Computing and Informatics, Vol. 26, 2007, 543–561

AN ONTOLOGY FOR NETWORK SERVICES

Pedro Alípio, José Neves, Paulo Carvalho

University of Minho
Department of Informatics
4715-057 Braga, Portugal
e-mail: {pma, jneves, pmc}@di.uminho.pt

Revised manuscript received 20 February 2007

Abstract. Most of the network service specifications are implemented using relational databases or XML schemas. However, those specifications are not flexible and expressive enough to be extended with new service classes, different corporate policies, network configurations and deployment strategies; thus, most of the QoS management operations are implemented as hard-coded software components. This paper presents a novel approach in the specification of IP network services, using F-logic knowledge representation framework, aiming to include, in the same specification, the high-level service requirements, the network model and the necessary operations for the deployment of multiple network services.

Keywords: Network service specification, multiservice ip networks, network service management

1 INTRODUCTION

Many network service management tasks such as service administration, service quality monitoring, service configuration, and resource optimization are often performed manually. This work can be time-consuming and very sensible to human errors. Moreover, it requires a growing number of highly skilled personnel, bringing huge costs to Internet Service Providers (ISPs).

Frequently, ISPs network services are expressed through Service Level Agreements (SLAs), where a technical part called Service Level Specification (SLS) is included. Several proposals of SLA and SLS specification have been presented, fostering a common ground for interoperability among domain and interdomain net-

work service configuration agents. However, none of those specification is expressive enough to include the necessary knowledge to map service requirements into network configurations.

A formal specification of network services semantics is required as the building blocks to create the reasoning mechanisms to allow their implementation and deployment. The explicit or formal characterization of atomic entities (concepts) in a domain and relations that may be established among them is called an ontology [1], i.e., an ontology defines a common vocabulary for information interchange in a knowledge domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them.

Here, a novel approach to SLSs specification and management is presented based on an ontology and using the F-logic (FL) [2]. The proposed framework includes several valuable features for both specification and implementation of a network service management engine. FL allows the development of frame based knowledge specifications including the concepts and relations necessary to reason about network services instances. Furthermore, meta-predicates may be included to check the network services instances consistency proving the system correctness.

This paper has the following structure: related work and state-of-the art concerning network service specification is presented in Section 2; the concepts and relations used to model service vocabulary and configuration mappings are explained in Section 3; the ontology specification language (F-logic) is presented in Section 4; the formal specification using F-logic is included in Section 5, and finally the conclusions and future work are summarized in Section 6.

2 RELATED WORK

The research community on network services management arena has been committed to SLS definition and management [3, 4, 5, 6]. Commonly, pure XML is the preferred network services specification language. However, XML has well known limitations, namely in creating non-hierarchical relations between elements. Lately, ontologies are being mostly used to bring semantics into the World-Wide Web (WWW). The WWW Consortium (W3C) is developing the Resource Description Framework (RDF) [7], a language for encoding knowledge on Web pages to make it understandable to electronic agents searching for information. The Defense Advanced Research Projects Agency (DARPA), in conjunction with the W3C, is developing DARPA Agent Markup Language (DAML) by extending RDF with more expressive constructs aimed at facilitating agent interaction on the Web [8]. More recently, the W3C Web Ontology Working Group is developing OWL (Web Ontology Language) [9] based on description logic, maintaining as much compatibility as possible with the existing languages, including RDF and DAML.

The Service Oriented Architecture (SOA) community is also using ontology based languages to specify semantic web services, such as DAML-S [10], OWL-S [11], Semantic Web Services Language (SWSL) [12] and Web Service Modeling Language

(WSML) [13], focusing on web services discovery, composition, choreography and orchestration.

Most of the ontology specification languages rely on XML and RDF only as underneath platform [8, 14, 9]. As a result, these ontologies may be validated, parsed or transformed with regular XML tools. Nevertheless, reasoning (queries, verification and taxonomical inference) is often performed by knowledge based systems using other formalisms. Several of these tools and formalisms, such as Flora-2 based on the FL and Transaction Logic (TR) frameworks, integrate frames, rules, inheritance, and transactions, consisting of far more powerful languages than those exclusively designed for the Semantic Web or for the Semantic Web Services. The main drawback of these languages is interoperability, i.e., exchanging information with other systems or software components. Nevertheless, efforts are being made to develop a FL XML Schema and tools to transform XML documents into FL [15] which may be used to overcome this problem. A Java package is also being developed for Flora-2 and it will allow using it as a reason engine for knowledge based desktop or web applications. Furthermore, the Web Service Modeling Language (WSML), which is based on the FL and TR, and the Web Service Modeling eXecution environment (WSMX) [16] are also in progress.

As WSML and WSMX are still in progress, the present proposal follows a FL based approach, which may be implemented and executed by the Flora-2 system, including features that allow reasoning over concepts, relations and changes of state, consisting of a flexible and robust ground for specifying and implementing autonomic and adaptive service management tasks.

3 NETWORK SERVICE SPECIFICATION ONTOLOGY

The main objective of the ontological representation of network services is to create a common vocabulary, including a service classification, and to map service attributes into network configurations. This ontology may be viewed from three perspectives: (i) the network service classification; (ii) the service level specification; (iii) the deployment of the network services. Although we focus on class-based networks such as the Differentiated Services (DiffServ), and follow most of the Diffserv configuration guidelines recommendations [17], we keep this specification abstract enough to allow the deployment of network services in other network architectures providing Quality of Service (QoS).

3.1 Service Classification

Network traffic can be classified in three major groups: (i) Network Control for routing and network control function; (ii) Operations, Administration and Management (OAM) for network configuration and management functions; and (iii) the User/Subscriber traffic group for ISP functions which may be divided into ten different categories [17], namely:

Telephony service – for applications that require very low delay variation and are of constant rate, such as VoIP (Voice over IP) and circuit emulation over IP networks;

Signaling service – for peer-to-peer and client-server signaling and control functions using protocols such as SIP, SIP-T, H.323, H.248, and MGCP;

Multimedia Conferencing service – for applications that require very low delay, and have the ability to change encoding rate (rate adaptive), such as H.323/ V2 and later video conferencing service;

Real-time Interactive service – interactive variable rate inelastic applications that require low jitter, loss and very low delay, such as interactive gaming applications that use RTP/UDP streams for game control commands, video conferencing applications that do not have the ability to change encoding rates or mark packets with different importance indications;

Multimedia Streaming service – for variable rate elastic streaming media applications where a human is waiting for output and where the application has the capability to react to packet loss by reducing its transmission rate, such as streaming video and audio, or web cast;

Broadcast Video service – for inelastic streaming media applications that may be of constant or variable rate, requiring low jitter and very low packet loss, such as broadcast TV and live events, video surveillance and security;

Low Latency Data service – for data processing applications where a human is waiting for output, such as web-based ordering, or Enterprise Resource Planning (ERP) application;

High Throughput Data service – for store and forward applications such as FTP, or billing record transfer;

Standard service class – for traffic that has not been identified as requiring differentiated treatment within the network and is normally referred as best effort;

Low-Priority Data service class – packet flows where bandwidth assurance is not required.

Although services are classified into ten groups (Figure 1), some are used by the same application category. In this model, four application categories are considered: (i) Application Control Category; (ii) Media-Oriented Category; (iii) Data Category; and (iv) Best Effort Category. Figure 1 also illustrates the relationship between an SLA and a service specification, where each SLA may in fact include several different service classes. This is the typical case of applications and services which require signaling with the network (e.g. VoIP).

3.2 Service Level Specification

As illustrated in Figure 2, an SLS should include the following sections: (i) the traffic classification section, defining the fields which identify an individual or aggregate

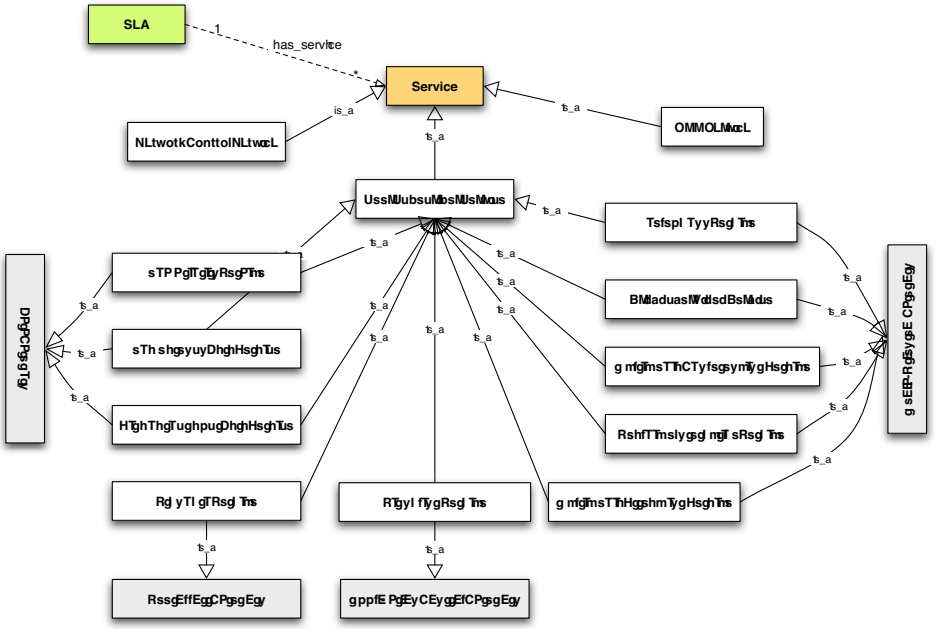

Fig. 1. User/subscriber services classification

flow; (ii) the traffic conditioning section, containing rules to identify in or out-of-profile traffic; (iii) the scope of the service, defining the boundaries of the region over which the service will be enforced and (iv) the expected QoS performance parameters. Although service scheduling and a service reliability sections could also be considered at SLS level, in our opinion, these parameters should occur at SLA level instead, as essentially they are technical service issues. As referred above, in many cases, a service may trigger or be divided, in technical terms, into several network services. Detailing each of the SLS's sections, we have:

Traffic classification is required to identify the network service traffic through a traffic classification key, which may consist of a microflow (identified by a combination of a Source Address, a Destination Address, a Source Port and a Destination Port) or a macroflow. A macroflow is an aggregate flow aiming at a specific service and may be specified in terms of: (i) a set of microflows – in this case, pairs of IP addresses (Source Address and Destination Address), transport information (Source Port, Destination Port) and Protocol Id; (ii) marking information (e.g. one or a set of DiffServ Codepoints (DSCPs)) – identifying traffic aggregates entering the domain; (iii) other IP information such as Protocol Id and IPv6 FlowLabel; (iv) as any set combining these fields. The classifier action usually consists of marking traffic as belonging to a certain service class. The deployment of this action depends on the underneath QoS architecture. While

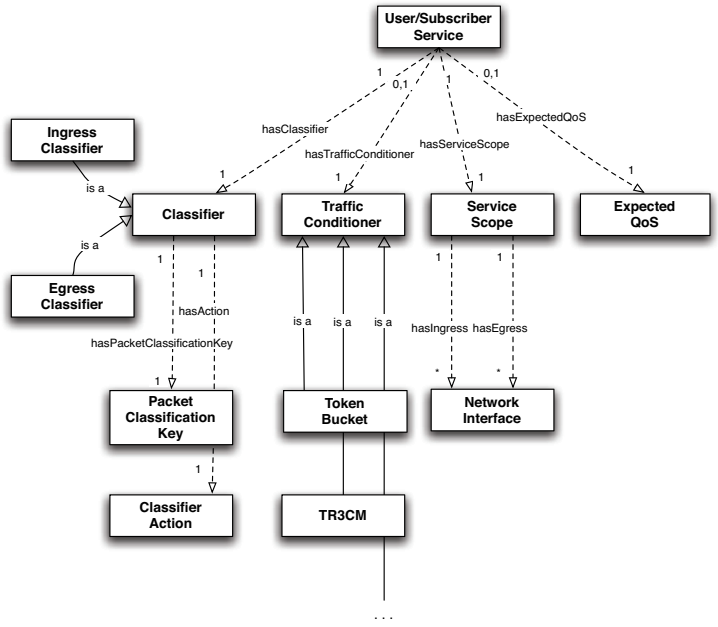

Fig. 2. Service level specification

in a Diffserv domain, it is deployed by assigning a value to DSCP field within IP packets, in a MPLS domain, it is deployed by assigning a label switching path to the packets.

Traffic conditioning. Traffic conditioning includes a conformance algorithm that identifies in-profile and out-of-profile traffic. The Diffserv configuration guidelines recommend a Single Rate Burst Size (Token Bucket) and a Two Rate Three Color Marker (TR3CM) [18] as possible conformance algorithms depending on the type of service. Nevertheless, in the present proposal we consider the possibility of extending the specification with other algorithms without affecting its consistency. Depending on the conformance algorithms, one or more actions may be applied to out-of-profile traffic. These actions may be, among others, changing the packets' drop precedence, de-promoting traffic to a lower QoS class, or dropping all out-of-profile packets.

Scope. The boundaries of topological regions must be specified as they are enforcement locations for service traffic classification and conditioning. It is expressed through a set of *ingress* and *egress* interfaces, denoting the entry and exit points of the network domain, respectively. We prefer to use the concept of ingress and egress interfaces instead of nodes because edge nodes have at least two interfaces, connecting the node to its network domain and to its neighbor domain.

Expected QoS. The expected QoS parameters express the required QoS to be provided by the network and are expressed by network performance parameters

such as: the *maximum interpacket delay*, *interpacket delay variation*, *packet loss ratio* and *throughput*. These parameters take qualitative values as recommended in the Diffserv configuration guidelines.

3.3 Network Service Deployment

The deployment of network services requires the definition of a set of network configuration rules and operations in order to establish and provide a consistent network behavior for traffic crossing an ISP domain. Although the location and configuration of traffic classifiers and conditioners may be obtained directly from the SLSs, the node forwarding behavior (e.g. Per Hop Behavior (PHB) in Diffserv) related configurations are dependent on the QoS architecture and must be specified in a network configuration model. This model includes network topology information, each node components information and mapping relations required to deploy each service class.

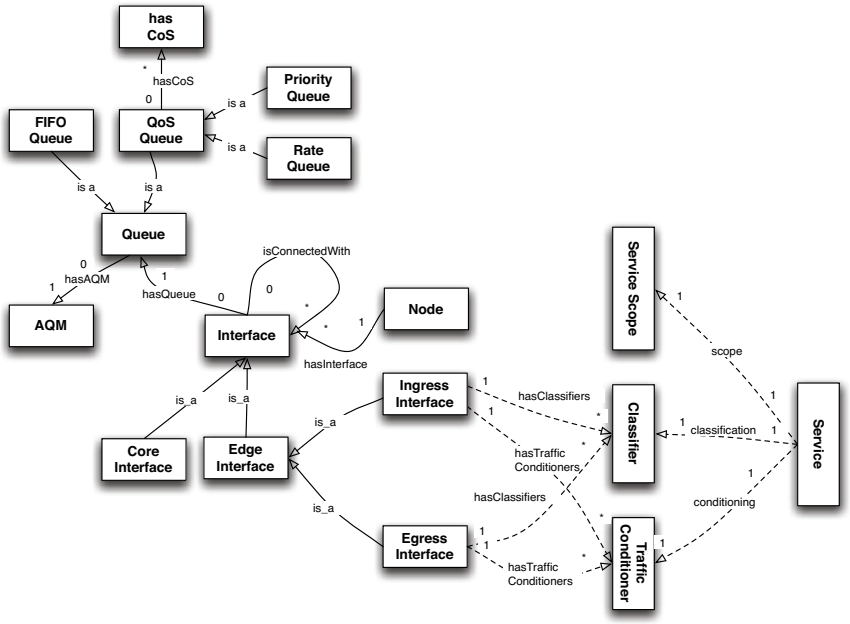

Fig. 3. Network configuration model

As shown in Figure 3, queues are associated with interfaces and interfaces with nodes. Ingress and egress interfaces are connected to classes *Classifier* and *Traffic Conditioner* while core interfaces are not. However, as *Edge Interface* is a subclass of *Interface*, it will also be connected to the class *Queue*, as a consequence of the principle of class generalization.

Associated with each node interface, there may be priority queuing and/or rate queuing disciplines. A priority queuing system is a combination of a set of queues

and a scheduler that empties them in priority sequence. Before dispatching a packet, the scheduler inspects the highest priority queue, and if there is data present returns a packet from that queue. Similarly, a rate-based queuing system is a combination of a set of queues and a scheduler that empties each of them at a specified rate, i.e., allocating, for instance, a proportional share of the interface bandwidth. In the proposed model, queues represented by the class *Queue* may be specialized into priority or a rate queue, represented by the subclasses *PriorityQueue* and *RateQueue*, respectively.

In order to prevent network congestion, queues may implement congestion control through Active Queue Management (AQM). AQM includes a variety of procedures that use specific packet dropping or marking to manage the depth of a queue.

4 F-LOGIC OVERVIEW

Although there are several ontology specification languages as previously mentioned in Section 2, F-logic [19] was chosen as the formal specification language for the network service ontology. In fact, most of those languages are not adjusted to general knowledge specification, as they focus on the Semantic-Web, being rather limited in their features.

F-logic, where F stands for frame, combines the advantages of the conceptual high-level approach typical of frame-based languages and the expressiveness, the compact syntax, and the well defined semantics of mathematical logic. The original features of F-logic include signatures, object identity, complex objects, methods, classes and inheritance. In addition, some F-logic implementations include other features, e.g., Flora-2[1] an F-logic implementation, support general rules (including recursive rules and rules with negation in the rule body), common-sense reasoning, and multiple inheritance. It may also include meta-programming in the style of HiLog, logical updates in the style of Transaction Logic, and dynamic modules.

F-logic uses first-order variable-free terms to represent Object IDentity (OID), e.g., *John* and *father*(*Mary*) are possible Ids of objects. Objects can have attributes as it is illustrated by the following example:

$$Mary[spouse \rightarrow John, children \rightarrow \{Alice, Nancy\}]$$
$$Mary[children \rightarrow Jack]$$

These formulae are called F-logic molecules. The first formula denotes that object *Mary* has an attribute spouse whose value is the OID *John*. It also denotes that the attribute *children* is set-valued and its value is a set that contains two OIDs: *Alice* and *Nancy*. Note that sets do not need to be specified all at once. For instance, the second formula above says that *Mary* has an additional child, *Jack*.

[1] `http://flora.sourceforge.net`

While some attributes of an object are specified explicitly (e.g. facts), other attributes can be defined using deductive rules. For instance $John[children \rightarrow \{Alice, Nancy, Jack\}]$ may be derived using the following deductive rule:

$$\forall_{Y,C,X}\ ?X[children \rightarrow ?C] \leftarrow ?Y[spouse \rightarrow ?X, children \rightarrow ?C]$$

F-logic objects can also have methods, which are functions that take arguments. For instance, $John[grade(net, cs) \rightarrow 100, classes(cs) \rightarrow \{prog, net\}]$ denotes that *John* has a *method*, *grade*, whose value on the arguments *net* (Networking class identifier) and *cs* (Computer Science course identifier) is 100; it also has a set-valued method courses, whose value on the argument *cs* is a set of OIDs that contains course class identifiers net and prog. Like attributes, methods can be defined using deductive rules.

The F-logic syntax for instances is *instance* :: *class* and for *class* hierarchies is *subclss* :: *class*. For example, *John* : *student*, means that *John* is an instance of class *student*, while *student* :: *person*, denotes that *student* is a specialization of a *person*. Since classes are treated as objects, it is possible for the same object to be considered as a class in one formula and an instance in another, e.g., in the formula *student* : *class*, the symbol *student* is considered an instance, while in *student* :: *person* it is considered a class.

F-logic also provides means for specifying schema information through signature formulae. For instance, $person[spouse\{0:1\} \Rightarrow person, name\{0:1\} \Rightarrow string, child\ \{0:*\} \Rightarrow person]$ denote a signature formula stating that class *person* has three attributes, the attributes *spouse* and *name*, which may have one value or none (indicated by the cardinality constraint $0:1$), and a set-valued attribute *child*, which may have any value or none (indicated by the cardinality constraint $0:*$). It further says that the first attribute returns instances of type *person*, the second of type *string*, and the last returns sets of objects such that each instance in the set is of type *person*.

5 NETWORK SERVICE SPECIFICATION

The network service specification follows the model described in Section 3, and it is organized in terms of four parts: (i) consistency checking including the predicates to verify cardinality and type checking; (ii) the ontology schema, defining the class hierarchy and method signatures; (ii) the default values for some of the class instances; and (iv) inferred relations and classes.

5.1 Consistency Checking

As Flora-2 does not include cardinality neither type checking, the following predicates were added to the specification for those purposes:

(1) $getCardinality(?O?M, ?Count) \leftarrow$
$?Count = count\{?Val|\ ?O[?M \rightarrow ?Val]\}.$

(2) $\mathtt{validCardinality}(?O, ?M) \leftarrow$
$\quad \mathtt{cardinality}(?IO?M, ?Min, ?Max)$
$\quad \mathtt{getCardinality}(?O, ?M, ?Count) \ \wedge$
$\quad ?Min =< ?Count \ \wedge$
$\quad ?Max >= ?Count.$

(3) $\mathtt{validCardinality}(?O) \leftarrow$
$\quad \mathtt{cardinality}(?O, ?M, ?Min)$
$\quad \mathtt{getCardinality}(?O, ?M, ?Count) \ \wedge$
$\quad ?Min =< ?Count.$

(4) $\mathtt{validType}(?O) \leftarrow$
$\quad ?O[?M \rightarrow ?V], \ ?O[?M \Rightarrow ?D], \ ?V : ?D.$

Predicate (1) obtains, through variable $?Count$, the cardinality of a method (or attribute) given by variable $?M$, of an object (or instance) given by variable $?O$. The cardinality is obtained by invoking the predicate *count* that counts all the elements of a set containing all the values that meet the formula $\exists_{?Val}.?O[?M \rightarrow ?Val]$, i.e., all the values bounded to the method $?M$ of the object $?O$.

Cardinality validation is performed by invoking predicates (2) and (3). These predicates check two types of facts with different signatures. While predicate (2) obtains $cardinality(?O, ?M, ?Min, ?Max)$ facts with 4 arguments, an object given by ?O, a method given by ?M, the minimum and the maximum cardinality given by $?Min$ and $?Max$, respectively, predicate (3) obtains $cardinality(?O, ?M, ?Min)$ facts where only the minimum cardinality is included because it is used to specify unbounded maximum cardinality.

Predicate (4) is used for type checking. A value bounded to a method is valid if its data type (or class) is the one specified in the method schema. Thus, the predicate takes an object as argument given by $?O$. The formula $?O[?M \rightarrow ?V]$ gets the values $?V$ bounded to the methods $?M$ of an object O. Then, the formula $?O[?M \Rightarrow ?D]$ gets the data type (class) $?D$ of method $?M$ defined for object $?O$ class. At the end, it just checks, through the formula $?V : ?D$, if the value $?V$ is an instance of $?D$.

5.2 Ontology Schema

The ontology schema describes the subclass-class relationships and method signatures. To specify the service classification presented in Section 3, the following F-logic formulas are used:

$networkControlService :: service$
$oamService :: service$
$userSubscriberService :: service$
$broadcastVideoService :: userSubscriberService$
$highThroughputDataService :: userSubscriberService$
$lowLatencyDataService :: userSubscriberService$
$lowPriorityDataService :: userSubscriberService$

$multimediaConferencingService :: userSubscriberService$
$multimediaStreamingService :: userSubscriberService$
$realTimeInteractiveService :: userSubscriberService$
$signalingService :: userSubscriberService$
$standardService :: userSubscriberService$
$telephonyService :: userSubscriberService$

SLS sections are specified as attributes of class *service*. The following formula consists of the *service* class attributes signature:

$$
\begin{aligned}
&service[\\
&\quad classification\{1:1\} \Rightarrow classifier,\\
&\quad scope\{1:1\} \Rightarrow serviceScope,\\
&\quad conditioning\{0:1\} \Rightarrow policer,\\
&\quad expectedQoS\{0:1\} \Rightarrow metrics,
\end{aligned}
$$

The *service* class has a set of attributes with different cardinalities: *classification* and *scope* are mandatory. All other attributes are optional. Attribute *classification* relates services with instances of *classifier* class. The *classifier* class has the following set of attribute signatures:

$$
\begin{aligned}
&classifier[\\
&\quad dscp\{0:*\} \Rightarrow integer,\\
&\quad flowlabel\{0:*\} \Rightarrow integer,\\
&\quad saddrs\{0:*\} \Rightarrow string,\\
&\quad daddrs\{0:*\} \Rightarrow string,\\
&\quad sport\{0:*\} \Rightarrow integer,\\
&\quad dport\{0:*\} \Rightarrow integer,\\
&\quad protocolid\{0:*\} \Rightarrow integer,\\
&\quad action\{1:1\} \Rightarrow policy].
\end{aligned}
$$

Attributes *dscp*, *flowlabel*, *saddr*, *daddr*, *sport*, *dport* and *protocolid* stand for DSCP, IPv6 Flow Label, source address, destination address, source port, destination port and protocol identification fields, respectively. All attributes are optional except *action*, which is used to specify which *policy* class instance will be used by the classifier. The *policy* class has the following schema:

$$
\begin{aligned}
&marker :: policy\\
&shaper :: policy\\
&dropper :: policy\\
&shaper[buffersize\{1:1\} \Rightarrow integer, rate\{1:1\} \Rightarrow integer]\\
&marker[class\{1:1\} \Rightarrow cos]\\
&cos[dscp\{1:1\} \Rightarrow integer]
\end{aligned}
$$

The classes *marker*, *shaper* and *dropper* are subclasses of class *policy*. Each has its own attributes as different policies have different parameters. Class *shaper* has two mandatory attributes: *buffersize* and *rate*, while marker has the attribute *class* which takes an instance of class *cos* (Class of Service) as a value. Class *cos* only contains the mandatory attribute *dscp*.

The *serviceScope* class has only the mandatory attributes *ingressNodes* and *egressNodes*. Both take one or a set of *edgeNode* instances as a value. The following formulae are involved in a service scope specification:

$$serviceScope[ingress\{1:*\} \Rightarrow edgeNode,\ egress\{1:*\} \Rightarrow edgeNode]$$

$$edgeNode : node$$
$$coreNode :: node$$
$$coreNode[core\_if\{1:*\} \Rightarrow link]$$
$$edgeNode[external\_if\{1:*\} \Rightarrow link,\ core\_if\{1:*\} \Rightarrow link]$$

The main difference between an *edgeNode* and a *coreNode* class is that core nodes only have interfaces with core links, while edge nodes also have interfaces with links located outside of the ISP domain. Links are specified by the class *link*, which is defined in terms of the following formulae:

$$\begin{aligned}&link[\\&\quad queues\{1:*\} \Rightarrow queue,\\&\quad classifiers\{1:*\} \Rightarrow classifier,\\&\quad policers\{1:*\} \Rightarrow policer,\\&\quad bandwidth\{1:1\} \Rightarrow integer,\\&\quad delay\{1:1\} \Rightarrow float]\end{aligned}$$

The *link* class has three multiple values attributes: *queues*, *classifiers* and *policers*, which relate *link* instances with instances of *queue*, *classifier* and *policer* classes, respectively. Attributes *bandwidth* and *delay* specify link maximum bandwidth and the link physical delay, respectively. All of them are mandatory.

The *queue* class has two subclasses: *priorityq* and *rateq*. The are given in terms of the following statements:

$$\begin{aligned}&rateq :: queue.\\&priorityq : queue.\\&rateq[weight\{1:1\} \Rightarrow integer].\\&queue[\\&\quad maxqueuesize\{1:1\} \Rightarrow integer,\\&\quad aqm\{0:1\} \Rightarrow aqmq,\\&\quad class\{1:1\} \Rightarrow cos].\\&aqmq[\\&\quad infl\{1:1\} \Rightarrow integer,\\&\quad supl\{1:1\} \Rightarrow integer,\\&\quad prob\{1:1\} \Rightarrow integer].\end{aligned}$$

Instances of class *priorityq* inherit the mandatory attributes *maxqueuesize* and *class*, while instances of class *rateq* must additionally specify a value for the attribute *weight*. Queues may have AQM and as result the optional attribute *aqm* had to be specified, linking *queue* and *aqmq* instances. Class *aqmq*, besides those inherited, has three mandatory attributes, namely attribute *infl*, which stands for the inferior limit over which packets have a probability of being removed from the the queue,

given by attribute *prob*. Attribute *supl* stands for the superior limit, which means that all packets over this value will be removed from the queue.

Two types of policers are considered in one model: single-rate/burst-size and a two rate three color marker, specified by classes *srbs* and *tr3cm*, respectively. These classes are subclasses of class *policer*. The following formulae illustrate the policer specification:

$$
\begin{aligned}
&srbs :: policer\\
&tr3cm :: policer\\
&policer[action\{1:1\} \Rightarrow policy]\\
&srbs[rate\{1:1\} \Rightarrow integer,\ bs\{1:1\} \Rightarrow integer]\\
&tr3cm[\\
&\quad greenclass\{1:1\} \Rightarrow cos,\\
&\quad yellowclass\{1:1\} \Rightarrow cos,\\
&\quad redclass\{1:1\} \Rightarrow cos,\\
&\quad rate1 * \{1:1\} \Rightarrow integer,\\
&\quad rate2 * \{1:1\} \Rightarrow integer,\\
&\quad bs * \{1:1\} \Rightarrow integer]
\end{aligned}
$$

Class *policer* has a mandatory attribute *action* which is inherited by both subclasses. This attribute links instances of *policer* and *policy* classes. On the one hand, class *srbs* has two mandatory attributes *rate* and *bs*, standing for the rate and burst size, respectively. In the other hand, class *tr3cm* has the attributes *greenclass*, *yellowclass*, *redclass*, which are used to map a CoS to the packet color; two rates given by *rate1* and *rate2*, standing for the rates for green and yellow packets, respectively; and *bs* which stands for burst size.

Expected QoS is specified by four attributes of class *metrics*. The following formulae specifies the expected service QoS.

$$
\begin{aligned}
&metrics[\\
&\quad thoughput\{0:1\} \Rightarrow integer,\\
&\quad loss\{0:1\} \Rightarrow qualitativeValues,\\
&\quad delay\{0:1\} \Rightarrow qualitativeValues,\\
&\quad jitter\{0:1\} \Rightarrow qualitativeValues]
\end{aligned}
$$

Attributes *throughput*, *loss*, *delay* and *jitter* are optional and stand for the minimum required bandwidth, packet loss ratio, inter-packet delay, and inter-packet delay variation, respectively. All except *throughput* take instances of *qualitativeValues* class.

Service scheduling is specified through the class *serviceScheduling* as it is illustrated by the following formulae:

$$
serviceScheduling[start\{0:1\} \Rightarrow range,\ end\{0:1\} \Rightarrow range]
$$

$$
\begin{aligned}
&range[\\
&\quad time\{1:*\} \Rightarrow timeOfTheDay],\\
&\quad weekday\{1:*\} \Rightarrow daysOfTheWeek],
\end{aligned}
$$

$month\{1:*\} \Rightarrow integer]$,
$year\{1:*\} \Rightarrow integer]$

serviceScheduling class includes two optional attributes: *start* and *end*, which take instances of class *range* as values. Class *ranges* includes the optional attributes *time*, which take one or several time values; *weekday*, which may take several days of the week values; *month* which take a month of the year as a value and *year* which takes an integer standing for the year.

Finally, reliability is given by the formula: $serviceReliability[mdt\{0:1\} \Rightarrow timeValue, mttr\{0:1\} \Rightarrow timeValue]$, where the optional attributes *mdt* and *mttr* stand for maximum down time and maximum time to repair, respectively.

5.3 Default Values

Default values are considered to map service specifications into network configurations, following the diffserv configuration guidelines[17], which includes information about traffic classification, traffic conditioning and queue configuration. Service classification and conditioning follows the following template:

S [
  $conditioning \rightarrow \ \_\# : P[PA_1 \rightarrow \ PV_1, \ PA_2 \rightarrow \ PV_2, \ \ldots, \ PA_n \rightarrow \ PV_n]$,
  $classification \rightarrow \ \_\# : classifier[$
    $action- > \_\# : Py[PyA_1 \rightarrow \ PyV_1, \ PyA_2 \rightarrow \ PyV_2, \ \ldots, \ PyA_n \rightarrow \ PyV_n]$
  ],
  $expectedQoS \rightarrow \ \_\# : metrics[loss \rightarrow \ L, \ delay \rightarrow \ D, \ jitter \rightarrow \ J]$
].

where the symbol _# stands for and anonymous object Id. In this template S, P, Py must be replaced by subclasses *service*, *policer* and *policy*. PA_1,PA_2, PA_n and PV_1, PV_2, PV_n must be replaced by attributes and values (instances of class *policer*), respectively. PyA_1, PyA_2, PyA_n and PyV_1, PyV_2, PyV_n must be replaced by attributes and values (i.e., instances of *policy* class), respectively. L, D and J should be replaced by loss, delay, and jitter qualitative values.

Queue configuration may now be given in terms of the following instances:

$qdf : rateq[aqm \ \rightarrow \ \_\#, class \rightarrow \ df]$.
$qcs0 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ cs0]$.
$qcs1 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ cs1]$.
$qcs2 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ cs2]$.
$qcs3 : rateq[class \rightarrow \ cs3]$.
$qcs4 : rateq[class \rightarrow \ cs4]$.
$qcs5 : rateq[class \rightarrow \ cs5]$.
$qcs6 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ cs6]$.
$qcs7 : rateq[class \rightarrow \ cs7]$.
$qaf11 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af11]$.
$qaf12 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af12]$.

$qaf13 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af13].$
$qaf21 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af21].$
$qaf22 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af22].$
$qaf23 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af23].$
$qaf31 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af31].$
$qaf32 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af32].$
$qaf33 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af33].$
$qaf41 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af41].$
$qaf42 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af42].$
$qaf43 : rateq[aqm \rightarrow \ \_\#, class \rightarrow \ af43].$
$qef : priorityq[class \rightarrow \ ef].$

These class *queue* instances are a common configuration for Diffserv domains, where each queue is mapped into a different CoS. All queues are rate queues except qef, which is a priority queue. All the AF class queues use AQM, while the others do not.

5.4 Rules and Inferred Knowledge

Some of the model classes do not need to be specified explicitly as they may be inferred. Application services categories classes include several service classes. Therefore, this knowledge may be materialized in terms of the following rules:

(1) $\forall_O \quad O : applicationControlCategory \ \leftarrow \ O : signalingService$

(2) $\forall_O \quad O : mediaOrientedCategory \leftarrow$
$O : telephonyService \ \vee$
$O : realTimeInteractiveService \ \vee$
$O : multimediaConferencingService \ \vee$
$O : multimediaStreamingService \ \vee$
$O : broadcastVideoService.$

(3) $\forall_O \quad O : dataCategory \leftarrow$
$O : lowLatencyDataService \ \vee$
$O : highThroughputDataService \ \vee$
$O : lowPriorityDataService$

(4) $\forall_O \quad O : bestEffortCategory \leftarrow \ O : standardService$

where $\forall$ stands for "for all". Rule (1) states that object O is an instance of the *applicationControlCategory* class, whenever it is a instance of class *signalingService*. Rule (2) states that object O is an instance of *mediaOrientedCategory* class, if it is an instance of any of the classes O : *telephonyService*, O : *realTimeInteractiveService*, O : *multimediaConferencingService*, O : *multimediaStreamingService*, or O : *broadcastVideoService*.. Rule (3) denotes that object O is an instance of *dataCategory* class, if it is an instance of any of the classes O : *lowLatencyDataService*, O : *highThroughputDataService*, or O :

lowPriorityDataService. Finally, rule (4) states that object O is an instance of the $bestEffortCategory$ class, whenever it is a instance of class *standardService*.

Attributes may also be inferred by a deduction process, e.g., the model shows that there may be several classifiers and policers associated with the node links. However, this information is not inserted directly into the node link classes. It is rather specified in the service scope through the ingress node information and by the classification and conditioning attributes. The following rules are used to report the attributes classifiers and policers of a link.

$$\forall_{L,C,S,S_c,I}\ L : link[classifiers \rightarrow\ C] \leftarrow$$
$$L : link\ \wedge$$
$$S[classification \rightarrow\ \ C]\ \wedge$$
$$S[scope \rightarrow\ S_c]\ \wedge$$
$$S_c[ingress \rightarrow\ I]\ \wedge$$
$$I[core\_if \rightarrow\ L]$$
$$\forall_{L,P,S,S_c,I} L : link[policers \rightarrow\ P] \leftarrow$$
$$L : link\ \wedge$$
$$S[conditioning \rightarrow\ P]$$
$$S[scope \rightarrow\ S_c]\ \wedge$$
$$S_c[ingress \rightarrow\ I]\ \wedge$$
$$I[core\_if \rightarrow\ L]$$

where the attribute $classifiers$ returns the classifiers associated with a *link* class by: (i) checking if the instance L is in fact a *link* instance; (ii) variable C takes the values of attribute classification of every instance of *service*; (iii) S_c takes the value of *service scope* attribute; (iv) I takes the value of *scope ingress* attribute; (v) only considers the service classifiers which uses ingress nodes (I) with core interfaces($core\_if$) connected to the *link* class instance. The other rule uses the same strategy over conditioning.

5.5 Queries

A F-logic allows high level queries to explore the ontology knowledge. In a network service context, queries may be addressed to the knowledge base, in order to know which configurations must be performed at each node of the network. As a result, this information may be used, together with a network management framework, to configure real network domains. Queries may also get information about the state of services, service scheduling, and other information useful for service management. Let us consider the following example.

$$\forall_{N,L,C,P,Q}\ nodeconf(N, \{C, P, Q\}) \leftarrow$$
$$N : edgeNode[core\_if \rightarrow\ L]\ \wedge$$
$$L[classifiers \rightarrow\ C]\ \wedge$$
$$L[conditioners \rightarrow\ P]\ \wedge$$
$$L[queues \rightarrow\ Q]$$

$$\forall_{N,L,C,P,Q}\ nodeconf(N,Q) \leftarrow \\ N : coreNode[core\_if \rightarrow\ L]\ \wedge \\ L[queues \rightarrow\ Q]$$

Such a rule returns a set of values, as a result of the unification process of the variables C,P and Q with the $classifier$, $policer$, $queue$ instances. As class $node$ has the subclasses $coreNode$ and $edgeNode$, two rules where defined. The first rule gets all links to core nodes and all the classifiers, policers and queues associated with the link instances by its attributes. The second rule only returns the core nodes queues. A query may therefore be posted in terms of the statement $nodeconf(ingress1, ?C)$. As a result, it will return classifier, conditioning and queue configurations for the $node$ instance $ingress1$. However, if the query $nodeconf(?N, ?C)$ is made, it will return all the domain node configurations.

6 CONCLUSION AND FUTURE WORK

This work intends to go much further than a service specification task. Service specifications usually include several sections, which describe the service requirements with different technical perspectives: traffic classification, traffic conditioning, scope, expected QoS, scheduling and service reliability. However, they never include network configuration and related information. In those approaches, mapping services into network configurations cannot be done conceptually because XML is often used and it imposes a hierarchical structure, which is not adequate to specify those complex relations between services and network devices.

By modeling network services in terms of an ontology, those limitations are overcome. Moreover, several classes and relations may not be explicitly defined, as they may be deducted through inference rules. Through high level queries and rules it is possible to retrieve any information kept in the knowledge base.

F-logic allows class attributes values specifications, which are inherited by instances and may be viewed as default values. Most of the ontology specification languages do not consider this feature. However, it is particularly important to create default configurations for service categories.

Work is currently in progress to create a service specification beyond its technical aspects, involving the administrative and management perspectives.

Acknowledgments

A PHD grant provided by Fundação para a Ciência e Tecnologia is gratefully acknowledged (SFRH/BD/17579/2004).

REFERENCES

[1] Gruber, T. R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. Guarino, N. and Poli, R. (eds.), Formal Ontology in Conceptual Analysis and

Knowledge Representation, Deventer, The Netherlands, Kluwer Academic Publishers, 1993.

[2] Kifer, M.—Lausen, G.—Wu, J.: Logical Foundations of Object-oriented and Frame-based Languages. Journal of ACM, No. 42, 1995, pp. 741–843.

[3] Morand, P., et al.: Mescal D1.3 – Final Specification of Protocols and Algorithms for Inter-domain SLS Management and Traffic Engineering for QoS-based IP Service Delivery and their Test Requirements. Mescal Project IST-2001-37961, 2005.

[4] Diaconescu, A.—Antonio, S.—Esposito, M.—Romano, S.—Potts, M.: Cadenus D2.3 – Resource Management in SLA Networks. Cadenus Project IST-1999-11017, 2003.

[5] Goderis, D.—T'Joens, Y.—Jacquenet, C.—Memenious, G.—Pavlou, G.—Egan, R.—Griffin, D.—Georgatsos, P.—Georgiadis, L.—Heuven, P. V.: Service Level Specification Semantics, Parameters, and Negotiation Requirements. Internet-Draft, drafttequila-sls-03.txt (work in progress).

[6] Alipio, P.—Lima, S.—Carvalho, P.: XML Service Level Specification and Validation. 10th IEEE Symposium on Computers and Communications (ISCC '05), 2005, pp. 975–980.

[7] Brickley, D.—Guha, R.: Resource Description Framework (RDF) Schema Specification. http://www.w3.org/TR/rdf-schema, W3C, 1999.

[8] Hendler, J.—McGuinness, D.: Darpa Agent Markup Language. IEEE Intelligent Systems, Vol. 15, 2000.

[9] Bechhofer, S.—van Harmelen, F.—Hendler, J.—Horrocks, I.—McGuinness, D. L., Patel-Schneider—P. F.,—Stein, L. A.: OWL Web Ontology Language Reference. W3C, 2004.

[10] Ankolekar, A. et al.: Daml-S: Semantic Markup for Web Services. Proceedings of the International Semantic Web Workshop, 2001.

[11] Martin, D. et al.: OWL-S 1.1 Release. http://www.daml.org/services/owl-s/1.1/, 2004.

[12] Grosof, B. N.—Kifer, M.—Martin, D. L.: Rules in the Semantic Web Services Language (SWSL): An Overview for Standardization Directions. Rule Languages for Interoperability, 2005.

[13] Lausen, H.—de Bruijn, J.—Polleres, A.—Fensel, D.: WSML – A Language Framework for Semantic Web Services. Rule Languages for Interoperability, 2005.

[14] Connolly, D.—van Harmelen, F.—Horrocks, I.—McGuinness, D. L.—Patel-Schneider, P. F.—Stein, L. A.: DAML+OIL Reference Description. W3C, 2001.

[15] de Bruijn, J.—Kifer, M.: F-logic/XML – An XML Syntax for F-logic. WSMO Working Draft. http://www.wsmo.org/2004/d16/d16.2/v0.1/20040324, 2004.

[16] Haller, A.—Cimpian, E.—Mocan, A.—Oren, E.—Bussler, C.: WSMX – A Semantic Service-Oriented Architecture. ICWS, 2005, pp. 321–328.

[17] Babiarz, J.—Chan, K.—Baker, F.: Configuration Guidelines for Diffserv Service Classes. RFC 4594, 2006.

[18] Heinanen, J.—Guerin, R.: A Two Rate Three Color Marker. RFC 2698, 1999.

[19] Kifer, M.— Lausen, G.: F-Logic: A Higher-Order Language for Reasoning About Objects, Inheritance, and Scheme. SIGMOD '89: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 1989, pp. 134–146.

Pedro Alípio is a project engineer at Critical Software and he is also a Ph. D. student at the Department of Informatics, University of Minho, Portugal, where he was a lecturer during the past three years. Previously, he worked as lecturer at Superior Institute of Engineering of Porto and Superior Institute of Industrial Studies and Management. He holds a Master degree in computer communications, distributed systems and computer architecture from University of Minho.

José Neves is full professor of computer science at the Department of Informatics, the University of Minho, Braga, Portugal, since 1999. He received a Ph. D. in computer science from Heriott Watt University, Edinburgh, Scotland, in 1984. His current research directions span the fields of extended logic programming, knowledge representation and reasoning systems, intelligent systems, cognitive robotics and the applications of artificial intelligence to medicine and to the law. He has published more than a hundred papers in international journals, conferences, workshops and symposiums.

Paulo Carvalho is currently assistant professor of computer communications at the Department of Informatics, University of Minho, Braga, Portugal. He graduated in 1991 in informatics and systems engineering at the University Minho and received his Ph. D. degree in computer science from the University of Kent at Canterbury, United Kingdom, in 1997. His main research interests include broadband technologies, multiservice networks and protocols, network management and mobile networks.