A MATHEMATICAL MODEL FOR INSTRUMENTATION CONFIGURATION

Charles H. Jones, PhD
Air Force Flight Test Center
Edwards Air Force Base, CA

ABSTRACT

This paper describes a model of how to configure settings on instrumentation. For any given instrument there may be 100s of settings that can be set to various values. However, randomly selecting values for each setting is not likely to produce a valid configuration. By "valid" we mean a set of setting values that can be implemented by each instrument. The valid configurations must satisfy a set of dependency rules between the settings and other constraints. The formalization provided allows for identification of different sets of configurations settings under control by different systems and organizations. Similarly, different rule sets are identified. A primary application of this model is in the context of a multi-vendor system especially when including vendors that maintain proprietary rules governing their systems. This thus leads to a discussion of an application user interface (API) between different systems with different rules and settings.

KEYWORDS

Instrumentation, Modeling, Multivendor Configuration, Constraint Satisfaction Problem

INTRODUCTION

As instrumentation systems have become more complex, so have the processes for configuring them. This is especially true in systems incorporating components from multiple vendors. With the advent of a true network backbone on a test vehicle (which is part of the network telemetry vision), multi-vendor systems will become more desirable and more common. From a user's perspective – both those configuring and controlling the system – it is desirable to have a single interface to all components on the system regardless of vendor origin. A stumbling block for such a single interface is that some vendors maintain proprietary rights to the rules used for determining valid configurations. Recognizing the existence of these proprietary rule sets, it becomes desirable to establish a common interface that allows third party software to interact with multiple vendors' software without requiring implementation of non-disclosure agreements.

By providing a formal mathematical model of instrumentation configuration, this paper provides a step towards understanding what the rule sets are and how to provide information exchange between systems implementing different rule sets. In addition to providing a single user interface to a multi-vendor system, this structure facilitates the ability to automate cross-vendor configuration verification, validation, and optimization. This ability becomes critical when considering this structure from the point of view of multiple systems vying for limited resources. As a simple example of conflict, consider setting up each vendor's subsystem independently. It is not difficult to imagine the aggregate bandwidth required exceeding that available on the network.

SETTINGS, CONFIGURATIONS, AND RULES

We start developing the model by defining the basic elements. The atomic unit of the model is a *setting*, s_i. A setting is something that can take on one or more values. These values may be discrete or continuous. A simple example is a gain setting that can take on values of 0, 5, or 10. Setting values are not necessarily numerical, for example, filter type. A single value (constant) setting might also be thought of more as a characteristic. For example, the total power available is simply what the power supply can provide rather than something that can be set to some value. We will be considering the set of all settings, $S = \{s_i\}_{i=1}^{n}$. That is, if a complete and systematic review of the control software were made to look at every dialogue box, *S* would be the catalog of every check box, pull down menu, numeric or text box entry, etc. That is, any item that can be set to some value. The set of *values* that a setting, s_i, can be set to is denoted by $V_i = \{s_{ij}\}_{j=1}^{n_i}$.

A *configuration* is thus some set of setting values, $C = \left\{s_{ij} \mid i = 1, \dots, n;\ j \in \{1, \dots, n_i\}\right\}$. (As a side note, consider that the number of possible configurations is the product $\prod_{i=1}^{n} n_i$.) We wish to be able to construct a *valid configuration*, C^V. In general, the assumption is that the set of valid configurations is a proper subset of all configurations.

Each setting may be dependent on the values taken on by other settings. That is, there is a *dependency rule* (or simply *rule*) for each s_i, denoted by $\mathcal{R}_i\left(\left\{s_{k_j} \mid j = 1, \dots, m_i; k_j \in \{1, \dots, n\}; k_j \neq i;\ m_i \leq n\right\}\right)$. Note that a dependency rule may return multiple values for a given set of inputs. That is, the rule may simply constrain the possible values of its related setting rather than identify one specific value.

Example

Let there be 4 settings (i.e., $n = 4$) so that $S = \{s_1, s_2, s_3, s_4\}$. Let the possible values of the settings be $V_1 = \{1,2,3,4\}, V_2 = \{11,12,13\}, V_3 = \{21,22\}$ and $V_4 = \{31,32\}$. This implies $n_1 = 4, n_2 = 3$, $n_3 = 2$ and $n_4 = 2$. (As a side note, this implies there are 48 possible configurations in C– not all necessarily valid.) An example configuration is $C = \{4,12,21,32\}$.

Let the valid values of s_1 be dependent on the values of s_2 and s_4. Then the rule for s_1 is denoted by $\mathcal{R}_1(s_2, s_4)$. That is, $i = 1$; $j = 1, 2$; $k_1 = 2$; $k_2 = 4$; and $m_1 = 2$.

An example rule might be $\mathcal{R}_1(11, 31) = \{1,3,4\}$. That is, when $s_2 = 11$ and $s_4 = 31$, then s_1 may take on the values 1, 3, or 4. It does not matter what s_3 is set to.

End Example

The set of dependency rules (or possibly just a subset of them) is what we are assuming to be proprietary. There is also the possibility that there are proprietary settings as well.

A setting may be a *derived setting*, $s_i^{'}$, with a corresponding dependency rule, $\mathcal{R}_i^{'}$. The distinction here is that the value of $s_i^{'}$, is uniquely determined by a specific set of inputs to $\mathcal{R}_i^{'}$. An example of a derived setting is aggregate gain which might be determined by a polynomial relation on primary and secondary gains. However, in addition to this polynomial relation, $\mathcal{R}_i^{'}$ might also contain a rule such as: the primary gain must be greater than the secondary gain.

The significance of derived settings can be seen when considering possible methods of developing a configuration. One approach might be to set the primary and secondary gains and see if you obtain the proper aggregate gain. Or, you can establish the desired aggregate gain and request appropriate settings for the primary and secondary gains.

This raises the question of what is the input to our algorithm for determining a valid configuration. Part of this question includes who is creating the input. An instrumentation engineer may very well want to set primary and secondary gains as part of a troubleshooting scenario. However, a test engineer is probably more interested in stating her requirements and having as much of the configuration set automatically as possible. Continuing with the gain example, you might consider aggregate gain to be the requirement in that you don't really care about the primary and secondary settings, only the end result. But a more realistic requirement is that the test engineer wants to measure temperature within a certain range, with a certain accuracy, and at a certain sample rate.

We thus introduce the set of *requirements*, $Q = \{q_i\}_{i=1}^{m}$. To emphasize some distinctions, we use $C_{q_i}^{VV}$ to indicate that the configuration satisfies the requirement q_i. That is, $C_{q_i}^{VV}$ is not only *valid*, but has been *verified* to meet this requirement. We extend this notation to C_Q^{VVC} to indicate that the configuration is *complete* and meets all requirements in Q. It is generally assumed that $C_Q^{VVC} \subset C_{q_i}^{VV} \subset C^V \subset C$. That is, as constraints are added, the set of configurations meeting those constraints is smaller than the entire set of possible configurations. It is possible that some of these inclusions are not proper, but highly unlikely in practice. To many test engineers, the ideal would be to provide Q and have an algorithm that returns C_Q^{VVC}.

INTERDEPENDENCIES CREATE CLOSED SETS OF SETTINGS

Let us look at the relation between settings and their interdependencies. If I change one setting, how does this affect other settings? How much of a ripple effect is there? It turns out there is a simple formal statement of this ripple effect.

Define $P_i = \left\{ s_{k_j} \mid j = 1, \dots, m_i;\ k_j \in \{1, \dots, n\}; k_j \neq i, m_i \leq n \right\}$ to be the *parameters* of the rule $\mathcal{R}_\mathrm{i}$. So, if a setting, s_i, is changed, the settings that are affected are $P_i^{-1} = \left\{ s_j \mid s_i \in P_j \right\}$. That is, if s_i is in the parameter set of another rule, $\mathcal{R}_\mathrm{j}$, then the setting, s_j, associated with that rule is potentially affected by changing s_i. (Note that a setting is not an element of its own parameter set, $s_i \notin P_i$, or there would be a circular dependency.)

Example Continued

Using the setup of the previous Example, we give an example of P_i^{-1}. From that example, $P_1 = \{s_2, s_4\}$. Let $P_2 = \{s_1, s_3, s_4\}$, $P_3 = \{s_2\}$, and $P_4 = \{s_1, s_2\}$. Then $P_1^{-1} = \{s_2, s_4\}$.

End Example

These definitions lead to an interesting and somewhat useful fact. The set of parameters of a rule for a given setting are *exactly* the set of parameters affected by changing that setting. Formally, we have:

Lemma: $P_i = P_i^{-1}$

Proof: It is true that $s_i \in P_j$ if and only if $s_j \in P_i$. That is, $s_i = x$ implies $s_j = y$ if and only if $s_j \neq y$ implies $s_i \neq x$.

A corollary of this Lemma is that settings form closed sets of interdependent settings. A simple example of this might be that the settings for different devices or for different vendor's systems form sets which do not affect other devices or systems. However, settings within a particular device might also be broken up into closed sets. For example, some of the settings for a given filter may not be applicable to other filters. (In the extreme case, some filter settings may not even make sense to other filters. Thus, it is clear that settings need to be able to take on a null or not applicable value.) More specifically, the corollary is that given two settings and their parameter sets there are two options. The first option is that there is a setting whose parameter set is a superset containing both original parameter sets. The second option is that the parameter sets are disjoint to the point that any super parameter set containing the first parameter set does not intersect with the second parameter set. Formally, we have:

Corollary: Given settings s_i and s_j *either* there exists *k* such that $P_k \supseteq P_i \cup P_j$ *or* for all *k* such that $P_k \supseteq P_i$, $P_k \cap P_j = \emptyset$.

A pragmatic aspect of this Corollary comes to light when considering requirements as derived settings. That is, requirements are a subset of all settings, $Q \subset S$. Although many people would think of requirements as coming first and settings following, once a system is configured, the values obtained for the requirements are directly derived from the values of the settings. This forms a hierarchy that can reach all the way up to the complexly derived values that are put into a final test report. At one level, requirements are things such as measurements of temperature or pressure at a particular point on the test vehicle. However, things such as flutter point or circle error probability are ultimately derived from measurements which are derived from lower level settings. (Mathematically this defines a partial ordering on the sets of settings.) This hierarchical interdependence has the potential to produce conflict. Consider two engineers in different disciplines, say avionics and flying qualities. Although, at first glance, these may seem to be unrelated to each other, if they have calculations that depend on the same measurement, say temperature at the nose, then the test settings for the two engineers are interdependent. Changing the timing or sample rate of that temperature measurement to satisfy one engineer may affect the other engineer's ability to do their analysis.

CONSTRUCTING P_i

Knowing the set of parameters, P_i, that are affected by changing a setting, s_i, is very useful. The elements of this parameter set are dependent on the scope of the settings. Does the set of settings, S, contain high level requirements or is it limited to a specific device?

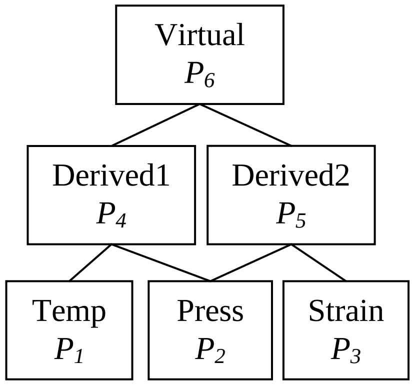

Figure 1
Example Hierarchy

As the corollary shows, interdependencies cause formation of closed sets. A simple example illustrates the hierarchical nature of these sets. Consider three measurements (derived settings) such as t*emperature*, *pressure*, and *strain*. Each of them, by themselves, have rules and a set of parameters, P_1, P_2, and P_3, that are independent of each other. Now consider a derived measurement, *Derived1*, which is dependent on *temperature* and *pressure*. The rule parameters for this include the parameters in P_1 and P_2. That is, P_4 contains the union of P_1, P_2, *temperature* and *pressure*, $P_4 \supseteq P_1 \cup P_2 \cup \{temperature, pressure\}$. This, by itself, is still independent of *strain* and its parameters. Now consider a measurement, *Derived2*, derived from *pressure* and *strain*. Because of the mutual dependence of both derived measurements on *pressure*, it is necessary to construct a *Virtual* measurement dependent on *Derived1* and *Derived2* which is dependent on the parameters of all three measurements. That is,
$P_6 \supseteq P_4 \cup P_5 \cup \{Derived1, Derived2\} \supseteq P_1 \cup P_2 \cup P_3 \cup \{temperature, pressure, strain\}$.

This example also illustrates a method for constructing P_i. Start with the lowest elements, the end nodes of the graph. In the example, these are $temperature,\ pressure,$ and $strain$. Then, work bottom up. As dependencies are identified, form unions of parameter sets. Recall that $s_i \notin P_i$, so as these unions are formed, s_i must also be added to this union (as was illustrated in the example). Formalizing this will require some further notation.

Define $s_g \in S$ to be a *greatest element* of S if $s_g \notin P_k$ for all k. Let $S_g^{'}$ be the largest subset of S having s_g as its *only* greatest element. Then $P_g^{'} = S_g^{'} - s_g$ is the set of parameters that s_g is dependent on when no other setting is dependent on s_g. That is, $S_g^{'}$ is a closed subset of S. Now consider identifying a setting, s_i, that is dependent on s_g. That is, $s_g \in P_i$. Then it is also true that $P_g^{'} \subset P_i$. This is illustrated in the example when constructing P_4 for the *Derived1* measurement.

Now consider two subsets $S_1^{'}$ and $S_2^{'}$ that are the largest subsets of S containing the only greatest elements s_1 and s_2, respectively. Let s_1 and s_2 also be greatest elements of S with $S_1^{'} \cap S_1^{'} \neq \emptyset$. That is, s_1 and s_2 have a common dependency but there is no setting dependent upon both of them. This requires a virtual setting, s_v, dependent on both s_1 and s_2 be created with $P_v \supseteq P_1 \cup P_2 \cup \{s_1, s_2\}$. This is illustrated in the example with the construction of the *Virtual* setting.

From a software point of view, this provides a mechanical means for giving feedback to the user regarding the consequences of changing a particular setting.

THE CONFIGURATION PROCESS

Consider walking through the process of configuring a complex data acquisition system with a user sitting at a computer and an application interface with a series of pull down menus, dialogue boxes, etc. Fundamentally, the user sets one setting after another (although going back to change settings is not unusual in practice). A simple form of automation that the software can provide is, at each point along the way, to present the user with only those values of settings that are valid with the settings already set. If this is not done, then it is very likely that the final configuration will not be valid. An underlying assumption here is that every value of every setting is part of a valid configuration. Formally, it is assumed that for every i, j, there exists C^V such that $s_{ij} \in C^V$. As the user contemplates selecting a value for the next setting, how does the software know what values are consistent with the settings already set?

To answer this question we start considering each rule, $\boldsymbol{\mathcal{R}}_i$, as a function that returns a set of values and use $\boldsymbol{\mathcal{R}}_i$ to denote that set of values. Recall that the set of parameters, P_i, for $\boldsymbol{\mathcal{R}}_i$ has been defined as a set of specific values. However, in our scenario not all settings have been given specific values. That is, instead of specific values, the parameters for the rule may actually be a set of values. Further, the values of the settings not set may be constrained by the settings that have already been set. Thus we define the *generalized parameters* of the *generalized rule*, $\widehat{\boldsymbol{\mathcal{R}}}_i$ as

$$\widehat{P}_i = \left\{ \widehat{\boldsymbol{\mathcal{R}}}_{k_j} \mid j = 1, \ldots, m_i;\ k_j \in \{1, \ldots, n\}; k_j \neq i, m_i \leq n \right\}.$$

This means that the set of values to put into the pull down menu for our user is $\widehat{\boldsymbol{\mathcal{R}}}_i(\widehat{P}_i)$. Since some settings have already been set, it would be appropriate to notate the i^{th} call as

$$\widehat{\boldsymbol{\mathcal{R}}}_i(s_{1,1}, s_{2,1}, \ldots, s_{i-1,1}, \widehat{\boldsymbol{\mathcal{R}}}_{i+1}, \widehat{\boldsymbol{\mathcal{R}}}_{i+2}, \ldots, \widehat{\boldsymbol{\mathcal{R}}}_n).$$

(The double subscripts on the settings indicate a specific value for that setting. The use of 1 for the second index is for convenience without loss of generality. Also for convenience, we allow parameters that are not strictly required for $\widehat{\boldsymbol{\mathcal{R}}}_i$.)

RULE SETS

Thus far we have established a basic concept of settings and related rules. We have also indicated the existence of a hierarchy of settings. If we now look at the real world application of this theory, it is not difficult to see that there is a functional breakdown of settings and their related rules. First, some vendors maintain proprietary rule sets. Second, the people that do data analysis are not usually the people that instrument the test vehicle. Third, there is a distinction between measurements and the details of how those measurements are made. These functional distinctions, along with identified rule sets, are illustrated in Figure 2. When considering these rule sets, recall that the concept of "setting" has been generalized to include "derived settings" and "requirements".

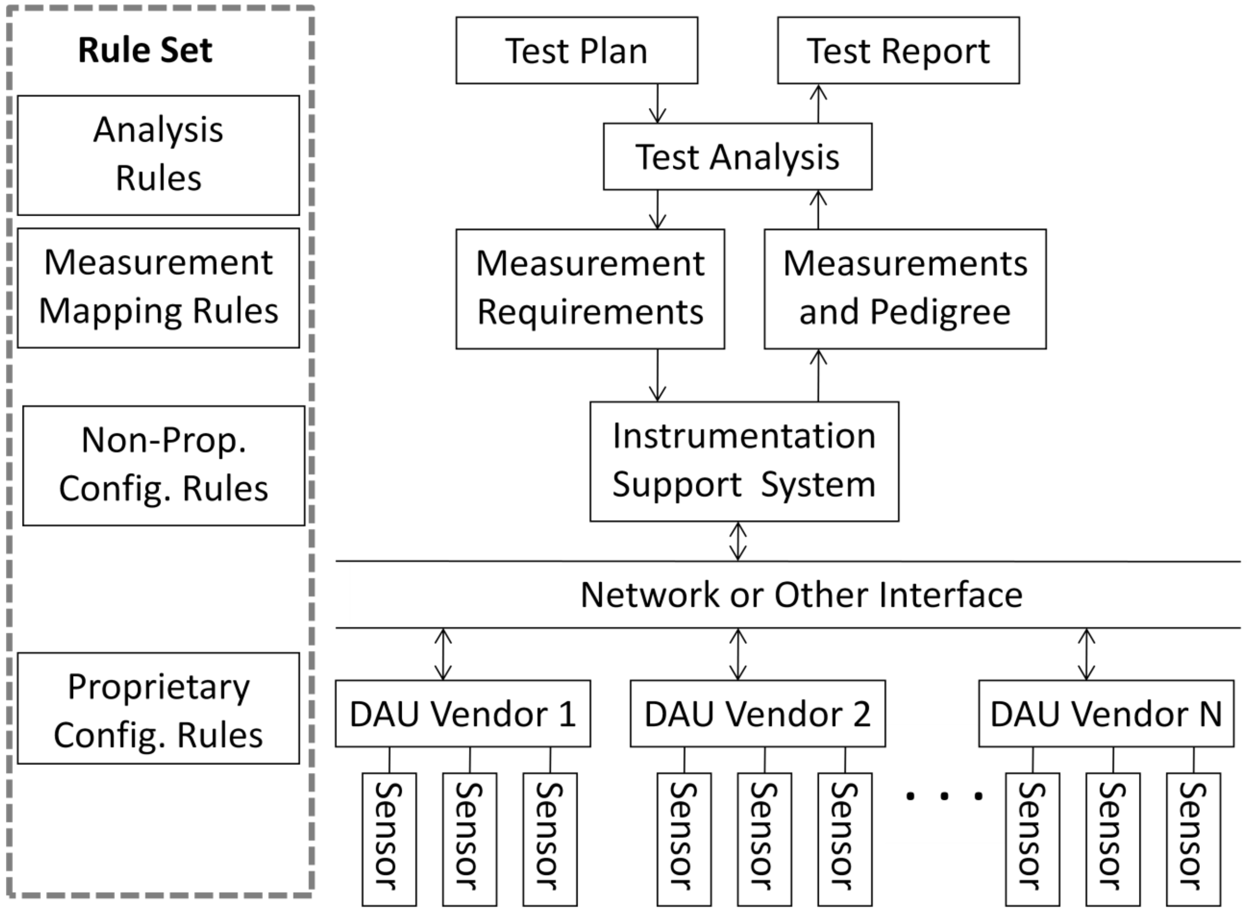

Figure 2 – Rule Sets to Function Mapping

Specific rule sets identified are:

1. *Proprietary instrumentation configuration rules*. These are vendor specific and not available to third party software developers that would implement a multi-vendor configuration interface. An example might be rules governing when a particular filter is used.

2. *Nonproprietary instrumentation configuration rules*. These are general rules that are open to everyone and, for some vendors, may actually be all the rules. An example might be that an aggregate gain is determined as the sum of two other gains.
3. *Measurement Mapping Rules*. These rules map general measurement requirements to specific instrumentation settings. An example might start with temperature on the left wing tip and map it to a specific sensor with specific calibration data.
4. *Analysis Rules*. These are the rules that take a high level test requirement and translate it into a set of measurements and then also map the results back up through the algorithmic data analysis process. An example might be that determining the flutter point requires an array of strain gauges on the test vehicle as well as what algorithms are used to calculate the flutter point.

The identification of distinct rule sets is certainly debatable and dependent on point of view. This paper and the model it presents has been developed from an instrumentation point of view and thus the rule sets are possibly more detailed at the lower levels than at the higher, analysis, levels. Although, even there, it might be appropriate to include a sensor level rule set. At the analysis level it might, for example, be appropriate to break the rules into those that are directly derivable from test vehicle measurements and those that require inputs from other sources.

APPLICATION NOTES

There are various uses of this model. The following outlines four major uses.

1. *Configuration Development and Transfer*. This is the primary motivation for developing this model. How is a multi-vendor instrumentation system configured through a single interface? Much of the notation and results explicitly support this function as will be discussed in the next section. Once a valid configuration is constructed it must be transferred to the systems using it.
2. *Verification, Validation, and Completeness (VV&C)*. Just developing a configuration does not mean that it does everything that it’s supposed to and is expected to do. By formalizing the functional hierarchy, VV&C can be more readily automated at different levels of the hierarchy.
3. *Optimization.* By creating a formal structure that allows for multi-vendor systems and by capturing all the settings – including the constraints – system level optimization can be automated.
4. *Pedigree.* Instrumentation and test engineers not only provide measurements and final analysis, they also provide information about where those measurements come from and what process was used to perform that analysis – the pedigree. The final configuration, C_Q^{VVC}, along with the rule sets have the potential to provide an absolutely <u>complete</u> documentation of this process.

APPLICATION PROGRAMMING INTERFACE (API) NOTES

The primary focus of this model has been on a single interface to a multi-vendor system in consideration of proprietary rules. A full API must include various support routines to allow initialization and actual transfer of information. But fundamentally the major sets defined in this model represent the types of information that has to be exchanged or some part of the system has to be cognizant of.

$\boldsymbol{\mathcal{V}_i}$ – The set of values that a setting can be set to.
$\boldsymbol{\widehat{\mathcal{R}}_i}$ – The rules governing a particular setting.
P_i – The parameters for a rule, i.e., the things that affect a particular setting.
P_g' – The parameters a greatest element is dependent on.

At the very least, there must be software engines that implement the rules, there must be a mechanism for passing the possible values a setting can have, and there must be some way of knowing the affects of setting or changing settings.

There are at least two methods of passing information between two pieces of software connected by an API – either pass all the information back and forth or use a shared memory block that maintains the current configuration. The latter of these seems simpler and requires less upfront information provided to a third party interface developer. For example, to make a request to change a setting without a common memory structure would require knowing P_i for every setting. Whereas, using a common memory block would allow a change request to be made and to allow the rule engine to pick up whatever information it needs out of this block. This may also aid vendor's to maintain their proprietary information.

SUMMARY

This paper has provided a theoretical structure for configuring instrumentation (which is a specific instance of the class of Constraint Satisfaction Problems). The underlying structure is defined in terms of instrumentation settings and the rules which govern valid settings. By combining this structure with real world functional considerations, a foundation has been laid for developing a single interface for a multi-vendor system. The model specifically identifies the sets of information that must be generated and exchanged for this interface. A simple lemma and corollary provide insight in to the structure of these information sets. By recognizing that higher level measurement requirements are, in essence, derived settings, the theory can be expanded beyond low level device settings. Overlaying this higher level functional reality onto the theoretical structure provides a mechanism for dividing rules into sets. This then facilitates formal development of these rule sets independently while maintaining the links between them. This lays the foundation for both fully automating the configuration process and for fully documenting the pedigree of measurements and data analysis.