RAPIDLY RECONFIGURABLE SYSTEM MANAGEMENT

Patrick J. Noonan Jr., Austin J. Whittington,
Hakima Ibaroudene, Myron L. Moodie

Southwest Research Institute®
San Antonio, Texas
patrick.noonan@swri.org, austin.whittington@swri.org,
hakima.ibaroudene@swri.org, myron.moodie@swri.org

ABSTRACT

The growth of network and distributed technologies in flight test instrumentation (FTI) has provided the benefits of flexibility, scalability, and compatibility with prevalent computing capabilities. However, to achieve these capabilities, the complexity of each piece of FTI and the overall system has increased dramatically. Even with systems composed of equipment from a single vendor, it is important to have management systems that provide the flexibility to adapt quickly to various system configurations and present unified information to the flight test users. The growth of network technologies and then standardized approaches such as iNET standards becoming accepted IRIG 106 standards is leading to the growth of multi-vendor systems. These multi-vendor systems further increase the need for rapidly reconfigurable management systems. This paper describes a constraints engine we have developed to enable flexible system management systems and reflects on how these techniques have been used successfully in the iNET System Manager.

KEYWORDS

Flight Test Instrumentation, Constraints, Rete, MDL, XML

INTRODUCTION

The growth of network and distributed technologies in flight test instrumentation (FTI) has provided the benefits of flexibility, scalability, and compatibility with prevalent computing capabilities. However, to achieve these capabilities, the complexity of each piece of FTI and the overall system has increased dramatically. Even with systems composed of equipment from a single vendor, it is important to have management systems that provide the flexibility to adapt quickly to various system configurations and present unified information to the flight test users.

Network-based telemetry systems have unprecedented amounts of flexibility due to the ability to manipulate configuration during a test. This flexibility, along with ever increasing data demands, imposes requirements not only on the devices acquiring, recording, and processing the data, but also on the underlying network transporting that data. Furthermore, the amount of spectrum available for tests is ever decreasing, so that optimal utilization of spectrum is required. This leads to complexities involved in managing and configuring the devices and the underlying network, which need to be carefully handled. In addition, the need to conduct multiple concurrent tests, remotely manage the FTI on a Test Article (TA), reconfigure devices in the middle of a test, reconfigure the test system for a different test, and dynamically modify certain parameters of the test system to maximize test efficiency are all now requirements of system management.

The flexibility and complexity of dynamic status and control is increased further when using a bidirectional communication link with the TA. This increase in flexibility creates challenges for test operations. For instance, a user might need to re-route data on the TA for a given test. To ensure proper configuration and performance, a network topology and loading prediction may be needed in order to forecast the potential impacts of the re-configuration, not just within the TA, but across the systems that transport data to the ground stations. The growth of network technologies and then standardized approaches such as Integrated Network Enhanced Telemetry (iNET) standards becoming accepted Inter-Range Instrumentation Group (IRIG) 106 standards has made these scenarios a reality. This flexibility and complexity, along with the growth of multi-vendor systems, has only increased the need for rapidly reconfigurable management systems.

Highly specialized applications, or management systems, are needed to fully manage network-based telemetry systems and are central in maintaining the system working seamlessly, reliably, and effectively. A key aspect of this is the speed and ease with which FTI and system level configurations can be built and deployed. This paper describes a constraints engine we have developed to enable flexible system management systems and reflects on how these techniques have been used successfully in the iNET System Manager.

CONFIGURATION VALIDATION

Constraints

Constraints are rules that state or control what something can or is allowed to do. These can range from how much weight someone can pick up without hurting themselves, to the tensile strength of a material, to the top speed of a car. In FTI, constraints generally describe the capabilities and limitations of devices and networks. A data acquisition device may have an upper limit on its sample rate, or a network switch can only handle 100 megabits per second from any individual data source.

Correct-by-Construction

In the practical case, these constraints are used as validation rules. A system integrates the vendor constraints that describe the devices, system constraints that describe the capabilities of all devices that participate in the system, and the user constraints that standardize or simplify device configuration [1]. User input is checked as they create a configuration, or at the very least in a verification action that can be triggered, and inputs are evaluated against the set of interwoven constraints, which point out any problems with the current set of values. As the user develops a test, the constraints ensure that the configuration will be valid, both for the individual devices and within the larger system.

Constraints Languages

Many of the existing flight test telemetry configuration languages use an XML format, including MDL, XidML, and TMATS. As such, XPath, the W3C query language for selecting nodes from XML documents, was a natural choice to describe the constraints on those files. This language, in conjunction with XForms, another XML-based technology made for gathering and processing XML data, was used by the Boeing Company's Modular Instrumentation Setup Tool (MIST), an implementation which uses constraints in the manner described above to aid its users in creating configuration using the correct-by-construction techniques [2].

As we have continued to explore the application of constraints to FTI, a more powerful constraints language, W3C's Rule Interchange Format (RIF) [3], has been identified as a logical successor to XPath. RIF was developed for expressing rules which computers can execute, which indeed matches this use case for constraints. RIF's main advantage over XPath is its lack of immediate mapping to XML. While XPath's direct mapping makes it easy to create constraints, it does require deep knowledge of the XML schema being constrained, and is then targeted to a specific version of that schema. Depending on the magnitude of changes between versions, this may be a minor issue, or it could require whole rewrites of the constraints. RIF can be indirect, with an individual constraint referring to nouns defined elsewhere. An XPath constraint has to define a specific instance, such as the path to a measurement in MDL:

```
MDLRoot/MeasurementDomains/MeasurementDomain/Measurements/Measurement
```

A RIF path could simply refer to a Measurement and have the particulars of what that means defined in a separate file matching the schema version currently in use. This means that only the mappings from these nouns to their specific instances would need to be updated between versions (or even sufficiently compatible schemas), instead of the XPath statements, which could contain complex logic that is hard to translate while maintaining certainty that the intent of the constraint is being preserved.

This also means that RIF constraints can refer to concepts and objects more abstractly, or be as specific as necessary. A RIF rule can reference a Transducer, which is covered by the broader term Device in MDL, and again a second rule can define what that term resolves to. Those developers or users creating new rules, given a sufficient set of prebuilt nouns, can use terms

which are more familiar to them to describe the constraints, which are then independent of any particular schema but for the mapping rules.

However, a switch to RIF is not incompatible with the XPath-based approach for a particular XML-based solution. Because the references in a RIF statement will eventually reduce down to XML entities, given that it is describing the constraints on XML configuration files, it is possible to create an XPath expression (or expressions, in cases where a rule may be too complex to capture with only one) that describe the same constraint for a particular schema. Thus, a set of RIF constraints can be used with an existing XPath system, after this conversion process takes place.

Research Areas

There are several active research areas seeking to improve these technologies. Constraint satisfaction technologies can be used to automatically build configurations, given a set of goals and the constraints that a configuration must be valid within [4]. Users could give the system a desired measurement list, and such a satisfaction system would come back with a set of possibilities, each of which acquires all of the specified measurements with varying combinations of devices or parameters. The user could then refine a search, eliminating devices they don't want to consider or optimizing for other factors such as weight and space, knowing that any generated configuration will be valid within the system's bounds.

There is also the area of meta constraint validation, validating the constraints themselves to make sure their interrelation does not lead to a validation paradox [5]. For example, the case where a vendor specifies a finite impulse response (FIR) filter as the only option for a particular acquisition technique, and a user constraint gives the rule that all filters must be infinite impulse response (IIR). In this case, if a user set up a measurement using that technique, there are no valid filters usable, which is a failure that could be detected by this higher-level validation.

CONSTRAINTS ENGINE

In previous systems, XForms has been used for the validation technology, evaluating the constraints and updating its object model as the user inputs new data. However, XForms provides a lot of presentation layer functionality that is not necessary in a tool that already has its own database and user interface (UI) frontend. Thus, these types of applications need a more specialized tool, a constraints engine that can work with existing objects and simply provide running validation of configurations to the rest of the system, rather than doing all the processing, storing, and rendering that a full XForms system uses.

This gives the blueprint of what the constraints engine does. It monitors a set of objects that reflect the current state of the database, and on a change, it evaluates the new information and determines whether any constraints have failed. In the case that they did, the constraints engine sends a report on each object that failed and the error message to show, and the UI decides how

to display the message. By dropping the requirements for any rendering or other presentation layer, the focus becomes performance and efficiency.

The main processing idea behind the constraints engine is the usage of the Rete algorithm [6], a standard pattern matching algorithm for implementing production rule systems, through an existing rules engine. A high-level illustration of this algorithm is shown in Figure 1 [7]. In general, the Rete algorithm matches facts against production rules. The production rules are "if/then" statements, such that if an initial condition is matched, the "then" action is fired. The Rete algorithm is generally used for business rules, such as "when a customer hits 10,000 airline miles, award them special status", but we will show how this algorithm can be used for our constraint evaluation use case.

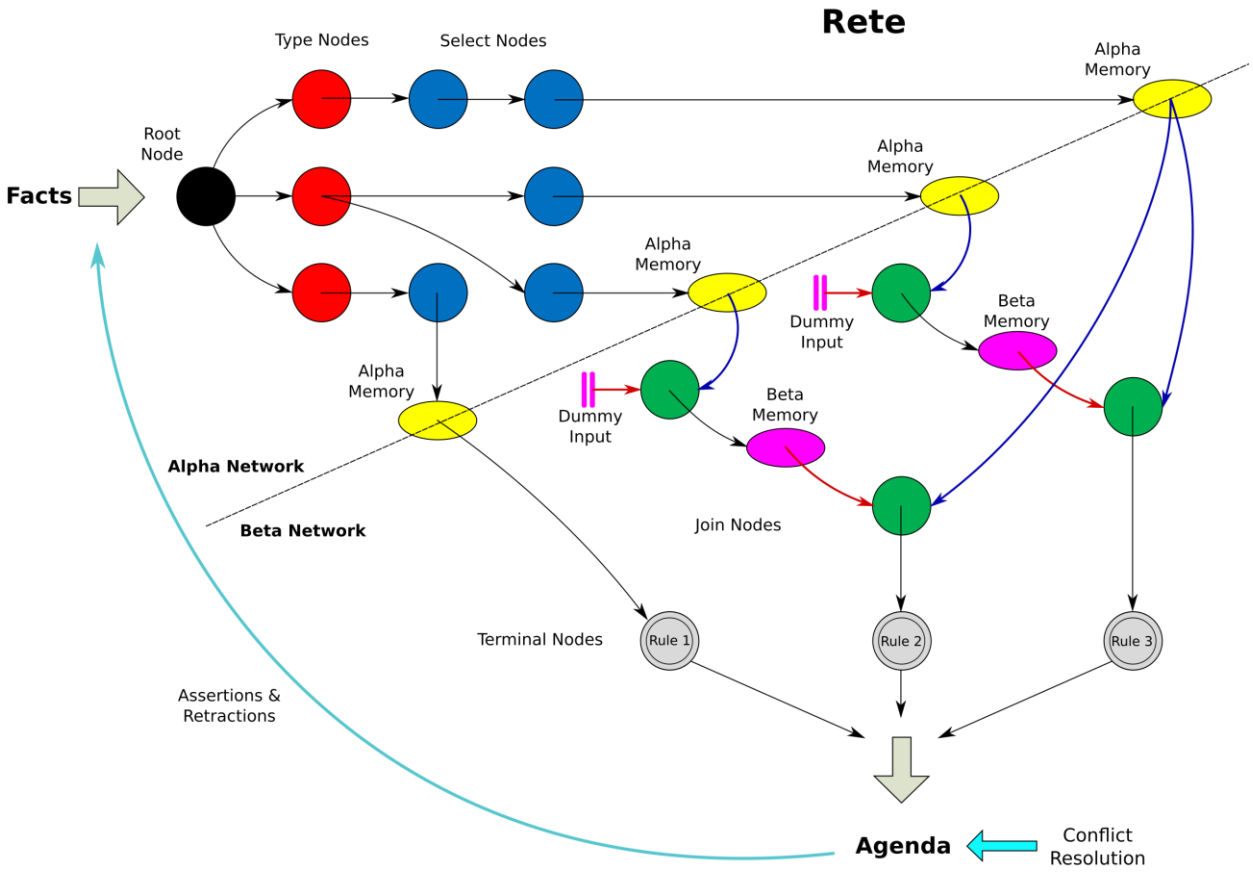

Figure 1 – Rete Algorithm

Before any useful evaluation can take place, we first convert our RIF constraints into Rete production rules, where the antecedent (the "if") is the failure case of the constraint, and the consequent (the "then") is the resulting error message and the triggering element. In this way, any rule which matches the conditions of failure will return the information necessary for the UI to display what went wrong to the user.

As the user builds up a configuration, the new facts (stored in the objects) are added to the Rete network. The Alpha Network generally categorizes facts based on simple conditional tests, and files facts into alpha memory as conditional chains are satisfied. For example, in FTI, such a chain could be "is this a piece of hardware", then "is this network-connected", and "is it made by vendor A", eventually categorizing as a Vendor A DAU. When a new fact is added, or an existing fact is modified, other facts do not have to be reevaluated because they are already stored in the network. The Beta Network joins between different memory elements. In our case, this applies after nodes have been categorized, and will be where many rules are evaluated, as it is very common that a constraint depends on more than one part of a system. For example, such a rule could be "on a Vendor A DAU, each measurement has an upper sample rate bound of 10000 Hz." Then, within that beta node, a list of measurements could be joined with the list of DAUs to determine which measurements need to satisfy this check. Finally, the rules are evaluated, and if such a check was to fail, the user would receive the error information. Note that the particular facts and commonality between alpha and beta nodes will be done programmatically by the rules engine implementing the Rete network, and so will not have to be explicitly captured by hand.

In this application, we do not currently need to make use of any conflict resolution, assertion, or retraction functionality. The Rete algorithm is only being used to validate the user's input and return errors. Thus, the constraints only provide errors, and do not cause other facts to be asserted or retracted, and because no facts are changing, conflict resolution is unnecessary, with no ordering dependency on which an error message is revealed, as the UI will display them all regardless.

SYSTEM MANAGEMENT

Rapidly reconfigurable system management is being realized by the System Manager application development for the iNET program, which provides an initial user application for describing, configuring, monitoring, controlling, coordinating, and visualizing the operations of a Telemetry Network Standards (TmNS)-based FTI system [8]. A key aspect of System Manager is the ability to describe flight test configurations and generate MDL configuration files used to configure FTI. The flexibility provided by MDL allows almost everything in a flight test system to be described, from low-level sensors to complex RF networks. However, with this flexibility comes complexity, and that is where the use of constraints can greatly simplify the user's choices.

The Southwest Research Institute (SwRI®) developed system management design is based around the principles of extensibility, modularity, and separation of concerns. Because of these principles, system management can occur with various levels of constraints or even none at all. The modular constraints engine described above can be seamlessly integrated with existing objects backed by a database, and the separation of the RIF constraints from code allows for user experience updates at any level of the constraints hierarchy (FTI, system, user) without the need for rebuilding the application. By taking advantage of these techniques, System Manager is itself rapidly reconfigurable, that is it can be quickly adapted to new or modified FTI, system, and/or user level requirements.

The result of rapidly reconfigurable system management is the ability to use constraints evaluation to alert the user of errors in the configuration and provide meaningful error messages so that the errors can be quickly addressed. An example of this is shown in Figure 2, where the user is in the process of configuring a Link Manager that is managing 10 concurrent test missions with each test mission consisting of two RF links (an uplink and a downlink).

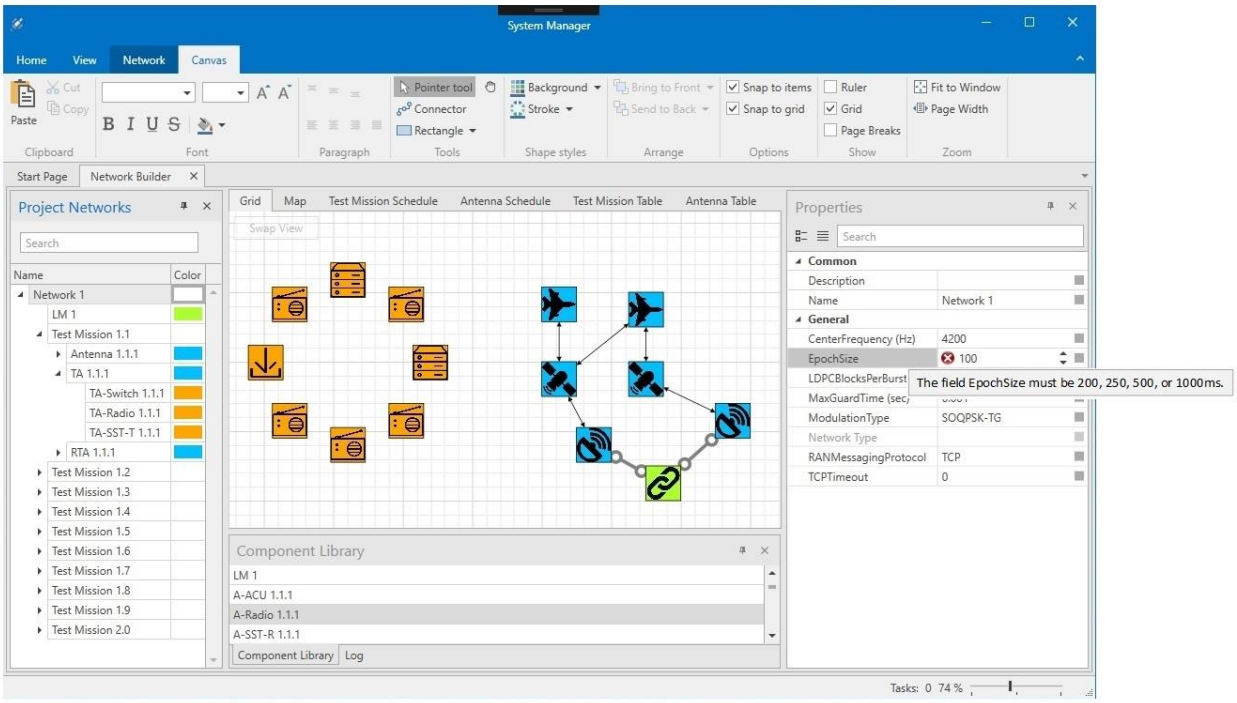

Figure 2 – Constraints Evaluation in System Manager

As shown in the example, the user has selected an epoch size of 100 ms which generates not only an error, but also an informative message stating that the epoch size must be either 200, 250, 500, or 1000 ms. This occurs because the constraints engine has evaluated a RIF rule stating that the Link Manager cannot operate correctly at an epoch size less than or equal to 100 ms if the guard bands are 1 ms and the burst size is 16 blocks per burst. Using this information, the user can quickly correct the error and continue describing the configuration. This small example shows how combining a familiar, user-friendly UI with the powerful constraints engine can lead the user to a configuration that is on its way to being correct-by-construction.

CONCLUSION

The rise of network technology in FTI and standardized approaches to system management have opened new capabilities and flexibility. Maximizing the benefit of these network systems requires equally flexible and rapidly reconfigurable management systems. Through our experience with a variety of commercial and military flight test programs we have developed a number of techniques that can be leveraged to build system management systems for the specific needs of a variety of flight test requirements. One of these applications, the iNET System

Manager, uses an extensible and modular design to incorporate a constraints engine that allows for rapid reconfiguration when FTI, system, and/or user level requirements change.

REFERENCES

[1] Noonan, P.J., H. Ibaroudene, A.J. Whittington, M.L. Moodie. “Simplifying Flight Test Configuration with Constraints.” Proceedings of the International Telemetering Conference, Glendale, Arizona, November 2016

[2] Neumann, M., J. Moore, S. Pantham, M.L. Moodie, P.J. Noonan, A.J. Whittington. “An Adaptable Constraints-based Metadata Description Language (MDL) System for Flight Test Instrumentation Configuration.” Proceedings of the European Telemetry and Test Conference, Nuremberg, Germany, May 2016

[3] https://www.w3.org/TR/rif-overview/

[4] Apt, Krzysztof. *Principles of constraint programming*. Cambridge University Press, 2003.

[5] Moskal, J., M. Kokar, J. Morgan. “Semantic Validation of T&E XML Data.” Proceedings of the International Telemetering Conference, Las Vegas, Nevada, October 2015

[6] Forgy, C.L. "Rete: A fast algorithm for the many pattern/many object pattern match problem." *Artificial intelligence* 19.1 (1982): 17-37.

[7] https://commons.wikimedia.org/w/index.php?curid=15901661

[8] Noonan, P.J., T.A. Newton, G.C. Willden, T.B. Grace, W.A Malatesta. “iNET System Manager.” Proceedings of the International Telemetering Conference, San Diego, California, October 2014