Ontological Configuration Management for Wireless Mesh Routers

Iván Díaz*, Cristian Popi†, Olivier Festor†, Juan Touriño*, Ramón Doallo*

{idiaz,juan,doallo}@udc.es*
Computer Architecture Group
Department of Electronics and Systems,
University of A Coruña
Campus de Elviña s/n, 15071 A Coruña,
Spain

{popicris,Olivier.Festor}@loria.fr†
MADYNES - INRIA Nancy
Grand Est - Research Center
615, rue du jardin botanique
54602 Villers-les-Nancy,
France

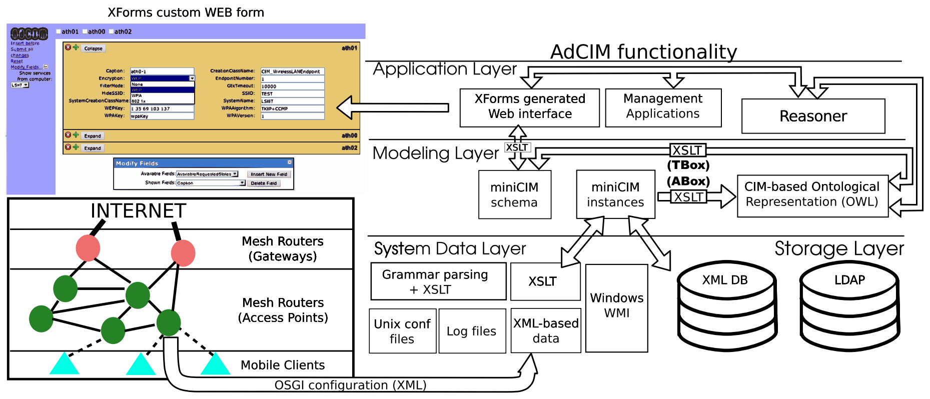
Abstract. Wireless mesh networks (WMNs) are a category of wireless networks that are self-organized, robust and which offer more flexible client coverage with less equipment requirements than wired networks. In WMNs, mesh routers constitute the network's "backbone". The distributed, ever-changing and ad-hoc nature of these networks poses new challenges in configuration management. In order to face them, we modelize the configuration and semantics of a preexisting mesh router using the CIM model and OWL ontology language and implementing XSLT transformations from the original configuration format to CIM/OWL and back. We thus represent it in a higher level of abstraction, an ontological representation that supports configuration semantic checking, policy enforcing and reasoning on the configuration of WMN nodes. We also use the capabilities of our AdCIM framework for persistence and the generation of web configuration interfaces.

1 Introduction

Wireless Mesh Networks [1] replace the classical wired network distribution with a wireless, self-organizing and self-healing infrastructure. This allows for an easy and cheap deployment of access points in places where cabling is costly. A WMN consists of access points, and client nodes. Access points (or mesh routers) form a mesh of fixed nodes, the "backbone", and have a double function: providing access to roaming clients, and relaying data for other routers and other networks (see left side of figure 1). The coverage of a wireless mesh network is extended by means of multi-hop communications. Therefore, mesh routers have additional functions to support mesh networking (i.e. routing capabilities), functions which are very important to manage for the performance and health of the network.

*This work was funded by the Ministry of Education and Science of Spain under Project TIN2007-67537-C03, and by the Galician Government (Xunta de Galicia) under Project PGIDIT06PXIB105228PR

†This work was partially supported by the French Ministry of Research project, AIRNET, under contract ANR-05-RNRT-012-01.



Caption:	ath0-1	CreationClassName:	CIM_WirelessLANEndpoint
Encryption:	WEP	EndpointNumber:	1
FilterMode:	None	GtkTimeout:	10000
HideSSID:	WEP	SSID:	TEST
SystemCreationClassName:	WPA	SystemName:	LSIIT
	802.1x	WPAAlgorithm:	TKIP+CCMP
WEPKey:	1 35 69 103 137	WPAVersion:	1
WPAKey:	wpaKey		

Available Fields: AvailableRequestedStates
Shown Fields: Caption

Fig. 1. Application of the AdCIM framework to WMN management

There is currently a lack of frameworks for the integrated configuration of WMN routers, so this work explores the application of the AdCIM framework [2] to the configuration of WMN routers. Our approach is shown in figure 1. The router configuration is mapped to the CIM object model [3] and then converted using XSLT into CIM instances (in our custom miniCIM format). Then the functionality of the AdCIM framework is exploited, including persistence in LDAP directories and the generation of web forms to manipulate these instances. The modified data is reverted to its original format and updated in the router, or transformed to an OWL [4] based semantic representation, to check internal consistency in the configuration and infer new data.

Using the CIM model and its OWL representation opens new possibilities to diagnose mesh network problems or to simulate the effect of a proposed configuration change globally. It also provides a higher level view that hides implementation details and that is generalizable to routers with other architectures which could then be managed homogeneously. We chose the configuration of a modular wireless router developed by the Image Sciences, Computer Sciences and Remote Sensing Laboratory (LSIIT) RP team [5]. This router configuration is managed using the OSGi [6] Java-based framework, which supports on-the-fly management and deployment of modules, and to support starting, stopping and uninstalling them independently.

This paper is structured as follows. First, Section 2 details background information and persistence and user interface aspects of the AdCIM framework related with this work. Section 3 contains an analysis of the router configuration and the entities it represents. After that, Section 4 elaborates about the mapping of these entities to CIM and the implementation of this mapping in CIM. The transformation from CIM to the ontological representation, and its use in semantic checking are developed in Section 5. Section 6 explores works related with this paper, and, finally, Section 7 the conclusions.

2 Background and Management Infrastructure

This section outlines some technologies used in this work and the management infrastructure provided by AdCIM.

From the W3C we use three standards: XSLT, XForms, and OWL. XSLT [7] is a template language for XML transformations. Its declarative nature allows transformations to be further optimized and parallelized by the interpreter. XForms [8] is designed as a replacement for HTML forms and uses XML data as input and output, supports dynamic form changes and off-line validation without server intervention or Javascript support. OWL (Web Ontology Language) [4] represents formally knowledge domains organized as hierarchical classifications and supports reasoning tasks on them, such as concept satisfiability and consistency. OWL supports three flavors, OWL Lite, OWL-DL, and OWL Full, which represent various compromises between expressivity and computability.

CIM (Common Information Model) [3] is a standard from the Distributed Management Task Force (DMTF) that defines an object-oriented and extensible information model to represent configuration data. It covers a vast spectrum of configuration information, ranging from the logical (such as device capabilities, operational status, software dependencies) to the physical (e.g., temperature, physical location, cabling, card placement), and relates all these entities via associations, which represent much of the semantical information in CIM.

To manage these CIM instances and associations, in this case modelling the configuration of the router, we use our AdCIM framework [2], of which figure 1 shows a rough overview. This framework supports the extraction of configuration data as CIM instances, even from unstructured sources. It validates and stores these data in LDAP directories and generate user interfaces via XForms. OWL data for the reasoning processes that will be presented in Section 5 are also generated using XSLT and divided into ABox (declarative) statements deriving from the instances, and TBox (terminological) statements deriving from the schema, as depicted in the figure.

Data persistence in AdCIM is modularized, but the preferred solution is LDAP [9], since it supports partial replication and scalability for efficient decentralized management. Also, it is more flexible than relational databases to store and retrieve miniCIM instances. LDAP storage of miniCIM data is handled with XSLT stylesheets that transform XML to and from LDAP format according to the miniCIM schema and the DMTF recommendations in [10].

Since the transformation to and from directory data must be called at every query, the stylesheet processor is configured to cache the CIM schema to ensure optimal response times and stylesheets make intensive use of the XSLT `<xsl:key/>` operator, as well as other optimization techniques.

AdCIM generates forms that retrieve the entities represented by CIM classes from the repository, modify them honoring schema restrictions, and support adding or deleting supported fields and instances. These forms are generated on-the-fly from the schema and a preexisting template, and then further processed to generate a standard HTML+Javascript form compatible with all major browsers.

```
<configurations>
  <level name="device">
    <level name="ethernet">
      <level name="interface">
        <configuration name="br31">
          <String name="ConfigurationType">interface</String>
          <Integer name="UpdateType">0</Integer>
          <String name="device.deviceType">ethernet</String>
          <String name="device.ethernet.broadcast">0.0.0.0</String>
          <Short name="device.ethernet.flags">1</Short>
          <String name="device.ethernet.ip">130.79.91.223</String>
          <Boolean name="device.ethernet.ipDesactivated">false</Boolean>
          <StringArray name="device.ethernet.ipv6addresses"/>
          <Integer name="device.ethernet.mtu">1500</Integer>
          <String name="device.ethernet.netmask">255.255.254.0</String>
          <Boolean name="device.ethernet.usingDHCP">false</Boolean>
          <String name="device.interfaceName">br31</String>
          <String name="device.virtualName">br31</String>
          <String name="service.bundleLocation"> file:ap-bundles/devmng_eth.jar </String>
          <String name="service.pid"> device.ethernet.interface.br31 </String>
        </configuration>
      </level>
    </level>
  </level>
</configurations>
```

Fig. 2. OSGi-based mesh router configuration for an IP interface

```
<CIM_OSGiConfSettingData namespace="dc=udc">
  <BundleLocation>
    file:ap-bundles/devmng_eth.jar
  </BundleLocation>
  <InstanceID>device.ethernet.interface.br31</InstanceID>
  <ConfigurationType>interface</ConfigurationType>
  <UpdateType>0</UpdateType>

<CIM_IPProtocolEndpoint namespace="dc=udc">
  <SystemCreationClassName>
    CIM_ComputerSystem
  </SystemCreationClassName>
  <SystemName>LSIIT</SystemName>
  <CreationClassName>
    CIM_IPProtocolEndpoint
  </CreationClassName>
  <Name>device.ethernet.interface.br31</Name>
  <Caption>br31</Caption>
</CIM_IPProtocolEndpoint>

<CIM_ElementSettingData namespace="dc=udc">
  <IsCurrent>Is Current</IsCurrent>
  <ManagedElement>
    <ref classname="CIM_IPProtocolEndpoint"
         namespace="dc=udc">
      <CreationClassName>
        CIM_IPProtocolEndpoint
      </CreationClassName>
      <Name>device.ethernet.interface.br31</Name>
      <SystemCreationClassName>
        CIM_ComputerSystem
      </SystemCreationClassName>
      <SystemName>LSIIT</SystemName>
    </ref>
  </ManagedElement>
  <SettingData>
    <ref classname="CIM_OSGiConfSettingData"
         namespace="dc=udc">
      <InstanceID>
        device.ethernet.interface.br31
      </InstanceID>
    </ref>
  </SettingData>
</CIM_ElementSettingData>
```

```
<CIM_IPAssignmentSettingData namespace="dc=udc">
   <InstanceID>br31</InstanceID>
   <AddressOrigin>Static</AddressOrigin>
</CIM_IPAssignmentSettingData>
<CIM_StaticIPAssignmentSettingData namespace="dc=udc">
   <InstanceID>br31-static</InstanceID>
   <IPv4Address>130.79.91.223</IPv4Address>
   <SubnetMask>255.255.254.0</SubnetMask>
   <GatewayIPv4Address>130.79.91.254</GatewayIPv4Address>
</CIM_StaticIPAssignmentSettingData>

<CIM_ElementSettingData namespace="dc=udc">
  <ManagedElement>
    <ref classname="CIM_IPProtocolEndpoint"
         namespace="dc=udc">
      <CreationClassName>
        CIM_IPProtocolEndpoint
      </CreationClassName>
      <Name>device.ethernet.interface.br31</Name>
      <SystemCreationClassName>
        CIM_ComputerSystem
      </SystemCreationClassName>
      <SystemName>LSIIT</SystemName>
    </ref>
  </ManagedElement>
  <SettingData>
    <ref classname="CIM_IPAssignmentSettingData"
         namespace="dc=udc">
      <InstanceID>br31</InstanceID>
    </ref>
  </SettingData>
</CIM_ElementSettingData>

<CIM_ConcreteComponent namespace="dc=udc">
  <GroupComponent>
    <ref classname="CIM_IPAssignmentSettingData"
         namespace="dc=udc">
      <InstanceID>br31</InstanceID>
    </ref>
  </GroupComponent>
  <PartComponent>
     <ref classname="CIM_StaticIPAssignmentSettingData"
          namespace="dc=udc">
       <InstanceID>br31-static</InstanceID>
     </ref>
  </PartComponent>
</CIM_ConcreteComponent>
```

Fig. 3. Excerpt from output of transforming fig. 2 configuration into miniCIM format

To find inconsistencies in the miniCIM data shown by these forms we use the programmatic Java interface of the Pellet reasoner.

3 OSGi Configuration Analysis

The router subject to study provides a naming schema and structure for the configuration attributes that conform to the standard OSGi Configuration Service which is the component of OSGi tasked with managing the settings of other services and their persistence. It defines *configuration objects* that contain *configuration dictionaries*, a collection of name-value pairs that represent the settings of an OSGi service; objects also have a *PID* (persistent identifier) as primary key. OSGi services can register themselves to a PID to receive a dictionary, or to a *configuration factory* to receive an arbitrary number of dictionaries registered in the factory.

The LSIIT router stores its configuration objects as XML data in the format seen in figure 2. Configuration objects have structured PIDs used for references and located in a hierarchy similar to that of Java packages. OSGi properties name, type and value are codified as an XML element attribute, name, and value, respectively. The router conceptual entities mapped by these objects are:

Services There are three services in the router configuration: SNMP, Bridging and Telnet. Each has a very different configuration: SNMP only needs to be set as started or stopped, Bridging needs a list of network interfaces; so the mapping of each service is different in each case.
IP Interfaces These entities represent various virtual interfaces on top of the wireless interfaces. Their set of properties includes IP address and net mask and DHCP configuration. While located in level `device.ethernet.interface`, they mostly represent IP configurations, with two properties (*MTU* and *Flags*) representing transport level properties. They can be related to wireless interfaces or be implemented by the bridging service.
Logical Wireless Interfaces They represent the aspects of wireless interface configuration that reside in a higher level than physical configuration. These aspects include encryption algorithms and settings, Radius server configuration, VLAN configuration, SSID (wireless network name), MAC filtering and related parameters like the link quality level, window size or link hysteresis control. Each one of these entities are generally associated with an IP interface and a physical wireless interface.
Physical Wireless Interfaces They represent the low-level settings of a wireless interface and are bound to logical interfaces. These settings include transmission channel or frequency and transmission power.
Other entities Other entities in the configuration include a generic IP routing default gateway setting and password information. There are also entities to represent virtual LAN settings.

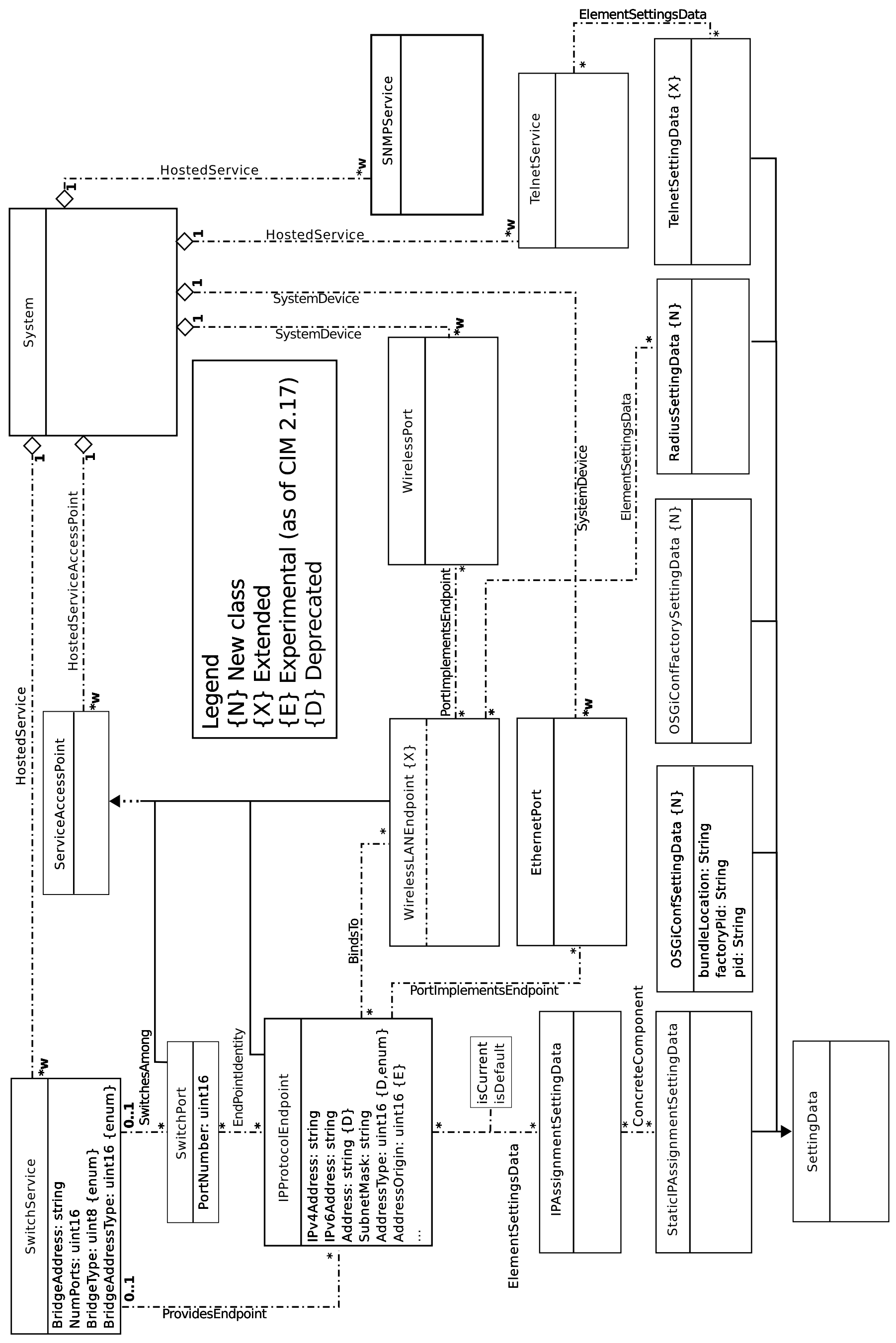


Fig. 4. CIM mapping class hierarchy

4 Mapping of the Router Configuration to CIM

This section covers the mapping from the original format of the router configuration to the CIM model that is later translated to OWL. According to the classification presented in Rivière et al [11], the mapping in this work follows the "recast" philosophy, since "concepts" are mapped. It also follows the principle of abstract translation, since redundant information is removed. Finally, the organization is independent, since the resulting model is standard. In figure 4, we show the CIM class hierarchy used to map the router configuration. We will explore both this mapping and the XSLT templates that implement it.

The CIM mapping of figure 4 can be separated in two abstraction levels, one including the `OSGIConfSettingData` and `OSGIConfFactorySettingData` classes, both representing OSGi-related structures and identifiers, and the other level representing more abstract entities. The two abstraction levels simplify the recovery of the original configuration file and, at the same time, avoid pollution of OSGi specific attributes on abstract entities.

The second abstraction level includes some specialized service classes, such as `SNMPService`. `SwitchService` represents the bridging facility and is associated to a list of `SwitchPort` instances, each associated to an `IPProtocolEndpoint`. The bridging service is also accessed as an IP interface of its own, related with `ProvidesEndpoint`. These `IPProtocolEndpoints`, which represent IP interfaces, are related to an `IPAssignmentSettingData` which indicates if the address setting is static or via DHCP. In the first case, it is further associated with a `StaticIPAssignmentSettingData` which contains the IP data. There are cases in which no IP assignment data will be given.

`IPProtocolEndpoint` instances can be associated to a `WirelessLANEndpoint` instance which represents logical wireless interfaces. Each one can have several Radius configurations, represented by the `RadiusSettingData` class. The IP interfaces maximum transfer unit value is moved to the `EthernetPort` class. Finally, physical wireless interface data are included in `WirelessPort` instances, related to `WirelessLANEndpoint` by the `PortImplementsEndpoint` association.

The properties of an OSGi dictionary are usually directly mapped to CIM properties, but there are some properties that are not mapped at all to CIM, such as boolean values that control the expression of others. For example, `device-.ethernet.ipDesactivated` shows if the IP interface has a valid IP configuration. Similarly, invalid fields can be omitted instead of being represented by dummy values.

4.1 XSLT implementation

The implementation of the transformations is done with two XSLT stylesheets: one that transforms the XML OSGi configuration data into CIM data and another one that does the opposite process. The result of the application of the first one to the configuration shown in figure 2 can be seen in figure 3. We use the miniCIM XML format (more detailed in [2]), which is a custom format that stores schema data separately, being much more compact and efficient than CIM-XML [12], the official XML mapping representing CIM data.

Internally, the first stylesheet can create any particular CIM association with the association endpoints and properties as arguments. Depending on the configuration PID of each entry, appropriate templates that create CIM classes and associations are invoked and a second pass adds relationships which would require backtracking in the first pass. The second template, which converts CIM data back to the OSGi configuration, can similarly follow CIM associations and thus rebuild the OSGi configuration retrieving the CIM instances pointed by OSGi-related associations. This template can also recover the level structure of the file by parsing their PID values. These two templates use pattern matching to allow extensibility: recognized elements trigger special case processing and unknown elements only are mapped at low level, without aborting the transformation.

5 Ontology Representation

Section 4 showed a semi-formal representation that covers taxonomical classification and domain knowledge, but this representation is not formal because many domain constraints and metadata are not expressed explicitly, so is not possible to infer and reason over the data without a priori knowledge of the semantics of the domain (see Quirolgico et al. [13]). For example, in the domain of WMNs, the channel information of a wireless interface actually maps into a range of frequencies that might be unusable because of national regulation or interference with other nearby equipment. The existence of several usable bands, and proprietary wi-fi protocols further complicates the issue. The use of an ontology and a reasoner (a program implementing logical reasoning) allows to deduce a conflict in those situations.

Configuration semantic checking is another motivator. Sinz et al. [14] and Glasner et al. [15], verify logical constraints in the Apache configuration file. These constraints are not concerned with mere well-formedness, but with semantic integrity. Checking this with ontologies has many advantages. For instance, problems in higher levels of abstraction are traced logically by the reasoner to lower-level causes and other configuration formats can be expressed with the same model, without changing the underlying logic.

Other advantage is the formal enactment of policies. A rule language like SWRL [16] allows to specify Horn-like rules of the form $H \leftarrow B_1, \ldots, B_n$, in which the head H is asserted if all the body atoms $B_{1 \ldots n}$ are true. Nevertheless, Motik et al. [17] show that the naïve combination of OWL-DL and unrestricted SWRL rules is undecidable (not guaranteed to end in the worst case), but it is decidable if rule variables are restricted to known individuals. SWRL rules extend OWL when more expressivity is needed (e.g. role composition like in a hypothetical property *uncleOf*) or there is no reasoner support (e.g. reasoning and mathematical operations with datatypes).

We chose OWL-DL as the format for our ontologies. Among the OWL flavors, OWL-DL has the best balance between expressivity and efficiency; Lite is too restrictive and Full is undecidable and inefficient. All are based on description logics, that are fragments of first order logic (FOL), in turn, propositional logic with existential and universal quantifiers. Full FOL is not used because of its undecidability and computational intractability. Description logics differ in the operations retained (or added) from FOL. OWL-DL supports these operators:

$$C \rightarrow \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid A \mid \exists R.C \mid \forall R.C \mid \geq nS \mid \leq nS \mid a_1, \ldots, a_n \quad (1)$$

where C and C_i are concepts or classes, A an atomic concept, $R.C$ describes a binary role or property, S is a property name, n is an integer number that indicates its minimum or maximum cardinality, and a_i are named individuals. Thus, the operations include negation, set union, set intersection, existential quantification, universal quantification, number restrictions and named individuals. Individuals can belong to several concepts and roles represent logic predicates involving concepts that can be transitive. Known facts or "axioms" are grouped in the *knowledge base*, divided in the TBox ("terminological box") which hierarchically groups axioms about concepts and roles and their mutual inclusion, and the ABox ("assertional box"), which contains knowledge about individuals (and their inclusion in a concept). A reasoner can perform several tasks with that information, such as classifying individuals in concepts, restructuring the concept hierarchy and detecting inconsistencies. We have used the open-source reasoner Pellet [18], which has support for both OWL and SWRL.

5.1 CIM transformation to OWL

Works like the ones by Heimbigner [19], Majewska et al. [20] and García et al. [21] have already implemented mappings of CIM to OWL-DL, and remark the lack of equivalences for some CIM constructs in OWL. Our mapping is closer to the second and third works, and is implemented with XSLT. Our mapping approach from the CIM schema to OWL TBox is roughly described as:

CIM entity	**OWL TBox mapping**
Class	`<owl:Class>`, defined as a closed set of individuals using `<owl:OneOf>`
SubClass	`<rdfs:subClassOf>`, all subclasses declared as disjoint to one another
Properties	`DataProperties` with appropriate types to map CIM types
References	cardinality 1 `ObjectProperty`. Inverse properties are inferred automatically by the reasoner
Association	subclasses of `Association`. Each instance limited to `DataProperty` and two cardinality 1 `ObjectProperty`
Cardinality	Normal in `DataProperty`. `ObjectProperty` in associated classes by limiting the cardinality of inverse object properties
Key	Not implemented, causes undecidability without being needed

We adapted some of the ideas of previous works and rejected others, for example, the approach for mapping property names in [19] is very cumbersome and we chose instead to append the class name as a prefix. Mapping CIM key properties requires OWL Full, which is undecidable, so instead a unique identifier is assigned for each instance conserving the semantics. Our approach of "splitting" associations tries to simplify their closure and preserves the semantics of cardinality. Since OWL properties are binary, we require separate instances for associations to house possible association attributes.

Closing the world Our mapping provides axioms to obtain world closure. OWL reasoners by default operate by the *Open World Assumption*, so unstated facts are not false, merely unknown. This also makes membership by negation (or *negation as failure*) very hard to verify, so it only works if there is no possible new knowledge that invalidates the negation. In this case, since the configuration file is a closed universe of discourse, we prevent this by declaring explicitly the inexistence of additional instances, restricting the cardinality of associations, and defining all individuals pairwise disjoint. This closes the world and allows negation as failure. This can be viewed as a drawback of OWL, but it also allows to open parts of the world, e.g. parts modifiable by unknown external imports.

5.2 OWL reasoning implementation

Once the information contained in the miniCIM schema and instances is translated to TBox and ABox axioms, additional axioms are introduced in an included file that verify some conditions. This section shows some examples of applicable restrictions for configuration checking. To better understand this section, refer to figure 4.

The first example shows a simple case in which unconfigured entities are detected; in this case, ports in a switching service that are not configured. This avoids the possibility of the switching service failing due to misconfiguration and causing a malfunction in the node.

$$SwitchPort\_Undefined \equiv SwitchPort \cap \tag{2a}$$
$$\neg(\exists EndpointIdentitySystemElement^{-}.EndpointIdentity) \tag{2b}$$

This restriction declares an *undefined* port in a switching service as a port not represented with a network endpoint. This is checked by the presence of an inverse property from association `EndpointIdentity`, x^{-} in 2b. The values of these inverse properties are inferred automatically by the reasoner. Since this restriction uses negation as failure, the possible instances of the `EndpointIdentity` association and `SwitchPort` must be closed for it to work.

The effects of a configuration error are made to cascade to other entities defining intermediate classes (OWL-DL does not support composition of properties). The reasoner automatically determines the proper evaluation order, and the cascading can be made arbitrarily deep. The type of errors detected by this process are important in wireless nodes, since obscure high-level errors might have simple motivations solvable on-the-fly:

$$
\begin{aligned}
&Not\_current\_IP\_setting \equiv && \text{(3a)}\\
&ElementSettingData \cap && \text{(3b)}\\
&(\exists ElementSettingDataIsCurrent = false \,|\, xsd : string) \cap && \text{(3c)}\\
&(\exists ElementSettingDataSettingData.IPAssignmentSettingData) && \text{(3d)}
\end{aligned}
$$

$$
\begin{aligned}
&Unconfigured\_IP\_Endpoint \equiv IPProtocolEndpoint \cap && \text{(3e)}\\
&((\forall ElementSettingDataManagedElement^{-}.Not\_current\_IP\_setting) && \text{(3f)}\\
&\cup (\neg(\exists ElementSettingDataManagedElement^{-}.IP\_setting))) && \text{(3g)}\\
&BindsTo\_Unconfigured\_IP\_Endpoint \equiv && \text{(3h)}\\
&BindsTo\ \cap (\exists BindsToAntecedent.Unconfigured\_IP\_Endpoint) && \text{(3i)}
\end{aligned}
$$

The term 3c selects IP setting instances that are not currently used, term 3d deselects `ElementSettingData` instances not related to IP Settings and the special intermediate class `Unconfigured_IP_Endpoint` is defined as one Endpoint with no IP settings (3e), or one in which none are current (3f). Finally, term 3h declares a subclass of the association `Binds_To` grouping those instances that bind with `Unconfigured_IP_Endpoint` instances. In that way, semantic errors are propagated so they can help diagnose problems in top-level entities.

Policies are also implemented by SWRL rule chaining. SWRL allows both straightforward composition without intermediate classes, and performing inequality comparisons with datatype ranges (instead of only supporting equality comparisons) . Some OWL reasoners translate the rules and relevant OWL axioms to another rule engine, sometimes changing the semantics, but the Pellet reasoner integrates them fully with OWL axioms. These rules, for example, detect wireless ports that have illegal frequencies depending on the legislation of the country:

$$
\begin{aligned}
&WirelessPortChannel(?x, ?y) \wedge swrlb : multiply(?y5, ?y, 5) && \text{(4a)}\\
&\wedge swrlb : add(?z, ?y5, 2412) \rightarrow WirelessPortFrequency(?x, ?z) && \text{(4b)}
\end{aligned}
$$

$$
\begin{aligned}
&located(Japan, ?y)\ \wedge\ ComputerSystem(?y) \wedge && \text{(4c)}\\
&\wedge\ SystemDeviceGroupComponent(?z, ?y) \wedge && \text{(4d)}\\
&\wedge\ SystemDevicePartComponent(?z, ?a)\ \wedge\ WirelessPort(?a) \wedge && \text{(4e)}\\
&\wedge\ WirelessPortChannel(?a, ?b)\ \wedge\ swrlb : greaterThanOrEqual(?b, 2500) && \text{(4f)}\\
&\rightarrow IllegalFrequency\_WirelessPort(?a) && \text{(4g)}
\end{aligned}
$$

$$
\begin{aligned}
&located(USA, ?y)\ \wedge\ ComputerSystem(?y) \wedge && \text{(4h)}\\
&\wedge\ SystemDeviceGroupComponent(?z, ?y) \wedge && \text{(4i)}\\
&\wedge\ SystemDevicePartComponent(?z, ?a)\ \wedge\ WirelessPort(?a) \wedge && \text{(4j)}\\
&\wedge\ WirelessPortFrequency(?a, ?b)\ \wedge\ swrlb : greaterThanOrEqual(?b, 2467) && \text{(4k)}\\
&\rightarrow IllegalFrequency\_WirelessPort(?a) && \text{(4l)}
\end{aligned}
$$

Since the legal spectrum depends on the country, it first determines the location of the system housing the wireless ports (4c, 4h), and then finding their frequency (4d-4g, 4i-4k). Since these frequencies will be usually expressed as channels, terms 4a and 4b calculate the frequency by using SWRL built-in operations. Finally, the frequency of each port is compared with the legal maximum (term 4f, 4k) and minimum (not shown). If this value is out of limits (for example, channel 13 in the USA), the ports are classified in the `IllegalFrequency_WirelessPort` subclass.

More complex policies and taxonomies are defined using SWRL to define new properties. In the following example, `WirelessPort`s are defined as interfering one another (i.e. a symmetric property) if their frequency difference is less than 25Mhz (5g). This property can be used in turn in more complex rules. For example, if throughput degradation between two ports is detected, interference can explain its origin.

$$ComputerSystem(?y) \wedge SystemDeviceGroupComponent(?z, ?y) \wedge \tag{5a}$$
$$\wedge\ SystemDevicePartComponent(?z, ?a) \wedge \tag{5b}$$
$$\wedge\ SystemDeviceGroupComponent(?z, ?y1) \wedge \tag{5c}$$
$$\wedge\ SystemDevicePartComponent(?z, ?a1) \wedge WirelessPort(?a) \wedge \tag{5d}$$
$$\wedge\ WirelessPort(?a1) \wedge WirelessPortFrequency(?a, ?b) \wedge \tag{5e}$$
$$\wedge\ WirelessPortFrequency(?a1, ?b1) \wedge swrlb : subtract(?delta, ?b, ?b1) \wedge \tag{5f}$$
$$\wedge\ swrlb : lessThanOrEqual(?delta, 24) \rightarrow interferes(?a, ?a1) \tag{5g}$$

6 Related Work

Some works have been done in the area of managing wireless mesh networks, for example, a simulator-based scheme for troubleshooting faults in WMNs by Qiu et al [22]. Zhang et al [23] present an attack–resilient security architecture for WMNs.

For the configuration and accounting of WMNs, commercial solutions are available from Nortel [24] or LocustWorld [25]. Staub et al [26] tackle the challenges of defective configurations or errors in mesh routers, and propose a distributed automated reconfiguration architecture. The approach uses cfengine [27] to distribute configuration and updates among the nodes in the WMN backbone, but it is based on the over-writing of the current configuration and does not allow for extraction and analysis of the current state of the network.

Other works use ontologies or logic models for configuration. Sinz et al. develop in [14] a CIM-based formal model similar to description logics to check for inconsistencies on the Apache configuration, as does Glasner et al. in [15] but using a custom OWL model and with more emphasis on decidability. García et al. [21] transform the CIM model into OWL and use SWRL rules, but do not give details about their implementation or undecidability. Quirolgico et al. [13] is an earlier exploratory effort using RDFS, but lacks cardinality restrictions, used in [14][15] and in our work for certain structural constraints.

7 Conclusions

We have presented support in our AdCIM framework for the configuration of wireless mesh networks as well as reasoning processes on their management data. We have implemented it analyzing the configuration of a real mesh network router, separating its format and underlying entities, and defining a CIM mapping that supports the complete expression of this configuration, taking special care to store format intricacies without losing abstraction. This mapping and its opposite was implemented with XSLT templates. AdCIM is model-driven, based on standards and supports automatically generated forms and LDAP-based persistence.

Our approach represents the current state of the WMN in an integrated manner that allows off-line analysis, useful to prevent broken configuration states before deployment. This analysis is supported with an ontological model that detects semantic errors and conflicts and supports policy enforcement. This semantical representation is exploited by discovering semantical equivalences (i.e., inferring individuals class pertenence and property values), and by navigating the associations in the underlying CIM model (e.g. using both the physical location and vendor of a product to enforce rules). We are currently working on the integration of our approach with other mesh routers, and reusing the same basic ontology and reasoning processes with other configuration formats and domains.

References

1. I.F. Akyildiz, X. Wang, W. Wang. Wireless Mesh Networks: A survey. *Computer Networks*, vol. 47, n. 4, pp. 445–487, March 2005.
2. I. Diaz, J. Touriño, J. Salceda, R. Doallo. A framework focus on configuration modeling and integration with transparent persistence. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Workshop on System Management Tools for Large-Scale Parallel Systems*, pp. 297a, Denver, Colorado, USA, April 2005. (More information in `http://adcim.des.udc.es`).
3. DMTF. *Common Information Model (CIM) Standards*, `http://www.dmtf.org/standards/cim`. [Accessed July 2009].
4. W3C. *OWL 1.0.* `http://www.w3.org/TR/2004/REC-owl-features-20040210/`, 2004. [Accessed July 2009].
5. *Image Sciences, Computer Sciences and Remote Sensing Laboratory Research Unit*, `http://lsiit.u-strasbg.fr/`. [Accessed July 2009].
6. *OSGi Alliance.* `http://www.osgi.org/Main/HomePage`. [Accessed July 2009].
7. W3C. *XSL Transformations (XSLT) Version 1.0*, `http://www.w3.org/TR/xslt`, 1999. [Accessed July 2009].
8. W3C. *XForms 1.0.* `http://www.w3.org/TR/xforms`, 2003. [Accessed July 2009].
9. P. Loshin. *Big book of LDAP RFCs*. Morgan Kaufmann, 2000.

10. E. Wood. *Guidelines for CIM-to-LDAP directory mappings.* `http://www.dmtf.org/standards/documents/DEN/DSP0100.pdf`, 2000. [Accessed July 2009].
11. A. Rivière, M. Sibilla. Management information models integration: From existing approaches to new unifying guidelines. *Journal of Networks and System Management*, vol. 6, n. 3, pp. 333-356, September 1998.
12. DMTF. *Specification for the Representation of CIM in XML.* `http://www.dmtf.org/standards/documents/WBEM/DSP201.html`, 2002. [Accessed July 2009].
13. S. Quirolgico, P. Assis, A. Westerinen, M. Baskey, E. Stokes. Toward a formal Common Information Model ontology. In *Proceedings of the 5th International Conference on Web Information Systems Engineering, WISE 2004*, vol. 3307, *Lecture Notes in Computer Science*, pp. 11–21, Brisbane, Australia, November 2004.
14. C. Sinz, A. Khosravizadeh, W. Kuchlin, V. Mihajlovski. Verifying CIM models of Apache web-server configurations. In *Proceedings of the 3rd International Conference on Quality Software, QSIC 2003*, pp. 290–297, Dallas, USA, November, 2003.
15. D. Glasner, V.C. Sreedhar. Configuration reasoning and ontology for web. In *Proceedings of the 4th IEEE International Conference on Services Computing, SCC 2007*, pp. 387–394, Salt Lake City, USA, July 2007.
16. W3C Member submission. *SWRL.* `http://www.w3.org/Submission/SWRL/`, 2004. [Accessed July 2009].
17. B. Motik, U. Sattler, R. Studer. Query answering for OWL-DL with rules. In *Proceedings of the 3rd International Semantic Web Conference, ISWC2004*, vol. 3298, *Lecture Notes in Computer Science*, pp 549–563, Hiroshima, Japan, November 2004.
18. K. Clark, B. Parsia. *Pellet: The Open Source OWL DL Reasoner.* `http://clarkparsia.com/pellet/`. [Accessed July 2009].
19. D. Heimbigner. DMTF - CIM to OWL: A case study in ontology conversion. In *Proceedings of the 16th Conference on Software Engineering and Knowledge Engineering, SEKE 2004*, pp. 470–473, Banff, Canada, June, 2004.
20. M. Majewska, B. Kryza, J Kitowsky. Translation of Common Information Model to Web Ontology Language. In *Proceedings of the 7th International Conference on Computational Science, ICCS 2007*, vol. 4487, *Lecture Notes in Computer Science*, pp. 414–417, Beijing, China, May 2007.
21. F. García, G. Martínez, J. Botía, A. Gómez-Skarmeta. On the application of the Semantic Web Rule Language in the definition of policies for system security management. In *Proceedings of 3th On The Move to Meaningful Internet Systems Conference, OTM 2005*, vol. 3762, *Lecture Notes in Computer Science*, pp. 69–78, Agia Napa, Cyprus, October 2005.
22. L. Qiu, P. Bahl, A. Rao, L. Zhou. Troubleshooting wireless mesh networks. *ACM SIGCOMM Computer Communication Review*, vol. 36, n. 5, pp. 17–28, October 2006.
23. Y. Zhang, Y. Fang. ARSA: An Attack-Resilient Security Architecture for multihop wireless mesh networks. *IEEE Journal on Selected Areas in Communications*, vol. 24, n. 10, pp. 1916–1928, October 2006.
24. *Nortel WMN solutions.* `http://www.nortel.com`. [Accessed July 2009].
25. *LocustWorld.* `http://www.locustworld.com`. [Accessed July 2009].
26. T. Staub, D. Balsiger, M. Lustenberger, T. Braun. Secure remote management and software distribution for wireless mesh networks. In *Proceedings of the 7th International Workshop on Applications and Services in Wireless Networks.* ASWN 2007, Santander, Spain, May 2007.
27. M. Burgess. Cfengine: A site configuration engine. *Computing Systems*, vol. 8, n. 3, pp. 1–29, 1995.