# Capturing and Matching Dynamic Behaviour in Case-Based Reasoning*

Peter J. Funk and Dave Robertson

Edinburgh University, Department of Artificial Intelligence
80 South Bridge, Edinburgh EH1 1HN, UK
E-mail: {peterf | dr}@aisb.ed.ac.uk

**Abstract.** In the telecommunications domain, reuse of service specifications is a major issue. However, it has proved difficult to modularise services because of the high degree of interaction between them. Direct application of formal logics to the specification of services has proved impractical because of the size of the services. However, much of this complexity stems from the details of implementation of the services; by contrast, the principal behaviours of a service are often approximated by simple varieties of logic which are easily accessible to users. We address the problem of determining, from a library of services, those which might be appropriate for reuse in constructing a new service. Simple behavioural sequences are used to provide features within a CBR system which matches these to behavioural examples supplied by users. By side-stepping the problem of formally specifying the entire service, we aim to promote greater reuse of services while avoiding a commitment to full logical specification.

Non-mathematicians often have difficulty in expressing requirements formally. By using a CBR approach the user can sketch out simple, familiar behaviours and with these examples the system is able to retrieve relevant cases and interactively produce a formal requirements sketch capturing the new required behaviour. A case in the case library encapsulates a particular formalised behaviour in a simple logic which is sufficient to capture the key dynamic behaviours of the domain. With a simulator the user can evaluate the behaviour without being confronted with the formal representation itself. Our domain is telephone features such as call waiting, redirect call, call back. These telephone services are stored in the case library as cases, each consisting of a set of transition rules. In previous papers we have described the general architecture of the system (see for example [Funk & Robertson 1994]). In this paper we focus on matching dynamic behaviour and the formal representation of the cases.

## 1. Introduction

The CABS system (see [Funk & Robertson 94]) approaches the design of formal requirements specification of telecommunication services. The user gives coarse-grained examples of the required behaviour, which are then matched against cases in a case library in order to identify similar parts of previous cases (requirements specifications). These previous cases are then used in the process of producing a new specification. This approach reduces the effort required to produce requirements

* This research was supported by the Marcus Wallenberg Foundation for Scientific Research and Education and ELLEMTEL Telecommunication Systems Laboratories, Sweden.

specifications since parts of existing specified, tested and integrated specifications may be reused to construct the requirements. This Case-Based approach relies on simple formal notation for features of the cases, capturing the required dynamic behaviour. This notation enables the comparison of cases with respect to these features, suggesting where they may exhibit the same behaviour and where they might differ.

The CABS system aims to cover a small section of the domain of telephone service requirements including the dynamic requirements of the services ”redirect call”, “wake up call” “call back”, etc. These are examples of services which are frequently reused. The case library contain cases which capture the behaviour of the requirements of a specific telephone service. In this and similar domains, it is not merely a matter of producing a new solution to capture the required behaviour of the new functionality, we also have to specify the requirements of interaction with other services as well as behaviour in exceptional circumstances. If we can reuse a past case in such a domain, we may benefit from the fact that the case is already integrated with other services, and that the behaviour of such exceptions may already have been specified.

This paper focuses on how to represent cases which themselves represent dynamic behaviour, and also on the comparison of cases. A brief outline of the CABS system is given in Section 2. In Section 3 the requirements of cases which capture dynamic behaviour are outlined. Examples and an outline of the formal logic used to store cases are given in Section 4. Section 5 discusses dynamic similarity measurements. Finally Section 6 contains a brief summary and conclusions.

## 2. Brief Description of the approach taken in CABS

In Figure 1 a brief overview of CABS is given, structured according to the four REs (Retrieve, Reuse, Revise, Restore) [Aamodt & Plaza 94] in the case-based reasoning cycle. In CABS the input is given as coarse-grained examples of the new behaviour. As described in Figure 1, the input is translated into a representation in which the necessary features for the matching can easily be accessed. In the matching process we identify transition rules (explained in Section 4) capturing a similar behaviour. Thereafter the modules (sets of transition rules) are ranked according to their similarity. The most common situation will be that there is one single case in the case library close enough to the new case to be used as a starting point for constructing new requirements specification. In some situations there are sets of rules from different cases which are similar to different parts of the input. They have to be merged and might need some adaptation in order to produce a proposed solution which is consistent. The adaptation might simply be to add a transition rule connecting two states not captured in the retrieved case but captured in the input. Finally the proposed solution is tested and further adapted to conform to the input example and the user’s intentions as closely as possible.

The representation of cases as used in CABS needs to meet a number of requirements, such as being able to reuse cases both in whole or in part, determine what parts of cases differ and what parts are similar, and identify inconsistency between parts of cases merged in order to produce a proposed solution. In the next Section we will outline the main features of such a representation.

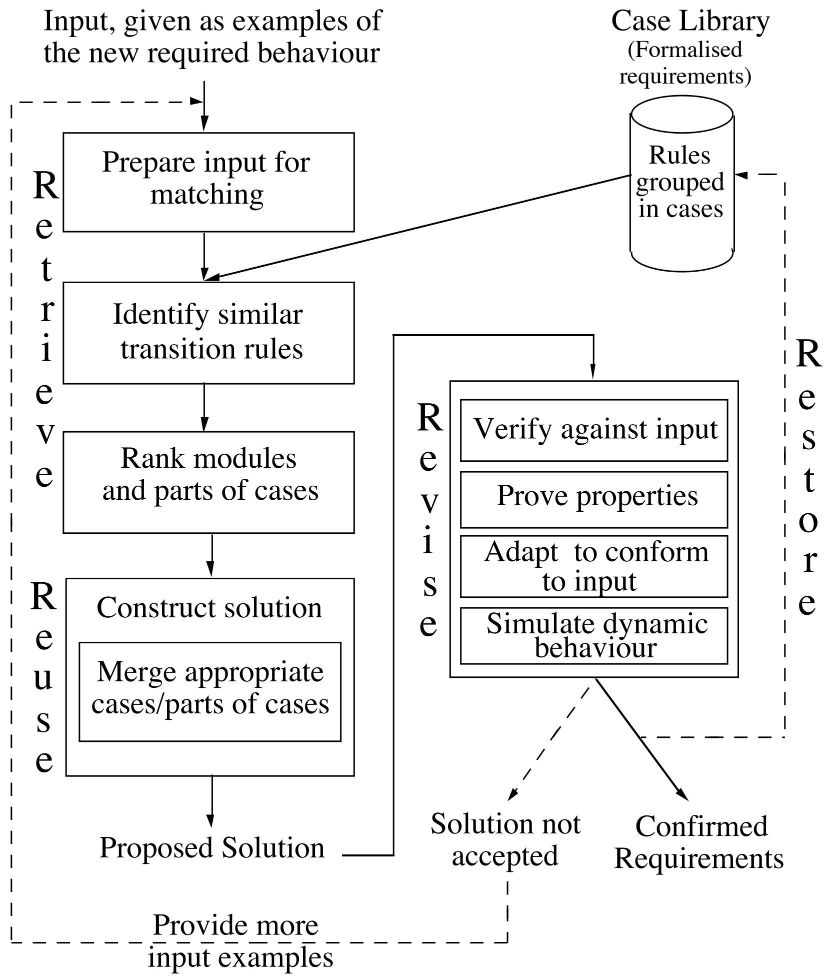


**Fig. 1**. Overview over the CABS system

## 3. Representing Dynamic Reactive Behaviour

Representing dynamic behaviour is an active research area and formalisms like event calculus, situation calculus, π-calculus, petri-nets, CCS, etc. have been widely explored in this context. These representations have the expressive power to reason about different aspects of temporal behaviour such as communicating processes, history, indeterminism, events valid in an open duration, etc. However, we are not concerned with describing all the possible behaviours of services – only an outline of the main features of their behaviour, which can assist in identifying appropriate services. It is important that the language we use should be simple enough to be communicated to non specialist users, since they must ultimately approve the service specified. If we restrict ourselves to specifying a single process which accepts discrete sequenced stimuli, a simple finite state machine represented in predicate logic (Moor-automaton, see e.g. [Lewis & Papadimitriou 81]) may be used.

This allows us to:

- store a particular behaviour in the form of a set of transition rules (a case).
- compare cases and determine if they capture the same behaviour.
- determine which parts of two compared cases correspond and which do not.
- produce a new behaviour by reusing parts of cases.
- determine which parts of the behavioural example given as input are covered by the proposed solution, and which are not.

## 4. A Simple Logic Capturing Change

The language we use [Funk 93, Echarti & Stålmarck 88, Gelfond & Lifschitz 93] contains transition rules (R), stimuli, (S), atomic terms (A) and states (T), which are sets of atomic terms. A stimulus is the only cause of change. Atomic terms are used to describe a state or part of a state. A rule contains a set of preconditions (atomic terms). If the stimulus S has occurred and the precondition is true, the conclusions are necessarily true in the next state.

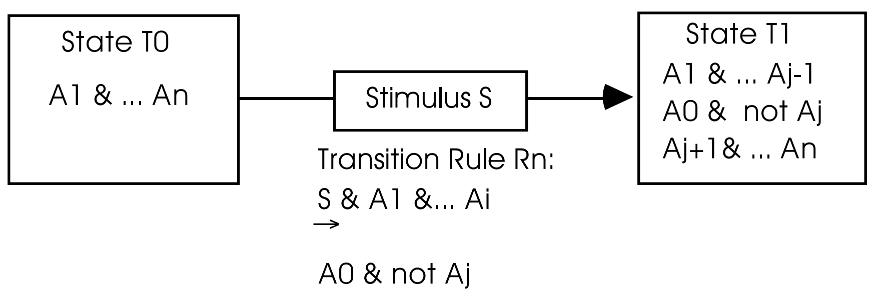


**Fig. 2**. Example of a transition.

State transition rules have been argued to be sufficient for outlining the main behaviour of simple telephone services [Funk & Raichman 90] and we have used these to specify 16 different services. Following is a simplified example of a transition rule:

```
           Stimulus :  dialling(A,Nr)
  Precondition at T:  answer_on(B,Nr) &
                      ¬  redirect(Nr,Nr2)&
                      idle(B).
Conclusion at T+1:   calling(A,B) &
                      ring_tone(A) &
                      ring_signal(B).
```

Our syntax assures an ordered sequence of time points and restricts us to only having preconditions about T and conclusions about T+1. This excludes reasoning about anything other than the immediate past, but gives a simple and computationally efficient implementation. Using predicate logic gives us access to a number of validation tools for consistency checking, simulation, transformation etc. (see for example [Bundy 92]). We may also add restricted natural language and graphical notations [Dalianis & Hovy 93, Davis 90] in order to further aid the user in the validation process of a new service.

## 5. Similarity Measurement of Dynamic Behaviour

There are two steps in identifying useful cases: we have to identify similar cases, and thereafter rank these cases according to how easily they can be adapted in order to produce a solution (see e.g. [Smyth & Keane 93]). To identify whether a case is similar to the input behaviour we have to determine:

- which transitions of the input example are covered by the case.
- which state transitions are missing in the case.
- what extra information the case captures and whether this extra behaviour is of interest for the proposed solution.
- if state transitions in the input are not covered, are there transition rules close to the input and are these candidates for adaptation.

The input examples are translated into partial transition rules – partial since it is assumed that the input examples are not complete but merely an outline of the required behaviour. Hence the partial rules' preconditions may have missing predicates and the conclusions may be incomplete.

If the matching algorithm (see [Funk & Robertson 1994] for the algorithm) does not find an appropriate rule for a state transition we have to either adapt a similar rule from the case library or use the transition rule generated from the input to fill the gap. This allows us, at the very least, to come up with a suggestion capturing the same behaviour as the input.

## 6. Conclusions

This paper has provided an overview of the different parts of a case-based reasoning system supporting the reuse of telecommunication services based on requirements expressed as dynamic behaviour. It has focused in particular on the representation of cases and the matching of dynamic behaviour.

We have outlined how a simple logic is used to capture the behaviour of cases. Cases are indexed using transition rules as features, which enables us to capture simple forms of dynamic behaviour and compare cases with respect to their behaviour. This allows us to reuse cases or parts of cases in order to produce a proposed solution.

## References


Aamodt, A. and Plaza, E., (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, Vol. 7, No 1, pp 39-59.

Bundy, A., (1992). Tutorial notes: reasoning about logic programs. *Second International Logic Programming Summer School, LPSS '92*. Proceedings, pp. 252-77, Comyn G., Fuchs N.E., & Ratcliffe M.J. (eds.), Springer-Verlag.

Dalianis H. and Hovy E. (1993). Aggregation in Natural Language Generation, *The Fourth European Workshop on Natural Language Generation*, *Proceedings,* Pisa, Italy.

Davis, E., (1990). *Representations of Commonsense Knowledge*, chapters 2 and 3. Morgan Kaufmann.

Echarti, J. P. and Stålmarck, G., (1988). A logical framework for specifying discrete dynamic systems, Technical Report, Ellemtel Telecommunication Systems Laboratories.

Engstedt, M., (1991). A Flexible Specification Language using Natural Language and Graphics. MSc thesis, University of Edinburgh.

Funk, P. J. and Robertson D., (1994). Case-Based Selection of Requirements Specifications for Telecommunication Systems. *Second European Workshop on Case-Based Reasoning*, *Proceedings*, Keane M., Haton J. P., Manago, M. (eds.), Chantilly, France, pp. 293-301.

Funk, P. J., (1993). Development and Maintenance of Large Formal Specifications Supported by Case-Based Reasoning. TP26. University of Edinburgh.

Funk, P. J., Raichman, S., (1990) ROS, An Implementation Independent Specification for ISDN, Technical Report, Ellemtel Telecommunication Systems Laboratories.

Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs, *Logic Programming*, pp. 301-321.

Lewis, H. R. and Papadimitriou C. H., (1981). *Elements of theTheory of Computation*, Prentice-Hall.

Smyth, B. and Keane M. T., (1993). Retrieving Adaptable Cases, In K-D. Althoff, K. Richter, & S. Wess (eds.), *First European Workshop on Case-Based Reasoning*. Kaiserslauten: Germany.