Appunti delle lezioni di Gestione di Sistemi e Reti

5. L'Informazione di Gestione in SNMP (Cont)

Riassumiamo velocemente quanto si era già detto sulla SMI (ritornare indietro per approfondire).

Come in tutti i sistemi di gestione, la gestione SNMP è fondata su un database che contiene informazione sugli elementi da gestire: *Management Information Base* (*MIB*). Ogni aspetto della risorsa da gestire è rappresentato da un oggetto. La collezione di questi oggetti costituisce la MIB.

Questa collezione di oggetti ha una struttura (organizzazione) e in SNMP questa è una struttura gerarchica (ad albero): gli oggetti sono "legati" fra di loro da una struttura ad albero. Ogni sistema da gestire (workstation, server, router, bridge, ...) mantiene (possiede ed espone) una MIB che riflette lo stato delle risorse che lo costituiscono.

Impostare la gestione come azione su una MIB significa che quando voglio realizzare una funzione di *monitor* la realizzo mediante la lettura di valori degli oggetti della MIB. Quando invece voglio effettuare una funzionalità di *controllo*, la realizzo mediante la modifica di valori degli oggetti della MIB.

Perché una MIB sia funzionale alla gestione deve soddisfare alcuni requisiti:

1. *Gli oggetti usati per rappresentare una certa risorsa devono essere gli stessi in ogni sistema dello stesso tipo.*
2. *Si deve usare uno schema comune per rappresentare le informazioni. Non è sufficiente la semantica per assicurare interoperabilità.*

Per soddisfare (2) è stata definita la *Struttura della Informazione di Gestione* (*Structure of Management Information*, *SMI*). Per soddisfare (1) viene specificato (usando la SMI) lo schema della MIB per ogni tipo di sistema/risorsa per cui deve essere implementato un agente di gestione. Avremo quindi tanti schemi (tante MIB) quanti sono i tipi di risorse che siamo interessati a gestire.

5.1. La Struttura della Informazione di Gestione (SMI)

La SMI è uno standard di Internet, definito nell'RFC1155. In SNMPv2 è stata fatta una ridefinizione chiamata SMIv2 (RFC1442). La SMI identifica:

- I tipi di dato che possono essere usati in una MIB
- Come le risorse descritte da una MIB possono essere rappresentate
- Come si può dare un nome alle risorse di una MIB

In altre parole, la SMI specifica come è ammesso fare le specifiche delle MIB, e tante MIB saranno definite (usando la SMI) quanti tipi di sistemi si vogliono gestire in modo standard.

I tipi di dati ammessi sono:

- Scalari (semplici)
- Tabelle bidimensionali di scalari

La semplicità è richiesta per:

- Semplificare la implementazione degli agenti e il costo della loro esecuzione
- Aumentare la interoperabilità

Per ottenere questi risultati occorre:

1. Fornire una tecnica standardizzata per definire la struttura di ogni MIB particolare
2. Fornire una tecnica standardizzata per definire i singoli oggetti che fanno parte di una MIB, incluso il tipo (sintassi) del loro valore
3. Fornire una tecnica standardizzata per codificare i valori degli oggetti, in modo che tali valori possano essere scambiati fra entità che implementano SNMP

Nell'ottenere gli obiettivi (1) e (2) bisogna evitare di fare riferimento a

- Architettura hardware usata dalle implementazioni
- Sistema operativo
- Linguaggio di programmazione
- Scelte implementative da lasciare all'implementatore

Nell'ottenere l'obiettivo (3) occorre invece essere precisi a livello di singolo bit e byte. Esprimere le specifiche in termini di codifica produrrebbe specifiche poco comprensibili. Occorre un *linguaggio* di specifica per esprimere (1) e (2), e delle *regole per codificare (trasmettere)* quanto definito usando il linguaggio: ASN.1 e BER.

Avendo visto i dettagli di ASN.1 e BER, possiamo ora considerare come si raggiungono i tre obiettivi elencati precedentemente.

5.1.1. La struttura della MIB

Tutti i managed object sono parte di una struttura ad albero (vederemo costruita come), le cui *foglie* sono i reali managed object che rappresentano aspetti delle varie risorse gestite. La struttura dell'albero definisce *raggruppamenti* degli oggetti, le *relazioni logiche* fra di essi, e i *tipi di oggetto* di cui i managed object sono istanza. I singoli managed object devono essere identificabili perché sono le uniche entità su cui si può operare: *devono avere un nome univoco*. Gli oggetti appartengono ad un tipo (sono una istanza di un object type) ed è comodo che ogni object type abbia un nome univoco, dal quale ricavare i nomi delle sue istanze.

Nelle MIB vengono definiti gli object type, mentre le istanze prendono vita solo nell'agente di gestione. Associato ad ogni tipo di oggetto vi deve essere nella MIB un identificatore univoco, e la SMI prescrive che il nome sia di tipo `OBJECT IDENTIFIER`.

Il valore di questo OID ha le funzioni di nome dell'object type (anche se il tipo non è accessibile) e funziona da "generatore" per i nomi delle istanze. Le istanze dell'object type, create dall'agente, saranno invece accessibili solo a run time, e avranno un nome derivato dal nome dell'object type. La SMI definisce le *regole* che permettono all'agent di *confezionare i nomi delle istanze* a partire dai nomi dell'object type. Quindi le varie MIB standard definiscono una grande quantità di costanti di tipo `OBJECT IDENTIFIER`, una per ogni object type definito nella MIB.

Dato che gli OID hanno una struttura gerarchica, la struttura dei nomi definisce anche la struttura dei tipi e delle loro istanze (managed object). Come si assegnano i valori degli OID?

I valori degli OID hanno una struttura gerarchica proprio per facilitare l'assegnazione di nomi univoci eseguita da gruppi di standardizzazione diversi, che operano indipendentemente e concorrentemente. Gli standardizzatori di Internet per lavorare hanno dovuto chiedere di poter

utilizzare un sotto-albero degli OID. Hanno deciso di chiedere a ISO, che per scopi simili aveva dedicato il sotto-albero `org`, al di sotto del quale ha loro riservato il sotto-albero `dod` perché Internet era una iniziativa del Department of Defense (DOD) del governo degli Stati Uniti di America, perché Internet non era una azienda mentre il DOD sì.

All'interno di questo albero, Internet si è riservata l'albero `internet`, la cui radice è identificata dal seguente OID, definito all'interno dell'RFC 1155

```
internet OBJECT IDENTIFIER ::= { iso (1) org (3)     dod (6) 1 }
```

Questa è una definizione in ASN.1 per un valore di tipo `OBJECT IDENTIFIER`, usando una delle tante notazioni definite per poter scrivere in un testo i valori degli OID. Usando questa notazione si definisce il nuovo nome simbolico e il suo valore, usando il valore di altri OID, che possono essere identificati dal loro nome simbolico ma anche con il numero intero che ad essi corrisponde.

Un'altra notazione molto usata per gli OID è la cosiddetta notazione puntata in cui i vari interi che costituiscono un OID vengono scritte in decimale, e sono separati dal carattere ".". Il valore di `internet` è quindi spesso scritto "1.3.6.1". Un'altra notazione ancora per gli OID fa uso di nomi simbolici, ad es. ancora `iso.org.dod.internet`. Questo valore di OID costituisce il prefisso di tutti gli OID definiti negli standard di Internet.

Il documento SMI definisce quattro nodi al di sotto di `internet`, e cioè

```
directory         OBJECT IDENTIFIER ::= { internet 1 }
mgmt              OBJECT IDENTIFIER ::= { internet 2 }
experimental      OBJECT IDENTIFIER ::= { internet 3 }
private           OBJECT IDENTIFIER ::= { internet 4 }
```

`directory` è il prefisso per gli OID degli standard sul Directory; `mgmt` per gli standard che riguardano la gestione SNMP; `experiment` per OID usati negli esperimenti di Internet; `private` per OID introdotti da aziende private (e vedremo che ne hanno bisogna proprio per implementare agenti SNMP con MIB standard. Gli OID che servono per le MIB sono contenuti nell'albero `mgmt`.

Qualunque agente di gestione deve implementare una MIB minima che descrive le risorse necessarie alla gestione stessa; gli OID di questa MIB minima si trovano nell'albero `mib-2`, così specificato:

```
mib-2 ::= { mgmt 1 }
```

Le MIB di ogni specifica risorsa sono definite come estensioni dell'albero di `mib-2`, quindi "appendendole" come sottoalberi ulteriori della `mib-2` minima (che contiene 11 sotto-alberi). In questo modo tutte le MIB sono "compatibili" cioè possono essere implementate all'interno dello stesso agente di gestione.

L'albero `private` contiene un solo nodo: `enterprises`.

Un sottoalbero di enterprises viene allocato ad ogni azienda che ne faccia richiesta, in modo che possa definire degli OID privati che sono necessari perché la gestione di Internet richiede che ogni costruttore identifichi con un OID distinto ogni diverso tipo e modello di prodotto che mette sul mercato, in modo che possa essere identificato sul campo.

Una volta che una azienda possiede uno spazio per gli OID può anche definire delle MIB proprietarie: ma lo scopo di Internet non è che ognuno si definisca MIB proprietarie, bensì che usi quelle standard. Se si vuole consultare l'elenco dei numeri assegnati alle aziende, si può andare su http://www.iana.org/assignments/enterprise-numbers. Se una azienda costruisce qualcosa che può essere gestita da un agent SNMP, implementato dall'azienda e che fornisce informazioni o controlli secondo le MIB standard, allora occorre farsi assegnare un OID "aziendale" al di sotto del quale allocare gli OID per i prodotti messi sul mercato. Su http://www.iana.org/cgi-bin/enterprise.pl, si può fare domanda per avere un OID.

5.1.2. La Sintassi degli Oggetti

Ogni oggetto usato da SNMP viene definito in modo formale, definendo in primo luogo il *tipo del valore* che contiene. Per definire il tipo del valore si usa ASN.1, ma in modo ristretto che ora illustriamo.

5.1.2.1. Tipi Universali

Solo i seguenti tipi della classe `UNIVERSAL` possono essere usati:

- `INTEGER (UNIVERSAL 2)`
- `OCTET STRING (UNIVERSAL 4)`
- `NULL (UNIVERSAL 5)`
- `OBJECT IDENTIFIER (UNIVERSAL 6)`
- `SEQUENCE`, `SEQUENCE OF (UNIVERSAL 16)`

Si noti che ci sono solo i costruttori `SEQUENCE` e `SEQUENCE OF`, che sono usati per definire tabelle. Il tipo `ENUMERATED` non è incluso e quindi se si vogliono definire delle enumerazioni (estremamente utili e comuni) occorre usare il tipo `INTEGER` e definire un sottotipo; la SMI prescrive però di ***non*** usare la costante 0.

5.1.2.2. Tipi Application-wide

I tag di classe `APPLICATION` servono per definire tipi che sono rilevanti all'interno di una certa applicazione. Anche SNMP fa uso di tipi application-wide, che vengono definiti all'interno del documento SMI; vediamone alcuni.

`IpAddress`: è un tipo definito in SMI come una stringa di byte di lunghezza 4:

```
IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (SIZE (4))
                                                    -- in network-byte order
```

Si noti l'uso della parola chiave `IMPLICIT`, il modo di esprimere il vincolo sulla lunghezza della stringa, e il vincolo che sia in network-byte order espresso con un commento in linguaggio naturale (ASN.1 non permette di esprimere in modo formale vincoli così sofisticati. Notate che `IpAddress` è un sotto-tipo di `OCTET STRING`, e non di `INTEGER`. Anche se siamo abituati a dire che un indirizzo IP è un intero a 32 bit, dobbiamo riconoscere che è più corretta la definizione della SMI perché l'indirizzo non viene mai trattato in modo "numerico".

Non ci sono tipi per gli indirizzi di IPv6, che saranno trattati da SMI Next Generation (SMING).

`NetworkAddress`: è un tipo per descrivere qualunque indirizzo di livello rete, anche non di Internet; quindi viene definito con una `CHOICE`; al momento però l'unica alternativa della `CHOICE` è costituita dall'indirizzo IP:

```
NetworkAddress ::= CHOICE {   internet IpAddress
                              }
```

`Counter`: un numero intero senza segno rappresentabile con 32 bit, che *può solo aumentare il suo valore nel tempo*, e *quando dovrebbe superare il massimo ritorna invece a zero*. Anche in questo caso vincoli sofisticati sul comportamento dinamico di un managed object di tipo `Counter` non sono esprimibili in modo formale in ASN.1. Nella SMIv1 viene definito così:

```
Counter ::= [APPLICATION 1]IMPLICIT INTEGER (0 .. 4294967295)
```

Tipicamente usato per contare eventi: pacchetti ricevuti, byte spediti, errori, ... Inevitabile che quando si arrivi al massimo valore "si perda il conto": deve essere il manager a tenerne conto, pollando

regolarmente il managed object per accorgersi del wrap-around.

`Gauge`: un numero intero senza segno a 32 bit che può aumentare o diminuire il suo valore nel tempo, senza mai superare il suo valore massimo. Anche in questo caso vincoli sofisticati sul comportamento dinamico di un managed object di tipo `Gauge` non sono esprimibili in modo formale in ASN.1. Nella SMIv1 viene definito così:

```
Counter ::= [APPLICATION 1]IMPLICIT INTEGER (0 .. 4294967295)
```

Tipicamente usato per indicare il valore corrente di qualche entità, ad es. il numero di pacchetti presenti in una coda.

C'è qualche discussione su come si deve comportare un gauge una volta che arrivi al suo valore massimo e tenti di superarlo. Qualcuno sostiene che sia meglio che rimanga fisso sul valore massimo, per indicare ai manager l'evento, e che un manager debba poi resettarlo per fargli di nuovo assumere il comportamento dinamico desiderato. Altri sostengono che in questo comportamento da origine a problemi nel caso di situazioni in cui i manager sono più di uno, e in ogni caso nessuno informa il manager che il comportamento dinamico del gauge sarebbe modificato a causa dell'evento avvenuto.

`TimeTicks`: è un numero intero senza segno a 32 bit che misura su un apparato, in centesimi di secondi, il tempo trascorso dal boot (uptime). Nella SMIv1 viene definito così:

```
TimeTicks ::= [APPLICATION 3] IMPLICIT INTEGER (0..4294967295)
```

Si noti che è un timer relativo e che l'unità di misura è alquanto inusuale.

`Opaque`: un qualunque tipo ASN.1. Nella SMIv1 viene definito così:

```
Opaque ::= [APPLICATION 4] IMPLICIT OCTET STRING
                                    -- arbitrary ASN.1 value, "double-wrapped"
```

Serve per indicare un qualsiasi dato di cui non si voglia "trattare" la semantica. Potrebbe anche non essere un dato ASN.1. È "double wrapped" nel senso che si aggiunge al suo (eventuale) tag il tag di `OCTET STRING`, di cui peraltro si elimina il tag con la direttiva `IMPLICIT` (se non ci fosse, ci sarebbero ben tre tag!).

Un importante tipo che *non* è definito dalla SMI è la soglia (threshold), meglio ancora se distinta in soglia in aumento e soglia in diminuzione. Una soglia è un valore che se viene superato oltrepassato (in salita o in discesa) da *una grandezza a cui la soglia si applica* genera un evento che viene inviato al manager. I progettisti di SNMP temevano che le soglie potessero dare origine a "innondazioni" di eventi in presenza di grandezze controllate che oscillano (velocemente) attorno al valore della soglia. Per questa ragione sono molto più utili le soglie ad aumentare e a diminuire, ovviamente settando la soglia ad aumentare ad un valore più alto di quello della soglia a diminuire. Il controllo della soglia dovrebbe quindi essere eseguito tramite polling dai manager. Le soglie sono però talmente utili in sistemi di grandi dimensioni e complessità che sono poi state introdotte nella *Remote Monitoring MIB* (*RMON*).

5.1.2.3. La Definizione degli Oggetti

Poiché una MIB è una collezione di oggetti, la definizione degli oggetti è centrale alla definizione di una MIB. Gli oggetti hanno un nome (un OID), un tipo e un valore: è allora opportuno introdurre il concetto di *object type*, cioè la definizione di un particolare genere di oggetti, una descrizione dotata di sintassi. Una *object instance* di un object type è una istanza di quell'object type che è stata "creata" secondo il modello definito dall'object type. Sono gli agent che creano le istanze degli object type, per implementare la MIB che descrive un apparato.

Da un lato si vuole poter definire liberamente object type di natura diversa, per catturare aspetti diversi degli apparati sottoposti a gestione. Dall'altro lato, si vuole *mettere ordine* in queste definizioni, in modo che le implementazioni degli object type siano il più economiche e robuste possibili, e il loro uso da parte dei manager sia anche esso economico e robusto. I progettisti di SNMP decisero quindi di

usare il concetto di macro dell'ASN.1 per definire nella SMI una particolare macro chiamata OBJECT-TYPE. Chi definirà una MIB dovrà usare questa macro per definire gli object type di suo interesse.

La definizione della macro presente nell'RFC 1212 è la seguente

```
OBJECT-TYPE MACRO ::=
      BEGIN
      TYPE NOTATION ::= -- must conform to RFC1155's ObjectSyntax
            "SYNTAX" type(ObjectSyntax)
            "ACCESS" Access
            "STATUS" Status
            DescrPart
            ReferPart
            IndexPart
            DefValPart
      VALUE NOTATION ::= value (VALUE ObjectName)
      Access ::= "read-only" | "read-write" | "write-only" | "not-accessible"
      Status ::= "mandatory" | "optional" | "obsolete" | "deprecated"
      DescrPart ::=     "DESCRIPTION"     value (description      DisplayString)
                        empty
      ReferPart ::=     "REFERENCE" value (reference DisplayString)
                        | empty
      IndexPart ::=     "INDEX" "{" IndexTypes "}"
                        | empty
      IndexTypes ::= IndexType | IndexTypes "," IndexType
      IndexType ::=           -- if indexobject, use the SYNTAX value of the
                              -- correspondent OBJECT-TYPE invocation
                        value (indexobject ObjectName)
                              -- otherwise use named SMI type
                              -- must conform to IndexSyntax below
                        | type (indextype)
      DefValPart ::=    "DEFVAL" "{" value (defvalue ObjectSyntax) "}"
                        | empty
      END
IndexSyntax ::= CHOICE {      number INTEGER (0..MAX),
                              string OCTET STRING,
                              object OBJECT IDENTIFIER,
                              address NetworkAddress,
                              ipAddress IpAddress
                        }
```

La macro prende alcuni parametri, fra i quali i più significativi e usati sono i seguenti

SYNTAX: è la sintassi astratta a cui appartiene il valore contenuto nelle istanze dell'object type *che sono accessibili tramite il protocollo SNMP* (questo dettaglio non è molto chiaro anche leggendo l'RFC, ma è importante per capire il seguito). Deve essere di tipo ObjectSyntax, importato da RFC 1155. La definizione è la seguente:

```
-- syntax of objects in the MIB
ObjectSyntax ::= CHOICE {     simple            SimpleSyntax,
```

```
                        application-wide  ApplicationSyntax
                        -- note that simple SEQUENCEs are not directly
                        -- mentioned here to keep things simple (i.e.,
                        -- prevent mis-use). However, application-wide
                        -- types which are IMPLICITly encoded simple
                        -- SEQUENCEs may appear in the following CHOICE
                    }
```

I valori possono appartenere ad una sintassi semplice oppure application-wide. Vediamole:

```
SimpleSyntax ::= CHOICE {     number      INTEGER,
                              string      OCTET STRING,
                              object      OBJECT IDENTIFIER,
                              empty       NULL
                    }
```

La definizione ci dice che una sintassi semplice può essere di solo quattro tipi. Notare che un oggetto può anche "non avere valore", ma in tal caso il valore esiste ed appartiene al tipo `NULL`, e quindi vale `null`. Invece una sintassi definita dall'applicazione SNMP è definita nella SMI nel seguente modo:

```
ApplicationSyntax ::= CHOICE {      address           NetworkAddress,
                                    counter           Counter,
                                    gauge             Gauge,
                                    ticks             TimeTicks,
                                    arbitrary   Opaque
                                    -- other application-wide types, as they are
                                    -- defined, will be added here
                              }
```

Solo quattro sono i tipi application-wide definiti dalla SMI, anche se prevede che altri possano essere definiti in evoluzioni dello standard, e addirittura che questi ulteriori tipi potranno essere definiti mediante SEQUENCE. Le definizioni dei quattro tipi sono quelle riportate precedentemente.

Il secondo parametro essenziale di `OBJECT-TYPE` è `ACCESS`: specifica il modo con cui si può accedere al valore dell'istanza dell'object type; può essere solo `read-only`, `read-write`, `write-only`, `not-accessible`. Vedremo a cosa servono gli oggetti definiti come `not-accessible`.

`STATUS` è il parametro mediante il quale si specifica quanto la implementazione deve supportare l'object type: può avere uno dei seguenti possibili valori:

- `mandatory` e `optional` sono di ovvio significato;
- `deprecated` significa che si deve implementarlo ma probabilmente sarà rimosso dalla prossima versione dello standard;
- `obsolete` significa che vecchie implementazioni possono ancora avere questo object type ma le nuove versioni non devono implementarlo

Il quarto parametro *molto importante* è `DESCRIPTION`, anche se formalmente è opzionale. Contiene la descrizione in linguaggio naturale della semantica dell'object type e quindi il suo significato per la gestione della risorsa. Definire un object type senza specificare questi aspetti non è molto utile!

`ReferPart` e `DefValPart` sono raramente usati.

Il parametro `INDEX` è necessario nella definizione delle tabelle (vedi sotto)

Infine, la `VALUE NOTATION` specifica il nome usato per indicare questo object type: deve essere un OID, e costituisce il prefisso dell'OID da usare per accedere alle istanze dell'object type, come vederemo nel seguito.

A prima vista si può solo definire solo oggetti di tipo semplice, ma esiste un modo (le "tabelle") per strutturare i dati.

5.1.2.4. La Definizione delle Tabelle

La SMI supporta un solo modo per strutturare i dati di gestione: un concetto molto semplice di *tabelle bidimensionali* i cui elementi siano costituiti da tipi semplici. La generica riga di una tabella ha una struttura che viene definita mediante l'uso del costruttore `SEQUENCE`. La tabella è definita mediante il costruttore `SEQUENCE OF`, che indica la ripetizione di un numero arbitrario di righe tutte con la stessa struttura. Né la tabella né la riga sono accessibili (non possono essere definite come tali: si può accedere solo ai singoli elementi delle righe, ma in un modo che risulta molto complicato a prima vista.

La posizione dell'elemento sulla riga è significativa, perché dipende dall'ordine con cui appaiono nella definizione usando il costruttore `SEQUENCE`: quindi la posizione dell'elemento può essere indicata da un intero positivo (quindi da un componente di un OID). La posizione della riga nella tabella invece non ha quasi mai un significato e quindi non si può usare "l'indice" della riga per indicare la riga. Nella maggior parte dei casi, una parte dei dati contenuti nella riga costituisce in modo *naturale*, *comprensibile all'operatore di gestione*, una sorta di *chiave di identificazione* della riga.

Usiamo come esempio illustrativo il caso della tabella delle connessioni TCP, che è certamente utile descrivere perché contiene informazioni essenziali per una entità TCP. Definiamo un object type `tcpConnTable`, e diamole un OID per identificarla: `1.3.6.1.2.1.6.13`. Per ogni connessione vogliamo che siano disponibili le seguenti informazioni (contenute nella riga della tabella):

`tcpConnState`: lo stato in cui si trova questa connessione. TCP definisce 11 stati; ne viene aggiunto un 12esimo, `deleteTCB`: quando il manager setta questo stato sulla connessione, l'entità TCP deve distruggere la connessione (abort);

`tcpConnLocalAddress`: l'indirizzo IP dell'end-point locale della connessione;

`tcpConnLocalPort`: la porta dell'end-point locale della connessione;

`tcpConnRemAddress`: l'indirizzo IP dell'end-point remoto della connessione;

`tcpConnRemPort`: la porta dell'end-point remoto della connessione.

Si noti che queste cinque informazioni sono solo una parte delle informazioni che una entità TCP mantiene per ogni connessione aperta: SNMP vuole essere semplice!

La definizione della tabella nella MIB `mib-2` è la seguente (tutte le definizioni di tabelle in SNMP seguono questo schema):

```
-- the TCP Connection table
-- The TCP connection table contains information about
-- this entity's existing TCP connections.
tcpConnTable OBJECT-TYPE
      SYNTAX SEQUENCE OF TcpConnEntry
      ACCESS not-accessible
      STATUS mandatory
      DESCRIPTION
            "A table containing TCP connection-specific information."
      ::= { tcp 13 }
```

La tabella ha una sintassi costituita dalla ripetizione (zero o più) di righe di tipo TcpConnEntry

```
TcpConnEntry ::= SEQUENCE { tcpConnState       INTEGER,
                            tcpConnLocalAddress    IpAddress,
                            tcpConnLocalPort  INTEGER (0..65535),
                            tcpConnRemAddress IpAddress,
                            tcpConnRemPort         INTEGER (0..65535)
                            }
```

Ogni riga è quindi costituita da cinque elementi scalari; notare la restrizione sugli interi che indicano il numero di porta (sono interi senza segno rappresentabili con 16 bit). A questo object type viene assegnato un OID, `tcp.13` (che si può scrivere anche come `1.3.6.1.2.1.6.13`) il cui uso vedremo in seguito.

L'`ACCESS di questo object type è not-accessible, perché non ha un`a sintassi semplice e quindi non è accessibile via SNMP: occorre definire un object type più semplice: la generica riga della tabella

```
tcpConnEntry OBJECT-TYPE
      SYNTAX TcpConnEntry
      ACCESS not-accessible
      STATUS mandatory
      DESCRIPTION
            "Information about a particular current TCP connection. An object of
            this type is transient, in that it ceases to exist when (or soon
            after)the connection makes the transition to the CLOSED state."
      INDEX {     tcpConnLocalAddress, tcpConnLocalPort,
            tcpConnRemAddress, tcpConnRemPort }
      ::= { tcpConnTable 1 }
```

La generica riga ha ovviamente come sintassi `TcpConnEntry` e quindi anche essa non è accessibile, ma almeno abbiamo "eliminato" il `SEQUENCE OF` esterno. Si noti che abbiamo assegnato un nuovo OID a questo object type, direttamente "sotto" a `tcpConnTable`, e facendo uso dell'OID definito precedentemente.

A questo punto però i singoli componenti della riga sono di tipo semplice, e se ne avranno tante istanze quante sono le righe della tabella.

Introduciamo questi (5) object type ; il primo:

```
tcpConnState OBJECT-TYPE
      SYNTAX INTEGER {    closed(1), listen(2), synSent(3), synReceived(4),
                          established(5), finWait1(6), finWait2(7),
                          closeWait(8), lastAck(9), closing(10), timeWait(11),
                          deleteTCB(12)
                        }
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
            "..."
      ::= { tcpConnEntry 1 }
```

la cui sintassi descrive gli stati della connessione, e a cui viene assegnato un OID specifico. Questo object type type potrà essere istanziato quanto necessario (tante volte quante sono le connessioni aperte nell'entità TCP). Object type di questo tipo vengono detti *colonnari*. Interessante è la descrizione di questo object type:

```
DESCRIPTION
        "The state of this TCP connection. The only value which may be set by a
        management station is deleteTCB(12). Accordingly, it is appropriate for an
        agent to return a 'badValue' response if a management station attempts to
        set this object to any other value. If a management station sets this
        object to the value deleteTCB(12), then this has the effect of deleting
        the TCB (as defined in RFC 793) of the corresponding connection on the
        managed node, resulting in immediate termination of the connection. As an
        implementation-specific option, a RST segment may be sent from the managed
        node to the other TCP endpoint (note however that RST segments are not
        sent reliably)."
```

Come vedete, prescrive in dettaglio il comportamento dell'entità TCP gestita, che deve implementare comportamenti richiesti dalla gestione.

Gli altri object type sono analoghi:

```
tcpConnLocalAddress OBJECT-TYPE
        SYNTAX IpAddress
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
                "The local IP address for this TCP connection. In the case of a
                connection in the listen state which is willing to accept
                connections for any IP interface associated with the node, the value
                0.0.0.0 is used."
        ::= { tcpConnEntry 2 }
```

Notate che la `mib-2` prescrive anche come l'entità TCP deve indicare le aperture passive di connessioni.

```
tcpConnLocalPort OBJECT-TYPE
SYNTAX INTEGER (0..65535)
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
            "The local port number for this TCP connection."
      ::= { tcpConnEntry 3 }
tcpConnRemAddress OBJECT-TYPE
      SYNTAX IpAddress
            ACCESS read-only
            STATUS mandatory
            DESCRIPTION
                  "The remote IP address for this TCP connection."
            ::= { tcpConnEntry 4 }
tcpConnRemPort OBJECT-TYPE
      SYNTAX INTEGER (0..65535)
      ACCESS read-only
      STATUS mandatory
      DESCRIPTION
            "The remote port number for this TCP connection."
      ::= { tcpConnEntry 5 }
```

Se la tabella contiene tre righe, ciascuno di questi cinque object type sarà istanziato tre volte, e si potrà finalmente avere una istanza con sintassi semplice a cui accedere con le operazioni del protocollo SNMP.

Ma per accedere ad un oggetto occorre specificare un OID che costituisce il suo nome: come è definito il nome (OID) di ciascuno degli elementi della colonna di un object type colonnare?

A questa domanda la SMI risponde usando il concetto di `INDEX`. Come abbiamo detto, le istanze di un object type hanno come nome un OID che inizia con l'OID che identifica l'object type di cui sono istanze. In questo modo la "parentela" è evidente: quando si cambia colonna, cambia il prefisso dell'OID; quando si cambia tabella, cambia il prefisso dell'OID.

Ma gli elementi della stessa colonna hanno lo stesso prefisso: come distinguiamo un elemento da un altro? La posizione della riga non ha significato e quindi non si deve usare. Ma la chiave unica che *identifica la riga*, che è comunque a tutti gli elementi della stessa riga ma diversa per elementi della stessa colonna, può essere usata per generare la parte finale dell'OID.

Quando vi sono più elementi che compongono la chiave, la parte finale dell'OID sarà fatta di tante sottoparti quanti sono gli elementi della chiave. Se un elemento è un intero non negativo, allora si può usare direttamente come componente dell'OID. Ma se ha un tipo diverso da intero?

La SMI definisce una regola di *mapping* da, o di *generazione* di sequenze di interi non negativi a partire da, valori appartenenti ai tipi che possono avere la funzione di componente di chiave. Gli indici possono solo avere tipo:

```
IndexSyntax ::= CHOICE {      number      INTEGER (0..MAX),
                              string      OCTET STRING,
                              object      OBJECT IDENTIFIER,
                              address     NetworkAddress,
                              ipAddress   IpAddress
                          }
```

`NetworkAddress` può solo essere di tipo `IpAddress`, quindi non aggiunge possibilità. `IpAddress` viene mappato sui quattro interi non negativi che corrispondono alla interpretazione dei quattro ottetti che lo costituiscono come interi rappresentati in binario.

In realtà la regola per `IpAddress` è solo l'applicazione della regole per le `OCTET STRING`, che prescrive di generare tanti interi non negativi quanti sono gli ottetti della `OCTET STRING`, ma solo nel caso in cui la `OCTET STRING` sia di lunghezza fissa (come è nel caso di IpAddress). Nel caso in cui la stringa di ottetti sia di lunghezza variabile, prima degli interi non negativi generati dal contenuto degli ottetti si genera il numero degli ottetti contenuti nel valore.

In questo modo esaminando l'OID si può estrarre il valore della chiave: se non fosse preceduta dalla sua lunghezza non saprei distinguere il valore della chiave dal valore della chiave successiva che compone l'OID completo

Anche per gli `OBJECT IDENTIFIER` occorre per prima cosa inserire la lunghezza dell'OID, e poi naturalmente tutti gli interi non negativi che costituiscono l'OID che vale come chiave vengono a far parte dell'OID della istanza.

È importante comprendere che con queste regole gli OID degli elementi di una colonna permettono di risalire ai valori della chiave della riga a cui l'elemento appartiene, senza bisogno di accedere agli elementi della tabella che valgono come chiave! Discuteremo dopo in che modo, e quando, potremo fare uso di questa "curiosa" proprietà.

È anche ovvio che se conosco gli end-point di una connessione posso generare l'OID della riga corrispondente ed accedere a `tcpConnState`. Se conosco gli end-point non mi interessa accedere agli altri 4 elementi della riga. Se invece non conosco le connessioni aperte nell'entità come posso "scandire" la tabella e ricavare tutte le connessioni e il loro stato? Vedremo che il protocollo SNMP ha una risposta a questa domanda; ma anche in questo caso mi interessa solo scandire la colona `tcpConnState` e ottenere l'OID della istanza oltre al suo valore: dall'OID ricavo gli end-point. Gli altri quattro elementi della riga non mi servono.

Quando invece un object type non è colonnare e ha una sola istanza, quale è il nome della sua istanza? La risposta è che si aggiunge 0 all'OID dell'object type.