An Overview of IDEA Network Management Platform

Manoel Camillo Penna

Centro Federal de Educação Tecnológica do Paraná
CPGEI

Av. Sete de Setembro, 3165
80230, Curitiba, Paraná, Brasil
CEFETPR@BRFAPESP.BITNET

Abstract: This paper is an overview of the IDEA network management platform. The goal of this platform is to provide a general solution for constructing network management systems over heterogeneous and distributed environments. IDEA platform proposes an implementation architecture to integrate all aspects of network management. The resulting platform corresponds to a runtime system, a repository for storing the management model and information for configuring the platform, and a collection of tools for customizing the runtime modules and creating the repository.

1. INTRODUCTION

This paper presents an overview of IDEA[†] network management platform. It is a set of software tools that allows the development of network management systems over distributed and heterogeneous computing systems. This platform corresponds to an IDEA support environment instantiated over a standard distribution infrastructure conforming to Open Distributed Processing (ODP).[1]

The support environment is an integrated repository of management information; a collection of tools; and a runtime system. Its goal is to provide the necessary means for constructing network management systems according to IDEA methodology. This methodology has been developed within MASI laboratory at University of Paris 6, and its goal is to define a

† Intelligence, Diagnostic, Expertise, Administration

instances within a module, and the latter supports their execution. The configuration manager is responsible for creating managed objects within modules. Three IDEA modules and respective tools are discussed in this paper: Integrated Network Data Base (INDB), Proxy and Configuration Manager. The repository and the platform construction tools are also discussed.

2. INDB

The INDB derives from the concept of "model-based network management." [5] This term describes network management systems in which management applications are supported in their tasks by a conceptual model. All shared management information is defined in this model. It also maintains the consistency and correctness of shared information, and provides a single mean to access it. A "shared conceptual schema" is the instantiation of a conceptual model.

The INDB module offers three services: a schema management service, a model updating service, and a notification service. Two functions support these services: storage function and representation function. They support INDB services by providing, respectively, object persistence and control of interaction with proxies (see Section 3).

The schema management service[6] provides a unified view of the underlying managed system. Requests to this service are resolved in two phases: the interpretation phase and the response phase. The interpretation phase accepts common management information service (CMIS)-compatible requests, translates them into simpler requests, and targets them at the appropriate INDB support function. The original request contains the necessary information to identify the needed function. The response phase collects all results and builds a response compatible with CMIS. The schema management service relies on shared conceptual schema and the storage function to perform its task.

The model updating service[7] monitors the state of real elements and updates object information stored within the management model. An interface is offered that allows the monitoring of managed-object states to be externally controlled. The model updating service relies on both INDB support functions to accomplish its task.

Event management is performed by the notification service. This task includes the notifying management applications about events and maintaining a log of those events. An interface allows the registration of management application interest in notifications coming from managed objects. Figure 1 depicts the INDB structure.

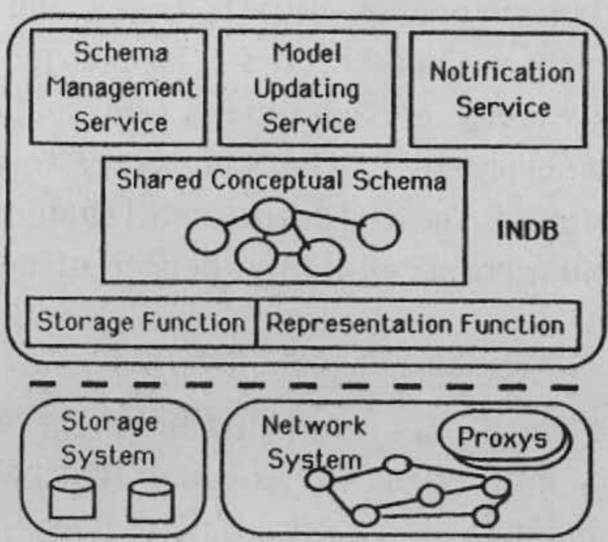

Figure 1 - INDB Structure

3. PROXY

A proxy provides a translation model for real resources within a specific environment. It contains the code necessary to access the real resources, unify the resource representation, and offer an object interface for the represented real resources. Its task is to interact with real resources by accessing information and capturing unsolicited events and to make real resource information available by an object interface. This object interface is compatible with ISO SMI (Figure 2).

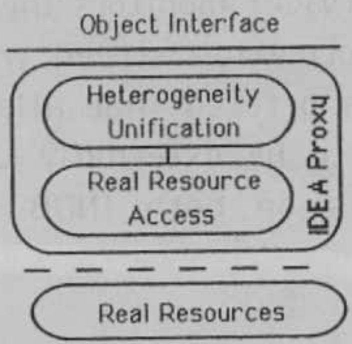

Figure 2 - IDEA Proxy

A proxy provides an object representation for real resources. One managed-object instance representing a real resource within a proxy is a structure composed of attributes and operations that encapsulates an internal realization. Two object properties are necessary at proxy level: encapsulation and spatial and temporal independence. Polymorphism is also necessary to allow one object model to be assigned to different realizations. Proxies are strongly typed, which enforces a high degree of correctness during compilation.

IDEA includes a method for unifying heterogeneous management information. This method splits the unification process in two logical phases: semantics and syntax unification. The construction of a translation model is a three-step procedure based on the definition of abstract data types: [8,9]

Step 1: A unified access interface (semantics interface) is specified as a set of attribute types and operation signatures. This interface has existential type. Its type is defined with respect to the existence of syntactical interfaces.

Step 2: The underlying heterogeneous information is structured in sets of homogeneous information. Each set is named a heterogeneity domain and has a syntax interface that is similarly specified in terms of attribute types and operation signatures. The definition of syntax interfaces depends on the information that is available within the domain.

Step 3: For each heterogeneity domain, the methods for interacting with real resources and for processing the retrieved information are coded by the expert's domain. Similarly, all methods for accessing syntax interfaces and for processing the semantics unification must be provided.

4. CONFIGURATION MANAGER

Dynamic configuration of IDEA platforms is performed by the configuration manager. The platform includes objects in two levels of granularity, leading to two major configuration functions. The first function provides for the creation and destruction of modules, and second for the creation and destruction of managed objects within modules. Each function maintains a hashed table indexed by the object identifier that contains all necessary information to control the object state.

The key element for interactions between objects is an interface reference. Any object possessing an interface reference can initiate an operation on the referred interface.[1] When a module is created, it returns an interface reference that allows the creation of managed objects within it. When a managed-object instance is created within a module, it returns another interface reference having an operation that allows the creation of further interface references pointing to the interface of this instance. These references are stored in the corresponding configuration manager table.

The configuration manager initiates all IDEA modules and managed objects. Its bootstrap procedure starts the initial configuration according to the repository information. Moreover, an interface is provided allowing for dynamic creation and destruction of objects at both levels. Another interface is provided for monitoring which allows an external object (a monitor) to obtain the control information stored within the internal tables.

5. TOOLS

Managed-object classes are defined by an object-oriented specification language. Declarations of managed-object classes are very similar to ISO SMI managed-object class definitions. The IDEA platform includes pre-processor for helping the translation from the model defined in the repository and the implementation of managed-objects within INDB.

The input to the INDB pre-processor is a collection of specifications of managed-object classes (e.g., the Circuit class defined in Annex 1). The pre-processor generates a runtime module with the corresponding INDB services. Managed-object implementations are coded in C++, and the INDB pre-processor type-checks this code with respect to specifications. An option is available to generate a C++ skeleton program containing only those class declarations that are to be filled with the appropriate C++ code.

Semantics and syntax unification are expressed in a proxy structuring language, which is an extension of the managed-object specification language. This language allows the structuring of C functions in software templates that implement the operations of a proxy class.

The compilation of a proxy software template produces a proxy module. This module is able to produce managed-objects having the specific knowledge necessary to implement the unification mechanism. This knowledge is composed by the C functions provided by an expert. According to IDEA unification method, managed objects within proxies must perform the semantics unification based upon several syntax unification functions. A specific paragraph named SIGNATURES is provided for specifying the semantics interface and a collection of syntax interfaces.

The semantics interface is specified by an OBJECT TYPE declaration. The methodology relies on the organization of the syntax unification functions in homogeneous groups, according to some ad-hoc criteria (e.g., functions to interact with real resources of a particular manufacturer). A group of functions is characterized by an interface type that is specified by an INTERFACE declaration containing function signatures. Semantics and syntax unification are implemented by collections of C functions. A REALIZATION paragraph identifies the multiple files containing these C functions. Annex 2 shows the code for both SIGNATURES and REALIZATION paragraphs for the same Circuit class defined in Annex 1.

6. REPOSITORY

The IDEA repository is a collection of structured files that contain all entries necessary to define an IDEA network management platform. Files are organized under three file directories: CLASS, CONFIGURATION, and TEMPLATE.

A managed-object class is defined in CLASS subdirectories. There is one subdirectory per managed-object class, which contains all entries concerning this class. The CLASS directory is structured as shown in Figure 3.

A SPECIFICATION file contains the definition of an object class. An example is given in Annex 1. The <Class Name>.cc file contains the C++ implementation of this class within the INDB. If a class is represented by a proxy, it must include a subdirectory named PROXY.. The SIGNATURES file contains the signatures of syntax interfaces (see Annex 2). Each object instance matching the proxy type corresponds to a subdirectory that includes a REALIZATION file, i.e., an OBJECT entry in the REALIZATION paragraph (Annex 2) and both the SEMANTICS file and SYNTAX files containing the C functions that implement the unification process for this object.

The TEMPLATE directory contains software template specifications used by the "extractor" in building input to pre-processors. An extractor is a utility that is responsible for constructing proxy software templates from repository entries. A proxy software template is described by a special repository entry containing a selection criterion. This criterion permits the extractor to build a file structure corresponding to a proxy software template.

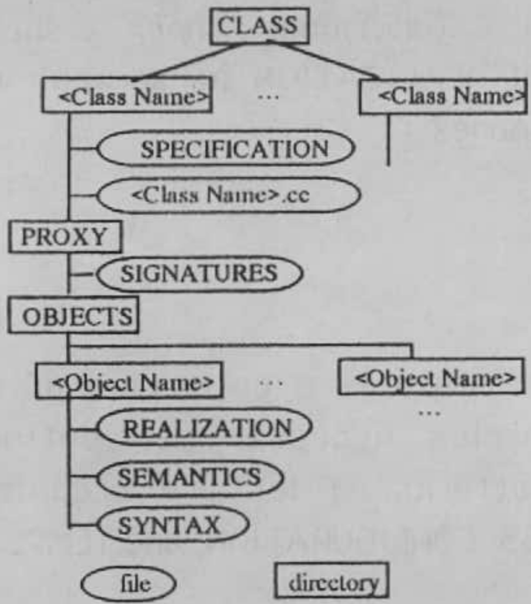

Figure 3 - Class Directory

The criterion can be specified either as the enumeration of all objects regrouped within the template or according to the CONTEXT entries founded in REALIZATION declarations. For example, the next two declarations specify two proxy templates. Template1 contains exactly the objects unixCpu1 and myCpu1, in contrast to Template2, which includes all objects having an internal representation that offers UnixCpu interface (unixCpu1 and unixCpu2 in the example).

```
PROXY TEMPLATE Template1 ::=
BEGIN
  OBJECTS
  { unixCpu1, myCpu1 }
END [ PROXY TEMPLATE Template1 ]
```

```
PROXY TEMPLATE Template2 ::=
BEGIN
   INTERFACE UnixCpu;
END [ PROXY TEMPLATE Template2 ]
```

The CONFIGURATION directory contains the information for configuring the platform. Two major kinds of configuration entries describe the configuration of modules and managed-object instances. Examples of both cases can be found in Annex 3.

7. CONCLUSION

The current version of the IDEA network management platform is implemented over ANSAware 1.0. In this implementation, IDEA modules correspond to ANSAware capsules. The IDEA configuration manager relies on factory functions to perform instantiation. It corresponds to approximately 3000 lines of code (C, DPL, and IDL). The proxy pre-processor was developed with lex and yacc, and its grammar contains 153 production rules. The code contains approximately 6000 lines of C code. We have advanced prototypes for the following INDB components: schema manager, representation function, and model updating.

IDEA concepts and tools have been validated in several research co-operations. Two European research projects have made experiments involving IDEA concepts and tools: ADVANCE (RACE Project 1009) and PEMMON (ESPRIT II Project 5371). The former applies Advanced Information Processing (AIP) techniques to building Network and Customer Administration Systems (NCAS). In this project, the IDEA unification technique is used to access heterogeneous real resources. The IDEA network management platform is the basis of the latter, whose goal is to construct a performance management platform over X.25 and Ethernet networks.

REFERENCES

1. International Standards Organization, ISON6080 Open Distributed Processing - Basic Reference Model (Part 3).

2. J. P. Claudé, Proposition d'une Spécification d'Administration de Réseaux Hétérogènes, Thèse d'Etat de l'Université Paris 6, Paris, France, September 1987.

3. J. P. Claudé et al., Unification of Heterogeneous Management by a Generic Object Oriented Agent, Proceedings of 4th RACE TMN Conference, Dublin, Ireland, November 1990.

4. International Standards Organization, ISO10165 Open Systems Interconnection - Management Information Model.

5. K. Manning and D. Spencer, Model Based Management Network, Proceedings of 4th RACE TMN Conference, Dublin, Ireland, November 1990.

6. PEMMON ESPRIT II Project 5371, Definition of a Performance Data Model and Environment Support - Deliverable 15, November 1991.

7. N. Agoulmine et al., Intelligent Model Updating System in Heterogeneous Network, Proceedings of International Conference on Communication Technology, Beijing, China, September 1992.

8. M. Penna, Mandataire Générique: Un Concept pour l'Intégration d'Environnements Distribués, Thèse de Doctorat de l'Université Paris 6, Paris, France, June 1992.

9. M. Penna, The Design of IDEA Network Management Platform, International Journal of Network Management, accepted for publication.

10. Architecture Projects Management Limited, ANSAware 3.0 Implementation Manual, January 1991.

Annex 1
SAMPLE SPECIFICATION FILE

```
CLASS CPU::=
BEGIN
DERIVED FROM (top)
CHARACTERIZED BY (
  ATTRIBUTES
    MUST CONTAIN (
      state,
      bust_time: READ ONLY
    )
  OPERATIONS (
    CREATE, DELETE,
    ACTIONS
    ( Activate, Deactivate )
  )
  NOTIFICATIONS (
    alarm_report,
    change_report
  )
)

LOCAL TYPES (
  AttributeChange ::=
    SEQUENCE OF
    RECORD (
      attrId:    STRING,
      oldVal:    ANY DEFINED
            BY CpuAttributes
    )
  Severity ::=
    ENUMERATED (
      cleared(0),
      informational(1),
      minor(2),
      major(3),
    )
  SeverityTrend ::=
    ENUMERATED (
      lessSevere(0),
      noChange(1),
      moreSevere(2)
    )
  AlarmReport ::=
    SEQUENCE OF
    RECORD
    (
      severity: Severity,
      severitytrend:
          SeverityTrend,
      backedUpStatus:
          BOOLEAN,
      backedCpuInst: STRING,
      eventId: INTEGER
    )
  State ::=
    ENUMERATED
    (
      preService(0),
      inService(1),
      outOfService(2)
    )
)

ATTRIBUTE state
WITH ATTRIBUTE SYNTAX State
  MATHES FOR ORDERING
  SINGLE VALUED
ATTRIBUTE busy_time
With SYNTAX ATTRIBUTE REAL
  MATCHES FOR ORDERING
  SINGLE VALUED

ACTION Activate
  ACTIONARG NULL
ACTION Deactivate
  ACTIONARG NULL

NOTIFICATION change_report
  EVENTINFO AttributeChange
NOTIFICATION alarm_report
  EVENTINFO AlarmReport

END [CLASS Cpu]
```

Annex 2
SAMPLE SIGNATURES AND REALIZATION FILES

```
SIGNATURES Cpu ::=
BEGIN
OBJECT TYPE ::= {
  CREATE : (STRING)->(FD, FD);
  DELETE : (STRING)->( ),
  EVENT REPORT:
  (FD, NotifyType)->(BOOLEAN);
  ACTIONS ::= {
    Activate:   ( )->( ),
    Deactivate: ( ) ->( )
  }
  ATTRIBUTES ::= {
    state {
      GET: ( )-> (State),
      SET: (State) ->
(BOOLEAN)
    },
    busy_time
    { GET: ( )->(REAL) }
  }
}

INTERFACES ::= {

  INTERFACE UnixCpu ::=
  BEGIN
    Login:        (STRING)->(FD),
    Logout:       (FD)->( ),
    UserBusyTime:
      (FD)->(REAL),
    SystemBusyTime:
      (FD)->(REAL),
    Reset:        ( )->( )
  END [ INTERFACE UnixCpu ]

  INTERFACE MyCpu ::=
  BEGIN
    MyCtrlnf: SEQUENCE OF
              OCTET,
    Connect:    (STRING)->(FD),
    Disconnect:  (FD)->( ),
    Ctrlinfo: (FD)->(MyCtrlnf),
  END [ INTERFACE MyCpu ]
}
END [ SIGNATURE Cpu ]

REALIZATION Cpu ::=
BEGIN
  OBJECT unixCpu1 ::=
    SYNTAX ::= {
      INTERFACE UnixCpu,
      METHODS
        unicCpu1.SYNTAX
    }
    SEMANTICS
      unixCpu1.SEMANTICS;
  END [ OBJECT unixCpu1 ]

  OBJECT unixCpu2 ::=
    SYNTAX ::= {
      INTERFACE UnixCpu,
      METHODS
        unixCpu2.SYNTAX
    }
    SEMANTICS
      unixCpu2.SEMANTICS;
  END [ OBJECT unixCpu2 ]

  OBJECT myCpu2 ::=
    SYNTAX ::= {
      INTERFACE MyCpu,
      METHODS
      myCpu2.SYNTAX
    }
    SEMANTICS
    myCpu2.SEMANTICS;
  END [ OBJECT myCpu2 ]

END [ REALIZATION Cpu ]
```

Annex 3
SAMPLE CONFIGURATION ENTRIES

```
INDB MODULE indb1 ::=
BEGIN
   HOST (aramis)
   BACKUP HOST (athos)
END [ INDB MODULE indb1 ]

INDB MODULE indb2 ::=
BEGIN
   HOST (portos)
   BACKUP HOST (milady)
END [ INDB MODULE indb2 ]

INDB MODULE indb3 ::=
BEGIN
   HOST (dartagnan)
   BACKUP HOST (richelieu)
END [ INDB MODULE indb2 ]

PROXY MODULE proxy1 ::=
BEGIN
   TEMPLATE (template1)
   HOST (treville)
   BACKUP HOST (milady)
END [ PROXY MODULE proxy1 ]

PROXY MODULE proxy2 ::=
BEGIN
   TEMPLATE (template1)
   HOST (richelieu)
   BACKUP HOST (milady)
END [ PROXY MODULE proxy2 ]

PROXY MODULE proxy3 ::=
BEGIN
   TEMPLATE (template2)
   HOST (richelieu)
   BACKUP HOST (athos)
END [ PROXY MODULE proxy3 ]

PROXY MODULE proxy4 ::=
BEGIN
   TEMPLATE (template3)
   HOST (dartagnan)
   BACKUP HOST (treville)
END [ PROXY MODULE proxy4 ]

INSTANCE xyz07B ::=
BEGIN
   CLASS (Circuit)
   INDB (indb1)
   PROXY (proxy1)
   OBJECT (X25IBM)
END [INSTANCE xyz07B]

INSTANCE xyz08B ::=
BEGIN
   CLASS (Circuit)
   INDB (indb1)
   PROXY (proxy1)
   OBJECT (UnixX25)
END [INSTANCE xyz08B]

INSTANCE xyz09B ::=
BEGIN
   CLASS (Circuit)
   INDB (indb1)
   PROXY (proxy1)
   OBJECT (UnixEth)
END [INSTANCE xyz09B]

INSTANCE xyz01C ::=
BEGIN
   CLASS (Circuit)
   INDB (indb2)
   PROXY (proxy2)
   OBJECT (X25IBM)
END [INSTANCE xyz01C]
```