Modelling Dynamic Systems with Known Data: An Object-oriented Modelling System

Carel van Dam and Carl Sandrock

***Abstract*—A number of modelling environments (languages) facilitate object-oriented model development. Although various methods exist for parameter estimation they are not well integrated into these environments and are treated as separate optimization problems. An environment was developed to investigate the integration of these tasks.**

***Index Terms*—plant data, parameter estimation, object-oriented modelling**

I. Introduction

MODELLING an existing chemical process involves identifying the relationships between the plant's various inputs and outputs. The modeller has to use the known parameters, measurable plant data and an understanding of the process to generate a model that is representative of the phenomena of interest. Even simple plants may have complex models and a number of modelling languages have been developed. These languages allow the modeller to describe a process in an object-oriented way, encouraging the reuse of known relations and grouping related process information into an organized hierarchy. This structure is then 'flattened' to a form suitable for simulation. The focus is on equation management and manipulation for simulation.

Parameter estimation has largely been left as a secondary process. The modeller has to determine the initial values, for all the unknowns in the model, to perform an initial simulation. The final values are then determined by comparing the results of a simulation against the known plant data and adjusting the unknown parameters. The problem is that plant data is not included in the model structure.

The objective of this work was to develop a system whereby plant data may be included in the model structure so that parameter estimation may be more readily performed during the development of a model and simulated results may readily be compared to experimental results.

A simple tank is considered that does not exhibit discontinuity or an index greater then unity.

II. Theory

The various components, namely the *Data*, *Process*, *Model* and *Criterion*, combine to define the various problems of *Simulation*, *Identification* or *Control and Optimization* encountered by the modeller [1]. Describing an existing plant falls under the *Identification* problem. Where the known *Data*, including certain parameters and the measured inputs and outputs of the *Process*, are used with a *Model* structure, selected from a possible set of structures, to determine the unknown parameters for that structure. The *Criterion* measures the ability of the structure to accurately represent the phenomena occurring within the process.

A. Model Representation

A number of modelling languages are available including Modelica [2], Simscape/Simulink [3] and gPROMS [4], that facilitate the modelling of physical systems. The requirements of such software is concisely described by Pantelides and Barton [5] and are considered briefly below.

These languages treat acausal, discontinuous, differential algebraic equations by allowing the modeller to group them and any supporting information into classes. Typically user-defined classes will inherit from one of two super classes. A 'Unit' class, which is used to model a volume or capacity and a 'Flow' class, which specifies the connectivity between these capacities [1, pg. 30]. The modeller then describes the various sub-systems under study by inheriting and expanding upon these and other previously defined classes. The model is then built by connecting these together to describe the behaviour of the entire system.

These languages are geared towards equation management and are generally limited in how they treat chemical processes. Simscape/Simulink requires 3rd party support and the commercial package Dymola provides this for Modelica. gPROMS is the only one of these languages that is designed to handle chemical processes.

B. Simulation

Before simulation can occur the hierarchical models must be flattened into a single set of equations. Simple integration routines like Euler or Runge-Kutta may require iterative solution of simultaneous equations [5]. While more advanced codes like DASSL may automate this process [6]. The language compiler must ensure the model is consistent, manage discontinuous equations and perform index reduction. The latter may require that certain equations be differentiated, which is best automated by a computer algebra system. The output of these systems is usually a compiled program that simulates the system.

C. Parameter Estimation

Parameter estimation is the process of determining the unknowns in a model given a structure that relates the input to the output data. An initial value is usually guessed for each unknown and iteratively adjusted until some metric or criterion is sufficiently minimized or maximized. The least mean square error method (LMSE), for example, minimizes

Manuscript received July 18, 2012; revised August 08, 2012. This work was supported by Anglo American PLC and the Technology and Human Resources for Industry Programme (THRIP).

C. Sandrock is with The University of Pretoria, e-mail: carl.sandrock@up.ac.za.

C. van Dam is with The University of Pretoria, e-mail: carelvdam@gmail.com

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the difference between the model's output and the plant's output. Various methods exist for this purpose [7].

This activity is closely related to optimization and the modelling languages mentioned above leave parameter estimation to their associated optimization tools. gProms provides the gPROMS Runtime Object (gO:Run) which supports maximum likelihood (ML) and LMSE [8]. Simulink/Simscape is supported by a very extensive Systems Identification Toolbox that treats linear and non-linear, grey and black box models and ordinary differential equations but not differential algebraic equations [9]. Modelica is supported by Optimica and features within Dymola [10], [11]. These tools tend to flatten the model prior to simulation and parameter estimation.

D. Plant Data

Plant data is typically retrieved from a plant historian in tabular form, where the first column represents the time of each measurement and the columns thereafter are the respective process measurements. The first row in the table is the plant tag for each measurement. Experimental data typically will be arranged in the same way.

III. Experimental

The focus of these languages is on the management of equations with variable objects existing only to support the model. Variables are often assigned upper and lower bounds, an initial value and possibly units as well as a short description. Much of this information is lost during model flattening but should still be available when interpreting the simulation results. Modelling languages share many aspects of their means of model representation and this motivates an intermediate representation for translation from one language to another. Ideally such a representation must have a flexible structure to accommodate the nuances in each language and the ability to expand as the supported languages grow. Validating such a representation requires an implementation in at least one programming language.

A. Structure

The proposed model structure or representation is shown in Figure 1. It aims to incorporate all the features mentioned by [5] and provides a set of classes towards these ends.

The `Relation` class is central to the entire structure as it manages both the equations and the associated variables. An instance of this class retains a set of equations, internal variables and references to external variables, present under other instances of this class. Discontinuity is handled by the `Relation` object which masks the equations that do not apply under the various conditions.

The 'Unit' and 'Flow' classes mentioned in section II-A are included called `Process` and `Stream` respectively. These classes include instances of `Relation` as children. `Process` may also include `Process` and `Stream` as children to handle hierarchical nesting of sub processes e.g. for distillation columns. Presently a mechanism for inheritance has not been defined and this needs further consideration.

Two helper classes `World` and `Reserved` are also included to manage global variables and constants respectively. Both include instances of `Relation` as children. `World` is meant to behave as a an infinite source or sink as required, but this is not rigidly enforced. `World` and `Reserved` instances are unique and may not be nested.

`Variable` instances that are not local to a particular instance of `Relation` are considered external to it. Three kinds of external variables are allowed namely reserved, world and relative, which correspond to `Variable` instances under `Relation` instances of `Reserved`, `World` and `Process` or `Flow` respectively. Variables may also have knowledge as to their Units, Range and Type, which defines them as Parameters, Inputs, Outputs or States. Importantly they will also retain information about their names in experimental investigations and as plant variables. This is done by supplying tag names to the variables which will match those in the data files. Equations may also need to retain information about their linearized form and under which discontinuous conditions they might apply. The `Data` class is included to manage data files and to connect the information in these files to the `Variable` instances in the model structure.

Instances of these classes have `Model` as their parent with the exception of `Process` and `Flow` instances which may have other instances of `Process` and `Flow` as their parent. The `Model` class exists to manage the structure of the model, its various components and any non-model classes such as the `Data` class.

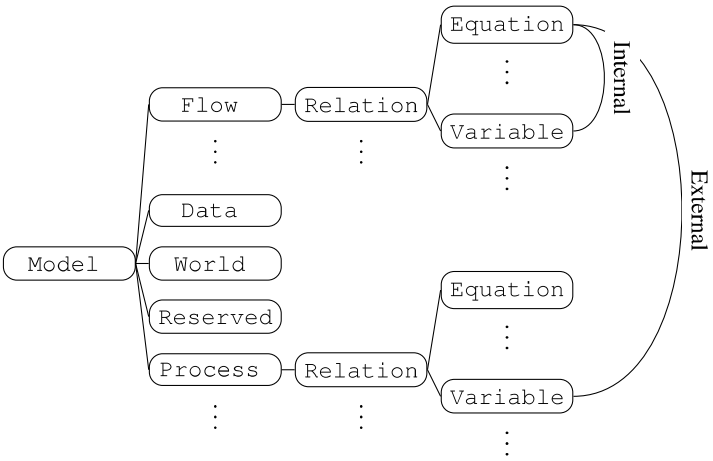

Fig. 1. The hierarchical structure of an instantiated model. Indicating how the instances of each class are arranged and the difference between an internal an external variable.

B. Representation

There are a number of marshaling or serialization formats for storing object-oriented structures including XML (eXtensible Markup Language), JSON (JavaScript Object Notation) and YAML (Yet Another Markup Language). JSON was selected as the preferred format as it is terse and readily translated, requiring only a single pass. Within this representation relative file paths to other JSON files are allowed, this enables models to be split up into parts. Any part may then be swapped out for another part enabling simple model comparison. Relative references to other objects are also allowed.

There is not presently a standardized schema for validating JSON files but some headway has been made [12]. For

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

now the structure is believed to be sufficiently clear and the implementation of a schema not an immediate requirement.

C. Implementation

Any language implementing this structure must support object-orientation, preferably providing a computer algebra system with differentiation capabilities, good plotting and user interface libraries. MATLAB and Python both provide such functionality. The former was selected as this project forms part of another in this language. Figure 2 indicates the implementation's class structure.

The main classes comprises `Odete`, `Base` and `Model`. The `Odete` Class acts as a project or model manager, handling the models that are loaded. This will eventually manage the user interface as well. A `Settings` class is provided to handle user settings and any other configuration information.

All the classes must inherit from a super class called `Base`, itself a subclass of MATLAB's `hgsetget`. This enables certain basic functionality such as the find and the set and get functions. The `Base` class essentially converts all classes to and from the JSON file. This reliably mirrors the structure presented in 1 and allows one to quickly alter the structure as the implementation matures.

The `Relation` class should handle both equation manipulation and index reduction [13]. At present it can rearrange the equations using an implementation of the Tarjan algorithm but index reduction is not yet implemented. In some cases the Type information contained within `Variable` may be used to identify system connectivity and assist in the equation arrangement. This leads to a representation similar in form to a Simulink model.

In the present implementation the modeller is able to select certain sections of the tree to flatten and simulate. The variables external to the selected set of relations are identified in this case and need to be specified for this to work. As `Variable` objects know the Tag Names of the plant measurements, direct comparison between the plant data and simulation results is possible. Parameter estimation is still being implemented.

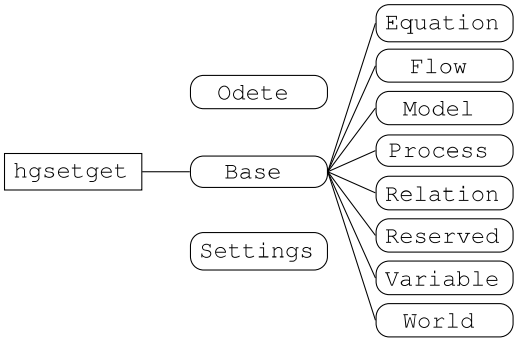

Fig. 2. The class inheritance diagram for the MATLAB implementation indicating how the classes are derived from `Base` which extends `hgsetget`.

D. Example

A simple tank system has been modeled to ensure the proposed structure loads and saves correctly and to ensure the model can be simulated. This system is represented by equations 1 to 3 and in the code listing in appendix A. The representation in JSON is fairly verbose for presentation purposes. The model becomes significantly more compact when the white space is removed.

$$\frac{dV}{dt} = V_{\text{in}} - F_{\text{out}} \tag{1}$$

$$V = Ah \tag{2}$$

$$F_{\text{out}} = kh \tag{3}$$

Such a simple system will not require index reduction but can verify that the equation management works in the implementation.

IV. Results and Discussion

The structure presented can be extended to represent more complex models, though these will require a better implementation which supports equation re-arrangement. The addition of information to the `Variable` class has helped to match the experimental data against the inputs, outputs and states in the model. To get to a point where parameter estimation is possible within this structure requires further development. JSON allows for a flexible structure where whole branches or localized parts of the Model may simply be swapped for other representations enabling the modeller to quickly compare different sub-models.

V. Conclusions and Recommendations

The inclusion of plant tags into the model structure simplifies the comparison of plant data and the results of a simulation. Parts of the model are also readily simulated enabling the modeller to investigate the lower levels of the structure. Further work must be done to enable parameter estimation and the various features expected of modern modelling environments.

Appendix

The following code listing describes a simple tank including references to the various plant tags in an associated data file.

```
{"Name": "Simple Model",
 "Processes": [
  {"Name": "Tank",
   "Relations": [{
    "Equations": [
     {"Formula": "V:t = In.V - Out.V"},
     {"Formula": "V = A*h"}],
    "Variables": [
     {"Name" : "V",
      "Description": "Volume",
      "Type" : "State",
      "Value": "Null",
      "Units": "l"},
     {"Name" : "A",
      "Description": "Area",
      "Type" : "Parameter",
      "Value": "Null",
      "Units": "cm^2"},
     {"Name" : "h",
```

```
      "Description": "Height",
      "Type" : "Output",
      "Value": "Null",
      "Units": "cm"}]}]}],
"Flows": [
 {"Name": "In",
  "Enters": "Tank",
  "Leaves": "World",
  "Relations": {
   "Equations": [
    {"Formula": "V = 0.5"}],
   "Variables": [
    {"Name": "V",
     "Description": "Flow Rate",
     "Type": "Input",
     "Value": "0.5",
     "Units": "l/s"}]}},
 {"Name": "Out",
  "Enters": "World",
  "Leaves": "Tank",
  "Relations": {
   "Equations": [
    {"Formula": "V = k*Tank.V"}],
   "Variables": [
    {"Name": "V",
     "Description": "Flow Rate",
     "Type": "Output",
     "Tag": "F_0",
     "Value": "0.5",
     "Units": "kg/s"},
    {"Name": "k",
     "Description": "Constant",
     "Tag": "F_1",
     "Type": "Parameter",
     "Value": "Null",
     "Units": "1/s"}]}}],
"Reserved": {
  "Name": "Environment",
  "Relations": {
    "Variables": [
       {"Name": "t",
        "Description": "Time",
        "Type": "Variable",
        "Value": "Null",
        "Units": "s",
        "Tag": "01-Time"}]}}}
```

REFERENCES

[1] M. R. Westerweele, "Five steps for building consistent dynamic process models and their implementation in the computer tool modeller," Ph.D. dissertation, Technische Universiteit Eindhoven, Eindhoven, 2003.

[2] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica : An international effort to design an object-oriented modeling language," in *Summer Computer Simulation Conference*, 1998, pp. 333-339. [Online]. Available: www.modelica.org

[3] A. C. W. Grace, "SIMULAB, an integrated environment for simulation and control," in *Proceedings of the 1991 American Control Conference*, Boston Park Plaza Hotel, Boston, Massachusetts, 1991, pp. 1015-1020.

[4] P. I. Barton, "The modelling and simulation of combined discrete and continuous processes," Ph.D. dissertation, Imperial College, London, 1992.

[5] C. C. Pantelides and P. I. Barton, "Equation-oriented dynamic simulation current status and future perspectives," *Computers & chemical engineering*, vol. 17, pp. 263-285, 1993.

[6] L. R. Petzold, "A description of DASSL: a Differential/Algebraic equation solver," *Scientific Computing*, pp. 65–68, 1983.

[7] K. J. Astrom and P. Eykhoff, "System identification : A survey," *Automatica*, vol. 7, no. 2, pp. 123-162, 1971.

[8] Process System Enterprise Limited, "Model validation and model-based data analysis," 2012. [Online]. Available: www.psenterprise.com

[9] The MathWorks, Inc., "System identification toolbox : User's guide (version 2012a)," 2012. [Online]. Available: www.mathworks.com

[10] J. Åkesson, "Optimica : an extension of modelica supporting dynamic optimization," in *Proc. 6th International Modelica Conference 2008*, 2008. [Online]. Available: www.modelica.org

[11] H. Elmqvist, H. Olsson, S. E. Mattsson, D. Brck, C. Schweiger, D. Joos, and M. Otter, "Optimization for design and parameter estimation," in *Paper presented at the 4th International Modelica Conference*, 2005.

[12] K. Zyp, "A JSON media type for describing the structure and meaning of JSON documents," 2010. [Online]. Available: www.http://json-schema.org/

[13] C. C. Pantelides, "Speed uprecent advances in process simulation," *Computers & chemical engineering*, vol. 12, no. 7, pp. 745-755, 1988.

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)