mn*} be the set of managed objects in the specified scope *s*; let {x1, ..., xm} be a set of attributes that can be derived for each managed object from the information permanently available in main memory without having to load the respective objects themselves into main memory.

Step 1: Determine the substitute filters *fTRUE* and *fFALSE* from *f* according to the above rules.

For each managed object *Mi* from the set {*M1, ..., Mn*}, steps 2 to 5 are performed (loop for (i=0; i++, i<=n))

Step 2: If *fFALSE* does not match for *Mi* (result FALSE), proceed with the next managed object *Mi+1*.

Step 3: If *fTRUE* matches for *Mi* (result TRUE), then load *Mi* into main memory and apply the request to *Mi* (step 5).

Step 4: Load the managed object *Mi* into main memory and apply *f* to *Mi*. If *f* matches for *Mi*, proceed with step 5; otherwise, begin with the next managed object *Mi+1* at step 2.

Step 5: Apply the request to *Mi*.

The following gives a simple example of a CMISE filter, the design of the substitute filters *fTRUE* and *fFALSE*, and the checking of the result:

```
original:  notFilter
orFilters
{
     filterItem equality objectClass OID_Fabric,
     filterItem equality operationalState disabled}
}

fTRUE:     andFilters
{
     notFilter filterItem equality objectClass OID_Fabric,
     orfilters {   }
}

fFALSE:    andFilters
{
     notFilter filterItem equality objectClass OID_Fabric,
     andfilters {   }
}

attrs:
{
     objectClass OID_Fabric,
     administrativeState locked,
     operationalState disabled,
     logRecordId 1
}
```

If applied to a managed object, *f* provides the result 0 (FALSE); *fTRUE* also provides 0, and *fFALSE* provides 0. The requirements placed on the design of the substitute filters are thus met: *fTRUE* is not satisfied and, therefore, does not permit an assertion about *f*. *fFALSE* is not satisfied; therefore, *f* is not satisfied, either.

Fig. 2 shows a controller 11 of a network element. Controller 11 contains a processor CPU, 13, a main memory RAM, 12, and a database MIB, 14, which is stored on a hard disk. The database contains managed objects M1, ..., Mn, 15, 16. The controller receives a request 17, which contains a filter in which those managed objects to which the request is to be applied are specified by means of their attributes. Processor 13 is responsible both for the evaluation of the filter and for the subsequent application of the request to the managed objects selected by means of the filter.

Processor 13 first determines from the filter at least one optimized, substitute filter which is applied to additional information NT, 18, about the managed objects M1, ..., Mn, 15, 16, stored in database 14, which additional information is permanently available in main memory 12. The substitute filter only contains attributes that are derivable from the additional information NT, 18. The additional information may be the naming tree NT, for example.

CPU 13 is responsible for applying the substitute filter to the additional information NT, 18, and, if necessary, i.e., if the evaluation of the substitute filter does not yield an unambiguous selection or nonselection for some managed objects, to load such managed objects into main memory 12 for further evaluation of the original filter, to load the selected managed objects into main memory 12, and to apply request 17 to the selected managed objects. After execution, the CPU sends a response to the network management facility that had sent the request. The response may be a confirmation of the successful performance of the request, an error message, or the result of the request.

The method according to the invention can be implemented using a computer program that is stored on a storage medium. The computer program includes a program coding means designed, for example, as a sequence of individual control instructions for a computer. All functions in the evaluation of a CMISE filter are executed by the computer program, for

instance a program function of the class FilterMgr. For each request, such a computer program is started, which is then responsible for the evaluation of the filter. One embodiment of such a computer program is given in the following as a program code in a programming language:

```
class FilterMgr
{
 public:
   FilterMgr(const CMISFilter& f);
   ReturnCode
     applyFilter(const DN& theMO,
                 bool& filterMatches);
 private:
   CMISFilter fTrue;
   CMISFilter fFalse;
   SetOfAttributeIds fastFilterAttrIds;
   SetOfAttributeIds allFilterAttrIds;
   SetOfAttributes& getFilterAttributes(const DN& theMO);
 };

 FilterMgr::FilterMgr(const CMISFilter& f)
 {
   // setup fTrue;
   // setup fFalse;
   // setup fastFilterAttrIds;
   // setup allFilterAttrIds;
 }

 ReturnCode
 FilterMgr::getFastFilterAttributes(const DN& theMO,
                                    const SetOfAttributeIds& theAttrIds,
                                    SetOfAttributes& theAttrs)
{
   SetOfAttributeIds getAttrIds(theAttrIds);
```

```
theAttrs.removeAll();
for (int i=0; i < getAttrIds.numMembers(); i++)
{
  assert(fastFilterAttrIds.contains(getAttrIds[i]);
  if (getAttrIds[i] == OID_ObjectClass)
  {
    theAttrs.unsafe_add(theMO.getObjectClass());
  }
  else
  {
    // must be the naming attribute
    theAttrs.unsafe_add(theMO.rdn().as_attribute();
  }
}
RETURN_OK();
}

ReturnCode
FilterMgr::getFilterAttributes(const DN& theMO,
                               const SetOfAttributeIds& theAttrIds,
                               SetOfAttributes& theAttrs)
{
  SetOfAttributeIds getAttrIds(theAttrIds);
  theAttrs.removeAll();
  // mGet(...) all filter attributes
  RETURN_OK();
}

ReturnCode
FilterMgr::applyFilter(const DN& theMO,
                       bool& filterMatches)
{
  // First evaluate the optimized filters using
  // the set of fast filter attributes...
```

```
SetOfAttributes theAttrs;
if (!getFastFilterAttributes(
                        theMO,fastFilterAttrIds,theAttrs))
{
  RETURN_ERROR(theErrno);
}
 if (!fFalse.setOfAttributesPasses(theAttrs))
 {
   filterMatches = FALSE;
   RETURN_OK();
 }
 if (fTrue.setOfAttributesPasses(theAttrs))
 {
   filterMatches = TRUE;
   RETURN_OK();
 }
  // ...then evaluate the original filter using
 //      the full set of filter attributes.
     if
 (!getFilterAttributes(theMO,allFilterAttrIds,theAttrs))
     {
       RETURN_ERROR(theErrno);
     }
     if (f.setOfAttributesPasses(theAttrs))
     {
       filterMatches = TRUE;
     }
     else
     {
       filterMatches = FALSE;
     }
     RETUNR_OK();   }
```

Patent Claims

1. A method of performing a request (17) from a network management facility to a network element of a communications network wherein physical and logical resources are represented in the network element as managed objects (15, 16) which are predefined in an information model, the request (17) including a filter which specifies the managed object or objects to which the request is to be applied, wherein the managed objects are stored in a database (14) of the network element, and wherein for the performance (5) of the request (17), the managed objects specified by the filter are loaded into a main memory of the network element, characterized in that
for the selection of the managed objects specified by the filter, at least one optimized filter is determined (1), which is applied to additional information (18) permanently contained in main memory (12) and relating to the managed objects (15, 16) stored in the database (14), and which only contains attributes that are derivable from the additional information (18).

2. A method as claimed in claim 1 wherein the additional information (18) comprises a naming tree in which all managed objects (15, 16) stored in the database (14) are registered with a unique name and arranged according to their object class membership.

3. A method as claimed in claim 1 wherein from the original filter *f*, a first optimized filter *fTRUE* and a second optimized filter *fFALSE* are determined which meet the following conditions:

   - If *fTRUE* matches for a managed object, then *f* also matches for this managed object; and
   - if *fFALSE* does not match for a managed object, then *f* does not match for this managed object, either.

4. A network element comprising a controller (11) for performing a request from a network management facility, a database (14) for storing managed objects (15, 16) which are representations of physical and logical resources of the network element, said representations being predefined in an information model, and a main memory (12) for loading those managed objects which are selected by the controller by means of a filter included in the request and specifying those managed objects to which the request is to be applied,
   characterized in that
   the controller (11) is provided for determining at least one optimized filter from the filter included in the request, said optimized filter being applied to additional information (18) permanently contained in main memory (12) and relating to the managed object (15, 16) stored in the database (14), and containing only attributes that are derivable from the additional information (18).

5. A computer program comprising a program coding means adapted to perform the steps of the method according to claim 1 when the program executes on a computer.

6. A computer-readable storage medium with a computer program stored thereon which comprises a program coding means adapted to perform the steps of the method according to claim 1 when the program executes on a computer.

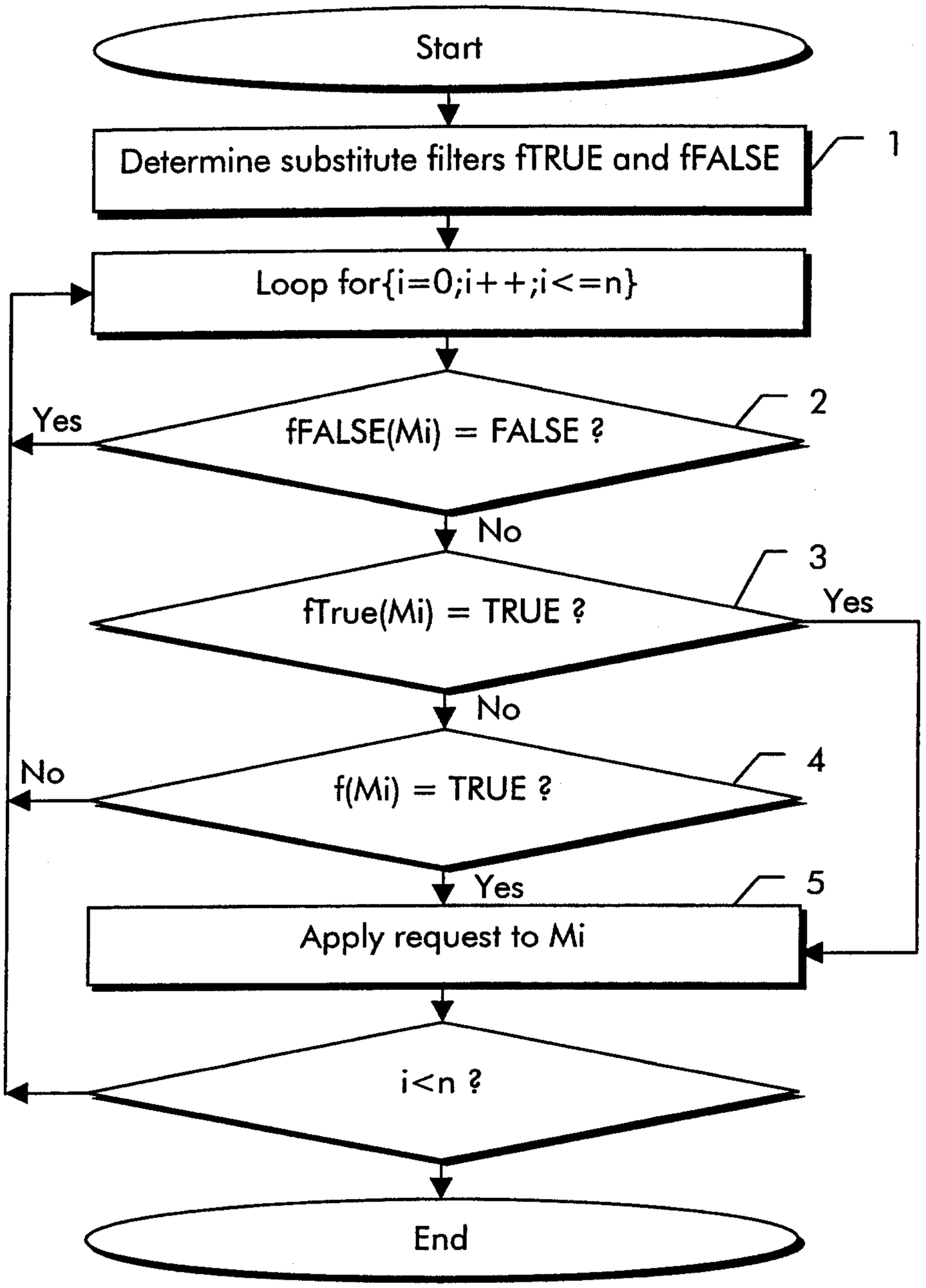

Start
Determine substitute filters fTRUE and fFALSE
1
Loop for{i=0;i++;i<=n}
fFALSE(Mi) = FALSE ?
2
Yes
No
fTrue(Mi) = TRUE ?
3
Yes
No
f(Mi) = TRUE ?
4
No
Yes
Apply request to Mi
5
i<n ?
End

Fig. 1

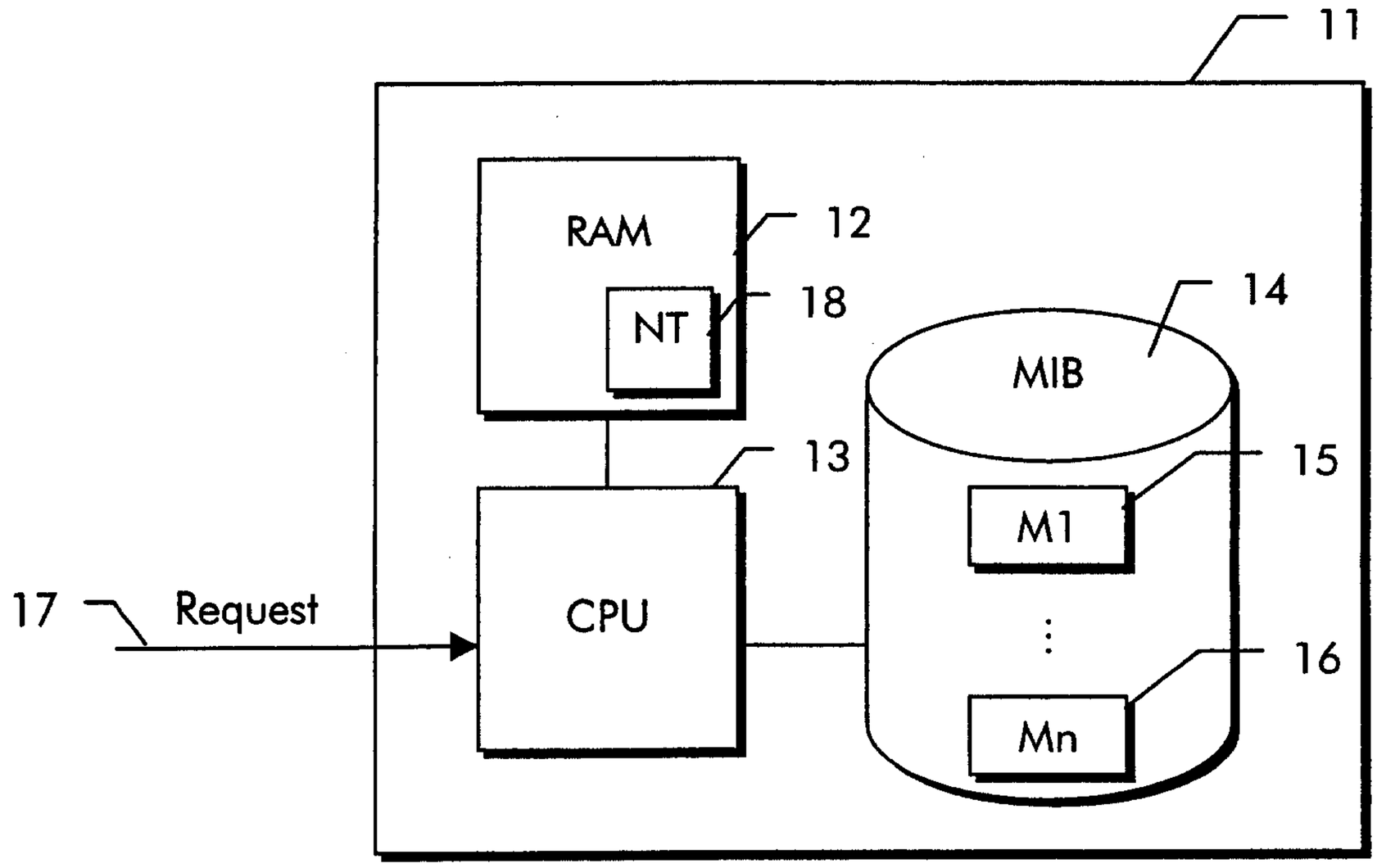
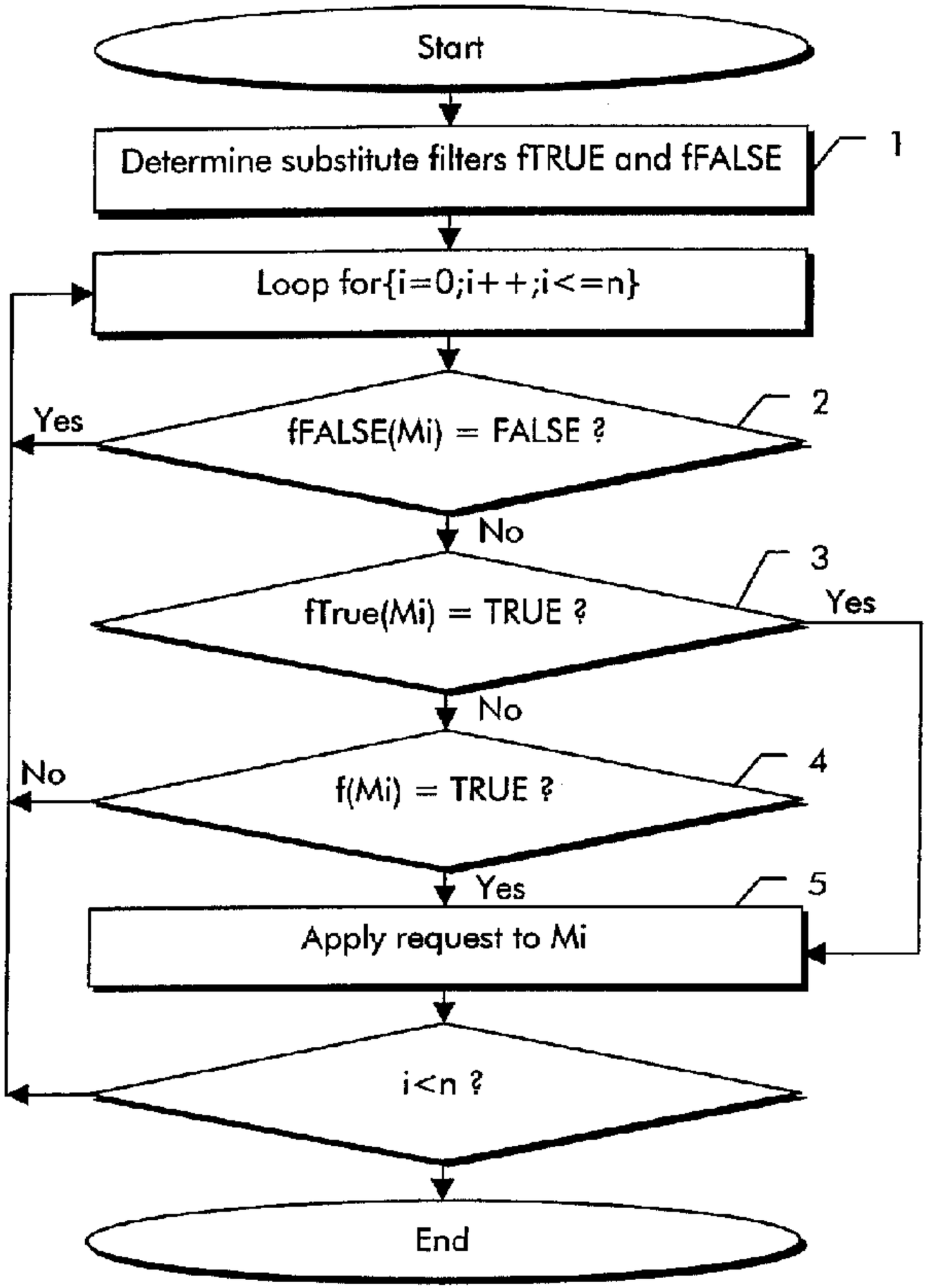

11
RAM
12
NT
18
MIB
14
13
15
M1
CPU
16
Mn
17
Request

Fig. 2


Start
Determine substitute filters fTRUE and fFALSE
1
Loop for{i=0;i++;i<=n}
fFALSE(Mi) = FALSE ?
2
Yes
No
fTrue(Mi) = TRUE ?
3
Yes
No
f(Mi) = TRUE ?
4
No
Yes
Apply request to Mi
5
i<n ?
End