The NOVI Information Models

Jeroen van der Ham[a,*], József Stéger[b], Sándor Laki[c], Yiannos Kryftis[d], Vasilis Maglaris[d], Cees de Laat[a]

[a] *System and Network Engineering research group, Informatics Institute, University of Amsterdam*
[b] *Department of Physics of Complex Systems, Eötvös Loránd University*
[c] *Department of Information Systems, Eötvös Loránd University*
[d] *Network Management and Optimal Design Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens*

Abstract

The NOVI Information Model (IM) and the corresponding data models are the glue between the software components in the NOVI Service Layer. The IM enables the communication among the various components of the NOVI Architecture and supports the various functionalities it offers. The NOVI IM consists of three main ontologies: resource, monitoring and policy ontology that have evolved over time to accommodate the emerging requirements of the NOVI architecture. This article presents the NOVI IM and its ontologies, together with an overview of how the NOVI software prototypes have benefitted from using the IM.

Keywords: Future Internet, knowledge representation, semantic web

1. Introduction

The NOVI project has defined and implemented an architecture that supports federation of e-Infrastructures towards a holistic Future Internet approach. Software components within the NOVI Service Layer allow users to have a unique interface to access and use resources in different testbeds. Access to the testbeds, authorization policies, monitoring information and selection of resources are integrated among platforms and implemented in the NOVI layer.

The definition of a common Information Model (IM) has from the start of the project been the essential element to achieve the federation goals. Various requirements have guided its development:

- Support of virtualization concepts to cater for virtualized resources;
- Semantics and context –awareness to support context-aware resource selection;

*Corresponding author: vdham@uva.nl

- Vendor independency as the different virtualized infrastructures have hardware and software from different vendors;
- Support of monitoring and measurement concepts, such that monitored entities along with the measurement units are uniformly described;
- Support of management policies.

The model that we have created and present here satisfies the above requirements and consists of the following features:

- Virtualization is modeled explicitly for both computing and networking devices;
- OWL is the language chosen to define the model;
- No vendor specific assumptions are made in the model;
- The model is modular and composed of three main ontologies: a resource ontology, the monitoring ontology and the policy ontology.

This document presents the NOVI IM ontologies, with a focus on the classes and properties that have been defined. We start the article with an overview of related work in Section 2. We then provide an overview of how the information model is used in NOVI by illustrating the lifecycle of a user-request in Section 3. In Section 4 we present the complete model, with its classes and properties. Section 5 provides a global overview of the implementation in the NOVI Service Layer and how the information model was incorporated. Section 6 concludes the article by highlighting the potential use of the model after the completion of the project.

2. Related Work

Numerous efforts have been done in the past few years to develop new information models that are capable to describe network resources, tools and management policies. However, each of them focuses on specific problems and is not general enough to express all the aspects of a such complex system as NOVI, which consists of virtualized and physical resources, substrate and slice monitoring, and network policies as well.

The Common Information Model [1] is produced by the Distributed Management Task Force (DMTF) [2] standards organization. CIM is an object-oriented information model described using the Unified Modelling Language (UML). This information model captures descriptions of computer systems, operating systems, networks and other related diagnostic information. CIM is a very broad and complex model, the current UML schemata of the network model span over 40 pages, the total model is over 200 pages. The model is capable of capturing the information with a very high level of detail, yet provides almost

no abstraction layer above this, making it very hard to reason generically using this model.

The Directory Enabled Network-new generation model [3] is an object-oriented information model that describes the business, system, implementation and deployment aspects of managed entities as well as their relationships. DEN-ng was created following the policy based management concept [4] employing policies to govern the behavior of the managed environment in a domain independent way. DEN-ng also included ontologies to augment its initial modeling framework, seeing the various data models as facts and ontologies as the means to infer based on the facts, i.e. the model themselves.

The Network Description Language NDL [5] is an information model developed by the University of Amsterdam to describe (computer) networks. NDL comprises of a series of schemas that categorize information for network topologies, network technology layers, network device configurations, capabilities, and network topology aggregations. The main use-cases so far have been generation of network maps, lightweight offline path finding and more recently multi-layer path finding, and network topology information exchange. Virtualization is not explicitly defined in NDL. However, it is quite possible that this can be solved with a simple extension, but NDL is currently not able to describe information required for measurement and monitoring, nor is there support for the description of policies or current and desired state modeling.

For monitoring and network measurement data, the MOMENT[6] ontology could be a very good basis. Originally, it was developed to describe various network measurement data collected by different research groups and stored in heterogeneous databases, providing us a formal description of a machine-readable vocabulary that represent concepts and formal semantics of the network measurement domain. The existing data in various databases are generally represented in infrastructure-specific ways, with no standardization of data types, units and structures. In order to enable the successful interoperability of these originally independent systems, the MOMENT ontology can help to generate mappings between the different information models, data types and units. However, MOMENT ontology does not deal with network resources, virtualization and policies at all.

After the analysis of the existing ontologies, we concluded that none of these information models fulfill all the requirements needed for describing such a complex communication network with the support of virtualization, monitoring and policies as we have in NOVI. Additionally, while some components of the existing models were suitable, they are impossible to combine with others. We have taken the existing studies as input for our model, especially the developments in the Network Markup Language WG[7], which was still in draft stage at the start of the NOVI project.

3. Example Use Case Explaining the NOVI Architecture

The high level overview of the NOVI Data, Control and Management architecture, reported in [8], is presented in Figure 1. In this section we will go

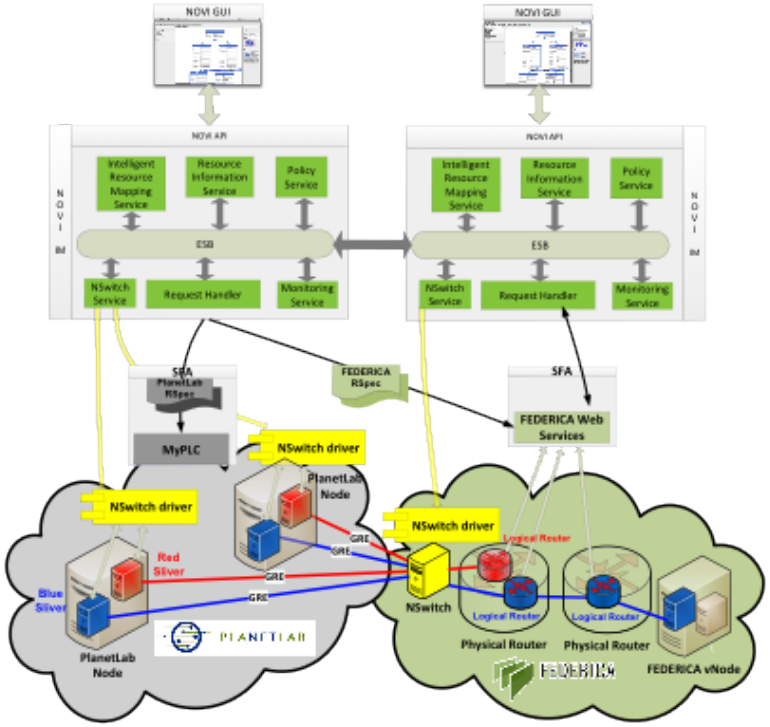

Figure 1: Overview of the NOVI architecture

through the lifecycle of a request and show how the different services of NOVI, contribute to a request, so as to explain the NOVI architecture. In this section we lay out everything in order to provide a better idea of what happens and exactly how intertwined the NOVI IM is with the working of all services. In NOVI we categorize requests into three different types:

- A request is unbound if all the virtual links and nodes are not mapped to physical resources.
- A request is bound if every virtual link or node is so mapped.
- A request is partially bound if only a subset of the requested virtual resources are mapped to physical resources or if the set of requested virtual resources are assigned to a specific platform (platform bound request).

In Figure 1 we see an overview of the NOVI architecture. At the top of the Figure is a small screenshot of the NOVI Graphical User Interface (NOVI GUI), which allows users to formulate requests and submit them to NOVI through the NOVI Application Programming Interface (NOVI API). When initiated, the GUI reads in the NOVI resource ontology to determine which concepts can be

used in formulating a request, and how these concepts can be linked together using the different relations. This is all presented in a graphical way to the user, via an intuitive drag-and-drop interface.

NOVI users are essentially users of the underlying federated platforms. The NOVI service layer acts as intermediary for passing through the authentication data between the users and the platforms, using the authentication API implemented by the corresponding virtualized platform. Via the NOVI GUI, the user can login using the credentials for his/her account at the virtualized platform he/she is registered at. The GUI can then also retrieve the list of resources that the user is authorized to use and their availability, utilizing the Resource Information Service (RIS) along with the Policy Service (PS) and the Monitoring Service (MS) of the NOVI architecture.

Once the user has finalized and submitted the request description at the GUI, the request is translated to the OWL model, and dispatched to the NOVI API. This makes sure that the request is in a format that the NOVI API can deal with and contains all the requirements that the user has imposed.

The NOVI API then forwards the request to the Intelligent Resource Mapping service (IRM) that orchestrates the creation of the slice, with resources drawn from the federated environment. In the case of an unbound or partially bound request, IRM is responsible for resource mapping, utilizing appropriate request partitioning and embedding techniques. To facilitate the mapping process, resource registration is handled by the RIS, using information from the RIS DB, the Monitoring Service (availability of substrate resources) and the Policy Service (authorization information for the requester). If the request is bound, IRM checks against RIS for availability and authorization information on the resources.

Once a suitable mapping is found by the IRM, the request is extended to contain all the bindings, and sent back to the RIS. The RIS then forwards the request to the Request Handler for slice creation at the actual platform(s). There, the request is translated from the NOVI IM to the format that the testbed supports, so far only RSpec. Resource Specification (RSpec) is an XML format, which describes testbed resources.

Should the request contain resources from different platforms, the NSwitch Service is invoked. The NSwitch service identifies the configuration details in the request for establishing data plane connectivity between virtual resources residing in different platforms. The NSwitch Service then uses these details to send configuration commands to the appropriate virtual resources, so that the inter-domain link is setup.

Once the resources are all configured by the platform, the Request Handler (RH), via the RIS and subsequently IRM, returns the outcome of the process to the requester, along with information that enables user's access to the slice.

The RIS saves the reservation and resource information to the RIS DB. Should the user require more detailed information about the current status of his/her resources, he/she can always invoke the MS via the NOVI - Slice Monitor application to request monitoring tools to run, or results from running monitoring tools. The user uses the identifiers contained in his request to unambiguously

communicate with the MS.

4. NOVI Information Model

This section describes the NOVI Information Model. The section provides a comprehensive documentation of the complete information model, divided in its three components. First we provide a general overview, In section 4.1 we describe the Resource Ontology, in section 4.2 we describe the Monitoring Ontology and its various parts, followed by a description of the Policy Ontology in section 4.3.

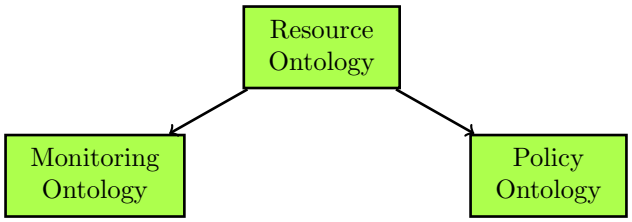

Figure 2: The NOVI ontologies

4.1. NOVI Resource Ontology

The NOVI Resource Ontology forms the basis of the NOVI Information Model. It describes the most important concepts and resources that the NOVI platform handles. This ontology is reused in the other ontologies, making sure that all services in the NOVI layer are using the same concepts. A graphical overview of the classes in the NOVI ontology is shown in figure 3.

The root element is the *Resource*, which is used as an abstract concept and has three direct descendants:

- **Node** is used to represent physical nodes, and the subclass *Virtual Node* is used to represent virtualized nodes, sometimes also called a 'sliver'.
- **Network Element** is used to describe network connectivity of resources. *Interface* is the point at which a resource connects (unidirectionally) to the network, while a *Link* is used to describe direct connectivity between *Interface*s, and a *Path* can be used as an alias for longer paths through the network. The *Virtual Link* is used to define a virtual connection through the network, either a channel on a *Link* or a tunnel through the network. The *NSwitch* is used to represent network connectivity provided by the NOVI NSwitch service.
- **Node Component** describes the capabilities of *Node*s. There is the regular elements of a computing node: *CPU*, *Memory* and *Storage*. Since we deal with networked virtualized nodes, the *Login Component* is important to describe how to log in to it. Finally, the *Switching Matrix* node

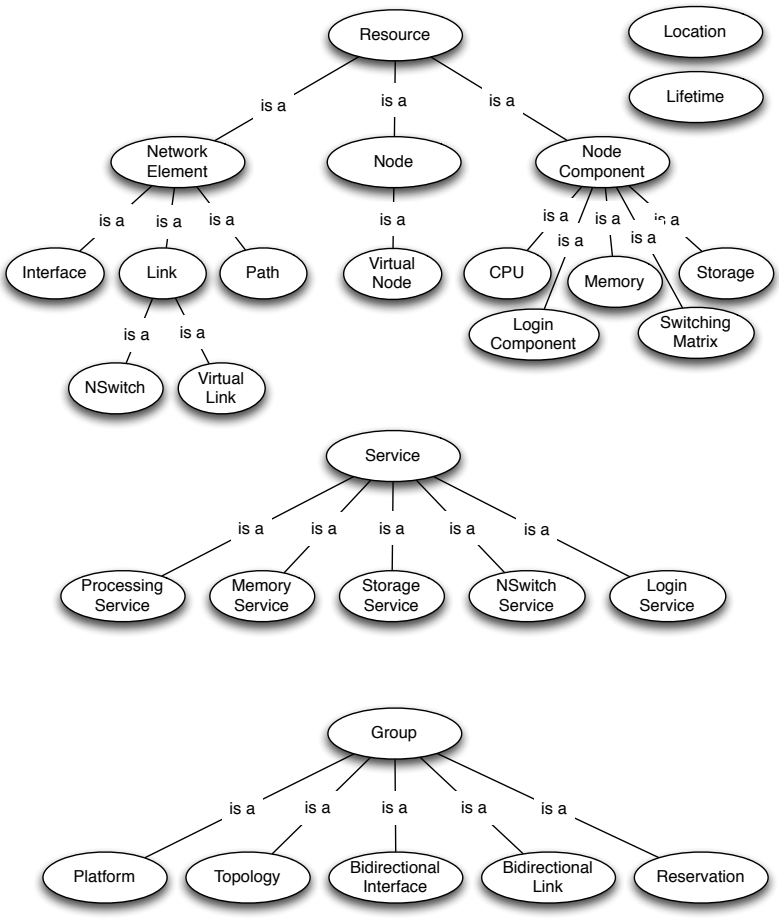

Figure 3: Graphical overview of the classes in the NOVI Resource Ontology

component is used with nodes that perform network switching or routing functionality.

It is important to note that the Network Elements in the ontology describe unidirectional elements. This has been a conscious choice to follow the NML model, which follows the philosophy that a unidirectional model can describe a bidirectional model, but not vice versa. This does make the model somewhat more verbose, but the network descriptions are not meant for human consumption, so this does not matter that much.

The *Service* class aims to allow the user to express the service level desired. This allows the user to decouple the desired service from the actual physical implementation. The types and goals of the *Service* classes mirror the *Node-Component* types. The only exceptions are the *NSwitch Service* and *Switching Matrix*. The *NSwitch Service* is used to represent the capability mentioned in the Network Element description above.

The *Group* element is used to create different kinds of groupings, some NOVI specific groups are:

- **Platform** describes a particular testbed in the NOVI federation. Resources can be linked to the class to denote membership of that platform, and it can provide pointers to other information such as the management service of that platform.
- **Topology** can define a group of resources that a user requests, or the implementation of a users' request.
- **BidirectionalInterface** defines how two (unidirectional) Interface objects are grouped together to form the physical interface.
- **BidirectionalLink** groups together two (unidirectional) Link objects are grouped together to form the physical connection.
- **Reservation** groups together the resources that have been requested or reserved for a user.

Finally we have two classes used for describing other metadata of reservations or resources:

- **Location** describes (approximate) geographical location to describe that resources share the same location. It can also be extended with properties to describe GPS coordinates.
- **Lifetime** is mainly used to describe the time dimension of a reservation (see below), but it can also be used to describe the availability of nodes, e.g. that a Node explicitly has no Lifetime during a maintenance period.

These properties and relations between the different classes in the NOVI ontology are discussed below:

- *id* and *hasName* define an identifier and a name respectively. Each Resource must have a unique identifier, which must be a Uniform Resource Identifier (URI), and should also have a human readable name.
- *startTime* and *endTime* describe the time elements of the Lifetime object, for example when a slice reservation starts and ends. The values must be specified in the XML Schema dateTime format.
- *locatedAt* is used to describe the relation between objects and their Location. This is used both for Nodes and Network Elements.
- *hasComponent* describes the relation between a Node and its NodeComponents, meaning that a Node has those components. Additional data properties are used to specify the properties of those components, such as *hasCores*, *hasMemorySize*, *hasStorageSize*, etc.
- *hasIPaddress* points to an IPAddress instance from the Unit ontology (see section 4.2), which describes the IP address (v4 or v6) and its netmask.
- *implementedBy* is to describe that a VirtualNode is implemented on a (physical) Node.
- *contains* describes that one element is part of a Group, for instance a user defines a Topology object in his reservation, and then uses the *contains* relation to describe which nodes are part of the Topology.
- *federatedWith* shows how Platforms are federated with other Platforms.

Figure 4 shows how the NOVI IM supports the description of network connections. These descriptions can be used in various contexts, ranging from an unbound reservation request (low detail) to a monitoring description (very high detail). The NOVI IM supports three levels of detail for describing network connections:

- *Node Level* A Node can be directly connected to another Node. This is an abstract way of saying that two Nodes should be connected through an underlying network connection.
- *Interface Level* An Interface can be directly connected to another Interface, either through the switchedTo or connectedTo statements. connectedTo describes a connection to an Interface on another Node, and switchedTo describes an internal connection on the same Node.
- *Link Level* Interfaces can be connected to (unidirectional) Links as a source and sink Interface to define a connection between two Interfaces.

To describe the relation between the Nodes, Interfaces and Links we have the following properties: *hasInboundInterface* and *hasOutboundInterface* to describe the relation between Interface objects and their Nodes. isSource and isSink to

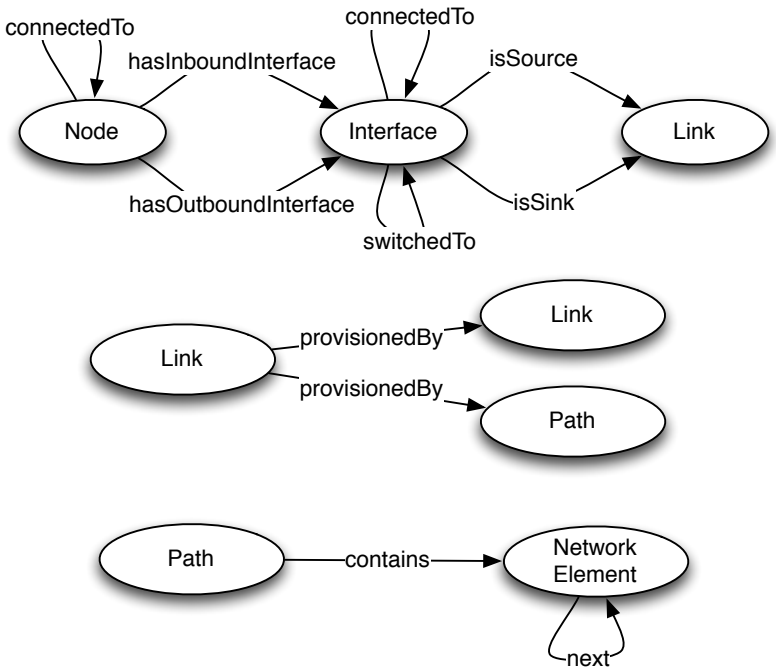

Figure 4: Network connectivity properties defined in the NOVI resource ontology

describe the relation of the unidirectional Interface objects to the Link object, which defines the connection between two Interfaces.

Finally, we have a **Path** object to allow the description of a path through the substrate network. This Path object should be an ordered collection of (at least) the Interface objects that uniquely define a path through the network. The level of detail on the Path description can be varied to allow domains to locally expand their part of the network path. Using the *contains* relation we can define that NetworkElements are part of a certain Path. The next relation is used to define the sequence within that group of elements.

4.2. NOVI Monitoring Ontology

In a federated virtual testbed environment monitoring is a fundamental task. On the one hand monitoring enables other service components intelligent decision-making, e.g. during the embedding of a virtual topology. On the other hand testbed-users can also follow the current state of the network. Due to the cross-domain nature of the system, however, it is not a trivial task to provide federated monitoring functionalities. The heterogeneity of the federated networks (including network elements and monitoring tools) poses a major challenge. NOVI tackled some of the most important related problems by elaborating a specific ontology to describe monitoring and network measurement

tasks. This semantic approach enables the flexible integration of a wide range of monitoring tools, frameworks and databases.

In Sec. 3 the life cycle of NOVI operations are described through a use case example. It is clear that many NOVI services are operating with the testbed resources, so for a full operation of these services a proper description of resources is necessary. In NOVI both the static and the dynamic information are required. By static information, we refer to characteristics, which are constant for the resource, or may change very infrequently in time. For example, the number of processor cores in a host machine is a constant static feature. Theoretically, it can change only when the hardware is upgraded. In contrast to this, one can collect characteristics, which are changing more dynamically, like the load of a CPU core in the same machine. In NOVI the description of resources are done at two abstraction levels. Besides characterizing the substrate resources we also have to handle the virtual resources. Speaking at the resource level, when the user requests a virtual testbed (a Topology), it may contain runtime, dynamical constraints. For example the creation of a topology, where virtual computing resources are allocated to hosts with a current memory utilization not exceeding a limit and virtual links are mapped to physical links of prescribed link utilization. Speaking at the virtual level instead, there is the monitoring support for the virtual testbed. Given that the user successfully gets a virtual topology, NOVI offers services to keep track of its certain temporal variables. For example, a user is interested in the evolution of the round trip delay on the given links of his topology. These two complementary abstraction levels split between *substrate monitoring* and *slice monitoring*.

Practically, substrate and slice monitoring have a lot in common and this fact was taken into consideration when designing the Monitoring Ontology. Naturally monitoring data comes from various data generators, which are tools, databases or services, and these generators speak different protocols to control them or fetch data from them. Therefore this ontology supports a uniform representation of monitoring functionalities in the federated environment extending the Resource Ontology to incorporate the relevant concepts. The Monitoring Ontology follows a modular design, see in Fig. 5. The idea behind defining smaller pieces rather than a huge ontology, comes from two facts. Firstly this model aims to be reusable by new future applications, secondly maintenance and development remains easier for each block. Thus, concepts closer in their meaning belong to the same small document component, and future applications may import only relevant pieces of the model. In the next subsections we highlight the use of each pieces of the model.

4.2.1. The Unit model

The fundamental concepts of the Monitoring Ontology are laid down in the Unit model, where the levels, dimensions, units and unit prefixes are defiined.

The *MeasurementLevel* or scales of measure comes from the statistics [9] i.e. all measurements conducted in science are claimed one of the four different types of scales that called “nominal”, “ordinal”, “interval” and “ratio”. These notions

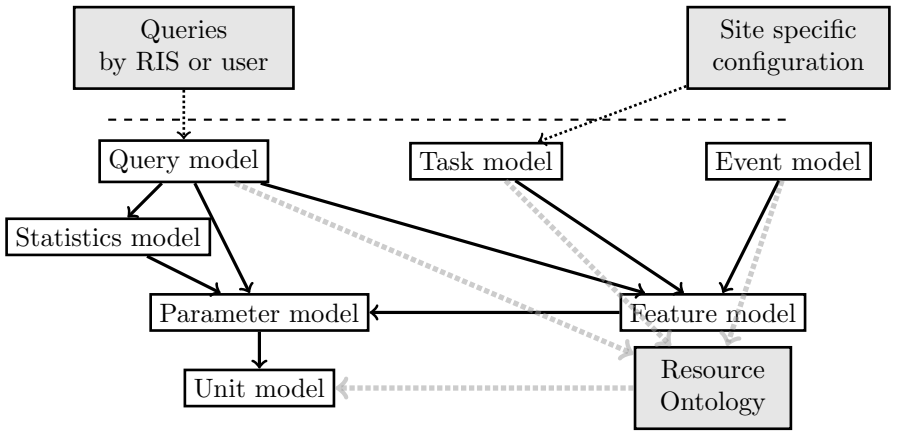

Figure 5: The monitoring model follows a modular design. The figure shows the dependency between the component modules. Items with a shaded background are components external to the monitoring model.

in the model tell what operations make sense to apply on the samples, like ordering samples, taking their difference or blowing them by a factor. Dimensions relate to one well defined level. For example if somebody is interested in the source port of IP packets captured in an interface card, this particular number is of nominal level, revealing the fact that ordering does not make sense for this type of data, whereas timestamps of packet reception, which are of interval level, both ordering and differencing makes sense.

A *Dimension* grasps the physical characteristic of a data entity. In the model we differentiate between basic and derived dimension. This module also models how derived dimensions relate to basic dimensions and/or other derived dimensions, so as to follow how various metrics root from each other. This knowledge is handy when there is no direct information of a given metric. For example duration derives from calculating the difference between two time-stamps, which are axiomatically taken fundamental.

For each dimension the corresponding natural *Unit* is indicated in the model. Using the former example, a time-stamp has a default unit *unixtimestamp*, whereas duration will fall back to second. Obviously, the derivations across units are dealt by the model similarly to that of the dimensions, so the hierarchy of dimensions and that of the units show numerous patterns in common. Model can handle transformation paths between compatible units using linear transformation, taking a product, raising to a power or applying transformations written in forms of regular expressions.

According to the model units may have various *Prefix*es. Sensible combinations of prefixes and a unit are indicated by object properties. The model handles prefixes of both base two and base ten. For instance one can infer from the model that data quantities of duration dimension can have prefixes like milli, micro, nano, etc.

4.2.2. The Feature model

The feature model is fairly flat, it declares the plethora of resource variables which can be measured and collected. The commonly used metrics form the classes, the names of the classes are self-explanatory. For example, different individuals of *MemoryFeature* specify the state of a physical or virtual memory resource (total, free, used, swapped) and analogously *StorageFeature* individuals of this class refer to the state of a storage resource (free, used, quota). *Processor-Feature* items refer to the state of a CPU (frequency, cores, load). Descendants of *DelayFeature* describe one-way delay or round-trip delay over a network link or path, whereas *BandwidthFeature* describes the capacity related metrics in a link or over a network path (available, consumed, utilization). The rest of the interesting features can be found in the *MiscFeature* class, like uptime of a resource, or packet loss over a path.

The Feature model extends the Unit model. It is important to emphasize here all features bear information about their dimension, so one can deduce the natural and all the possible units of a given feature.

4.2.3. The Task model and the Parameter model

It is true that there are more tools that may measure the same metric. So it makes sense to unify them to an extent and hide as much details about the data generators from a data consumer as it makes sense. Still it is essential to bind the tools operable in the resources or their results stored in repositories and the feature metrics a user or a service look for together. The Task model makes this bridge.

Notions of the Task model describe data generators and guide the unification of the result data as well.

An instance of the *Task* class represents a measurement session, that is a set of operations including the initialization, execution and termination of measurement processes, and the gathering of measurement results. To generalize the access to the monitoring tools, the task will have a unique driver implementing the communication protocols, such as ssh, SOAP, XML-RPC, HTTP-REST or DB. In order to properly set up the communication channel with the tools that require authentication Task has an authentication type relation like username and password or RSA-key based authentication.

A monitoring task also bears information of the execution plan for a given network measurement. To this end, it consists of five predefined procedures to be implemented. These are the *prehook* to initialize a monitoring session, the *starthook* to launch an asynchronous jobs, the *retrievehook* specifying fetching and digesting of logs, the *stophook* to terminate a job and the *posthook* to clean up. Some of these hooks are optional depending on the tool. One can observe that using these hook procedures, we can easily define both synchronous and asynchronous measurements for short and long term monitoring scenarios.

The model targets to cover a wide range of monitoring tools, which are available to enable experimenters to monitor various metrics of the resources and the network. The tools developed for the past decades serve for different purposes, many of them only focus on a subset of the metrics and provide their users

with basically different capabilities. Some tools allows users to parameterize and perform active measurements on demand, while others do not provide any customization feature, merely return measurement data. The Parameter model guides the unit aware description of the runtime parameters of the former set of tools.

4.2.4. The Site specific configuration

In the federation for each site a configuration document contains Task individuals pinning down the available tools and binding them to the appropriate features. As a proof of concept, in NOVI numerous command line applications were mapped by the model, like `ping`, `traceroute`, `iperf`, etc. Furthermore, some more complex measurement frameworks were used to demonstrate the generality of the monitoring model.

Frameworks used in NOVI included the Service Oriented Network Measurement Architecture (SONoMA) [10] a Web Service based network measurement platform that is scalable, adaptable and open for scientists and other network developers. SONoMA realizes a common and extensible active network measurement framework that aims at decreasing the required time and efforts of network experiment implementation. Its services can be accessed via a standardized SOAP-based web services interface, not constraining the used programming language. The SONoMA architecture consists of a central management system, which submits experiments on a proper set of remote measurement agents in a completely distributed manner. In addition, the results are not only forwarded back to the user, but they are automatically stored in a public data repository, called Network Measurement Virtual Observatory (VO) [11] as well.

Another mapped architecture is the Packet Tracking tool (PT) [12], which is a multi-hop packet tracking architecture that provides the experimenter with information on the paths that packets take throughout the network. Besides that, it carries other hop-by-hop metrics like delay and loss, as well, and this tool enables to measure the extent of cross-traffic and its influence on the experimenter's traffic. PT uses a hash-based packet selection technique that ensures a consistent selection throughout the network while maintaining statistically desired features of the sample to reduce the traffic overhead caused by the probes.

The Hades Active Delay Evaluation System (HADES) [13] is a network monitoring tool providing performance measurements following the IETF approach of RFC3393. It provides its users with different performance metrics such as one-way delay, delay variations and packet loss. HADES was designed for multi-domain delay evaluations with continuous measurements. HADES measurement agents are generating bursts of UDP probes continuously throughout the day. The experimenter can access the measured data via a web service interface, but has no direct influence on the measurement process.

4.2.5. The Query model and the Statistics model

The purpose of this module is to cleanly bridge information request exchange between data consumers and data generators. The following queries were defined in the model: the *BundleQuery* is the simplest information request binding

a resource and a metric together, the *BatchQuery* is a similar request binding a set of resources and one metric together and the *SampleManipulationQuery* that describes statistical post processing of the data among them the formulation of conditional checking, which can serve as watchdog signaling. A query optionally may provide extra parametrization to fine tune task control and it carries instructions how to serialize data, indicating e.g. CSV or JSON format.

Our intention was to go beyond just modeling data collection so we extended the model with trivial data manipulating methods. Most often data consumers are interested in aggregate results rather than a whole dump of log. Further more, the use of this model can help automatize recognition of some malfunctioning of a topology component or a stress in the requested virtual slice. The three corresponding notions of the Statistical model are the *Sample*, the *SampleOperator* and the *Condition*.

Sample represents data quantities. It stands both for the raw data generated by a tool and also for the data that have undergone some transformation. With an abstract terminology, one can say that the domain and the range of data operations are the same set, whose elements are called Samples. Magnifying into this class we have the *ReducedSample* representing the reduction of the input, such as the calculation various aggregates or resampling of the source of data. To complement *UnionOfSamples* enumerates a combination data from different sources of compatible type, i.e. matching dimension. The SampleOperator class manifests the operations on the data, its disjoint subclasses are the *Aggregator* projecting the sample set to the minimum, maximum, average, standard deviation or the percentile, the *Resampler* filtering the sample set, like truncation, ordering and shuffling, and the *Condition* that enables comparison of Samples.

It possible to express a whole chain of operations, based on the model the non-trivial example expression can be formulated: "Calculate the 90% percentile of the last 100 samples".

4.3. NOVI Policy Ontology

The Policy Service provides the functionality of a policy-based management system, where policies are used to define the behaviour governing the managed environment.

Policies are expressed using the NOVI Policy Ontology, which supports the following policies:

- Authorization policies that specify authorization rights of users within the federation
- Event-condition-action policies that enforce control and management actions upon certain events within the managed environment.
- Role-based-access control policies that are used to assign users to specific roles, and where different permissions/usage priorities on virtualized resources are granted to each role.

- Mission policies are used to define inter-platform duties, i.e. the management obligations that a platform must fulfill against its peer platform in a NOVI federation.

The NOVI policy ontology provides the primary support for the operation of the Policy Service. These concepts are linked to the networking resources described using the NOVI resource ontology. The OWL representation of the Policy ontology can be seen in [14].

The main classes of Policy Ontology are:

- **ManagedEntity** is the entity that the Policy Manager is able to manage. This includes virtual resources and services within the NOVI federation;
- **NOVIUser** is a user of NOVI. All users of NOVI are basically users of the underlying platforms, but after authentication, they are considered as NOVIUsers and the appropriate object is created to keep the necessary information;
- **Platform** describes a particular testbed in the NOVI federation. This class is defined in the core ontology;
- **PolicyAction** is an action that can be performed by the Policy Service;
- **PolicyEvent** is an Event that can be caught by the Policy Service to trigger the proper actions;
- **PolicyCondition** is a condition that needs to be checked before triggering the actions;
- **ManagedEntityMethod** is a method of a **ManagedEntity** which is used to interact with it. Each **ManagedEntity** has a set of such methods;
- **ManagedEntityProperty** is a property of a **ManagedEntity**. These properties can be used as conditions for Policies applied on a **ManagedEntity**. For example if we want to reserve a Virtual Node that has a **ManagedEntityProperty** equivalent to a maximum reservation time we can have a Policy that uses the condition that the user requested reservation time is less that a **MaxReservationTime**.

A **ManagedEntity** has several subclasses:

- **ManagementDomain** is a Domain that may contain other **ManagementDomains** and **ManagedEntities**. It gives the ability to manage the entities in a common way. It is also a subclass of **ManagedEntity** to give the ability to be managed by itself;
- **Policy** is an abstract class to define policy rules that will be used to manage the **ManagedEntities**;
- **Resource** is the resource on which the Policies are applied. This class is the same as the one defined in the core ontology;

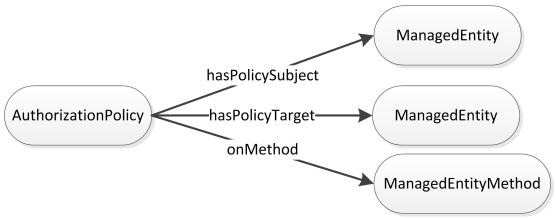

Figure 6: Visual representation of Object Properties of authorization Policies

- **Service** is the service on which the Policies are applied. This class is the same as the one defined in the core ontology;
- **Event** is the event as it is stored in the Policy Engine;
- **NOVIRole** is a role that a user of NOVI may have;
- **MissionController** is the class that loads and manages the **MissionPolicies** on the remote Platform;
- **MissionInterface** is the class that provides a generic way of expressing missions in order to specify later the specific class that will be used to the policies included in a **MissionPolicy**.

The **Policy** class has also a number of subclasses:

- **ECAPolicy** is an Event-Condition-Action Policy Rule, i.e. when an event occurs, and the conditions are met, the action triggers;
- **MissionPolicy** is a set of Policies that define the duties of a Platform with respect to other Platforms within a NOVI federation;
- **Role-based-access Control Policy** is a Policy used to define access rights of the users according their Role;
- **Authorization** (Access Control) Policy is a Policy used to define Authorization rights to the users according their Role in NOVI.

Figures 6 7 8 provide visual representation of the different types of Policy classes and their relations via the properties defined in the NOVI Policy Ontology.

As we can see in 6 **AuthorizationPolicy** ***hasPolicySubject*** a **ManagedEntity** and ***hasPolicyTarget*** also a **ManagedEntity**. This is to say that each Authorization policy can be applied on a target Managed Object (ex. a Resource, Service, etc) and it will refer to a specific ManagedEntity as a subject for the policy. The subject can be a NOVIRole, but in the description it was preferred to use the ManagedEntity to leave the definition more generic. Additionally Authorization Policies have some data properties for the definition of the Enforcement Points for the policies. They can be on Target or on Subject side and for request or for response. Each Event-Condition-Action Policy

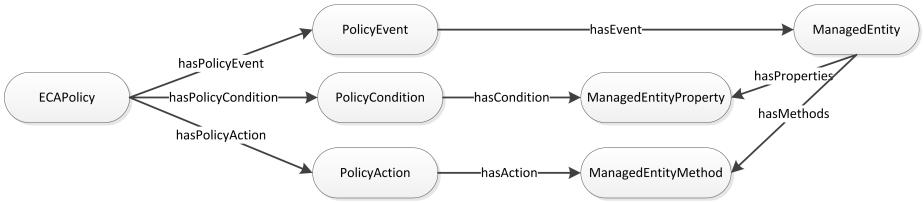

Figure 7: Visual representation of Object Properties of ECA Policies

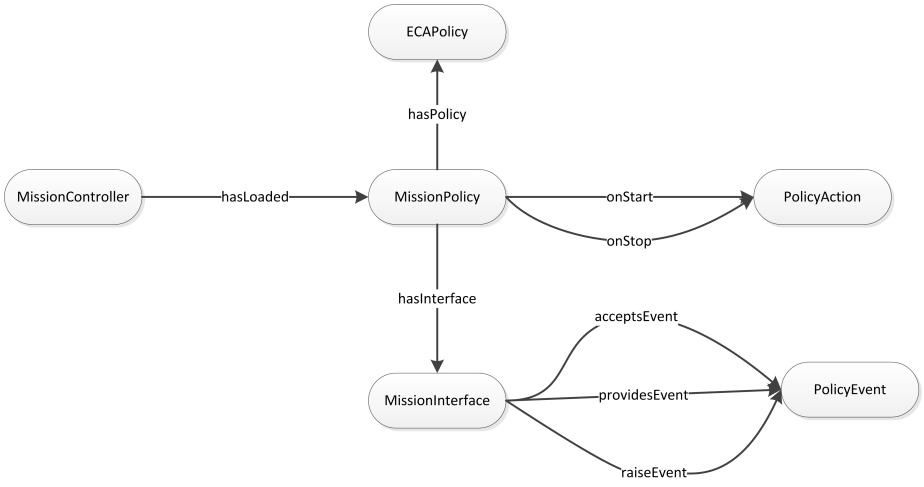

Figure 8: Visual representation of Object Properties of Mission Policies

is consisted of three parts. The event that triggers the policy, the conditions that should be satisfied in order to execute the actions and the actions. So as we can see in 7 each **ECAPolicy** ***hasPolicyEvent***, ***hasPolicyCondition*** and ***hasPolicyAction***. **PolicyEvent** ***hasEvent*** a **ManagedEntity**. **PolicyCondition** ***hasCondition*** **ManagedEntityProperty** since conditions are Managed Entity methods and **PolicyAction** ***hasAction*** a **ManagedEntityMethod** since actions are Managed Entities methods. In 8 we can see that **MissionPolicy hasPolicy** an **ECAPolicy**. This provides the ability to define the **ECAPolicies** included in a **MissionPolicy**. **onStart** and **onStop** introduce the relationships between a **MissionPolicy** and **PolicyAction**. They define the **PolicyActions** that should be executed when a **MissionPolicy** is started and terminated. **hasInterface** introduces the relationship between a **MissionPolicy** and **MissionInterface**. Each **MissionPolicy** has a set of local and remote **MissionInterfaces** on which the **ECAPolicies** are applied. **acceptsEvent**, **providesEvent** and **raiseEvent** introduce the relationships between **MissionInterface** and **PolicyEvent**. Each interface is able to accept, provide and raise **PolicyEvents** in order to interact with other **ManagedEntities**. **hasLoaded** introduces the relationship between **MissionController** and **MissionPolicy**. Each **MissionController** can load some **MissionPolicies**, on a remote **Platform**.

5. Implementation

This section describes how the NOVI Information model was implemented and used in the NOVI Services. A first important step for using an information model is to pour it into a data model. Then we describe how the different services have benefitted from the NOVI IM. For a more detailed discussion of the implementation of the NOVI IM, see [15].

5.1. Data Model

The NOVI Information Model has been created using the semantic web approach. So as a data model we have used OWL and RDF. For storing the RDF triples in the NOVI repository we use the Sesame[16] RDF triple store. For some of the services in NOVI, it is easier to deal with Java objects than RDF data. So the Data Model is represented as Java Classes and the Alibaba library[17] is used in the creation of the data model, and in the conversion of RDF triples to Java objects. This data model makes processing of data easier and facilitates the communication between the NOVI services.

5.2. NOVI Services

The GUI and API are the user interaction components and they are used by the user as entry point for interaction with NOVI Service Layer. The GUI is used for composing slice requests. These are then submitted using the NOVI API. Further interaction with the user always involves the NOVI API, which involves

the conversion of these requests to the appropriate NOVI IM representation for processing.

The Resource Information Service uses both NOVI Resource and Policy IM to store information in the database. The testbed physical substrate is stored in RIS as a Topology. The slices are stored as a Reservation, which contain all the necessary slice information. The RIS can also accept a resource discovery request, it then contacts the Monitoring Service to get the non-functional characteristics (availability) of the appropriate physical machines, utilizing the Monitoring ontology. RIS exploits the semantic query language SPARQL [18], in order to find the available resources which meet the requirements posed by the user. It dynamically constructs SPARQL queries based on the given topology request.

The Intelligent Resource Mapping utilizes the IM for identifying functional and non-functional characteristics of the requested Resources before reservation and for expressing mapping information in the resulting bound request that is further processed by the NOVI service layer.

The Policy Service uses the Resource and Policy ontologies to communicate and enforce policies via the NOVI Service layer. All Physical Resources provided by each Platform, described using the NOVI IM, are instantiated as Ponder2 Java Objects. Ponder2[19] is the policy Engine and specifies policies in a subject-action-target format, with optional fields such as constraints, triggers etc.

The Request Handler allows the Resource Information Service (RIS) to retrieve the substrate topology and the description of available substrate resources including their functional characteristics. The RH service provides an abstraction of the virtualized platform characteristics, according to the NOVI IM.

The Monitoring Service heavily relies on the Resource and Monitoring ontologies. Queries to the Monitoring Service are represented as an OWL document importing the relevant parts of the NOVI Information model complemented by the internal site-specific knowledge. Data returned to the requester can be of various formats as requested in the query, but in the response the meta-information with all the mappings to the corresponding concepts of the NOVI Information model are indicated.

The NSwitch manager receives Topology class objects and based on the Resource ontology it decomposes them and then performs actual configuration of federated links on resources that participate in the slice federation.

6. Conclusion

The NOVI information model defines the semantics to describe NOVI resources and services, express requests, supports the NOVI policy based management system and unifies active and passive monitoring information of the testbeds. It enables the interoperation among the NOVI software components.

The development of the NOVI IM has closely followed the evolution of the NOVI architecture as a whole. Since the delivery of the first draft of the model several changes have been introduced to better match the emerging requirements. The choice of Semantic Web languages, and specifically the choice for

OWL, has proven to be a good one: additions, extensions and changes have been easily incorporated.

The strength of the NOVI model is that while it fully satisfies the NOVI needs, it is also generic enough to be usable outside the scope of the project. In fact, we have received requests from other FI projects who are looking to adopt the NOVI IM, such as GENI, GEANT, MANTYCHORE, and others.

6.1. Future of the NOVI Information model

At the start of the NOVI project there were not many information models for Future Internet platforms, and certainly none for creating a federated environment of these. In section 2 we have analyzed the state of the art. We have highlighted different models and selected the most relevant models to work from. During the project we leveraged the definitions and experiences of these models.

During the creation and implementation of the NOVI models we have also shared our experiences with other projects, such as GEYSERS, and most importantly the Network Markup Language (NML) standardization working group at the Open Grid Forum. With our practical experiences in NOVI we have been able to provide an important contribution to the now standardised NML schema[7].

During the project several external parties have expressed interest in our work: we also exchanged our experiences with the NDL-OWL team in GENI. In the final stages of the NOVI project we have also started collaborating with the MANTYCHORE project, which expressed an interest in using our model for describing their infrastructure and integrating it into their software. Together with the NML standardization effort, this shows that the NOVI information model has provided an important contribution to the FI community. Given the number of the current interested users and projects, and some early adopters we expect the NOVI information model will be used in its current form or with possible extensions by the Future Internet community in the coming years.

Acknowledgments

This work was partially supported by the European Commission, 7^{th} Framework Programme for Research and Technological Development, Capacities, Grant No. 257867 – NOVI. Furthermore, this publication was supported by the Dutch national program COMMIT and the GigaPort 2013 Research on Networks project. The authors thank the partial support of the EU FP7 OpenLab project – Grant No.287581 –, the EIT ICTLabs FITTING project, the MAKOG Foundation and the EITKIC 12-1-2012-0001 project supported by the Hungarian Government, managed by the National Development Agency.

References

[1] DMTF's common information model, `http://www.dmtf.org/standards/cim`.

[2] Distributed Management Task Force (DMTF), http://www.dmtf.org/.

[3] Directory Enabled Networks – Next Generation, http://autonomic-management.org/denng/index.php.

[4] M. Sloman, Policy driven management for distributed systems, Journal of Network and Systems Management 2 (1994) 333–360.

[5] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, C. de Laat, Using the network description language in optical networks, in: Tenth IFIP/IEEE Symposium on Integrated Network Management, 2007.

[6] MOMENT Project Deliverable, D3.2 Final specification of the unified interface.

[7] J. van der Ham, F. Dijkstra, R. Łapacz, J. Zurawski, Network markup language base schema version 1, GFD.206 (2013).

[8] L. Lymberopoulos, M. Grammatikou, M. Potts, P. Grosso, A. Fekete, B. Belter, M. Campanella, V. Maglaris, Novi tools and algorithms for federating virtualized infrastructures, Future Internet – From Technological Promises to Reality (2012) 213–224.

[9] S. S. Stevens, On the theory of scales of measurement, Science.

[10] B. Hullár, S. Laki, J. Stéger, I. Csabai, G. Vattay, SONoMA: A Service Oriented Network Measurement Architecture, in: TridentCom 2011, 2011.

[11] P. Mátray, I. Csabai, P. Hága, J. Stéger, L. Dobos, G. Vattay, Building a prototype for network measurement virtual observatory, in: Proceedings of ACM SIGMETRICS, 2007, mineNet.

[12] T. Santos, C. Henke, C. Schmoll, T. Zseby, Multi-hop packet tracking for experimental facilities, in: Proceedings of the ACM SIGCOMM 2010 conference, SIGCOMM '10, 2010.

[13] P. Holleczek, R. Karch, R. Kleineisel, S. Kraft, J. Reinwand, V. Venus, Statistical characteristics of active ip one way delay measurements, in: Proceedings of the International conference on Networking and Services, ICNS '06, 2006.

[14] Policy IM OWL file, http://wiki.fp7-novi.eu/pub/WP2/Documents/policy_imV4.owl.

[15] J. van der Ham, W. Adianto, F. Farina, P. Grosso, Y. Kryftis, A. Monje, C. Papagianni, B. Pietrzak, C. Pittaras, J. Steger, C. V. Lopez, Novi deliverable 2.4: Final information and data models plus report on prototypes, http://www.fp7-novi.eu/deliverables/doc_download/71-d24.

[16] Sesame rdf triple store, http://www.openrdf.org.

[17] AliBaba, website: http://www.openrdf.org/alibaba.jsp, http://www.openrdf.org/alibaba.jsp/.

[18] SPARQL query language for RDF, http://www.w3.org/TR/rdf-sparql-query/.

[19] Ponder2 policy framework, http://ponder2.net/.