THE *GUM TREE* DESIGN PATTERN FOR UNCERTAINTY SOFTWARE

B. D. Hall
Measurement Standards Laboratory of New Zealand,
Industrial Research Ltd, PO Box 31-310, Lower Hutt, New Zealand.

Abstract

The paper describes a simple approach to software design in which the 'Law of propagation of uncertainty' is used to obtain measurement results that include a statement of uncertainty, as described in the Guide to the Expression of Uncertainty in Measurement (ISO, Geneva, 1995). The technique can be used directly for measurement uncertainty calculations, but is of particular interest when applied to the design of instrumentation systems. It supports modularity and extensibility, which are key requirements of modern instrumentation, without imposing an additional performance burden. The technique automates the evaluation and propagation of components of uncertainty in an arbitrary network of modular measurement components.

1 Introduction

This paper describes a design technique for software that applies the 'Law of propagation of uncertainty', as recommended in the Guide to the Expression of Uncertainty in Measurement (henceforth, the Guide) [1]. It automates the task of evaluating and propagating components of uncertainty and produces software that is both modular and extensible, both of which are requirements for modern instrumentation. The technique can be applied to instrument and smart-sensor firmware, or to higher level software coordinating measurement systems for specific measurements. It is both efficient and simple to implement. Its most striking feature is that measurement equations are differentiated automatically, allowing uncertainty to be propagated with only a direct statement of measurement equations. The next section shows that the Guide's formulation of the uncertainty propagation law can be easily applied to modular measurement systems. By decomposing the evaluation of a measurement value and the associated uncertainty into an arbitrary set of intermediate calculations, we show that each step can be considered as a modular component of a system. Section 3 uses this formulation to identify the basic requirements of modular software components capable of encapsulating the calculations.

2 Calculation of value and uncertainty in modular systems

2.1 The measurement function

According to the Guide, a measurement may be modelled by a single 'measurement function', f, interpreted

"... as that function which contains every quantity, including all corrections and correction factors, that can contribute a significant component of uncertainty to the measurement result" [1].

The inputs to this function are uncertain quantities so a measurement result is estimated as

$$x_m = f(x_1, x_2, \cdots, x_l) \ , \tag{1}$$

where f is evaluated at the estimated mean values of the inputs, $x_1, x_2, \cdots, x_l$. For example, electrical power, P, dissipated in a resistor may be estimated by measuring the potential difference across the resistor terminals, V, and the resistance, R. The measurement function for this is

$$P = \frac{V^2}{R} \ , \tag{2}$$

where V and R are the two inputs.

2.2 Decomposing the measurement function

In the Guide's approach, the function f represents the entire measurement procedure. In complex measurement systems it will be difficult to state is function explicitly. However, f can be conveniently decomposed into a set of simpler functions of the form

$$x_j = f_j(\Lambda_j) \ , \tag{3}$$

where 'j' identies the decomposition step, x_j is the intermediate value at the step and Λ_j is the set of direct inputs to the function f_j.[1]These functions can be used to evaluate the measurement result by using a simple iterative algorithm. If there are $m-l$ decomposition steps and l direct inputs to f (called 'system inputs'), then

$$x_j = f_j(\Lambda_j) \ , \text{for } j = l+1, \cdots, m \tag{4}$$

The decomposition steps of a measurement function are referred to from now on as 'modules' because we will show that both value and uncertainty calculations can actually be encapsulated at each step and thought of as a discrete component of a measurement. The algorithm above can be interpreted as a distributed calculation over the modules comprising a system. Referring to the simple power measurement example, a possible decomposition is:

$$x_1 = V \tag{5}$$

$$x_2 = R \tag{6}$$

$$x_3 = x_1^2 \tag{7}$$

$$P = x_4 = x_3/x_2 \ . \tag{8}$$

There are two system inputs, x_1 and x_2, so $l = 2$, and two more modules that decompose equation 2, so $m = 4$.

[1]The labels j, are assigned so that $j > k$, where k is the subscript of any member of the set Λ_j.

2.3 Standard uncertainty and components of uncertainty

The term 'standard uncertainty', denoted $u(x_i)$, is an estimate of the standard deviation of the random variable associated with the quantity x_i. The standard uncertainty in the measurement result, $u(x_m)$, often referred to as the 'combined standard uncertainty', depends on the input uncertainties $u(x_1), u(x_2), \cdots, u(x_l)$, their possible correlations and on the function f. The calculation of $u(x_m)$ is simplified by defining components of uncertainty, which are related to the sensitivity of the measurement function to its inputs. The 'component of uncertainty in x_m due to uncertainty in an input quantity x_i is defined as[2]

$$u(x_m) = \frac{\partial f}{\partial x_i} u(xi) \ , \tag{9}$$

where the partial derivative is evaluated at the values of $x_1, x_2, \cdots, x_l$.

The combined standard uncertainty can be evaluated from the $u_i(x_m)$ and the associated correlation coefficients

$$u(x_m) = \left[\sum_{i=1}^{l} \sum_{j=1}^{l} u_i(x_m) r(x_i, x_j) u_j(x_m) \right]^{1/2} \tag{10}$$

where the sums are over the l system inputs and $r(x_i, x_j)$ is the correlation coefficient between inputs x_i and x_j. In terms of the power example, the components of uncertainty are

$$u_V(P) = u_1(x_4) \quad = 2\tfrac{V}{R} u(V) \tag{11}$$

$$u_R(P) = u_2(x_4) \quad = -\tfrac{V^2}{R^2} u(R) \tag{12}$$

and if the measurements of V and R are uncorrelated, the combined standard uncertainty in the power measurement is

$$u(P) = \frac{V}{R} \left[4u^2(V) + \frac{V^2}{R^2} u^2(R) \right]^{1/2} \tag{13}$$

2.4 Decomposition of the uncertainty components

The key point made in this paper is that evaluation of equation (9) can be decomposed into steps to obtain an algorithm of the same structure as (4). This makes it possible to encapsulate both the calculation of components of uncertainty, and the calculation of value, within modules. Each module need only evaluate the components of uncertainty of its own and the results will propagate through the system correctly. Equation (9) can be decomposed by applying the chain rule for partial differentiation to the decomposition functions in algorithm (4). A component of uncertainty, for any given i, can then be calculated iteratively:

$$u_i(x_j) = \sum_{x_k \in \Lambda_k} \frac{\partial f_j}{\partial x_k} u_i(x_k) \ , \text{for } j = l+1, \cdots, m. \tag{14}$$

[2]The *Guide* defines a component of uncertainty as the *modulus* of $u_i(x_m)$ here.

This algorithm propagates the sensitivity of each module function, f_j , to a module input x_k, weighted by the component of uncertainty $u_i(x_k)$ in that input's value due to uncertainty in the system input x_i.

3

This algorithm, together with equation (4), show that the uncertainty component and value calculations of a step can be encapsulated in one module. The algorithms effectively distribute both value and uncertainty calculations over a network of interconnected modules. In the power example, referring to equations 5–8, the two uncertainty functions ($j = 3$ and $j = 4$), for arbitrary i, are

$$u_i(x_3) = \frac{\partial x_3}{\partial x_1} u_i(x_1) = 2x_1 u(x_1) \quad (15)$$

$$u_i(x_4) = \frac{\partial x_4}{\partial x_2} u_i(x_2) + \frac{\partial x_4}{\partial x_3} u_i(x_3) \quad (16)$$

$$= -\frac{x_3}{x_2^2} u_i(x_2) + \frac{1}{x^2} u_i(x_3). \quad (17)$$

In these equations, together with equations (7) and (8), a module need only obtain information from its immediate inputs (the arguments to f_j). So the $j = 3$ module (equations (5) and (15)) only refers to x_1 (the voltage V) and the $j = 4$ module (equations (7) and (16)) only refers to x_3 (the voltage squared, V^2) and x_2 (the resistance, R).

3 A design pattern for uncertainty software

A design pattern describes a generic solution to a broad class of problem. Patterns are widely used in software development and are an effective way for software developers to communicate and share successful techniques. They provide flexibility (e.g. allowing implementation in various programming languages) while concisely describing a solution strategy. The GUM Tree design pattern, shows how to incorporate, into the design of measurement systems, the mathematics underlying uncertainty propagation. The GUM Tree is closely related to the more general and well documented 'Interpreter' pattern 2. There are two important software classes in the GUM Tree: a Module interface class and a Context class. The former is a base class for system modules. It constrains a module's implementation class to meet the basic requirements necessary for uncertainty propagation. The Context class encapsulates calculations that must be handled outside of modules and data associated with the overall system. 3.1. The requirements of a module interface To evaluate algorithms (4) and (14) the j^{th} module must provide two pieces of information:

- an output value, x_j;
- a component of uncertainty in the (module) output due to uncertainty in the i^{th} system input, $u_i(x_j)$.

[3] Equation (14) implies that $u_i(x_i) \equiv u(x_i)$. This is compatible with (9), when x_m is replaced by x_j and f by f_j.

Every module's software interface should include functions that return these numbers.[4] In C++ pseudo code, such an interface can be declared as a pure abstract base class[5]

```
class ModuleInterface {
  public:
    virtual double value() = 0;
    virtual double uComponent(ModuleInterface& i) = 0;
};
```

3.1 The Context

A separate class is required, in addition to the library of module classes, to encapsulate global calculations and manage global data. For example, the calculation of combined standard uncertainty, equation (10), has to be performed on the system modules and needs access to correlation co- efficients. An entity called the '`Context`' can be used to encapsulate this type of information.[6] The `Context` is also useful for handling language and implementation specific details and can extend native language features if necessary. For instance, it can provide support for memory management and expression parsing with abstract data types.

3.2 Applying the pattern

Implementation of the GUM Tree pattern requires that a set module classes be defined, each of which implements the module interface in a different way. Module objects can be created from this class library and linked together to form a system. When in operation, a `Context` will interrogate the network of modules to determine measured values and their uncertainties. Equations (4) require only that a module be characterised by a simple function, f_j , which may have an arbitrary number of arguments. Hence modules may be distinct pieces of instrumentation, connected to the system by some kind of physical communications interface, or they may be abstract software entities in computer memory. An example of the latter is a class library for uncertainty calculations [6]. This effectively introduces a new abstract data type for measurement uncertainty calculations – an entity that has attributes for both value and components of uncertainty.[7] Module classes can implement simple maths functions and arithmetic operations, allowing arbitrary measurement functions to be written directly in software. The explicit derivation of sensitivity coefficients is then unnecessary: a measurement function will be parsed automatically into a tree of linked module objects, which can be interrogated to obtain values and uncertainties. As an indication of how simple such a class library is, consider the definition of a module class for division. The function definitions were given in equations (7) and (16). In addition, the class will need two 'references' (x_2 and x_3), which link to a module's inputs. Member functions use these

[4] Other module interface functions will generally be useful. For instance, an interface can include a function that returns an iterator for the set of system inputs on which the module depends directly or indirectly, this helps sum over i and j in equation (10).

[5] In practice, a distinct interface class, derived from `ModuleInterface`, can be useful to distinguish between system inputs and module inputs. This could be used as the argument type in the second function declaration.

[6] Another calculation that the Context would typically handle is the evaluation of effective degrees-of-freedom [1].

[7] Much like complex numbers, and mathematical support for them, can be added to some programming languages.

references to obtain information (i.e., an output value or an uncertainty component). In C++ pseudo-code, the class definition for division looks like [6]

```
class Division : public ModuleInterface {
  protected:
    ModuleInterface& x2;
    ModuleInterface& x3;
  public:
    Division(ModuleInterface& l,ModuleInterface& r) x3(l), x2(r) {}
    double value() { return x3.value() / x2.value(); }
    double uComponent(ModuleInterface& i) {
      double v = x3.value() / x2.value();
      return
        2 * v * x3.uComponent(i) -
          v / x2.value() * x2.uComponent(i);
    }
};
```

The member function `value()` corresponds to equation (7) and the member function `uComponent(ModuleInterface& i)` to equation (16) with `i` a reference to any system input module. The class constructor,

`Division(ModuleInterface& l,ModuleInterface& r)`,

establishes the required links to the module inputs. This type of implementation of the GUM Tree is closely related to the 'reverse' form of automatic differentiation [5]. The extensibility of GUM Tree designs relies on the common `Module` interface. For example, the input links in the pseudo code above gave no hint has to the extent of the module network connected to each input. A link could point to a simple input or to a complicated module network. This generality allows exchange and extension to occur dynamically in a system. For instance, equation (6) attributed a system input value of R to the measurement equation. However, as a modification to the system, a temperature dependent resistance

$$R = x_2 = R_0[1 - \alpha(T - T_0)] \tag{18}$$

could be associated with that step in the calculation. The network of modules representing the new resistance calculation are then connected at this point (i.e., a decomposition tree of modules representing equation (18)). This change does not require any changes to code in other modules.[8]

4 Discussion and Conclusions

The GUM Tree design pattern is a simple technique that represents a significant improvement over current practice and should ultimately result in more reliable and robust measurement and control systems throughout society. It could be deployed in most modern instrument designs where it would allow systems to report measured values together

[8]The calculation of combined standard uncertainty will naturally be different and will include the additional uncertainty terms. However, this calculation can easily be made generic and parameterised on information available from the module interface.

with uncertainties. Current trends in the test and measurement industry are towards intelligent and flexible system components with industry-standard interfaces to facilitate exchange, re-configuration and longterm maintenance requirements [3, 4]. The modularity and extensibility of the GUM Tree pattern is fully compatible with this trend and has the potential to substantially enhance the expected benefits. The GUM Tree design enables systems to dynamically report uncertainty as operational parameters change, which greatly enhances the integrity of a system's measurement results. One interesting possibility is to reduce the cost-of-ownership burden associated with revalidating test and measurement systems when components are changed. The information required for the uncertainty function of the GUM Tree module interface would be known during the design of an instrument or sensor, so there will be little difficulty in providing software to report it. On the other hand, legacy instruments could be endowed with a GUM Tree compatible interface, by introducing an additional software layer similar to the 'Role control module' approach described in [3].

The new software data type, representing measurement results by encapsulating value and uncertainty information, is particularly interesting. It could be used directly as a convenient tool for data processing (an alternative to 'uncertainty calculator' applications). However it is potentially more powerful in the internal software of measuring instruments. An instrument's calculations would then be following the *Guide*'s recommendations and handling uncertainty at a very low level. This would not compromise proprietary algorithms, but would give external access to the sensitivity coefficients required for uncertainty calculations. In conclusion, the GUM Tree design pattern is an elegant approach to instrument software design that allows the 'Law of propagation of uncertainty' to obtain measurement results that include a statement of uncertainty, as described in the Guide. The technique is applicable to measurement systems of many kinds, large and small. It supports modularity and extensibility, which are key requirements of modern instrumentation systems, and does not impose significant performance requirements. There is increasing pressure on test and measurement manufacturers to provide support for uncertainty that follows the recommendations of the Guide. The GUM Tree design pattern offers a solution to this problem.

Acknowledgements

The author is grateful to R. Willink, for many helpful discussions regarding the GUM Tree and measurement uncertainty in general, and to T. Armstrong for careful reading of this manuscript. Aspects of the GUM Tree design technique have been filed in a PCT application: Uncertainty propagation system and method (PCT/NZ02/00107, 6 June 2002).

References

[1] ISO, *Guide to the expression of uncertainty in measurement*, International Organisation for Standardization, Geneva, 2nd edition, 1995.

[2] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of reusable object-oriented software*, Addison-Wesley, Boston, 1995.

[3] J. Mueller and R. Oblad "Architecture drives test system standards" *IEEE Spectrum* **September** (2000) 68.

[4] K. L. Lee and R. D. Schneeman, "Internet-based distributed measurement and control applications," *IEEE Instrum. Meas. Mag.*, pp. 23–27, June 1999; T. R. Licht, "The IEEE 1451.4 proposed standard", *IEEE Instrum. Meas. Mag.*, pp. 12–18, March 2001.

[5] L. B. Rall and G. F. Corliss, "An introduction to automatic differentiation", in *Computational Differentiation: Techniques Applications, and Tools*, M. Berz, C. H. Bischof, G. F. Corliss, and A. Griewankpp, Eds., pp. 1–17. SIAM, Philadelphia, September 1996.

[6] A number of documents that describe the GUM Tree in more detail and give examples of its implementation are available for download at `http://www.irl.cri.nz/msl/mst`. The available material includes software libraries and examples.