Requirements for Configurer Tests: A Position Statement

David C. Brown
Artificial Intelligence in Design Research Group
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609, USA
E-mail: *dcb@cs.wpi.edu* Web: *http://www.wpi.edu/~dcb/*

Abstract

The purpose of the paper is to discuss the requirements for a general set of tests that would both test any Configurer, and allow different Configurers to be compared.

Introduction

It would be nice if there was a good, general way to compare different configurers according to different criteria.

The obvious approach is to produce a test suite, in some chosen domain or domains, in some 'neutral' representation language. Each configurer could then be tested by converting the test suite into the input format required by each system. The performance of different systems could then be compared.

For this paper, the main issues that are raised by this idea are:

- What should the test suite "test"?
- What is it that one might want to compare?

Hence the general question, *what are the requirements for configurer tests?*

Not all configurers view configuring in the same way. Some assume the availability of a template of key components, or a high-level functional description. Some consider the configuring to be a Constraint Satisfaction Problem. A test suite would have to adequately test and compare systems however they view configuring.

As not all configurers use the same AI techniques to implement their problem-solving (e.g., using a search vs. arc consistency given a CSP view) then this too needs to be taken into account by a test suite.

As not all configurers are implemented equally well, we would like to isolate the effects of the implementation if possible. So, while we'd like to know that the code was fast, we'd also like to know when that speed was being produced by a better algorithm.

The Approach

We are proposing that the requirements for a configurer test should separate out the different issues that might be tested. The qualities of a configurer that should be tested can be divided into a hierarchy of views.

From most general, to most specific, the views are:

- *The Software view;*
- *The Algorithm view;*
- *The Search view;*
- *The Configuration Problem-Solving Method (PSM) view;*
    - AI Techniques and Implementation Methods subview.
- *The Domain view;*
- *The Particular Problem view.*

The qualities introduced at higher levels can be inherited and specialized at lower levels. For example, the "absolute speed" quality is an issue however we view the configuration process. However, at lower levels we can ask much more specific speed questions, about portions of the process, such as the time it takes to retrieve a component description from a database.

By using different views, the intent is to try to produce tests that work to reveal the functionality of the system, and not just surface aspects that are tied more to implementation.

In general, we require a test of configurers, to provide, reveal or be responsive to all the issues from all views.

Software

As configurers are software, then their quality can be measured and compared by using measures that have been developed for software in general. CMU's Software Engineering Institute's "Quality Measures Taxonomy" (CMU 1997) provide an excellent set of measures for evaluating software. The measures include:

- *Needs Satisfaction:* including correctness, and effectiveness.
- *Performance:* including reliability, capacity, and throughput.
- *Maintenance:* including maintainability and complexity.
- *Adaptivity:* (i.e., adaptability) including portability and re-usability.
- *Organizational:* including cost of operation and productivity.
- *Dependability:* including robustness and security.
- *Cost of Ownership:* including training and maintenance.
- *Trustworthiness:* including vulnerability and accountability.

Algorithm

When viewing a configurer as an algorithm, we are concerned with:

- Whether it always halts, or whether looping conditions can be induced;
- Its Absolute Speed;
- The Total Memory used;
- Its Sensitivity to small changes in inputs;
- Its Performance relative to how much input is provided, e.g.:
    - Speed and memory;
    - Quality of solution;
- Ablation or Lesion studies (Kibler & Langley 1990): how the algorithm responds to reduced amounts of knowledge, or partial data;
- Whether all the answers are needed or just one;
- Whether the best answer is provided or just a satisfactory answer.

Search

If we view a configurer as searching over a space of partial or complete configurations, then we can add those issues that relate specifically to searching to the requirements for testing. For example, the search performance of a configurer may depend on:

- The size of the search space;
- The number of operators;
- The number of goals states (i.e., how does the configurer respond in the presence of many, versus few, satisfactory configurations given a set of requirements?);
- The average branching factor for a state;
- The availability of abstract state descriptions (i.e., is it possible to represent a configuration in terms of *types* of components?);
- How it responds to being overconstrained;
- Whether prior knowledge is available and used:
    - About prior solutions; i.e., Can the search be guided, and potential states pruned, by using knowledge of the characteristics of a configuration that corresponds to a particular set of requirements?
    - To evaluate states;
    - To detect closeness to a goal;
    - To prefer known partial paths.

Configuration

Once we get to the level of considering a configurer as carrying out a configuring process then we can be much more specific about the issues that might be tested. For example, are "Key Components" known? Is a "Functional Schema/ Template" available that describes the target configuration? Are functional and component hierarchies available? How do these affect the performance? How does the system degenerate if it is denied their use?

More precise tests can be described if we consider Configuring as a Problem Solving Method (PSM) (Benjamins & Fensel 1998). It's worth noting that ideas in this paper are similar to what is described by Fensel & Motta (1998), i.e., the idea of PSM being a specialization of a generic problem-solving paradigm, such as search.

It has been pointed out that PSMs themselves have not been well evaluated (Menzies 1998). Although that appears to be true, that does not prevent the use of knowledge of a PSM to guide tests of configurers.

The PSM approach includes an Operational Specification that describes the inference steps in the reasoning process, the control flow between them, and the knowledge used. A portion of what a configuring PSM might look like is provided in my paper for the last AAAI

Configuration workshop (Brown 1998). This presents configuring as logical subtasks:

Configuring = Selecting + Associating + Evaluating;

where:

Selecting = Choosing components;

Associating = Establishing abstract and then specific relationships between components;

Evaluating = Compatibility Testing + Goal Satisfaction Testing.

The requirements for configurer tests would include the need to test each "inference step" (or subtask) in the PSM.

Selecting: How does selection performance vary depending on the number of currently active constraints? What is the effect of increasing the number of available components? What is the effect of having to use all of the components supplied versus some of them?

Associating: How does the number of 'ports' per component (i.e., relationships to establish) affect performance? How does the availability of more abstract relationships (e.g., 'above') affect performance -- does it serve as a least commitment approach that reduces failure-producing early choices?

Evaluating: How does the form and number of requirements on the desired configuration affect the performance?

Implementation, Domain & Problem

It should be clear that once a method, such as logical inference or heuristic search, is used to 'implement' a subtask, additional method-specific tests can be applied that might cause the method to fail or to behave badly.

While different domains may vary the nature of the relationships that might be established between components (e.g., electrically connected vs. surfaces touching), that shouldn't in and of itself affect the tests to be done.

However, the particular problems chosen as tests will make a very large difference. Some might be pathological in various ways, perhaps with a single possible configuration or even none. Others might vary key ingredients, such as the number of components.

Conclusions

Comparative testing of configurers is very hard, due to the great differences in approaches used. As the number of variables that would explain those differences is large, and their interactions many, it will be difficult do make trustworthy performance predictions about configurers.

A very large number of tests will be required to reveal all aspects of the differences between two configurers. However, following an approach such as this, should indicate what to concentrate on, and should produce better tests.

In this paper we have sketched some requirements for configurer tests, and outlined a way of thinking about them as a hierarchy of increasingly more specific views. Clearly there is much more to be done in this area.

References

V. R. Benjamins & D. Fensel (1998) "Editorial: problem-solving methods", *International Journal of Human-Computer Studies*, Vol. 49, pp. 305-313.

D. C. Brown (Sept. 1998) "Defining Configuring", invited paper, *AI EDAM*, special issue on Configuration, (Eds.) T. Darr, D. McGuinness & M. Klein, Cambridge U.P., Vol. 12, pp. 301-305.
{also available at *http://www.cs.wpi.edu/~dcb/Config/EdamConfig.html*}

D. Fensel & E. Motta (1998) "Structured Development of Problem Solving Methods", *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling, and Management* (KAW'98), Banff, Canada.

D. Kibler & P. Langley (1990) "Machine learning as an experimental science", In: *Readings in Machine Learning*, (Eds.) Shavlik & Dietterich, Morgan Kaufmann.

T. Menzies (1998) "Evaluation Issues for Problem Solving Methods", *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling, and Management* (KAW'98), Banff, Canada.

SEI (1997) "Quality Measures Taxonomy", *Software Technology Review, http://www.sei.cmu.edu/activities/str/taxonomies/qm_tax_body.html,* Software Engineering Institute, CMU.