INTRODUCING TACL — A PROPOSAL FOR A NEW STANDARD T&E CONSTRAINT LANGUAGE

Jakub Moskal[1], Austin Whittington[2], Mitch Kokar[1], and Ben Abbott[2]

[1]VIStology, Inc., Framingham, MA
[2]Southwest Research Institute, San Antonio, TX

ABSTRACT

It is expected that XML-based languages for configuring telemetry systems like MDL and TMATS will eventually replace their non-XML predecessors. However, despite its numerous benefits, XML does not solve all the related problems. In particular, it cannot harness the complexity of constraints that may pertain to vendor hardware or to express system-level constraints that span across entire networks of devices.

In this paper, we present TACL, a T&E extension to W3C Shape Constraints Language (SHACL) for formulating constraints on configurations represented in MDL and TMATS, independently of any configuration software. TACL introduces high-level components that help to form constraints close to the user's intent and are less concerned with the low-level syntax details. It exhibits much better resilience to changes in the XML schemas than the languages that refer directly to the XML trees. A proof of concept TACL engine has been successfully developed and applied to MDL/TACL configurations.

INTRODUCTION

Configuring multi-vendor T&E systems today poses a number of challenges that cannot be fully addressed without proprietary solutions that are often too expensive to develop. In order to build a proper multi-vendor configuration, it must be validated with respect to RCC standards, vendor constraints pertaining to each of the devices that comprise it, and to system-level constraints that may be imposed by the user. Because there is no standard language to express all of these constraints, proprietary solutions are developed independently by each vendor. Thus, to build a configuration with devices developed by n vendors, the engineers must use n configuration tools to validate each device setup. In addition, they must leverage additional software to validate system-level constraints that do not pertain to any specific device, for instance, to limit the maximum weight or power usage of the entire configuration (c.f. left side of Figure 1). When a system-level constraint is violated, the engineer must make appropriate changes and run the validation with each vendor's software. This could result in a lengthy process that is not suitable for last-minute changes that are often required before a test is conducted. Each software suite used in the process must be updated to keep up with

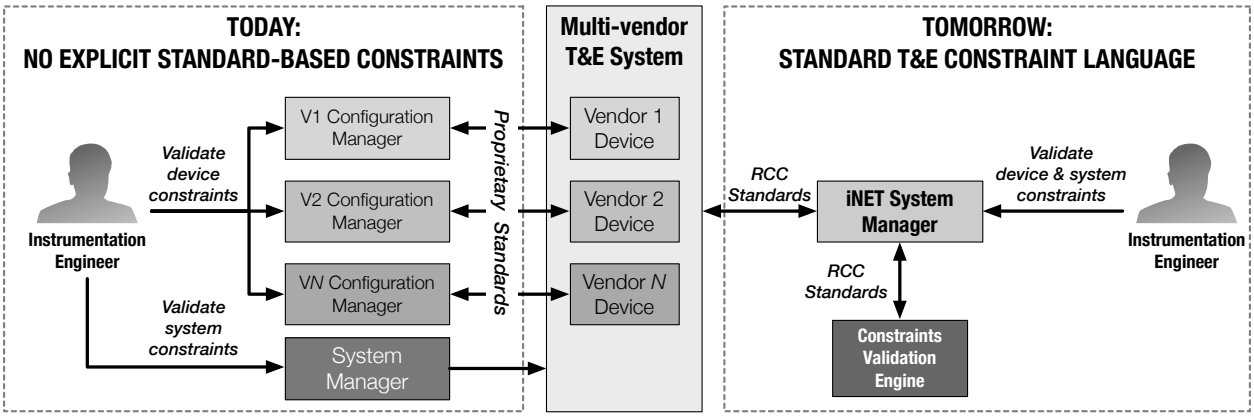

Figure 1: Configuring multi-vendor T&E systems today and tomorrow.

the changes in the RCC standards (and maintained, if built in-house), and requires separate training for new engineers, thus further increasing the overall engineering time and effectively, the cost.

In practice, multi-vendor T&E systems are avoided both in the military and the commercial domain, and instead, users stick with a single vendor to provide all the hardware, allowing them to use a single software package to build the configuration. While this allows building multi-device configurations, it comes with caveats. In particular, the user is locked-in with the vendor and cannot use competitive devices, even if they offer better performance, price, or provide a new functionality. The main obstacle to effectively building multi-vendor systems is the lack of a standard language to express constraints. If such a language was present, the instrumentation engineers could rely on a single piece of software that interacts with cross-vendor devices using RCC standards and validates all types of constraints at once (c.f. right side of Figure 1). Under such paradigm, the users could easily build cross-vendor configurations, and the vendors could attract the customers that were previously reluctant to use their hardware as it meant increased cost. Both the configuration management software and the validation engine could be implemented by third-party vendors, further facilitating a competitive market that significantly benefits the end-user (the instrumentation and test engineers) in terms of improved performance, engineering time, and cost. Most notably, upon a release of a new version of an RCC standard, such as MDL [1] or TMATS XML [2] schema, it would be in the interest of these third-party vendors to keep their software up to date, removing this burden from the users, as it is the case today. In addition, having all constraints in one place opens up opportunities for new functionality that was not even possible before, e.g. development of T&E constraint satisfaction problem solvers, tools that could automatically suggest configurations given a set of measurement requirements specified by the test engineers.

In summary, a standard T&E constraint language would have a positive impact on all stakeholders by providing the following benefits:

- Decreased engineering time spent on developing a configuration (fewer iterations) by combined validation of all constraints
- Decreased software development cost via access to third-party tools that implement the standard

- Decreased acquisition cost by open access to a competitive market of alternative hardware solutions
- Decreased training time by training engineers with a single management software as opposed to a number of vendor-specific tools
- Increased chance of mission success by supporting last-minute, or even in-test modifications via automated system-level validation
- Significantly reduced engineering time by assisting the engineers with a (semi-)automated deconfliction and optimization

While the T&E community largely agrees there is a need for a constraint language, no formal document has been published to date to formally specify the requirements for such language. There is an ongoing RCC-TG task with the purpose of collecting such requirements from various US military bases, although it will not address the vendor constraints, and undoubtedly be incomplete. For any constraint language candidate to become a successful standard, it must address all of the stakeholders' constraints, otherwise it will merely serve a subset of use cases and will not be adopted by the wide community. Being fully aware of this challenge, VIStology and SwRI have been encouraging various stakeholders to participate in the process of requirements elicitation, with a rather limited success so far. Although the language proposed in this paper, TACL, targets MDL and TMATS XML data, the general approach could as well be applied to data expressed in other data exchange standards that are based on XML, JSON, RDF [3], or any other format that is well-defined and could be automatically converted into a semantic data model. The following lists the requirements for a T&E constraint language that our team has developed at the beginning of the work on TACL:

- The language should support expressing all stakeholders' constraints in a declarative manner
- All MDL and TMATS concepts must be expressible in the language
- It shall relate to the XML models of MDL and TMATS so that it is intuitive to the engineers already familiar with these standards
- It shall facilitate appropriate level of abstraction, i.e. be intuitive enough for the engineers to use without requiring them to learn too much about a new technology
- It shall not be tightly-coupled with a particular version of the XML schema and exhibit resilience to future changes in the schema
- It shall not rely on XML schema-specific constructs, such as ID/IDREF, which have no counterpart in the conceptual level of the MDL/TMATS data
- It shall be extensible, i.e. allow adding new concepts without the need for modification

TACL meets the requirements described above and is defined as an extension to the Shapes Constraint Language (SHACL) that became an official W3C standard "for validating RDF graphs against a set of conditions" [4] as of July 20, 2017. SHACL is domain-agnostic and exhibits a lot of features that are desired in the T&E domain. Please note that the team had reviewed several other languages as potential candidates before settling on SHACL, but due to space limitation, the respective discussion is not included in this paper.

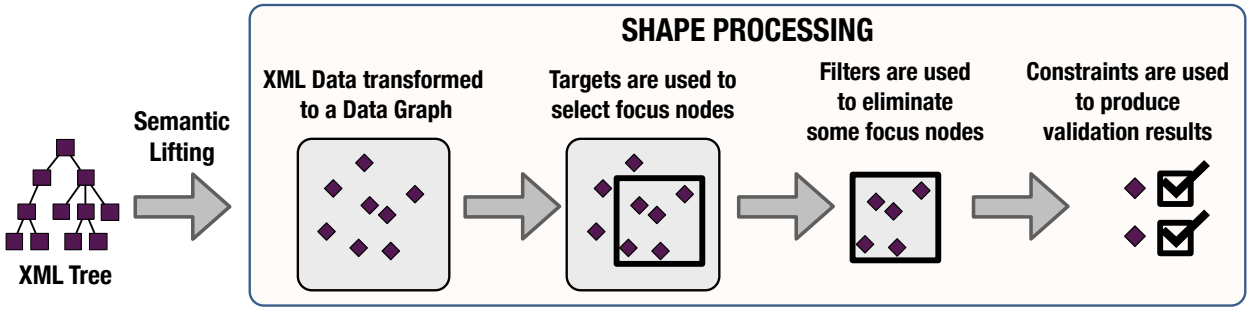

Figure 2: Using Shapes to validate XML data, based on [9]

SHACL

SHACL is a product of a long effort led by a community of various stakeholders, which faced a similar problem to the T&E community: lack of a standard constraint language to validate data originating at disparate sources. While in the T&E domain the primary concern is the MDL XML data, SHACL targets RDF documents, widely used to represent Linked Data [5] on the Web. Beyond validation, SHACL is also intended to support other use cases such as building UI, generating validation code and data integration. The following key features lead our team to development of TACL:

- SHACL is an official W3C standard and will soon be followed by an ecosystem of software tools that support it. In fact, an open source SHACL API [6] is already available for Java applications.
- All XSD constraints can be expressed in SHACL, but many SHACL constraints cannot be expressed in XSD because the latter is a lot less expressive.
- It can be integrated with OWL [7] ontologies and utilize the power of OWL inference to introduce a high-level of abstraction in constraint definitions.

SHACL was specifically developed to validate RDF graph data and as such it cannot be directly used to validate XML data. Fortunately, any XML tree can be represented as an RDF graph through the process called *semantic lifting* [8], and then be subject to standard SHACL processing. There are two kinds of SHACL shapes: node and property. The node shapes specify constraints on a class of nodes or a specific node, and for instance, can be used to enforce a naming convention on the resources in the data graph. The property shapes specify constraints that need to be met with respect to nodes that can be reached from a focus node by following a property path, e.g. *The power needs of a DAU cannot exceed its power sourcing capacity*. The data flow presented in Figure 2 illustrates how a single SHACL constraint is processed against an XML input:

1. XML instance data is lifted to an equivalent RDF data graph
2. Based on the targets specified in the shape, initial focus nodes are selected
3. Filters defined in the shape eliminate those nodes that meet the constraints
4. Remaining (invalid) nodes, if any, are described in the validation result as an output

Shapes can target any of the following: a concrete node in the graph (e.g. Device1), a class of nodes (e.g. all thermocouples), subjects of a specific triple (e.g. all devices that have a power

sourcing capacity), objects of a specific triple (e.g. all measurements with specific attribute), or nodes selected by a SPARQL [10] expression (e.g. all devices that weigh more than 10 lbs.). Note that the target class of nodes may be a reference to a class defined outside the input data graph, in an OWL ontology, and as such may leverage the inference capability of OWL to also target any node that is *inferred* as a member of the class (e.g. Thermocouple class, where the ontology provides its logical definition akin to "Any device that is capable of measuring temperature").

To address the wide range of use cases developed by SHACL community without defining a large standard that is hard to implement, SHACL Working Group defined the following:

1. SHACL Core — a set of common constraint components (constraint types) that every SHACL-compliant engine must support: *and, less-than-or-equals, or, unique-languages, property, qualified-min-count, max-length, min-length, disjoint, qualified-max-count, min-exclusive, node, exactly-one, SPARQL, not, in, max-count, max-inclusive, closed, min-inclusive, min-count, datatype, max-exclusive, class, has-value, language-in, pattern, equals, less-than, node-kind, derived values*. These types of constraints are very similar to those found in XSD.
2. SHACL Extensions — mechanisms that customize the standard to specific user needs. The most notable mechanism is the notion of SHACL Constraint Components, which represent user-defined, parameterized constraint types that can be reused in different contexts. Users can define domain-specific components and effectively create a high-level abstraction vocabulary that is close to their domain and processable by standard SHACL processors. Components are declared using standard SHACL RDF statements and the business logic of the associated validation can be defined in any of the extension languages. The standard supports SPARQL and JavaScript, but other languages could be used as well.

To better illustrate SHACL, a few examples are given below, with increasing complexity:

```
# The weight of every device must be 40.
:Shape1
  sh:targetClass :Device;
  sh:property [sh:path :hasWeight; sh:hasValue 40; ].
```

```
# The value of volume of every device must be less than the value of its weight.
:Shape2
  sh:targetClass :Device;
  sh:property [sh:path :hasVolume; sh:lessThan :hasWeight; ].
```

Note that :Shape2 involves relating the values of two different properties, which cannot be expressed in a language like XML Schema.

```
# Each console must have at least 1 FPGA.
:Shape3
  sh:targetClass :Console;
  sh:property [sh:path :hasPart ; sh:class :FPGA ; sh:minCount 1; ].
```

While the basic constraint components can cover a lot of the basic constraints found in various data exchange standards, it is the advanced examples that really demonstrate the power of SHACL. These examples leverage SHACL's more advanced features, such as SHACL-SPARQL, which allows using SPARQL expressions to define the constraint. SPARQL is to RDF what SQL is to relational data, and gives the constraint author a very rich language to inspect the data.

```
# The total weight of all devices in a radar sensor must not exceed 200.
:Shape4
  sh:targetClass :RadarSensor;
  sh:sparql [
    sh:message "The total weight of {$this}({?value}) exceeds the maximum of 200.";
    sh:select """
      SELECT $this (sum(?weight) AS ?value)
      WHERE { $this hasPart/hasWeight ?weight .}
      GROUP BY $this HAVING (?value > 200)"""] .
```

In the shape above, the message template property includes variables in the curly brackets. When the particular constraint is violated, SHACL processor produces an error message based on this template using concrete values from the focus nodes that did not pass the validation. For instance, the shape could result in a message like "The total weight of RadarSensor15 (350) exceeds the maximum of 200".

The SPARQL expressions used inside SHACL shapes, while very powerful, may be hard to accept by the T&E community due to the fact that the new users would have to learn a new language to use them. This is where SHACL constraint components come into play — they can be used to encapsulate the SPARQL expressions and users can make use of them just as they do with the Core components. Because they can be parameterized, they can introduce a higher level of abstraction and be aligned closely to a specific domain. To illustrate this, consider the following constraint component developed as a generalization of :Shape4, shown above:

```
:Shape4Component
  a ConstraintComponent ;
  sh:parameter [sh:path ex:maxWeight ; sh:datatype xsd:integer ; ];
  sh:nodeValidator [
    sh:message "Total weight of {$this} ({?value}) exceeds the maximum of {$maxWeight}." ;
    sh:select """
      SELECT $this (sum(?weight) AS ?value)
      WHERE {$this :hasPart/:hasWeight ?weight .}
      GROUP BY $this HAVING (?value > $maxWeight)"""].
```

The SPARQL query inside the component looks almost exactly like the SPARQL query inside `:Shape4`, with the exception that the maximum weight is now parameterized. Also, note that the component does not specify any target class and can be applied to any target that fits the data pattern. With the component definition, the `:Shape4` constraint is now much simpler:

```
:Shape4usingComponent
  sh:targetClass :RadarSensor ; ex:maxWeight 20.
```

The new constraint is clearly much simpler and no longer requires using SPARQL. SPARQL is used inside the definition of the component, but the constraint author does not need to know anything about it. The same component can now be used in different contexts, for instance, it could be applied to a specific network of devices:

```
:Shape5 sh:targetNode :Network5 ; ex:maxWeight 15.
```

The component could be even more generalized by replacing `:hasWeight` with another parameter, allowing it to be used in any constraint that needs to quantify the value of any property within the target defined by the shape, e.g.:

```
:Shape5 sh:targetNode :Device4 ; ex:maxAttributeName :hasVolume ; ex:maxAttributeValue 25.
```

It can be seen how powerful is the component mechanism of SHACL and how extensible it is without requiring a custom software to process it. In addition to components, SHACL has other extension mechanisms (not covered in detail due to space limitations), most notably user-defined targets (e.g. all devices with a module from Vendor X) and user-defined functions (e.g. a custom algebraic operation for conversion between derived and base units of measurement).

TACL

Given the powerful extension mechanisms in SHACL, we are proposing T&E Extension for SHACL, or TACL, which is a set of extensions to SHACL Core, and consists of the following:

- *TACL Ontology* — semantic data model that imports MDL and TMATS ontologies, which reflect the T&E XML schemas and adds additional concepts and logical axioms to facilitate automatic inference and effectively a more powerful constraint language.
- *TACL Constraint Components* — set of commonly used T&E constraint types that refer to the ontological model. They will involve potentially very complex SPARQL expressions inside, which the end users will never have to see. Instead, they will simply use a TACL processor that already imports the component definitions and leverage a SHACL engine internally.
- *TACL Functions* — set of functions commonly used inside the constraints, components, or target definitions.

TACL bears a similar relationship to SHACL as MDL does to XML — a customization of a standard language to a specific domain that can be processed with the tools developed for the core language. Just like MDL or TMATS XML documents can be processed with domain-agnostic XML tools (editors, schema validators, query processors), TACL can be processed with those developed for SHACL. With the W3C behind the standard, SHACL user community is quickly growing, which is important for the relatively small domain of T&E. If the community decided to develop an entirely new language, it would take a much longer time before appropriate tools would be in place, and there would be a lot fewer resources (user groups, forums, etc.) to rely on.

Naturally, the community could use pure SHACL instead of standardizing a set of extensions, but then it would result in a lot of redundancy for the commonly used constraints, and force the engineers to learn the more complex SPARQL expressions. The benefits of pursing TACL as the standard are manifold:

- Easier constraint authoring and maintenance due to a more succinct syntax and domain-specific terms, while still being compliant with SHACL tools. For instance, consider custom target class tacl:usesForeignModules to point to any device that contains a module that is manufactured by a foreign vendor.
- A looser coupling with the MDL schema (resilience to updates) — if the domain model changes, the TACL component definitions may need to be updated, but the individual constraints are likely to be entirely unaffected. Consider the following scenario as an example:
    1. A TACL component (e.g. tacl:maxPowerNeeds) is developed against a variety of vendor-specific generic parameters to facilitate development of constraints on maximum power

needs (not currently supported by MDL).
2. The generic parameters are eventually standardized and become first-class citizens in the MDL schema.
3. The component is reimplemented to reflect the changes, but the constraints that used the component do not need to change at all.

- Improved time required to author constraints due to the reusability of TACL components in different contexts (c.f. :Shape5 above).

The following are some examples of constraints that could be developed with TACL. Note that TACL is just a proposal and that it requires more involvement from the wider community before it can be subject to standardization. The primary purpose of this work is to convince the community of its value and to show how this approach would benefit all stakeholders.

```
# Devices with 3-5 transducers must have at least two power sources
:TaclShape1
  sh:target [a tacl:hasTransducers ; tacl:min 3; tacl:max 5]; tacl:hasPowerSourceCount 2.
```

```
# DAUs from VendorX can be mapped to at most 5 measurements and have a maximum
    power need of 20
:TaclShape2
  sh:targetClass mdl:DAU ; sh:property [sh:path mdl:manufacturedBy "VendorX" ];
  tacl:hasMeasurmentsMax 5; tacl:hasMaximumPowerNeed 20.
```

Given that MDL and TMATS XML are both XML languages, they can be subject to processing with other standard languages that refer to the XML data model, such as XPath or XQuery (which uses XPath internally). These languages are a lot more expressive than XML Schema and could be considered as a basis for a T&E constraint language, however, they are not meant to serve that purpose, which leads to undesired consequences:

- Because XPath works directly on the XML tree structure, any change in the schema must be followed by changes in the respective XPath expressions. In large projects this could easily amount to a major task.
- Because XPath does not provide extension mechanisms, it results in a lot of redundancy.
- Because XPath does not have a notion of parameterized components, the constraints must use syntax that is far from the language of the domain.
- Because XPath is not a constraint language, it requires coupling with non-XPath technologies such as XForms to actually provide the constraint validation.

For a comparison, consider two examples of constraints, encoded both in XPath/XForms and in TACL.

Constraint 1: *"DAU1" device sample rates are* 2^0 *to* 2^{10}*, and only powers of 2*

XPath:

```
<!-- Scope -->
//mdl:Measurement[@ID = //mdl:PortMapping[mdl:PortRef/@IDREF = //mdl:NetworkNode[mdl:Name
    eq "DAU1"]//mdl:Port/@ID]//mdl:MeasurementRef/@IDREF]/mdl:DataAttributes/mdl:
    DigitalAttributes/mdl:SampleRate

<!-- Constraint -->
mdl:ConditionParameter/mdl:ConditionValue = ('1', '2', '4', '8', '16', '32', '64', '128', '
    256', '512', '1024')
```

TACL:

```
:TaclShape3
  sh:target [a tacl:MeasurementOfNetworkNode ; tacl:name "DAU1" ];
  sh:property [
    sh:path (mdl:hasSampleRate mdl:hasConditionValue ) ;
    sh:in ("1" "2" "4" "8" "16" "32" "64" "128" "256" "512" "1024"); ];.
```

Constraint 2: *Uncertainty of "DAU1" measurements must be in range between .03 ms to 80 ms*

XPath:

```
<!-- Scope -->
//mdl:Measurement[@ID = //mdl:PortMapping[mdl:PortRef/@IDREF = //mdl:NetworkNode[mdl:Name
    eq "DAU1"]//mdl:Port/@ID]//mdl:MeasurementRef/@IDREF]/mdl:DataAttributes/mdl:
    Uncertainties/mdl:Uncertainty

<!-- Constraint -->
 (num(mdl:ConfidenceInterval/mdl:ConditionParameter/mdl:ConditionValue) ge . 3e-5)
 and (num(mdl:ConfidenceInterval/mdl:ConditionParameter/mdl:ConditionValue) le 0.08)
```

TACL:

```
:TaclShape4
  sh:target [a tacl:MeasurementOfNetworkNode ; tacl:name "DAU1" ];
  sh:property [
    sh:path (mdl:hasDataAttributes mdl:hasUncertainty mdl:hasConfidenceInterval mdl:
        hasConditionValue) ;
    sh:minInclusive ". 3e-5" ;
    sh:maxInclusive "0.08" ; ];.
```

Note the intricate scope expression in both XPath constraints — it traverses the XML tree to find all measurements that are bound to a device with a specific name. Not only it cannot be reused and must be retyped in each similar constraint, it is tightly coupled with the schema and all such expressions would have to be updated if the related schema changed. By contrast, in TACL constraints, this lookup is replaced by a single component tacl:MeasurementOfNetworkNode that hides all the details. Schema changes would not affect these constraints at all, only the component would have to be recoded. Furthermore, using ontological axioms, the paths can be much shortened, e.g. in the first constraint, the mdl:hasDataAttributes/mdl:hasDigitalAttribtues/mdl:hasSampleRate could be defined as an OWL property chain that effectively adds a shortcut in the graph that relates the measurement directly with the sample rate value. As a result, the constraint expression is much more succinct.

All of the constraints described in this paper were implemented and tested against input XML data, including the MDL examples that are bundled with the schema. The tests were performed with xVISor, VIStology's TACL constraints validation engine, which includes a semantic lifter that automatically converts the XML input to RDF/OWL.

CONCLUSIONS

VIStology and SwRI are actively collaborating to develop more candidates for TACL components and functions and to extend the TACL ontology with richer semantics. It is a work in progress that

we hope the community will join and eventually adopt. Once TACL is standardized, cross-vendor configurations will be much easier to build and it will pave the way to new and more advanced use cases, such as semi-automatic generation of configurations given the measurement requirements and a set of constraints pertaining to all available devices, and those imposed by the user.

While TACL syntax is reasonably succinct and easy to read and write, it may still pose a challenge to those engineers that never worked with any semantic standard. For this reason, our team will explore the possibility of generating basic TACL constraints based on the input from the user given via graphical user interface or through predetermined templates. Although it is plausible that a controlled natural language could be used to author some constraints and then to generate TACL from it, it is unlikely that it could ever be expressive enough to cover all use cases. Consequently, we are proposing that only TACL is standardized, allowing vendors to develop their own user-friendly interfaces to interact with their users, but to require that TACL is used in any exchange crossing their systems' boundaries.

Acknowledgements: This work was performed under Air Force contract FA9302-16-C-0017 "Rule-Based XML Validation for T&E (RuBX)". Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Air Force.

REFERENCES

[1] M. S. Moore, J. C. Price, A. R. Cormier, and W. A. Malatesta, "Metadata description language: The inet metadata standard language," in *International Telemetering Conference Proceedings*, International Foundation for Telemetering, 2009.

[2] B. Downing, "An xml vocabulary for tmats," in *International Telemetering Conference Proceedings*, International Foundation for Telemetering, 2000.

[3] R. W. Group, *Resource Description Format (RDF) Primer*. W3C Recommendation, February 10, 2004. Available at http://www.w3.org/TR/rdf-primer/.

[4] H. Knublauch and A. Ryman, "Shapes constraint language (SHACL)," *W3C Candidate Recommendation*, vol. 11, p. 8, 2017.

[5] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International journal on semantic web and information systems*, vol. 5, no. 3, pp. 1–22, 2009.

[6] TopBraid, "TopBraid SHACL API." https://github.com/TopQuadrant/shacl, April 2018.

[7] W3C, "Owl 2 web ontology language document overview," 2009. Retrieved from http://www.w3.org/TR/owl2-overview/.

[8] J. Moskal, M. Kokar, and J. Morgan, "Semantic validation of T&E XML data," in *International Telemetering Conference Proceedings*, International Foundation for Telemetering, 2015.

[9] R. Cyganiak, "SHACL: Shaping the big ball of data mud." https://www.slideshare.net/cygri/shacl-shaping-the-big-ball-of-data-mud, November 2016.

[10] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," in *The Semantic Web-ISWC 2006*, pp. 30–43, Springer, 2006.