Rule-Based CIM Query Facility for Dependency Resolution

Shinji Nakadai, Masato Kudo, and Koichi Konishi

NEC Corporation
s-nakadai@az.jp.nec.com

Abstract. A distributed system is composed of various resources which have mutually complicated dependencies. The fact increases an importance of the dependency resolution facility which makes it possible to check if there is given dependency between resources such as a router, and to determine which resources have given dependencies with other resources. This paper addresses a CIM query facility for dependency resolution. Its main features are ease of query description, bi-directional query execution, and completeness of query capability to CIM. These features are performed by a rule-based language that enables interesting predicates to be defined declaratively, unification and backtracking, and the preparation of predicates corresponding to CIM metamodel elements. To validate this facility, it was applied in servers dynamically allocated to service providers in a data center. The basic behavior of the query facility and the dynamic server allocation was illustrated.

1 Introduction

Today's computer network systems have become huge and heterogeneous, and the situation has induced operational mistakes from system administrators and increased operational costs. To solve these problems, the interest in autonomic computing has been growing. From the users' viewpoint, a fixed investment in servers and networks increases management risk and total cost, because the depreciation cost is a fixed cost, even though the business environment is dynamic. To solve these problems, several studies have been made on utility computing.

In this study, we focus on a dependency resolution facility [1,2] as one of the important functions in autonomic computing and utility computing. This dependency resolution facility is a facility that makes it possible to check if there is a given dependency between resources and to determine which resources have given dependencies with other resources. As for dependencies, the authors regard that some dependencies are directed and others are undirected. For example, in the case of an online bookstore, this service is hosted on a server, and the dependency *hosted* is thought to be directed. If the server has a connection with one switch, the dependency *having-connection* is thought to be undirected. It is noticeable that the dependencies such as *hosted* and *having-connection* can be combined into another dependency, an example of which is the dependency *Bookstore-Switch*. From the viewpoint of the

A. Sahai and F. Wu (Eds.): DSOM 2004, LNCS 3278, pp. 245–256, 2004.

ease of query description, this is the key-point in this study. The details of the ease of query description are described in Section 3.1.

The reason such a dependency resolution facility is important for autonomic computing is explained as follows. Suppose that under the circumstances of the above-mentioned bookstore service, some trouble occurs on three services simultaneously. Discovering a switch that has the dependency *Bookstore-Switch* with all three services may be useful for root cause analysis. The discovered switch can thus be regarded as a possible root cause. At the time of the recovery from the switch failure, an impact analysis is required, because separation of the switch may affect other irrelevant services. This analysis is realized by finding other services that have the above-mentioned dependencies with the switch. It is noticeable that service identifications should be retrieved from a switch identification in impact analysis, but vice versa in the case of root cause analysis. The capability to query bi-directionally thus enhance a reusability of query descriptions. As for utility computing, the dependency resolution facility makes it possible to match resource requests. In the following, we take the example of a service provider such as an online bookstore that is utilizing several servers provided by a data center (DC) and requests an additional server in the face of a workload increase. When a DC receives a request from a service provider, the dependency resolution facility makes it possible for the DC to resolve complicated requirements for a server, which include complicated dependencies with other resources.

Our approach to the dependency resolution is an association traversal on an information model representing dependencies between system components. The query description for the dependency resolution is realized by the declaration of what kind of dependency is to be traversed. In this paper, we adopt a Common Information Model (CIM) [5] as a target information model. The overview of CIM are described in Section 2.1.

Ease of query description, reusability of described query, and the completeness of capability are all required for retrieving data represented in CIM. To put it more concretely, ease of query description means that the description must be similar to the system administrator's concept that an interesting dependency (e.g., *Bookstore-Switch*) is composed of pre-known dependency (e.g., *hosted* and *having-connection*). In addition, the capability to query CIM without being aware of the CIM schema also contributes to the ease of query description, because the schema is strictly defined by Distributed Management Task Force, Inc. (DMTF) and is less readable. Reusability of a query means that an information retrieval is possible in both ways, even if it is composed of directed dependencies. It is desired, for example, that the same query can be used by root cause analysis and impact analysis. Completeness means that the query language should have sufficient capability to retrieve data of CIM. Our approach meets these requirements with the following features: use of a rule-based language, unification, backtracking, and unique built-in predicates.

The rest of the paper is organized as follows. In Section 2, we present backgrounds of the discussion and review related works. Section 3 describes the features and the architecture of our work. Section 4 shows the implementation applying utility computing. Finally, we conclude our paper in Section 5.

2 Background and Related Work

This section presents the backgrounds: CIM and Meta-level. CIM is a information model and Meta-level is an analysis framework for an information model. Related works are also described in this section.

2.1 CIM

DMTF is an industry organization that has provided a conceptual information model called CIM [5] in order to promote management interoperability among management-solution providers. The heterogeneity of the present management repositories makes it hard for system administrators to coordinate management information [3]. Differences in repository structures and query formats, for example, have worsened the interoperability and the reusability of management applications. To resolve these problems, it is important to divide data models, which represent a particular type of repository, from an information model that is independent of repositories. The latter is desired to be vender-neutral [3,4]. CIM is one of the industry-common conceptual views of the management environment. And Web-Based Enterprise Management (WBEM) is an implementation of management middleware that utilizes CIM. The dependency resolution facility described in this paper makes use of application programming interfaces (APIs) of WBEM.

2.2 Meta-level

The concept of Meta-level, which is discussed in the Object Management Group (OMG), is applied for a comprehensive discussion about an information model. The Meta-level is composed of four layers: the instance layer (short M0), the model layer (M1), the metamodel layer (M2), and the meta-metamodel layer (M3). Elements at lower layers are defined by upper layers. The element at M0 is a so-called *instance* which maintains state (e.g., ComputerSystem.Name = “host0”), and the element at M1 is a type of *instance*, that is, a *so-called class* (e.g., ComputerSystem class). M2 defines how to represent M1 elements. For example, the M1 element of CIM is defined by *class*, *property*, and *association*, which are the M2 elements. In this layer, CIM differs from other models such as Shared Information and Data (SID) [3,6], which is promoted by the TeleManagement Forum (TMF). For example, CIM defines that *association* is derived from a *class*, whereas SID defines that an *association* is not derived from a *class* and an *association-class* is derived from both *association* and *class*. Such relationships between the M2 elements are defined by M3. In this paper, CIM Metaschema is regarded as a metamodel (M2), CIM Core Model and Common Model are regarded as a model (M1), and CIM instance is regarded as an instance (M0).

2.3 WQL

Our proposal provides CIM with a query facility. As regards the query facility, WBEM Query Language (WQL) is a possible query language, which is a subset of SQL, and its basic structure is described below.

Select <Property> From <Class> [Where <Condition>]

Conditions on properties of CIM are inputted into the *Where* clause, and the *Select* clause indicates properties which are to be retrieved as output. This means that there is a static relationship between input and output and the query is thereby "one-way". Although the WQL is advantageous in terms of its well-known syntax, the M2 elements of RDB (e.g., *table* and *column*) do not correspond to those of CIM (e.g., *class*, *property*, *association*, and *qualifier*). This fact may make it difficult to retrieve *qualifier* or *property* of an *association instance*, even if the semantics of clauses are transformed.

2.4 XML, XPath, and RDF

An approach for managing dependencies with XML, XML Path Language (XPath), and the Resource Description Framework (RDF) has been proposed. Dependencies are defined using *class* and *property* of an RDF Schema (RDFS), which is a vocabulary definition language, and the model element might be one of CIM [7]. And actual dependency data is retrieved as an XML document from managed resources with instrumentations such as WBEM. The query is realized by an XPath Query Language, which does not have any reverse query mechanisms. The reverse query should hence be described, if it is required. This approach is, nevertheless, promising because it may utilize several advanced Semantic Web technologies.

3 Management System Using the Rule-Based CIM Query Facility

This section addresses the architecture of the management system using our CIM query facility. Section 3.1 describes the basic concept of the facility and overview of the architecture. Section 3.2 describes the basis of the query description. We discuss the sufficient capability to query CIM in Section 3.3 and the enhancement of the query usability in Section 3.4. Section 3.5 describes the interaction with external management applications and shows the capability to query bi-directionally.

3.1 Overview

In the following, M0 elements such as CIM *instances* and *association instances* are regarded as query targets. An *instance* represents the existence of a particular type of system component in a managed system, and *association instance* represents an existence of a particular type of relationship between system components. The types of *instance* and *association instance*, which are the M1 elements, therefore can be regarded as predicates that may become true or false depending on variables

representing the state of system components. The basic concept of our approach is that CIM model (M1) elements can be treated as predicates and such M1 predicates can be defined by M2 predicates, because M1 elements are defined by M2 elements. The definition is realized by a rule-based language. The details of M2 predicates are described in Section 3.3.

It is easy for system administrators to describe a query based on the rule-based predicate definition, because the concept is similar to one's way of thinking about a dependency in an actual management environment. For example, an interesting dependency such as *Bookstore-Switch*, as described in Section 1, can be regarded as a combination of the dependencies *hosted* and *having-connection*. This predicate definition is shown in Section 3.5.

The proposed CIM query facility, which deals with above-mentioned predicates, is similar to a Prolog processor. One predicate is replaced with a combination of other predicates recursively, unless it is a built-in predicate. If a built-in predicate is called, WBEM API is utilized to obtain M0 elements instead of unifying facts within the processor. This unification process including a backtracking-algorithm makes it possible to retrieve the M0 elements, which makes the interesting predicate true.

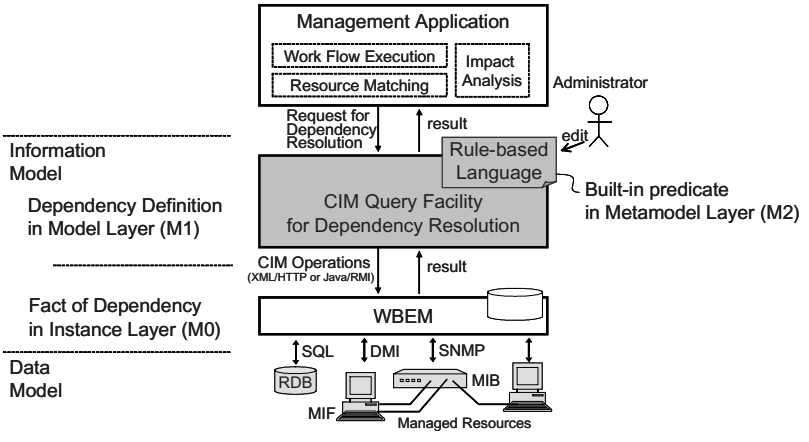

Fig. 1. Architecture of Management System

Fig. 1 shows the whole architecture of a management system using developed dependency resolution facility. Management applications are components with some specific management functions such as work flow execution, impact analysis, and resource matching. These management applications request the confirmation of dependency existence or query resources with some dependencies. The dependency resolution facility retrieves information one after another from WBEM in accordance with dependencies described by administrators. The actual dependency information is stored in WBEM as CIM *instances* and *association instances* (M0). These instances might be dynamic data or static data. Dynamic data might be retrieved from managed resources via WBEM on demand, while static data is stored in the repository of WBEM. The model in the managed resource can be thought as a data model, because

it might depend on some repository formats. The correspondences with the Meta-level are listed in Table 1.

Table 1. Correspondances to Meta-level

Meta-level		Example	This System
Instance Layer	M0	CIM_ComputerSystem.Name="host0"	Query Target (e.g. WBEM)
Model Layer	M1	CIM_ComputerSystem	Definition of Dependency
Metamodel Layer	M2	CIM::Class, Association, Property	Built-in Predicate
Meta-metamodel Layer	M3	MOF::Class	

3.2 Basis of Query Description

The way to describe dependencies is syntactically similar to Prolog. Fig. 2 shows samples of the description. As for the definition of a new predicate using a rule, the variable should be selected from a free variable or a bound variable. A free variable, which is shown by a question mark, is able to become a variable whose value is not yet decided. And a bound variable, which is shown by an exclamation mark must be a variable which value must be determined. In Fig. 2(c), the predicate is defined using a bound variable, so the term should be filled with a concrete variable as an input. The predicate with some bound variables has some restrictions on the direction of the query.

```
(a) computerSystem( ?compSys):-
        class( "ComputerSystem", ?compSys).
-----------------------------------------------------------------------------
(b) fileServer( ?fServer):-
        computerSystem( ?fServer),
        property( "Dedicated", ?fServer, 16).
-----------------------------------------------------------------------------
(c) linuxFileServer( !LFServer):-
        fileServer( !LFServer),
        association( "InstalledOS", !LFServer, ?opSys),
        class( "OperatingSystem", ?opSys),
        property( "OSType", ?opSys, 36).
```

Fig. 2. Examples of Query Description

3.3 Built-in Predicate

The examples of the built-in predicates, which are key-components of this query facility, are shown in Fig. 2. There are three built-in predicates: class predicate (Fig. 2(a)), property predicate (Fig. 2(b)), and association predicate (Fig. 2(c)). Furthermore, these predicates correspond to CIM operations: enumerateInstances, getProperty, and associator. This means that these predicates have restrictions on variables. The first term of each predicate should specify a model (M1) element, because each predicate represents the metamodel (M2) element. The second terms of a property predicate and an association predicate should be a bound variable, because these are input parameters of CIM operations. The second term of the class predicate and the third terms of the property predicate and association predicate should be free

variables, because these are dealt with the outputs of the operations. These correspondences are listed in Table 2.

Table 2. Built-in Predicates Corresponding to CIM Metamodel Elements

Predicates (M2)	Terms 1st (M1)	2nd (M0)	3rd (M0)	Corresponding CIM Operations
class	*bound variable* (name of a class)	*free variable* (instance)		enumerateInstance( )
property	*bound variable* (name of a property)	*bound variable* (belonging instance)	*free variable* (value of the property)	getProperty( )
association	*bound variable* (name of an association)	*bound variable* (associating instance)	*free variable* (associated instance)	associator( )

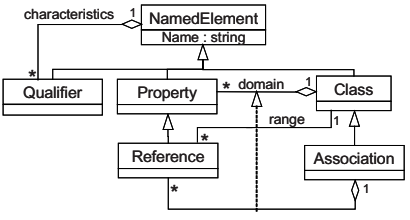

Fig. 3. CIM Metamodel (extracted from CIM Metaschema)

The reason these predicates are prepared is as follows. As described in Section 2.2, the M1 element is defined by the M2 elements. A predicate corresponding to a CIM model (M1) element is thus defined by the CIM metamodel (M2). The design of our predicates is as follows. Fig. 3 shows an extracted CIM metamodel. Since all M2 elements have a *name* property, all built-in predicates have a *name* term, which can specify an M1 element. A *Property* is aggregated by a *Class* and an *Association* aggregate multiple *References*, each of which is associated with a *Class*. These relationships are reflected on the 2nd terms and 3rd terms of the predicates. Though we list only three predicates in Table 2, another predicate can be mentioned as long as it reflects the relationship in Fig. 3. An example of the relationship is as follows: a *Qualifier* can be aggregated by any element and an *Association* can aggregate *Properties*. Usage of the *Qualifier* predicate may enable M0 elements of particular version of CIM to be queried. This design concept of built-in predicates is applicable to SID, which has a different metamodel from CIM.

3.4 Enhancement of Usability

This section describes a macro of the query for the enhancement of the usability. The macro described here means that pre-described predicates are combined into more readable predicates. This facility is important because CIM is designed on the basis of the concept that reusability among the industry is more important than the usability and readability. To enhance the reusability, managed resources are modeled in functional aspects. For example, a router is not modeled as a Router class, but as a combination of functional classes such as ComputerSystem and IPProtocolEndpoint. It is true that such a *divide-and-conquer* strategy is useful for reusability, but it is not so readable. It is therefore useful to re-organize these functional predicates into a

more usable and readable predicate. For example, predicate *fileserver* shown in Fig.2(b) is quite readable, while it is not so readable that a value of the *Dedicated* property of *ComputerSystem* class means the type of server.

Table 3. Predicate Stack

Predicate Stack		Example	Feature
Dependency Predicate	Definable	ActiveConnectionBetweenRouterAndFileServer(*?a*, *?b*).	Usability
Component Predicate		Router(?a). FileServer(?b).	↕
Model Predicate (M1)		ComputerSystem(*?a*). ActiveConnection(*!a*, *?b*).	↕
Metamodel Predicate (M2)	Built-In	Class("ComputerSystem", *?a*). Association("ActiveConnection", *!a*, *?b*).	Reusability

Since our rule-based language enables a new predicate to be defined by using predefined multiple predicates, it is easy to define the macro of the query naturally. Table 3 indicates a predicate stack as the guideline of macro definition. The predicates in the upper layer are defined by the predicates at the lower layer. Fig. 2(a) shows an example that a model predicate is defined by a metamodel predicate, and Fig. 2(b) shows an example that component predicate is defined by a model predicate. Predicates at the lower two layers depend on CIM, while predicates at the upper two layers are independent of CIM and are suitable for management applications and system administrators. We thereby suppose that the predicates at the lower layer are defined by those who are familiar with CIM and predicates at the upper layer are defined by those who describe a query for some management applications.

3.5 Usage of the CIM Query Facility

The interaction between this rule-based CIM query facility and management applications is as follows. There are two patterns in a dependency resolution. One is the pattern that resources are queried in accordance with defined dependencies (Pattern 1), and the other is the pattern that the existence of the dependency is checked (Pattern 2). These patterns have the same semantics as a Prolog.

Pattern 1: The input to CIM query facility is a predicate and its list of parameters (Fig. 4). If some parameters are filled with data and the others are filled with *null*, the filled data act as a key to a query, and the parameters filled with *null* can be retrieved from WBEM. It is therefore possible to execute a reverse query using a same query description, which is impossible using WQL. In the example of bookstore service discussed in Section 1, Fig. 4(c) shows the query for impact analysis, while Fig. 4(b) shows the query for root cause analysis.

(a)
```
BookstoreSwitch( ?bookstore, ?switch) :-
    Bookstore( ?bookstore),
    hosted( ?bookstore, ?server),
    Server( ?server),
    HavingConnectivity( ?server, ?switch),
    Switch( ?switch).
```

(b)
```
Input  : BookstoreSwitch( Bookstore_1, null)
Output : { ( Bookstore_1, switch_1),
           ( Bookstore_1, switch_2),
           ( Bookstore_1, switch_3) }
```

(c)
```
Input  : BookstoreSwitch( null , switch_1)
Output : { ( Bookstore_1, switch_1),
           ( Bookstore_2, switch_1)}
```

Fig. 4. Examples of Input and Output

Pattern 2: Filling all terms with some values makes it possible to check if given dependencies exist or not. If there is a given dependency in WBEM, the value *true* is returned.

4 Prototype Implementation

This section describes the use case of utility computing for an illustration. In Section 4.1, a service model is described and a required sequence are described. In Section 4.2, we show an example of the query description, and an evaluation of query performance is shown in Section 4.3.

4.1 Service Model

We utilize our dependency resolution facility for utility computing. We suppose that servers in a data center (DC) are shared by several service providers. When workloads on allocated servers increase, the service provider may request an additional server with some requirements on resources such as the type of operating system and IP address range. It is regarded that the service provider is a virtual organization (VO) as defined in [8]. In addition, it is assumed that whether the administrators of a VO can monitor resource information such as servers and network devices depends on the access control policy of a DC. For example, Fig. 5 shows the context that the monitor of the identifications of network devices such as firewalls are restricted to a VO, and what can be monitored is the assigned servers and their locations such as DMZ or internal-LAN. Therefore, when the DC receives a request for an additional server, it needs to retrieve detail information about pool servers and network devices in order to realize a resource matching [9] and filling parameters of workflow templates. The CIM query facility is applied to this information retrieval. To put it more concretely, among the following steps that consists an overall sequence of our implementation, the facility is used in Step 2 and Step 5.

Step 1: VO requests an additional server with some requirements on resources.
Step 2: DC collects servers which match the request among the pool servers.
Step 3: DC selects the most suitable server among the collected servers.
Step 4: DC prepares a workflow template required for the configuration change.
Step 5: DC fills the workflow with parameters.
Step 6: DC executes the completed workflow.

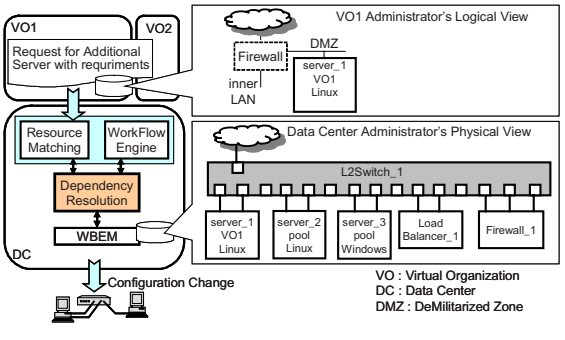

Fig. 5. Architecture of the Prototype System

The configuration change is realized by the control of servers and network devices such as a layer 2 switch (L2SW), a firewall, and a load balancers. In the prototype system, we use an NEC ES8000 L2SW, a Cisco PIX 515E firewall, and an Alteon

ACEDirector3 load balancer as managed resources. And we use WBEMServices as a WBEM server, and its runtime environment is as follows: Linux RedHat 7.3, Celron 1.7 GHz CPU, and 512 MB Memory.

4.2 Predicate Definition for Resource Collection

This section describes the outline of the experimental implementation according to the above-mentioned sequence. In particular, we introduce a sample query and an object diagram which represents objects existing in a WBEM server.

In Step 1, a VO generates the request for a server with following requirements.

Requirement 1: Linux OS is required.

Requirement 2: The domain at which the server is to be allocated is DMZ.

Requirement 3: The server's IP address should be within the subnet 192.168.10.0.

In Step 2, the resource matching facility collects pool servers which have Linux OS. In our prototype, pool servers are represented as the servers belonging to the *pool* organization. The collection is thereby realized by the query shown in Fig. 6, and the object diagram which represents the target of query is shown in Fig. 7.

```
CompSysInVLANofOpSysOrg( ?compSys, ?vlanid, ?osType, ?org):-
   Organization            ( ?orginst),
   property                ( "Organization",              ?orginst,          ?org),
   OrganizationDependency  ( ?orginst,                    ?vlan),
   property                ( "VLANId",                    ?vlan,             ?vlanid),
   EndstationInVLAN        ( ?vlan,                       ?vlan_endstation_endpoint),
   EndpointIdentity        ( ?vlan_endstation_endpoint,   ?ip_protocol_endpoint),
   EndpointIdentity        ( ?ip_protocol_endpoint,       ?lan_endpoint),
   PortImplementsEndpoint  ( ?lan_endpoint,               ?ether_port),
   SystemDevice            ( ?ether_port,                 ?compSys),
   OperatingSystem         ( ?os),
   property                ( "OSType",                    ?os,               ?osType),
   RunningOS               ( ?os,                         ?compSys).
```

Fig. 6. Query Description

If the CIM query facility receives the predicate *CompSysInVLANofOpSysOrg* and its parameters *(null, null, 36, "pool")*, the list *("server_2", 5, 36, "pool")* is returned to the resource matching facility. The fact that dedicated property is 36 means that the type of OS is Linux, as shown in Fig.2(c). If there are plural appropriate servers, the plural lists are returned. The resources are thereby collected.

In Step 3, one server is selected from the collected servers on the basis of a first-match strategy. In Step 4, a hard-coded workflow template is retrieved. This workflow is filled with appropriate parameters in Step 5. The retrievals are realized by the CIM query facility, that is, specifying of network device such as a switch, a load balancer, and a firewall, and the retrieval of required configuration data such as administrative IP addresses, port numbers, and VLAN numbers. In Step 6, the completed workflow is executed and the result of the execution is reflected on WBEM.

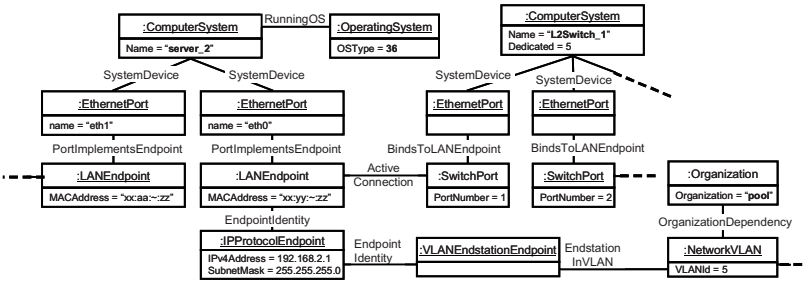

Fig. 7. Object Diagram of Query Target

4.3 Performance Evaluation

The execution time used in the all steps was 59.6 seconds. This performance was observed under an experimental condition that a CIM query facility was connected with WBEM using XML/HTTP. After the protocol of the connection was changed to Java/RMI, the execution time improved 45.2 seconds. This implies that our approach, which can exclude an XML parser, has an advantage in terms of the execution time. Furthermore, we provided a cache and connection-pooling with a CIM Query Facility and we obtained the execution time of 32.5 seconds. Under this condition, the execution time from Step 1 to Step 5 is about 16 seconds, while the method enumerateInstances, getProperty, and associators was called 12 times, 114 times and 114 times respectively. We have confirmed much time was consumed by the responses from WBEM and the overhead of the query facility was negligible.

The scalability issue was investigated by changing the total number of instances and association instances existing in the CIM repository of WBEM. Fig. 8 shows the execution time of a similar query. The figures in the graph indicate the number of pool servers. This result indicates the scalability problem, though the cause is supposed to stem from the usage of the WBEM API of the enumerateInstances method.

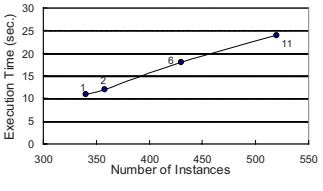

Fig. 8. Result of Scalability Investigation

5 Conclusion

We focus on the dependency resolution facility among the important issues in autonomic computing and utility computing. The discovery of system components

which have particular dependencies is realized by an association traversal, and therefore we enhance a query facility of CIM. The required features of the facility are ease of query description, bi-directional query execution, and sufficient capability to query CIM. This CIM query facility is based on a predicate logic and a rule-based language. The ease of query description is realized by the rule-based language that can combine multiple predicates into a new predicate representing a query, because it is similar to the system administrator's concept that an interesting dependency is combined with pre-known dependencies. The capability to define a macro also contributes to the ease of description, because the model element of CIM is based on the design concept that a reusability takes priority over an usability and readability. Bi-direction query execution can be realized by the unification process. Sufficient capability to query CIM can be realized by the preparation of the built-in predicates corresponding to CIM metamodel elements. The discussion based on a Meta-level indicates that our approach is independent of CIM. The proposed CIM query facility was validated by implementing it in a utility computing application. The basic behavior of the query facility and the dynamic server allocation was illustrated.

References

[1] A. Keller, U. Blumenthal, and G. Kar, "Classification and Computation of Dependencies for Distributed Management," 5th IEEE Symposium on Computers and Communications (ISCC), July 2000.
[2] A. Keller, and G. Kar, "Determining Service Dependencies in Distributed Systems," IEEE International Conference on Communications (ICC), June 2001.
[3] J. Strassner, "Policy Based Network Management : Solutions for the Next Generation," Morgan Kaufmann, Aug. 2003.
[4] A. Westerinen, et al., "Terminology for Policy-Based Management," IETF RFC3198, Nov. 2001.
[5] CIM standards. http://www.dmtf.org/standards/standard_cim.php
[6] TMF, "GB922: Shared Information/Data (SID) Model: Concepts, Principles, and Business Entities," July 2003.
[7] C. Ensel, and A. Keller, "Managing Application Service Dependencies with XML and the Resource Description Framework," IFIP/IEEE International Symposium on Integrated Management (IM2001), May 2001.
[8] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid," International J. Supercomputer Applications, 2001.
[9] H. Tangmunarunkit, S. Decker, and C. Kesselman, "Ontology-based Resource Matching in the Grid - The Grid meets the Semantic Web," 1st Workshop On Semantics in P2P and Grid Computing at the 12th International World Wide Web Conference, May 2003.