Rethinking network management: Models, data-mining and self-learning

Article · April 2012

DOI: 10.1109/NOMS.2012.6212003

CITATIONS
4

READS
90

3 authors:

Stefan Vallin
Luleå University of Technology Alumni
24 PUBLICATIONS **130** CITATIONS
SEE PROFILE

Christer Ahlund
Luleå University of Technology
69 PUBLICATIONS **606** CITATIONS
SEE PROFILE

Johan Nordlander
Data Ductus AB
37 PUBLICATIONS **314** CITATIONS
SEE PROFILE

Rethinking Network Management : Models, Data-Mining and Self-Learning

Stefan Wallin, Christer Åhlund and Johan Nordlander
Luleå University of Technology 971 87 Luleå
Email: stefan.wallin@ltu.se, christer.ahlund@ltu.se, johan.nordlander@ltu.se

***Abstract*—Network Service Providers are struggling to reduce cost and still improve customer satisfaction. We have looked at three underlying challenges to achieve these goals; an overwhelming flow of low-quality alarms, understanding the structure and quality of the delivered services, and automation of service configuration. This thesis proposes solutions in these areas based on domain-specific languages, data-mining and self-learning. Most of the solutions have been validated based on data from a large service provider.**

We look at how domain-models can be used to capture explicit knowledge for alarms and services. In addition, we apply data-mining and self-learning techniques to capture tacit knowledge. The validation shows that models improve the quality of alarm and service models, and enables automatic rendering of functions like root cause correlation, service and SLA status, as well as service configuration.

The data-mining and self-learning solutions show that we can learn from available decisions made by experts and automatically assign alarm priorities.

***Index Terms*—network management, data models, data mining, neural networks**

I. Introduction

Alarm monitoring and *service management* are fundamental processes for a network service provider. Network administrators monitor alarms in order to make sure the provided services works as expected. They analyze the alarms and take appropriate actions to minimize the effects and correct the fault. Therefore the alarm quality is vital, the information such as severity levels and alarm texts must be as accurate as possible in order to optimize this process. *Service management* can be broken down into *monitoring and activation*. Service *monitoring* tries to measure or estimate the service quality. The service monitoring function is used both for validating that service level agreements with customers are fulfilled and to guide the network administrators towards a proactive management process. Service *activation* drives the service provider business in that this is the function that turns up the service ordered by a customer. Therefore quick and error-free service activation is critical.

The above mentioned areas are the target for this paper and the corresponding Ph. D. thesis[1]. The work is fully described in previously published articles [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13].

[1]http://staff.www.ltu.se/~stewal/noms-wallin-thesis.pdf

978-1-4673-0269-2/12/$31.00 © 2012 IEEE

We studied the problem and challenges both based on our industrial experience [1] and with a structured survey [2]. Our approach for this broad scope is to look for core models that can ease the overall subject. Therefore we looked at *domain-specific languages* in order to simplify and automate alarm monitoring and service management. Domain-specific models can be used to capture explicit knowledge from domain experts but not everything can actually be modeled. In our work we turn to *data-mining* to capture this kind of tacit knowledge or information that can be used in *self-learning* techniques.

The main contributions of this thesis are:

- We give a useful *definition* of "alarm" and *statistical alarm analysis* within an *alarm analysis taxonomy*.
- We validate that a *data-mining and neural network* self-learning process can assign valid alarm priority levels.
- We have designed *domain-specific language, BASS,* that can improve the quality of the alarm interface definitions and enable automatic correlation of alarms.
- We show that domain-specific models can automate and simplify service monitoring and activation. For service monitoring, we propose a time-aware functional domain-specific language, SALmon. To address service activation, we show that a solution based on the IETF standardized domain-specific language YANG can automate parts of the solution.

II. The Chaotic Alarms

Network administrators are flooded with alarms which require action, either to resolve the problem or to define the alarm as irrelevant. Therefore, the alarm quality is vital. Despite this, we are still in a situation where we see fundamental problems in alarm systems with alarm floods, standing alarms, a lack of priorities, and alarm messages that are hard to understand. The problem has been around for decades without much improvement.

After considering various attempts to define what an alarm is, we have adopted the following definition [5]:

Definition 1 (Alarm): An *alarm* signifies an undesired state in a resource for which an operator action is required. The operator is alerted in order to prevent or mitigate network and service outage and degradation.

At the core of this definition is that every alarm needs manual attention and manual investigation and possible actions. Many alarms do not adhere to this definition in that they are just status information changes more targeted for logging systems. System vendors do not always realize that every alarm is actually an operational cost for the service provider.

In the following sections we summarize our findings based on data-mining analysis, Section II-A, domain-specific models for alarms in Section II-B and finally data-mining and self-learning in Section II-C.

A. Alarm Taxonomy and Related Findings

In order to understand the characteristics of alarms we defined an alarm taxonomy partly borrowed from linguistics and semiotics. The taxonomy is useful for studying alarms, alarm interfaces and alarm standards.

- *Phenomenon* : the resource state change or event that is interpreted as an alarm
- *Syntax and grammar* : the protocol and information modeling language to define the alarm interface. Since this only defines the envelope for the alarm, we do not study this.
- *Semantics* : what does the alarm say?
- *Pragmatics* : what is the meaning and effect of the alarm when using contextual information?

In the rest of this section we use the above taxonomy to describe our findings and recommendations [5], [6].

1) The Alarm Phenomenon: We studied an alarm and trouble-ticket database from a mobile service provider in order to understand the first level of the taxonomy. The alarm database contains close to 20 million alarms over a three-month period. The investigation [5] shows that the alarms do not adhere to the basic alarm definition (Definition 1). Only about 10 % of the alarms required manual actions and only these should really qualify as alarms.

Studies also showed that 20% of the alarms are cleared by the reporting device in less than 5 minutes and 47% of the alarms are closed by administrators or automatic rules within 5 minutes. This is again an indication of alarms that could be filtered out or never sent.

Another interesting observation is the “rule of vital few”. In total, there are over 3500 different alarm types defined in the sample alarm database, but most of them are very rare. Given the distribution of tickets and alarms on different alarm types we get 90% of all our trouble-tickets from less than 30 of the most common alarm types. On the other end of the scale, 10% of all alarms belong to alarm types that have never given rise to an actual ticket.

2) Alarm Semantics: The *semantic level* deals with understanding the alarm information. An alarm carries a defined set of parameters according to the grammar but the contents need to be understood to provide meaning to the network administrator.

At this level we continued to study the meaning of *severity*. We looked at the correlation of alarm severity levels as set by the equipment, versus associated trouble-ticket priorities set by network administrators. The results shows that there is very little correlation between these two. This is even more disturbing when realizing that the alarm severity is actually used for the initial sorting. On average, critical alarms were acknowledged 500 times faster then warning alarms. See Section II-B on how we address alarm semantics with domain-specific models.

3) The Alarm Pragmatics: Finally, at the *pragmatic level*, we need to understand the impact and context of the alarm. Network administrators use contextual information such as topology and (informal) service models to understand the alarm pragmatics. Most important of all is their informal expert knowledge. We see two primary ways to improve the situation at this level: data-mining with self-learning, Section II-C, and service models, Section III-A.

B. Addressing Alarm Semantics with a Domain-Specific Language

In this section we will define the basic concepts needed to define alarms and alarm types. These definitions are then used to form the alarm definition language, BASS [7]. We will show how BASS can be used to improve the quality of existing informal alarm documentation and how it enables automatic alarm correlation. The correlation is validated using real alarms from a service provider.

We need to distinguish different *types* of alarms. An *alarm type* is a tag that assigns a name to a specific resource state and it is fundamental in order to define good alarm interfaces.

Many alarm standards refer to an *alarm type* without giving it a precise definition. These standards only give the tags of alarm types without further definitions. These tags can sometimes be understood by humans but not by automatic systems. Another problem in this flat naming approach is that there is no way to subtype existing alarm types. The only way of extending an alarm type is by adding a completely new alarm type, which creates unnecessary complexity in the management system.

Definition 2 (Alarm type): An alarm type $\mathcal{T}$ identifies a unique alarm state for a resource. It corresponds to a list of predicates P on the resource attributes, each with an associated severity S. Alarm types are defined as specializations of previously defined alarm types.

$$\mathcal{T} = \overline{(P, S)}$$

The alarm types $\mathcal{T}_i$ are named using a hierarchical naming scheme with dotted names. The first level of alarm types includes the X.733 event types (Communications, QualityOfService, ProcessingError, Equipment, Environmental), on the second level we have all the standardized probable cause values. As an example, a power problem would have the alarm type *Equipment.powerProblem* and an UPS vendor could extend this to the alarm type *Equipment.powerProblem.UPSProblem*. Using the alarm type definition we can now give a definition of what we consider an alarm to be:

Definition 3 (Alarm): An alarm A signifies an *undesirable state* in a resource R, corresponding to an alarm type $\mathcal{T}$ and associated state changes C, for which an operator action is required to prevent or mitigate service degradation.

$$A = (R, \mathcal{T}, \overline{C})$$
$$\text{where}$$
$$C = (t, S, I)$$

The state changes $\overline{C}$ contains a timestamp t, a severity S, and auxiliary information I.

1) The Alarm Type Language, BASS: To formalize the alarm type specification process we designed a domain-specific language called BASS. It is used to create alarm models independently of the underlying alarm protocol. The language supports two kind of constraints: *semantic* and *information* constraints. Information constraints are relations that give information about alarms, they help us perform tasks such as grouping, correlating or filtering by allowing us to use the information gained from the relations. For example stating that one alarm type is the cause of another. We identify root cause alarms as faults, and the consequent alarms as errrors. Semantic constraints exists on the meta level and are used to verify that the model is correct. This could for example state for which resource types an alarm type is valid.

Listing 1 shows how to define alarm types in BASS. In this example we map the X.733 event type, probable cause and specific problem parameters into an hierarchical alarm type identifier *equipmentAlarm.replaceableUnitProblem.switchCoreFault*. Reasoning about alarms using a hierarchy rather than the historical tuple matching approach from X.733 will give us several benefits later when we want to perform semantic checks on the alarm model.

We also show an example of a semantic constraint (Line 2) in that `equipmentAlarm` can only be used for subclasses to `equipment`.

Listing 1. BASS Alarm Type Definitions

```
1  abstract alarm equipmentAlarm {
      resource equipment;
3  }
   abstract alarm
5   equipmentAlarm.replaceableUnitProblem {}

7  abstract resource equipment {}

9  alarm
   equipmentalarm.replaceableUnitProblem.switchFault
11 {
    kind = fault;
13  resource = plugInUnit; // subclass to equipment
    minor = plugInUnit.switchModule == disabled;
15  clear = plugInUnit.switchModule == enabled;
   }
17
   alarm
19 qualityOfServiceAlarm.lostRedundancy.SwitchRedundancy
   {
21  kind = error;
    resource = SwitchModule;
23  ....
    cause[Switch Core Fault] =
25   equipmentalarm.replaceableUnitProblem.switchFault;
   }
```

Line 9-16 defines a concrete alarm type, *switchFault* as a subtype to *replaceableUnitProblem*. Since this is an equipment alarm we require the resource to be a subclass of equipment, a semantic constraint. In Line 18 we define another alarm type *switchRedundancy* that models a casual dependency (information constraint) to the previously defined alarm type, (Line 24-25).

The primary goal of the BASS compiler is to ensure the correctness of alarm models and give semantic feed-back based on analysis of the models. The BASS compiler can also generate various outputs, most importantly it generates a Python library so that custom output modules can be easily written for agents or managers.

We transformed the alarm documentation from a large equipment vendor into BASS specifications and were able to transform all informal documentation with about 300 alarm types into BASS. We found more than 150 semantic warnings by compiling the corresponding BASS specifications. Correction of the semantic warnings led us to improve on the quality of the specification in several ways.

Since the alarm model contains causality definitions it can serve as the input to a correlation engine. To evaluate this possibility we used a historical database from a large service provider containing over 3 million alarms from more than 3,000 distinct alarm types from several vendors. We matched the database against the BASS model, which gave us a subset of 245,759 alarms with 71 different alarm types. Finally, we expanded the BASS specifications for 5 randomly selected alarm types to enable generation of correlation rules. We had to adopt the resource identification to the naming conventions of the service provider. From the BASS compiler we then generated SQL queries that grouped alarms using the causality-model, these queries were then run against the alarm database. For the randomly 5 selected alarm types we could identify the root cause alarm in 42% of the cases.

This shows that current informal documentation is an unused asset, by applying a domain-specific language to the alarm interface documentation we can enable automation of the alarm processes.

C. Addressing Alarm Pragmatics With Data-Mining and Neural Networks

This part of our work focus on assigning relevant priorities to alarms at the time of reception by using neural networks [3], [6]. We let the neural network learn from the experienced network administrators rather than building complex correlation rules and service models. To validate this approach we used a database with 85814 trouble-tickets associated with 260973 alarms. 10% of the tickets were used as training data and 90% as test data. Based on discussions with network administrators we could select the most relevant alarm attributes and feed that to the training process along with the manually assigned priority. After the training process we tested the remaining 90% of the alarms and matched the priority set by the neural network with the original severity from the device and the manually assigned priority in the trouble-ticket. The prototype assigns the same severity as a human expert in 50% of all cases, compared to 17% for the severity supplied in the original alarm.

This solution has several benefits:

- Priorities are available immediately as the alarms arrive.
- Captures network administrators' knowledge without disturbing their regular duties.

III. The Unmanaged Services

Managing complex services requires a service modeling language that can express the service models in an efficient way. We have used two languages; SALmon and YANG, Figure 1 illustrates the context of the two languages in this thesis. SALmon focuses on expressing the services status as a function of service and device attributes and time-awareness of the calculations. In order to study service configuration we used the newly released IETF standard YANG [14]. YANG focuses on the structural part and validation of correct configurations. The reason for these two languages in this thesis is primarily because YANG was not available at the start of our work.

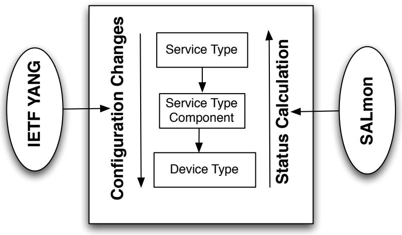

Fig. 1. The two DSL for Service Management

Service monitoring and service activation are in many ways orthogonal. Service status can be calculated as a function of service component and device attributes. Service activation implies a calculation from desired service state to new service component and device configurations.

A. Addressing Service Monitoring with SALmon

SALmon [4], [9] is a tailor-made language for expressing service models. SALmon combines object-oriented structuring for service model decomposition and functional expressions for status calculations. Due to the nature of service modeling, the programming language must have the capability to treat time as part of the normal syntax: all variables are seen as arrays indexed by a time stamp. It is possible to use the time-index syntax to retrospectively change the value of variables. This is important in many scenarios, such as, late arrival of probe data. This also enables SALmon to recalculate SLA status whenever new status indicators are reported.

The evaluation of attributes is performed by time-indexing. Time indexes are restricted to constants or constant functions of the NOW parameter such as

```
system.status@(NOW-1h)
```

The expression can be used both as right-hand value and left-hand value, the latter to change a value retrospectively.

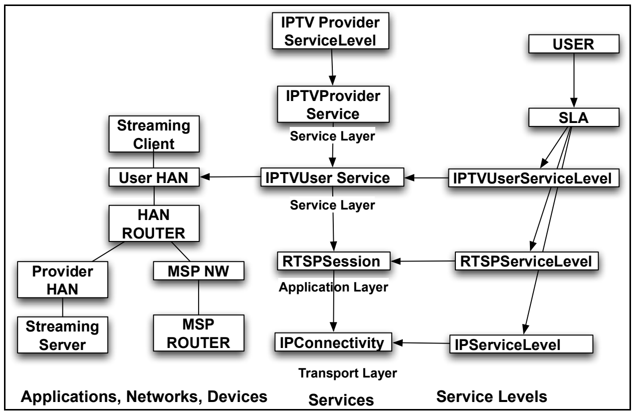

Fig. 2. Service Model

Intervals of a time-variable can also be retrieved by specifying a time range as in the following example:

```
dailyStatus = worstOf system.status@(NOW, NOW-1day)
```

The language has two fundamental layers: the *Definition Layer* and the *Instantiation Layer*. The *definition layer* defines the classes and calculations in the model. Core concepts that we want to represent as classes are Services, Service Levels and SLAs. Classes have inputs, anchors, attributes and properties. Inputs represent external sources to the models like alarms, trouble-tickets and performance probes. Anchors are references to other instances in the models. Attributes represent the values we want to calculate and properties represent values bound at instantiation. The `instantiation layer` creates instances of the service classes, assigns properties and establishes connections between instances through anchors.

The best way to understand SALmon is probably by looking at an example. In order to validate SALmon we built a prototype that calculated the quality of an IPTV stream, and associated Service Level Agreements [10], [11]. We also showed how to use SALmon in the context of wireless access point selection [8]. The rest of this section will use the IPTV prototype to explain the core concepts of SALmon. The first step was to study a technical report from DSL Forum on triple-play services quality: TR-126 "Triple-play Services Quality of Experience (QoE) Requirements"[15]. This is an excellent document that describes an "informal service model". The first step in the validation was to make sure that we could capture the report in a domain-model. This turned out well. Figure 2 illustrates the classes in the resulting IPTV model.

Listing 2 shows fragments of the `IPConnectivity`, and `IPServiceLevel` classes. The `IPConnectivity` class represents a transport layer link, which is fairly straightforward from a model point of view, it has a set of inputs representing lower level Key Performance Indicators, (KPIs), from probes.

Listing 3. Service Level Template

```
def IPTVServiceLevel = IPTVUserServiceLevel(
      max_channel_change_time => 2s,
      max_startup_time => 10s,
      min_availability => 0.9994,
      max_video_quality_impairment => 5,
      time_interval => 24h)
```

Listing 2. IP Connectivity

```
   class IPConnectivity
 2  input jitter, packet_losses, packets, delay,
     datarate
 4
   class ServiceLevel
 6  violated = false, jeopardized = false
    property jeopardized_th = 0.8, time_interval
 8  I = (NOW, NOW-time_interval)

10 class IPServiceLevel inherits ServiceLevel
    anchor ip_connection
12  property max_jitter, max_loss_rate, max_delay,
     min_datarate
14  losses  = ip_connection.losses
    packets = ip_connection.packets
16  // Lambda expression for calculating
    // loss rate in the s,e interval
18  loss_rate = (s, e) -> (losses@e-losses@s) /
     (packets@e-packets@s)
20  violated = (max jitter@I) > max_jitter or
          (loss_rate I)  > max_loss_rate or
22        (max delay@I)  > max_delay
    jeopardized = (max jitter@I) >
24    max_jitter*jeopardized_th  or ...
```

We then define an abstract `ServiceLevel` class (Line 5) which enforces subclasses to implement `jeopardized` and `violated` status attributes. The Service Level class also defines an expression that refers to the current SLA measurement interval:
`I = (NOW, NOW-time_interval)`
This is a first illustration of SALmon's time-awareness and its usefulness in the context of service monitoring. `NOW` is a keyword referring to the current time. `I` becomes a time interval from the time of evaluation back to the corresponding `time_interval`. So assume the `time_interval` is set to 1h and we evaluate an expression at 12.00, `I` would refer to the interval [12.00, 11.00]. The `time_interval` property is defined when the Service Level is instantiated and should correspond to the SLA measurement period, for example be hourly or daily.

Finally the *IPServiceLevel* class in Line 10 defines an anchor to the monitored *IPConnectivity* class, loss-rate calculation based on a time interval and the Service Level thresholds based on network KPIs.

With this general definition of a IPTV Service Level we can create templates for different service levels where we assign values to the properties (thresholds). Listing 3 shows a service level for IPTV based on the requirements defined by DSL Forum. The service level is measured daily by setting `time_interval` to `24h` and the other properties defines the thresholds, for example it shall take maximum 2 seconds to change channel.

SALmon is different from many of the upcoming products targeting the problem domain in that the model can be directly executed. There is no additional implementation layer layer to define the functional behavior. Also we remove the mapping to storage in that we automatically persist everything directly from the model to the Berkeley DB. We believe that a small and efficient domain-specific language will be successful. While graphical approaches may seem attractive at the outset, they often face obstacles when faced with the full complexity of the operator's reality. This approach also eases the transition from current monitoring solutions where most of the integration is performed by using modern, regular, languages like Python. A SALmon-based approach will attract skilled integrators and give them a tool where they can rapidly change and develop models.

B. Addressing Service Configuration using YANG and NETCONF

With SALmon we looked at a functional approach to calculate service status from lower layers in the model. When we turn to service configuration things are turned up-side-down: it is a transformation from higher level models to lower levels and it is not state-free, service configurations can result in almost any changes in the network.

We looked at service configuration using the YANG language rather than continuing with SALmon, mainly since YANG is now an IETF standard. YANG is a data modeling language used to model configuration and state data, and it is tree-structured rather than object-oriented. Our main thesis is to reuse a device modeling language also at the service layer and thereby creating a two-layered YANG solution.

We studied YANG as a service modeling language using a YANG and NETCONF based management system, Tail-f NCS [12]. YANG does not cover functional or imperative expressions but is strong in expressing model constraints. This lead to a two-layered mapping solution:

1) Constraint Layer : we can use YANG "must" (XPATH 1.0) expressions to constrain the service and device configuration.
2) Configuration Mapping Layer : a Java based mapping layer that calculates the device changes based on a service model change.

Constraints can ensure that the service model is consistent including any references to the device model. We do this at compile time by checking the YANG service model references to the device model elements. At run-time, the service model constraints can validate elements in the device-model including referential integrity of any references. At the *configuration mapping layer* we use a "traditional" imperative approach in Java. This since a service configuration can have effects anywhere in the network. We simplified this layer as much as possible by providing a lazy-evaluated DOM-tree representing the hierarchical YANG model. This means that developers manipulate data-structures that correspond to the domain-model. Software developers are trained on manipulating data-structures rather than working with network management protocols.

Validation of the approach was performed by a large equipment vendor in building a Carrier Ethernet Service activation solution for Juniper and Cisco routers. The amount of code for the complete solution was less than 400 LOC Java. According to them, this was a code reduction of several orders of magnitude compared to their previous traditional approach.

IV. Visualization of Models

Modeling is a crucial part of network and service management. The models must be understood by human domain-experts as well as management applications. This dualistic requirement for modeling languages has led to trade-offs in either direction. There is a risk of losing the initial domain experts if the YANG, BASS and SALmon models cannot be communicated in other means than the language itself. We have defined and implemented YANG to UML mapping [13] and also UML and causality graph views for BASS [7].

V. Related Work

A. Alarm Models

Most research efforts related to alarm handling focus on alarm correlation. To present a complete overview of alarm correlation is the role of a whole other paper, see for instance the paper by Meira [16]. We focus on domain-specific models for alarms to capture the alarm semantics and enable automatic rendering of alarm functions.

Alarm standards like X.733 [17] and 3GPP [18] focus on defining the alarm protocol and the parameters. None of these standards address the constraints of an alarm interface or try to model the actual alarm types.

Event algebras [19], [20] define the basic mechanisms involved in mapping a sequence of primitive events to a complex event. This corresponds more to the problem of alarm correlation in that notifications in our model are mapped to alarm states.

Alarms are often modeled as finite state machines as in the work by Rouvellou [21]. In our previous work [5] we also tried to establish a general state machine for alarms. But we have come to the conclusion that a state-machine will always depend on the actual context and usage of alarms. Therefore we moved to a model based on discrete mathematics which allows for various interpretations of the alarm state depending on the context.

Duarte et al. [22] present an interesting approach, with their SNMP focused specific language ANEMONA. ANEMONA focuses on generating a alarm monitor solution and not the alarm model as such. Other relevant model-based approaches are illustrated by Frolich et. al. [23] and Yemini [24] et. al. Both use modeling approaches to describe the monitored resources and the associated alarms. This approach is very much in line with our model. While these approaches focus on generating an alarm correlation engine we focus primarily on validating the alarm model as such and give feed-back to the alarm designers based on static analysis.

B. Data-Mining and Self-Learning for Alarm Monitoring

Klemettinen [25] presents a complementary solution to alarm correlation, using semi-automatic recognition of patterns in alarm databases. The output is a set of suggested rules that the user can navigate and understand. This approach shows that data-mining is a way forward to solve the alarm management problem. Wietgrefe [26] uses neural networks to perform alarm correlation in order to find the root cause alarm, the learning process is fed with alarms as inputs and the triggering alarm as output. The results presented by Wietgrefe show that telecom alarm correlation based on neural networks behaves well in comparison to traditional rule-based approaches.

C. Service Management and Modeling

There are various proposed models and language for service modeling, the most relevant being: SID, “System Information Model” [27], “Multi-Technology Operations Systems Interface MTOSI [28], “Common Information Model”, CIM [29], “Service Modeling Language”, SML [30]. These focus mostly on the static structure of the models and most of them are UML based. We try to, in addition, look at functional definitions and constraints. Also some of the above languages do not have a canonical text-based representation which we believe is fundamental for distributed authoring and tooling.

There are many well-designed configuration management tools like: CFengine [31] and Puppet [32]. These tools are more focused on system and host configuration whereas we focus mostly on network devices and network services. This is mostly determined by the overall approach taken for configuration management. In our model the management system has a data-model that represents the device and service configuration. Administrators and client programs express an imperative desired change based on the data-model.

VI. Conclusion

This thesis shows that there is a lot to gain in formalizing service and alarm models using domain-specific languages. It enables both early validation of fundamental network information as well as rendering parts of the network and service management solution. While models capture explicit knowledge we also show that data-mining and self-learning can be used to capture tacit knowledge in order to enrich the alarm information.

Acknowledgements

This paper partly describes work undertaken in the Context of the Magneto-Celtic Telecommunication Solutions Project, which is partially funded by the Commission of the European Union.

Thanks to Data Ductus developers who participated in our projects: Johan Enmark, Ulrik Forsgren, Andreas Jonsson, Jörgen Öfjell and Leif Landén. We would also like to thank all the Tail-f developers, especially Claes Wikström, Martin Björklund, Sebastian Strollo, Johan Bevemyr and Joakim Grebenö who developed the YANG based product we used for service activation.

References

[1] S. Wallin and V. Leijon, “Rethinking network management solutions,” *IT Professional*, vol. 8, no. 6, pp. 19 –23, Nov. 2006.

[2] ——, “Telecom network and service management: An operator survey,” in *12th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS)*, October 2009.

[3] S. Wallin and L. Landen, “Telecom alarm prioritization using neural networks,” in *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, Mar. 2008, pp. 1468 –1473, dOI=10.1109/WAINA.2008.105.

[4] S. Wallin and V. Leijon, “Multi-Purpose Models for QoS Monitoring,” in *Advanced Information Networking and Applications Workshops, 2007, AINAW ’07. 21st International Conference on*, vol. 1, May 2007, pp. 900 –905.

[5] S. Wallin, “Chasing a definition of “alarm”,” *Journal of Network and Systems Management*, vol. 17, pp. 457–481, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10922-009-9127-3

[6] S. Wallin, V. Leijon, and L. Landen, “Statistical analysis and prioritisation of alarms in mobile networks,” *International Journal of Business Intelligence and Data Mining*, vol. 4, no. 1, pp. 4–21, 2009.

[7] S. Wallin, “The semantics of alarm definitions: Enabling automatic reasoning about alarms,” *International Journal of Network Management*, 2011, DOI: 10.1002/nem.800.

[8] C. Åhlund, S. Wallin, K. Andersson, and R. Brännstrom, “A service level model and internet mobility monitor,” *Telecommunication Systems*, vol. 37, pp. 49–70, 2008, dOI=10.1007/s11235-008-9072-6. [Online]. Available: http://dx.doi.org/10.1007/s11235-008-9072-6

[9] V. Leijon, S. Wallin, and J. Ehnmark, “SALmon A Service Modeling Language and Monitoring Engine,” in *Service-Oriented System Engineering, 2008. SOSE ’08. IEEE International Symposium on*, Dec. 2008, pp. 202 – 207.

[10] S. Handurukande, S. Wallin, and A. Jonsson, “IPTV service modeling in Magneto networks,” in *Network Operations and Management Symposium Workshops (NOMS Wksps 2010), IEEE/IFIP*, Apr. 2010, pp. 51 –54.

[11] S. Handurukande, S. Fedor, S. Wallin, and M. Zach, “Magneto Approach to QoS Monitoring,” in *IM 2011 - TechSessions*, Dublin, Ireland, may 2011.

[12] S. Wallin and C. Wikström, “Automating network and service configuration using netconf and yang,” in *Proceedings of Usenix Large Installation System Administration Conference (LISA 11)*, Dec. 2011.

[13] S. Wallin, “UML visualization of YANG Models,” in *IM 2011 - DANMS Workshop*, Dublin, Ireland, may 2011.

[14] M. Björklund, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF),” RFC 6020 (Proposed Standard), Internet Engineering Task Force, Oct. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc6020.txt

[15] Tim Rahrer and Riccardo Fiandra and Steven Wright, “Technical Report TR-126 Triple-play Services Quality of Experience (QoE) Requirements,” DSL Forum, Tech. Rep., 2006, http://www.broadband-forum.org.

[16] D. M. Meira, “A Model For Alarm Correlation in Telecommunications Networks,” Ph.D. dissertation, Federal University of Minas Gerais, November 1997.

[17] ITU-T, “X.733: Information technology - Open Systems Interconnection - Systems Management: Alarm reporting function,” 1992.

[18] 3GPP, “3GPP TS 32.111-2: Alarm Integration Reference Point (IRP),” 2007.

[19] D. Zimmer and R. Unland, “On the semantics of complex events in active database management systems,” in *Data Engineering, 1999. Proceedings., 15th International Conference on*, Mar. 1999, pp. 392 – 399.

[20] A. Hinze and A. Voisard, “A parameterized algebra for event notification services,” in *Temporal Representation and Reasoning. TIME 2002. Proceedings of the Ninth International Symposium on*, 2002, pp. 61 – 63.

[21] I. Rouvellou and G. Hart, “Automatic alarm correlation for fault identification,” in *INFOCOM ’95. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People. Proceedings. IEEE*, 2-6 1995, pp. 553 –561 vol.2.

[22] E. P. Duarte, M. A. Musicante, and H. D. H. Fernandes, “Anemona: a programming language for network monitoring applications,” *International Journal of Network Management*, vol. 18, no. 4, pp. 295–302, 2008. [Online]. Available: http://dx.doi.org/10.1002/nem.655

[23] P. Frohlich, W. Nejdl, K. Jobmann, and H. Wietgrefe, “Model-based alarm correlation in cellular phone networks,” in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. MASCOTS ’97., Proceedings Fifth International Symposium on*, jan. 1997, pp. 197 –204.

[24] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, “High speed and robust event correlation,” *Communications Magazine, IEEE*, vol. 34, no. 5, pp. 82 – 90, may 1996.

[25] M. Klemettinen, H. Mannila, and H. Toivonen, “Rule discovery in telecommunication alarm data,” *Journal of Network and Systems Management*, vol. 7, pp. 395 – 423, 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1018787815779

[26] H. Wietgrefe, K.-D. Tuchs, K. Jobmann, G. Carls, P. Fröhlich, W. Nejdl, and S. Steinfeld, “Using neural networks for alarm correlation in cellular phone networks,” in *International Workshop on Applications of Neural Networks to Telecommunications (IWANNT)*, 1997.

[27] TM Forum, “GB922 Information Framework (SID),” 2005, http://www.tmforum.org.

[28] F. Caruso, D. Milham, and S. Orobec, “Emerging industry standard for managing next generation transport networks: TMF MTOSI,” in *Network Operations and Management Symposium (NOMS 2006). IEEE/IFIP*, Apr. 2006, pp. 1 – 15.

[29] DMTF, “CIM Specification v2.15.0,” 2007.

[30] W3C, “Service modeling language,” http://www.w3.org/TR/sml/, May 2008.

[31] M. Burgess, “A tiny overview of cfengine: Convergent maintenance agent,” in *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO*. Citeseer, 2005.

[32] L. Kanies, “Puppet: Next-generation configuration management.” *login: the USENIX Association newsletter*, vol. 1, no. 31, 2006.