Formal Techniques in the Development of Blackboard Systems

Iain D. Craig
Department of Computer Science
University of Warwick
Coventry CV4 7AL
UK EC
JANET: idc@uk.ac.warwick.dcs
INTERNET: idc@dcs.warwick.ac.uk
TEL: +44 203 523682
FAX: +44 203 525714

January 24, 1995

Abstract

The blackboard architecture has been an essentially informal construct from its inception. This has led to different, though essentially similar, interpretations of the original metaphor. In a recent book, the author presented a mathematical specification of a blackboard system shell. The exercise was successful and suggested a method for developing blackboard systems with the aid of formal methods. Given the application of blackboard systems to real-time and safety-critical problems, better guarantees of correct functioning than that provided by a working prototype are required.

This paper describes a formal method for the development of blackboard systems. The method is based, in part, on an informal one. Apart from the difference in emphasis (formal rather than informal), the new approach rests upon a formal definition of the architecture. The essential idea is that the mapping between the formal model of the problem domain (which is intended to be similar to the formal models proposed by, for example, Hayes) and the blackboard shell should be supported by formal proofs of correctness in a way identical to formal software engineering. As part of this mapping process, formal semantics must be given to the attributes that appear on the blackboard, as well as to the abstraction levels themselves. These formal semantics give a guarantee that the objects represented on the blackboard satisfy a variety of properties that are determined by the formal domain model. The semantics also determines the range of legal transformations that may be effected by Knowledge Sources—this is of importance when developing the final control structure. The control structure for the resulting system is derived by considering the blackboard as being transformed from an initial state to a set of potential final states. Each of the final states represents one of the possible answers that the system may produce. The paths from initial to final states can be treated either as traces or by regarding the entire control problem as formalizable in a temporal logic.

We present an informal method for constructing blackboard systems. The informal method forms the basis of the formal method whose initial stages are then described. We outline the formal treatment of control and suggest the use of temporal logic as a tool for reasoning about control. The paper ends with a review of the method.

Keywords: Blackboard Systems, Development Methodology,
Formal Specification, Temporal Logic

1 Introduction

During the last few years, there have been a number of proposals to employ blackboard systems in real-world environments. Blackboard systems, as is well known, can be used in complex domains that require the application of different kinds of knowledge. The characteristics of the blackboard architecture that make it particularly suited to problems of these kinds include:

1. The integration of different kinds of knowledge within a single framework;
2. The incremental generation of solutions;
3. The property that solutions merely degrade in quality or accuracy when input data degrades;
4. The architecture's capacity simultaneously to follow different, potentially contradictory, lines of reasoning;
5. The ability of the architecture to integrate incomplete information within an emerging solution, and
6. The ability of the architecture to interrupt one line of reasoning in favour of another.

Although other architectures (for example, production rules) have some of these features, the blackboard architecture has them all. This makes for a powerful architecture, ideally suited to real-time and safety-critical applications. Indeed, many of the more famous blackboard applications have been in real-time domains: HEARSAY-II [11] understood continuous speech in approximately real time; HASP/SIAP [13, 31] interpreted sonar data; the BB* patient management system described in [25] monitored the respiration of intensive care patients in real time. These systems indicate the actual (rather than theoretical) suitability of the application of blackboard systems to problems that require solutions in real time.

As with conventional software, real-time and safety-critical domains impose constraints over and above those which apply to software constructed for other domains. Both kinds of domain require high-integrity systems which remain operational even under adverse conditions. Furthermore, safety-critical problems (such as intensive-care patient monitoring, nuclear reactor management, air traffic control, and flight management systems) impose the constraint that system failure, when it occurs, should not lead to catastrophic results. For AI systems, these constraints must of course be respected. With AI systems, there are additional constraints which must be satisfied. As was argued in [7], the additional constraints are:

1. The accuracy and relevance of domain knowledge;
2. The correctness of domain knowledge representation;
3. The correctness and appropriateness of inferences drawn from domain knowledge;

4. The correctness of the mapping between domain knowledge and the system's representational structures, and

5. The correctness of the implementation of the system software.

As noted in [7], it is the last point that is typically addressed in the software engineering literature. The other points must be addressed before an AI system can be allowed to go live in a real environment.

The focus of this paper is on how to go about providing the guarantees of correctness that should be expected of a system upon which lives may depend. Specifically, the paper proposes a development method for blackboard systems that is formal in nature, is a formal development of the one proposed in [8], and relies, for its success, upon specification exercises such as those presented in [6]. The method expects the blackboard system builders to construct a mathematical specification of the blackboard shell and other software they intend to use, and to describe the problem domain mathematically. The most obvious mathematical tool that can be used is logic. The description process must be supported by proofs. Some proofs deal with the correctness of mappings between domain theory and implementation structures; others deal with termination and control.

Because blackboard systems are being applied in domains where high standards of reliability are expected, their correctness is crucial. For this reason, it is essential that the best possible guarantees of correctness must be given. As far as we know, *mathematics* is the best way to guarantee correctness. The method proposed in this paper is the first step in providing the necessary high levls of confidence in AI systems. Of course, it will never be possible to give *complete* guarantees: that would presuppose the existence of infallible experts with perfect knowledge. However, mathematical techniques can assist in the development of AI systems, not only of the basic software, but also in assuring the quality of domain-specific information and its mapping into the structures of the resulting system. Formal methods have been successfully used in the development of conventional software, and in the development of at least one AI system [9]. The previous uses of formal methods have shown the feasibility of constructing software that is correct with respect to its initial specification. With AI systems, mathematical methods also afford the opportunity to detect, amongst other things, inconsistencies in the information supplied by domain experts (this is a point we will explain below).

The method we propose entails large amounts of proof. Initially, we expected that the necessary proofs would be extremely complex. Experience (e.g., [6]) suggests that the software-specific proofs are relatively straightforward. The proof-oriented approach to commonsense reasoning proposed by Randell [33], although entailing large amounts of proof, requires no proofs that could not reasonably be done with pencil and paper[1]. Although there is a lot to do, it is not necessarily of an excessively taxing nature, and we expect it to be mostly a matter of routine.

[1] D. Randell, personal communication, 1990.

In the next section, the informal method presented in [8] is briefly described. The informal method forms the basis of the formal method whose initial stages are described in section three. Section four contains an outline of the treatment of control and suggests the use of temporal logic as a tool for reasoning about control. In the final section, the method is reviewed.

2 The Informal Method

In this section, we outline the informal method for blackboard system construction given in [8]. The present version is highly condensed, and the reader should consult [8] for more details. The method assumes that knowledge acquisition has either been completed, or that it continues while the system is being constructed. The method is, therefore, one for the construction of the *system*, and does not contain any prescriptions on how to elicit knowledge.

In this section, we assume a BB1-like architecture [23]. That is, we assume, in general, that the blackboard is divided into a hierarchy of abstraction levels, and that events convey significant information about changes to the current blackboard state. The AGE [31] family of blackboard systems differs slightly from our assumptions, and there are other interpretations of the blackboard model that differ more significantly (e.g., [3]). The selection of an appropriate architecture is clearly an important aspect of any blackboard project, but we must ignore this point at present.

Below, in the description of the formal method, we will occasionally point out some alternatives and generalizations, and their relationships to our approach. However, for reasons of space, we are unable to explore the range of potential differences.

2.1 The blackboard

The blackboard database is the central structure in a blackboard system. The organization of the blackboard has been described by some workers (for example, Nii [30]) as an outline plan for organizing the system's problem-solving activities. Others regard the blackboard structure as a general organizing principle around which problem solving occurs (this appears to be the view of Hayes-Roth in [22]). The blackboard database remains the central structure in blackboard systems.

The blackboard database is organized as a partially ordered hierarchy. Each level in the hierarchy can be thought of as a different view of the problem or of the structures required for the solution of the problem. Thus, in HEARSAY-II, the speech signal is viewed as being composed of a sequence of signal features or as successively more abstract objects, phonemes, for example. The informal method requires that the organization of the blackboard be determined as the first step in building a system: once the organization has been decided upon, it should not be altered unless there are compelling reasons for doing so.

The organization of the blackboard is fundamental because it determines the structures which will form the basis for reasoning within the system, and because it determines the general form of the solution process. For example, in HEARSAY-II, the solution process is basically bottom-up: the signal is transformed into increasingly more abstract structures until linguistic representations of the utterance as a whole have been determined. The system actually operated a two-phase process in which the first phase consisted of bottom-up processing until a collection of words had been hypothesized. Thereafter, context became available for a more opportunistic phase in which hypotheses were refined or integrated to form a better, more abstract, model of the utterance. The organization of the HEARSAY-II blackboard implies a fundamentally bottom-up method of operation.

The OPM planning system [21], the solution was generated in a top-down manner, but, again, the general movement of the solution from top to bottom was mixed with periods of opportunistic problem solving. The initial list of errands was refined in a top-dowm fashion into more detailed sequences of actions which were integrated in an opportunistic manner into a general, satisficing, course of action. In both cases, the interpretation of the abstraction levels that comprise the blackboard is a significant factor in determining how the system will eventually solve the problems it is to solve.

Once the blackboard organization has been determined, the types or classes of entry that reside at each level must be determined. The method in [8] interprets each abstraction level as holding entries of one or more type ('type' being interpreted in the way familiar from logic). The role of types is somewhat controversial: Nii, for example, is wary of them[2]. The idea that entries belong to types is based upon the interpretation of an entry as a set of attribute-value pairs. The values that are held in the attributes are determined by the meaning or definition of each attribute. The collection of attributes which are permitted to be present in an entry defines the type of that entry. Without some kind of a notion of type (even a weak one such as the constructive interpretation of types as sets), it is impossible to differentiate levels.

The method requires that the objects to be represented at each level of abstraction be defined, their important characteristics identified and recorded. In HEARSAY-II, the structures on the Phoneme level represented phonemes that had been hypothesized as appearing in the speech signal; those at the Lexical level represented words. In the abstract, phonemes and words have different properties, and different things can be said of them. In the context of speech understanding, there are common properties: for example, both have a start and end time in the signal, both are temporally contiguous with the immediately previous and immediately succeeding word or phoneme (or the end of the signal). Each abstraction level entails that its entries are composed of attributes of different types. Of course, there will be intersections, but they should not contain members that are definitional (the NAME or CREATION-TIME attributes can appear all over the blackboard, but they

[2] Personal communication, 1985.

are hardly definitional)[3].

The definition of required objects at each level implies that the attributes comprising these objects also be defined. Attributes can be thought of as properties or relations. They serve to define the structures of which they form a part. In other words, the method takes entry types to be defined by the sets of attributes. When attributes are defined, the values which can be associated with them must also be defined. Again, this amounts to the definition of a type. For attributes, it is also important to consider their role in the abstraction hierarchy as a whole. Thus, when a particular attribute is defined, the rest of the blackboard must be considered in order to determine whether the new attribute is unique to the level for which it was first defined, or whether it is more general (and, hence, does not form part of the definition of an entry type). The method requires that the definitions of attributes be recorded. Apart from good documentation, this requirement is imposed so that equivalences can be checked.

Hayes-Roth[4] has noted that during the development of PROTEAN, a number of attributes, each with different names, was defined. Later, when the system was reimplemented using the frameworks described in [24], it was found that some attributes which were formerly considered to be distinct were, in fact, equivalent. The method seeks to remove such duplications by requiring system builders to make equivalence checks as part of the routine. Furthermore, the approach advocated by the method entails that changes to attribute and entry definitions can be more easily and reliably propagated through the rest of the system than would be the case if all were unstructured.

The definition of the blackboard and the objects it contains is, of course, informed by the knowledge acquisition process. At the time of writing [8], it was not clear that the method had implications for knowledge acquisition. In order to determine the structures required by the method, the knowledge acquisition team must ask the experts for descriptions of the objects they consider important in the domain, and they must ask how these objects are related. Clearly, without this information, the analysis process described above is rendered more difficult. As will be seen from the next subsection, Knowledge Source identification also depends on a particular way of acquiring expert knowledge.

2.2 Knowledge Sources and Control

Once the blackboard and its contents have been defined, and the abstraction hierarchy fixed for ever, Knowledge Sources are defined. The method suggests a top-down approach to the development of Knowledge Sources, and it also acknowledges that Knowledge Sources and entries are often quite interdependent in the sense that a Knowledge Source may require (or even presuppose) the existence of a particular en-

[3] In HEARSAY-II, a common set of attributes was used for the entire blackboard.

[4] B. Hayes-Roth, personal communication, 1986.

try or attribute which has not yet been determined. Here, we ignore this dependency and consider the two phases as wholly separate.

The method considers a Knowledge Source to be a task or unit of activity. The identification of Knowledge Sources entails the identification of the inferential tasks the system must perform. An initial set of Knowledge Sources should be defined (say, a set which is adequate to perform some, though not all of the inferential tasks that will eventually form part of the behaviour of the system). These Knowledge Sources form the basis of an initial implementation of the system's inferential component, and they can also form the basis of a refinement activity.

The method assumes that the very first Knowledge Sources that are identified will be general inferential tasks. They would be developed, for instance, by asking the expert what the basic inferential tasks of the system are to be. In speech understanding, one important task is to hypothesise words on the basis of sequences of phonemes; in errand planning, an important task is to decompose a complex goal into a set of less complex ones. These initial Knowledge Sources are then refined into their component actions or sub-tasks. This refinement leads to the identification of other Knowledge Sources. For example, if two Knowledge Sources require a common subtask to be performed, it might be worth defining a third Knowledge Source to perform it. Another example is where a set of Knowledge Sources require a subtask to be performed and where the time at which the subtask is performed is irrelevant (they might, for example, just expect the task to have been performed by the time they execute).

There are clearly many ways in which subtasks can be identified. Whether all subtasks are made into Knowledge Sources, or whether they are made into procedures called by the Knowledge Sources depends upon the application.

The task-based approach also interacts with the formation of the control structure for the system. Above, we noted that tasks are interdependent in a number of ways. Part of the control problem is defined by the organization of the blackboard, and part by the relationships between inferential tasks. The formation of the control structure proceeds by examination of these relationships. General strategies and tactics should be formed. Strategies and tactics should be expressed in terms of the tasks that are covered by them, but they must also be related to the blackboard—that is, they must be related to the changes in blackboard state that result from Knowledge Source execution. This entails that strategies should be conceived in terms of how attributes change on the blackboard—what the significant types of event that occur during the solution process are. This approach to control has the advantage that it is not tied to the existence of particular Knowledge Sources, but is dependent upon the general ways in which the blackboard state alters over time.

3 Formal Method: Motivation and Initial Stages

In this section, we introduce our proposed formal method for the construction of blackboard systems. In broad outline, the formal method is similar to the informal one described above, but differs in detail. The new method requires that the blackboard be defined first and that the types of entry and attribute that will appear on the various abstraction levels also be defined. The difference is that this activity is now viewed as a formal (i.e., mathematical) exercise and can only take place *after* the problem domain has received a formal description (i.e., after a formal model of the domain has been produced). In this section, we will be concerned with the domain and the blackboard representations. In the next section, we consider Knowledge Sources and control.

The formal method attempts to give a mathematical guarantee that the blackboard system is correct. For this, it is necessary to have a formal specification of the blackboard interpreter (or shell) software similar to that in [6]. Furthermore, it is necessary to define the semantics of the representations used by the interpreter: this is to ensure that the mapping between the formally expressed domain knowledge and the structures manipulated by the interpreter is well-founded. Part of the semantics comes from the formal model of the domain, and part comes from the formal specification of the interpreter (the interpreter specification can be viewed as a form of semantics since it attempts to capture the 'meaning' of the architecture).

3.1 Domain models

The basis of the new method is a model of the domain in which the system will solve problems. Hayes [20] argues that formal domain models offer advantages over informal ones: the ontology can be determined more easily, the range of possible inferences discovered, absurdities detected, and so on. A formal theory of a domain also brings with it the valuable property that properties of domain objects can be established by *proof*. Hayes suggests that first-order logic (or some, possibly intensional, enrichment of it) is the best representational candidate for these models because logic has both a syntax and a semantics: that is, it has a proof and a model theory. Formal models are used to give a *precise* account of a problem domain in a way that facilitates detailed analysis and also allows the proof of properties.

The formal model should be constructed from the information provided by domain experts. An example of its use might be as follows. A model is constructed, and implications are drawn (backed up by proof). The implications can be used to ask the domain experts further questions. If something strange is inferred, it is possible to go back to the experts and point out this anomaly. The result might be a refinement of the domain model, or it might be that the apparently strange consequence is, in fact, correct..

The use of a formal domain model also has implications for the knowledge elicitation process. In domains such as those chosen for blackboard systems, more than one expert is usually involved in the elicitation process. It has often been a worry that the knowledge elicited from the different experts may not mesh together, and that inconsistencies can creep in. By use of a formal model of the domain, the assumptions of the experts can be made explicit and the areas of overlap can be checked. If the process of eliciting knowledge from an expert is thought of as the process of constructing a theory, there follows the idea that the domain model is itself a theory whose subtheories are the theories representing the knowledge elicited from the various experts. The deductive closure of the entire domain model is then the union of the closures of the subtheories: consistency can be shown for each subtheory, and for the model as a whole. This may not be a simple or speedy process, but it is in principle possible. It should be noted that completeness is not a property we expect of domain models, but the more modest requirement of consistency is. Naturally enough, we cannot expect the domain model to be exactly the union of its component theories: in practise, it will probably be the case that additional axioms will be required to connect the subtheories (for example, translating terms of one subtheory into terms of another), but this can be handled with relative ease, or so it would appear.

The formal model also serves another purpose. Once constructed and checked, the model serves as a standard against which the rest of the construction process can be judged. This is because the remainder of the derivation of the system is based on the domain model in ways that will become clear below. The domain model must, of course, have the sanction of the domain experts. It is worth stressing, at this point, that, although there is a considerable amount of proof required, it is of a relatively straightforward nature.

Now, it should not be thought that the domain model will require the development of new and exotic logics. This is because, as has often been pointed out, first-order logic (or some variant of it) can be considered as a *meta-language*: first-order logic is powerful enough to describe different inference procedures—the development of domain models of the kind under consideration *does not* depend upon developments in logic. The remaining arguments for the construction of a formal domain model are similar to those in the literature (e.g., [20]).

3.2 Blackboard and entries

The domain model is the starting point for further work. The formal method, like the informal one, requires that the abstraction hierarchy be defined first and that the defining properties of each abstraction level be identified. In the new method, the development of the system-specific structures parallels the steps of the informal one, but requires formal, mathematical, statement of properties. Furthermore, the domain model is already there to assist in the development of the blackboard. Below, we will often refer to the domain model and to the form of its axioms.

It will probably always be the case that the abstraction hierarchy will have to be defined by intuition rather than by formula manipulation. This is no bad thing, as long as the intuitions are good enough. However, the domain model can assist in at least *checking* the hierarchy. In the formal method, the hierarchy is determined by examining the classes of object hypothesized by the domain model: this is similar to the informal model. However, the hierarchy is generated by a *abstraction relation* which can be stated explicitly: the abstraction relation is a binary relation that totally orders its domain[5]. For example, the *part-of* relation generates abstraction hierarchies for tasks such as construction (protein structure analysis is also a domain in which the *part-of* relation generates an adequate abstraction hierarchy). The abstraction relation has the following properties:

$$\begin{array}{l} \forall x\ R(x,x) \\ \forall x,y\ R(x,y) \wedge R(y,x) \Rightarrow x = y \\ \forall x,y,z\ R(x,y) \wedge R(y,z) \Rightarrow R(x,z) \end{array}$$

i.e., R is a partial order.

Once the abstraction relation has been defined, the structures hypothesized in the domain model must be organized in terms of the relation. Furthermore, if the abstraction relation is stated formally, it becomes possible to *prove* that various structures (unary relations in particular) belong to various levels in the hierarchy. Formally, if R_A is the abstraction relation, and if P_1 and P_2 are two unary predicates, we need to show that:

$$\begin{array}{ll} \forall x & \forall y\ P_1(x) \wedge P_2(y) \wedge R_A(x,y) \Rightarrow \\ & \ell(y) \sqsubseteq \ell(x) \end{array}$$

where $\sqsubseteq$ is the partial ordering induced by R_A and ℓ is a (Skolem) function from objects to their abstraction levels. In words, this implication states that if two objects (x and y) satisfy the abstraction relation, then the level to which each is mapped (the function ℓ) satisfies the $\sqsubseteq$ relation. That is, if two objects satisfy the abstraction relation, they will reside on at most two abstraction levels (note that they may reside on the same level—this is possible because the abstraction relation is a partial order). Note that R_A is a relation over objects and that $\sqsubseteq$ is a relation over abstraction levels; also note that other views on how the blackboard is organized can still be accommodated within this approach, provided that these other views regard the blackboard as being a partially-ordered structure (we call R_A the 'abstraction relation' only for reasons of familiarity).

Above, we used two predicates P_1 and P_2: these predicates appear in the domain theory. These predicates can be considered as determining *sorts* or *types* ('type' is, again, to be interpreted in the logical rather than the algebraic sense). Unary predicates usually define kinds of object: above, they are used to denote objects which satisfy some definition (of which more below). The point of the above is to show that the abstraction relation can be used to order the sorts defined in the domain

[5] The extension of the abstraction relation is always finite, note.

theory. The aim of this is to separate the types of the domain theory into types that represent objects at different levels of abstraction. The purpose of the separation is to provide the beginnings of a *theory* for each abstraction level by partitioning the original domain model into (one hopes non-overlapping) subtheories.

Let us consider the structure of some of the sentences that appear in the domain theory: in particular, the *unary* predicates. Typically, unary predicates are defined by sentences of the form:

$$\forall x\ \Phi(x) \leftrightarrow \bigwedge_{i=1}^{i=n} P_i(x)$$

where the right-hand side is a conjunction of relations P_i (which can be thought of as being in conjunctive normal form—this is no limitation, for any formula can be rewritten in this form). The definition of a unary predicate involves other predicates and relations. We can use these definitions in the formation of blackboard objects as follows.

Each unary predicate defines a type of object. The objects in the type can be related to the abstraction relation as we have seen. This entails that the abstraction relation relates types to abstraction levels. Some unary predicates can be considered as defining types of entry. The attributes which an entry possesses are defined by the relations that appear in the *definiens* of the unary predicate. In the above example, any object which satisfies Φ, also satisfies all the P_i: the satisfaction of the P_i may involve the introduction of other objects (defined by unary predicates or taken as primitive—for example, elements of $\mathbf{R}$ or $\mathbf{N}$). Thus, if Φ is defined as:

$$\forall x\ \Phi(x) \leftrightarrow \exists y\ P(y) \wedge R(x,y)$$

we know that entries of type Φ have an attribute R and that the second argument of R is an object of type P. Furthermore, we also know that the arity of R is two. The translation from an n-ary relation in the formal description to an attribute is to consider the attribute as an n-ary relation. The old first argument is the entry in which the instance of the relation is to be found. For example, if we define a *CHILD* type as:

$$\begin{array}{lll} \forall x & \mathit{CHILD}(x) \Leftrightarrow & \\ & (\exists y, z & \mathit{MALE}(y) \wedge \mathit{FATHER}(y,x) \wedge \\ & & \mathit{FEMALE}(z) \wedge \mathit{MOTHER}(z,x) \end{array}$$

we have the following translation into entries and attributes:

CHILD_101:

FATHER y
MOTHER z

where *CHILD_101* is an object of the type *CHILD*. Note here that y and z are taken to be *references* to other entries (they could be names for people, however). The mapping between logic and entries is taken to be very similar to the representation of frames suggested by Hayes [19].

Before going on, let us be clear about what we have so far. We have a way of relating the types (unary predicates) of the domain theory to the abstraction relation. This relation tells us where on the blackboard the various types are to be instantiated. In other words, it tells us where the various kinds of entry will eventually reside. Also, we have a way of translating the predicate calculus definitions of unary predicates into entries (entries construed as attribute-value pairs). This mapping tells us what the attributes of any type of entry will be: it tells us what the attributes that comprise an entry type are.

Seemingly, all we need to do is to apply the above with rigour and we are done. This is not correct. Care is needed in mapping down to entries. Three cases are immediately apparent. Consider the case of two unary predicates, P_1 and P_2 such that:

$$\forall x\ P_1(x) \leftrightarrow \Delta_1(x)$$

and

$$\forall x\ P_2(x) \leftrightarrow \Delta_2(x)$$

Now, let R_1 be the set of relation symbols in Δ_1 and R_2 be the relation symbols in Δ_2 and $R_1 \cap R_2 = \emptyset$. This clearly suggests that P_1 and P_2 have different extensions. Thus, they represent different types. However, in the context of the domain theory, it is still possible that $\forall x\ P_1(x) \leftrightarrow P_2(x)$. That is, the two predicates represent the same type. Unless the number of entry types is to expand indefinitely, this kind of equivalence must be detected (by proof, of course) in order to keep the blackboard manageable. It should be noted that

Equivalent unary predicates can cause problems when the abstraction relation is applied. This is because the abstraction relation might assign P_1 to level l_1 and P_2 to level l_{10}, for example. Since they are equivalent, this cannot be and there may be an inconsistency somewhere; of course, it is possible for the abstraction relation to impose different *views* of the same object, but if the relation is *part-of*, there is a problem in need of solution.

Finally, consider the case in which a relation, R say, appears in the definitions of both predicates, and in which we know that each predicate is assigned a different abstraction level by the abstraction relation. Here, the type of R is important. If we were slavishly to follow the vocabulary approach suggested in [8], we would want to try to define non-overlapping vocabularies as far as possible. However, if the same relation appears in many definitions which represent entries that will appear at different abstraction levels, there will be an overlap in the vocabularies.

The above discussion has concentrated on *types* as a way of organizing the blackboard. Individual entries are thought of as belonging to types, and the attributes which actually appear in entries are thought of as instances of more general types. The notion of type underpins the formal method and is an aid in proofs of correctness. However, there will always be more types that must be dealt with than there are entry or attribute types. For example, a common operation on the blackboard is to form sets of entries (a solution island is an example of a set, it should be remembered). The entries may be from different abstraction levels, and *a fortiori* have different types. The set which is composed of all entries on the blackboard containing a specified attribute whose value satisfies some predicate will also contain entries of different types. In the last case, the operation to find all these entries usually returns the set of entry identifiers, its type being (usually) defined as:

$$f: B \times ((A \times V) \rightarrow \mathbf{B}) \rightarrow E$$

where B is the type of the blackboard, $(A \times V) \rightarrow \mathbf{B}$ is the type of the predicate (A is the type of attributes, V that of values, and $\mathbf{B}$ is the set of truth-values), and E is the set of entry *identifiers*. However, by the argument above, the objects 'named' by elements of E are entries, and they may have different types. In other words, the f operation which finds those entries which satisfy a particular predicate or relation has a sum type as its range. A simple analysis of the types produced by the domain model may not reveal the existence of these other types: that is, these more complex types may only become evident when Knowledge Sources have been produced.

The view of the blackboard and its contents that results from a formal analysis of the domain (particularly one of the kind given above) leads to a strongly typed view of the blackboard database. In addition, various operations defined over the blackboard are also typed. The use of rich type systems has not yet been explored in the context of AI systems, and the consequences for the construction of reliable AI systems has yet to be fully explored (although MXA [28] employs a type system inherited from Pascal).

Returning to the method, we must consider the role of formal proof. In common with other formal methods, for example VDM [27], Z [34] or [29], the current method requires as much formality as possible. The basis of the development of the specification, the domain model, is expected to be relatively rich in axioms (particularly so if relations are also defined), thus allowing a potentially enormous number of proofs. Randell [33] points out that many such models (for example, those to be found in [26]) suffer from the problem that their consequences are not drawn out: they seek as many axioms as possible in order to cover their chosen domain without determining the consequences of the axiom sets. Randell therefore urges a smaller axiom set and the increased use of proof. In the conversion of domain model sentences to blackboard structures, the current method also urges a considerable amount of proof—fortunately, many of the proofs are expected to be quite trivial. Specifically, the proofs fall into two categories:

- Refinement proofs, and
- Realization proofs.

Refinement proofs are of the kind familiar from the software engineering literature: they serve to demonstrate that a less abstract structure is adequate to represent a more abstract one. Realization proofs are those which show that it is possible to construct an object which satisfies a particular set of axioms. For the former proof category, it is necessary to show that the domain model's representations map satisfactorily onto the representations of the blackboard. Given the definitions of types and relations in the domain model, it is necessary to show that their representation as entries (which might be viewed as mappings from a set of attributes to a set of values) adequately represents the range of the unary predicate. For the latter category, it is necessary to show, for example, that if an attribute is to be filled by a particular type of entry, then that entry can exist on the blackboard (a stronger version is that of showing *how* to construct the required entry).

4 Knowledge Sources and Control

The development of Knowledge Sources and control régimes is much closer in spirit to specification in the more conventional sense. In order to develop these aspects of the system, it is necessary to introduce the concepts of blackboard event and blackboard state. A blackboard event is an alteration to an entry or the posting of a new entry on the blackboard. The blackboard state can be thought of as being the contents of the blackboard at any time. Knowledge Sources can be considered to be objects that transform the blackboard. The control structure can be thought of as a mechanism for determining which changes to make at any one time. The point of the control structure is to bring the blackboard into such a state that one or more goals is satisfied.

For reasons of space, and because we want to present our ideas in a form that is as immediately relevant to blackboard systems as possible, we are unable to address the relationships between control in blackboard systems and in AI systems in general. Instead, we are content with outlining the basic problems and proposing what appears to be a relatively coherent approach to the formalization of control: this approach is intended to be suited to the transition from abstract specification to implementation. Bachimont [1] proposes a logical framework within which to reason about coherence and convergence: the relationships between the current proposal and Bachimonts are worth exploring at a later date.

4.1 Knowledge sources

In the formal method, the same analytic technique is employed as in the informal one: Knowledge Sources are viewed as inferential tasks. Blackboard events determine, in conjunction with the control structure, which Knowledge Sources (strictly, Knowledge Source instances) to execute in order to transform the blackboard state. Once major inferential tasks have been identified, it is necessary to determine when to apply them: partly, this is the role of events (we return to this point below). Major inferential tasks must, of course, be ones that are natural as far as domain experts are concerned.

Knowledge Sources are treated as inferential tasks. This relates to the domain model in the following way. If the set of possible inferences has been determined in advance, the inferences performed by Knowledge Sources will be a subset of these. Of course, for a large set of axioms, the set of consequences will be very large, so a complete enumeration will be unlikely for any practical application. However, as part of the domain model construction process, it is suggested that the principal inferential moves be determined. One way of stating these is as theorems or lemmata, with a Knowledge Source taken to stand for one (or possibly more) of these. It is important to note which axioms will be used in a proof. By the Deduction theorem, a theorem in logic can be thought of as an implication (biconditionals being expanded in the usual way—this generates two lemmata), the relationship to inference is immediate: indeed, one can think of proofs of theorems and lemmata as being proofs of derived inference rules, possibly of a domain-specific nature—once these rules have been proved, they can later be used. Of course, not all the theorems that can be proved in the domain model will be suitable as bases for Knowledge Sources: theorems concerned with properties of relations and unary predicates—theorems about the representation—may not turn out to be very suitable for conversion into Knowledge Sources; theorems that deal with the existence and transformation of known objects may, though, form the basis from which Knowledge Sources are developed.

Once theorems of the right kind have been identified, their proof can be attempted. This may be regarded as a separate step in the construction of the system: this is because it deals with its inferential properties and not with representation. A proof of a Knowledge Source (i.e., of the theorem which initially states the transformation representated by a Knowledge Source), if sufficiently formal, will be based on domain axioms (as well as general results from logic). This fact is of importance for two reasons:

- It tells us which domain structures are required in order to perform the large inference represented by the theorem;
- It tells us what the inferential steps in the solution are (it tells us about subgoals, for example).

A consequence of these facts is the fact that the proof determines, in outline, the construction of the Knowledge Source.

In the informal method, the decomposition of the initial Knowledge Source set was achieved by attempting to refine them into smaller units and by looking for common subtasks, a process which may suffer from the fact that important tasks may be overlooked. In the formal method, although this can be done, the refinement of large Knowledge Sources can be achieved by lemma discovery. If a number of theorems rest on a small set of lemmata, it may make sense to view the lemmata as independent theorems: this leads naturally to considering them as Knowledge Sources. However, given the formal nature of the domain model, the development of Knowledge Sources on the basis of theorems appears more natural. Furthermore, the formal approach to Knowledge Source development solves a problem with the informal method: the problem that the representation was, in some ways, separated from Knowledge Sources and incrementally adjusted as a result of Knowledge Source definition. In the present case, this defect does not appear because the identification of Knowledge Sources with theorems in the domain theory closely relates the two: Knowledge Sources are now defined as inferences over domain structures, a property supported by proof. By defining Knowledge Sources in this way, those aspects of the domain which are relevant or necessary to the Knowledge Source can, in effect, be read off from the proof.

The refinement of Knowledge Sources from important theorems in the domain model to code becomes relatively straightforward. When domain objects are represented as entries composed of attribute-value pairs, Knowledge Sources can be represented as operations acting on these structures. The form that Knowledge Sources take will be fairly close to the structures used in [6]. What is then necessary is to prove that the structures that result from this specification exercise are equivalent to the theorems. In other words, what is necessary is to prove that, given a theorem of the form

$$\vdash \phi \Rightarrow \psi$$

that the corresponding Knowledge Source, when started in a state that satisfies a refinement of ϕ will terminate yielding a state that satsifies a refinement of ψ. Put this way, the process of verifying a Knowledge Source is very similar to verifying a procedure. If, on the other hand, the Knowledge Source is produced by refinement, the verification is immediate (it is an immediate consequence of the refinement proofs).

Knowledge Sources have, so far, been unrelated to blackboard events. The proposal for identifying Knowledge Sources has been based entirely upon the ontology of the domain. That is, it is closely connected to the predicates, relations and constants that appear in the domain model. Blackboard systems, though, have an evolving and dynamic structure. Above, we saw that a theorem in the domain model of the form

$$\vdash \phi \Rightarrow \psi$$

can be thought of as representing a Knowledge Source which, when started in a state that satisfies ϕ will terminate with a state ψ. As the state of the blackboard evolves, it is necessary to determine *when* a Knowledge Source will be activated: this is where events become important.

The assumption about blackboard events is that the relevance of Knowledge Sources can be partially determined by small changes to the blackboard state. Some Knowledge Sources will always be applicable, but most others will only apply once the state has been transformed in some way. Events are intended to signal changes of state (thus, not all events will trigger Knowledge Sources, for the changes represented by these events will be irrelevant as far as inference is concerned). The account given so far does not take change into account.

There are two ways in which state change can be catered for:

1. It can be considered as an efficiency-improving technique and its introduction postponed until the major parts of Knowledge Sources (their state-based preconditions and actions in a BB*-like model) have been specified, or

2. It can be considered from the start.

Our view is slightly equivocal: we view events as important sources of information about the way the current solution is developing, while at the same time seeing events as a way of making the system faster. Events are also related to the problem of partial information in blackboard systems: at any stage in the development of a solution, the information available to the Knowledge Sources of the system may only be partial—inference is needed in order to complete the solution. In the next subsection, we consider the formal specification of control in a blackboard system and relate control to the focus-of-attention problem: this will allow us to integrate events with the remainder of the specification in a natural fashion.

4.2 Control

Control, as is noted in [8] is possibly the hardest aspect of building a blackboard system; it is also one of the most important, for, without an adequate control structure, the system may never find acceptable solutions. The role of the control structure is to select currently applicable Knowledge Sources for execution. This selection process is based on the current state of the blackboard, currently applicable Knowledge Sources and the (sub)goal to be satisfied.

In order to see how to develop a control structure, it is instructive to examine the purpose of control. We view control as being expressed in terms of plans, strategies, tactics and goals (see [23] for a careful and detailed exposition). The purpose of a control structure is to satisfy a goal: it matters little whether the goal is satisfied by forward or backward reasoning. For blackboard systems, a goal can be expressed as a blackboard state to be achieved. The state can be expressed as a sentence in the first-order language used to construct the domain model for the reason that a state description amounts to a description of those entities which must be present on the blackboard in order that the goal be satisfied.

In real systems, one would expect to express the specification of a goal as a logical normal form. Once expanded, the goal statement says what objects must be present on the blackboard in order satisfy the goal. As is usual, the satisfaction of a goal may require the joint satisfaction of subgoals. The problems for blackboard systems are: detecting when a strategy is applicable and determining when it has succeeded (or failed, which is equally as important). A further question is that of deciding what to do when more than one strategy or tactic is applicable.

By the above, the application of a strategy or tactic is dependent upon some blackboard state obtaining. Matters are somewhat more complicated by the fact that a state which should cause the application of a strategy may only be partially defined at runtime. For the time being, we will ignore this case in favour of the simpler one in which states are totally defined (we can get round the problem, in any case, by ensuring that there are sufficient preconditions for the application of a strategy). Furthermore, given the possibility that other strategies may be simultaneously active, it is necessary to characterize the precondition in such a way that there is no interference between strategies.

The prescriptions of a strategy impose a temporal ordering on blackboard changes. In other words, when each blackboard change is viewed as an event, there is an ordering imposed by the strategy currently in use. For each step, it is possible to describe the state which results from the application of a step in a strategy: that is, we can associate pre- and post-conditions with strategies (and with tactics). These pre- and post-conditions, we suggest, can be expressed as formulae in a temporal logic (e.g., those in [14]). For each precondition, ρ and each postcondition ξ, we can then assert that:

$$\rho \Rightarrow \Diamond\xi \tag{1}$$

where $\Diamond$ is the 'eventually' operator of the temporal logic. A strategy which contains a number of separate tactics can be expressed as something of the form:

$$\phi_1 \wedge \bigcirc\phi_2 \wedge \cdots \wedge \underbrace{\bigcirc\cdots\bigcirc}_{n \text{ times}}\phi_n \tag{2}$$

where $\bigcirc$ is the 'next' operator of the logic (the temporal operator binds tightest in this example and in the one above) and the ϕ_i are formulae expressing the pre- and post-conditions of each step of the strategy (i.e., they are implications).

Within this framework, we can account for blackboard events is a reasonable way. Events are caused by alterations to the blackboard state: with each event, the (logical) description of the state changes. We assume that changes cannot be reversed: i.e., we assume that all blackboard operations are *monotonic*, and so we do not need to bother about nonmonotonic inference, or about individuals coming into and going out of existence. Events are caused by Knowledge Source actions, so we can associate a temporal formula with each action stating what the next state of the blackboard will be. That is, with any action A, we can associate a temporal formula of the form:

$$A \Rightarrow \bigcirc\phi \tag{3}$$

where ϕ is a (possibly temporal) sentence that describes the next state of the blackboard.

At this point, it is worth pointing out that we have a choice as to the level at which we describe the temporal consequences of each action. We can do this by producing a temporal formula to accompany each theorem derived from the domain model, or we can do it when we have specified the Knowledge Sources from their theorems. If the Object Z specification language [10] is being used, temporal formulae can be included in the specification rather than with domain-model theorems. The question of the best place to include temporal formulae remains open.

The reader may have noted that we have used temporal formulae with what amounts to two different scopes. In the specification of strategies and tactics, we feel free to use the $\bigcirc$ operator to mean the next step in the strategy; in actions, we are using a finer notion of what next means—in this context, it means the next state. Furthermore, because of the possibility of concurrent execution of strategies, the steps of one strategy may be interleaved with steps of another. This causes no problems, for we intend there to be a hierarchy of descriptions with those of the form (2) being the most general, and those of the form (3) to be the most specific. In other words, a strategy defines a sequence of abstract actions that is implemented as a sequence of events: that is that the statement of a strategy in the form of (2) can be viewed as a general specification of *that* strategy (i.e., it does not refer to any other strategy). It must also be noted that there is no real problem with the use of the same temporal operator with different scopes: the aim is to refine strategies to the level of events, and, as long as the difference in the view of time taken at different levels of specification is kept in mind, there should be no confusion.

It should not be thought that the ultimate goal of this refinement process is the determination of deterministic sequences of actions (or sequences of Knowledge Source executions). For some applications, this might be suitable, but, in the general case it is not: this is because search in blackboard systems is assisted by opportunism. Without opportunism, it can be argued, some problems will never be solved because of the impossibility of exhaustive search. What the aim of the refinement process boils down to is the formal statement of conditions which must obtain before strategies can apply and the formal statement of the classes of states which count as satisfying their goals. The possibility of concurrent strategies makes the development of deterministic sequences difficult in any case, for there may be interactions which prohibit or delay the continuation of another strategy or tactic (proofs of interaction are, note, possible).

It remains to relate the control structure to the design of the blackboard database. Above, it was noted that the blackboard's abstraction hierarchy defines a general plan for the solution of problems. From the above discussion of control, the reader may believe that the blackboard's organization has been forgotten: this is not the case, and careful examination of the statement of a strategy shows this. Each strategy is composed of formulae which are expressed in terms of the representation used in the system. For example, it may contain unary predicates which define entry types. Such a connection closely relates strategies and tactics to the blackboard, and this property can be used to define plans for the general control of the system. In the case of HEARSAY-II, we noted above that the general plan (movement of solutions) was bottom-up as far as the Lexical level, and opportunistic thereafer.

By applying the approach advocated above, but this time in a more abstract way, the general control plan for the system can be specified (of course, the control plan would normally be defined first, but we have presented things in the reverse order for the sake of clarity).

5 Conclusions

In this paper, we have proposed a formal method for the specification of blackboard systems and have also examined some issues raised by it. The method is a development of an informal one stated in [8]. The method is characterized by a completely formal approach, and consists of the following steps:

1. Development of a *formal* domain model. This theory consists of the formal description of the objects of the domain and the relationships which obtain between them. The domain model also requires theorems and lemmata that represent the major inferential steps that are needed in order to solve certain problems in the domain. The domain model can be treated to logical analysis, and the consequences of the definitions can be determined by proof

2. The refinement of the formal description of the domain's objects into entries and attribute-value pairs. This refinement begins with the definition of the blackboard abstraction hierarchy and the abstraction relation which defines the partial ordering over abstraction levels. The unary predicates of the domain theory are interpreted as types of entries, and relations are interpreted as attributes. This refinement determines the types of entry that each abstraction level will hold and also determines the range of attribute types that will appear in each entry type.

3. The refinement of the theorems of the domain theory into Knowledge Sources. Knowledge Sources are considered to be the major inferential tasks that the system will eventually perform. The refinements are based upon proofs of the theorems in the domain model: these proofs are seen as providing valuable information about the resources Knowledge Sources will eventually need. The refinement process retains links between the domain model's structures and Knowledge Sources: these connections may easily be lost using informal techniques. Refinement from the domain model further serves to identify

4. The specification of the control aspects of the system using temporal logic. The control structure is defined in terms of the goals the system must satisfy. The implementation of the control structure involves the association of temporal formulae with Knowledge Source actions. The intermediate refinement is in terms of temporal logic. The specification of the control structure depends upon the objects in the domain model because the non-logical vocabulary with which this refinement process starts is the domain model.

Above, we have emphasized the proposition that the blackboard is to be considered as being composed of *typed* objects, and that the range of types is not entirely determined by the domain model for the reason that some blackboard operations may require the introduction of new types[6]. We believe that the explicit introduction of types into blackboard systems is to be welcomed (types were present in the HEARSAY-III system [2, 12]), whether one chooses to use a type-theoretic logic such as [4, 17, 18] or a many-sorted logic with a classical interpretation. The use of typed objects at runtime can reduce errors, and the specification in terms of typed objects reduces the risk of faulty reasoning.

The second point to be made concerns temporal logic. In the last section, we merely introduced the idea of temporal logic without saying which system we prefer. Given the range of logics and the basic choices as to temporal ontology (discrete versus continuous time, branching versus linear time), we do not as yet have a clear view of which is the best. The question of temporal logics that allow reasoning about the past is also open (it seems that the ability to reason about past actions could profitably be applied to blackboard systems). Questions about temporal logic will only answered by more work, both on the logical systems themselves and by their application to blackboard systems.

Also, and this is more of an aside, the formal method allows the development of formal *meta-theoretic* models of the system. We have not examined this in detail at present, but note it as an interesting possibility, particularly for expressing a semantics.

We must next ask whether all of this is worth the effort. Clearly, it is a difficult exercise to follow the method exactly: it has already been shown that the construction of *complete* domain models of the sort recommended by Hayes in [20] is exceptionally hard, perhaps impossible. This fact should not deter one, however, for the aims of the two projects are different. In Naive Physics, the aim is the production of a model that captures *all* intuitions; for us, the problem is that of producing a formal domain model that is *sufficient* to allow the development of the system—a much more modest requirement.

At this point, it is natural to wonder whether the proposed method will be adequate to support the development of large-scale systems. Blackboard systems are best suited to highly complex domains that require the application of large amounts of knowledge. We have advocated the formal specification of the problem domain as a first step in the development of a system. In the last paragraph, we argued that the range and depth of the domain model will be different for a blackboard system than it would be for a Naive Physics system. The last fact entails that we expect somewhat less from our domain models than would others: this has the implication that domain models for blackboard systems should be easier to produce. The development of the control component will, of course, be a complex and difficult

[6] We believe that the concept of dependent types affords the best analysis, but have not taken up the idea here because the concept may be unfamiliar to the reader.

activity, but it is often the most complex component in a blackboard system that is developed informally. We believe, though, that our approach to control brings benefits: in particular, it is based upon the idea of at first being as abstract as possible and then becoming increasingly more concrete (a method that is strongly reminiscent of formal specification of a conventional kind).

Next, there is the problem of proving the large number of results that are required to support the derivation of the blackboard structure and the Knowledge Sources. Certainly, there will be many theorems to prove, but we expect that many results will require only moderate effort. Finally, there is the problem of reasoning about time: another complex process, requiring much proof. It may be argued that the state of automatic theorem-proving programs is insufficient to support this amount of proof: the reply is that one need not do it all by hand, and that machine *support* is available—for example, LCF [15, 32] or HOL [16].

In support of all this effort, we offer the following remarks. The first is of practical. If the system that is to be built is intended to be safety-critical, it must be realized that lives may depend upon its correct functioning. The best guarantees of correct functioning that can be given must be given: the best possible guarantee is that the system is mathematically correct. If the system is to be subject to real-time constraints, a similar argument (although, perhaps, with less force) applies. The method proposed above appears to provide the construction of blackboard systems with a method which allows the best possible guarantees of correct functioning. Furthermore, such a method also has the advantage of providing unambiguous documentation to be used in later modification and in maintenance. Finally, the approach advocated above allows the separation between the progam and the theory which it is supposed to implement: the clear distinction between the two is something that AI reseach currently lacks.

References

[1] Bachimont, B, A logical framework to manage coherence and convergence in blackboard architectures: a proposal, *Proc. Fifth Blackboard Systems Workshop*, Anaheim, CA, 1991.

[2] Balzer, R., Erman, L., London, P. and Williams, C., HEARSAY-III: A Domain-Independent Framework for Expert Systems, *Proceedings of the First Annual Conference on Artificial Intelligence*, pp. 108-110, 1980.

[3] Bisiani, R., Alleva, F., Forin, A., Lerner, R., and Bauer, M., The architecture of the AGORA environment, in Huhns, M. N. (ed.), *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, pp. 99-117, Pitman, London, 1987.

[4] Constable, R.L., et al., *Implementing Mathematics with the Nuprl Proof Development Systenm*, Prentice Hall, New Jersey, 1986.

[5] Craig, I.D., Wilson, D. and Richards, A., *CONFER*, Research Report No. 103, Department of Computer Science, University of Warwick, 1987.

[6] Craig, I.D., *Formal Specification of Advanced AI Architectures*, Ellis Horwood, Chichester, England, 1991.

[7] Craig, I.D., The Role of Formal Specification in Real-time AI, *IEE Colloquium on Rule-based Planning and Control*, IEE, Savoy Place, London, October, 1991.

[8] Craig, I.D., *Blackboard Systems*, Ablex Publishing Corp., Norwood, NJ, *in press.*

[9] Craig, I.D., A Reflective Production System, *Kybernetes*, to appear.

[10] Duke, R., King, P., Rose, G. and Smith, G. *Object Z*, Technical Report No. 91-1, Software Verification Research Centre, Department of Computer Science, University of Queensland, Queensland, Australia, 1991.

[11] Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R., The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty, *ACM Computing Surveys*, Vol. 12, pp. 213-253, 1980.

[12] Erman, L.D., London, P. and Fickas, S., The Design and an Example Use of HEARSAY-III, *Proceedings of the International Joint Conference on Artificial Intelligence 7*, Vol. 1, pp. 409-415, 1981.

[13] Feigenbaum E.A., Nii, H.P., Anton, J.J. and Rockmore, A.J., Signal-to-signal transformation: HASP/SIAP case study, *AI Magazine*, Vol. 3, pp. 23-35, 1982.

[14] Goldblatt, R., *Logics of Time and Computation*, CSLI Lecture Notes No. 7, Center for the Study of Language and Information, Stanford University, 1987.

[15] Gordon, M, Milner, R. and Wadsworth, C., *Edinburgh LCF*, LNCS No. 78, Springer-Verlag, Berlin, 1979.

[16] Gordon, M., *A Proof Generating System for Higher-order Logic*, Technical Report No. 103, Computer Laboratory, University of Cambridge, 1987.

[17] Harper, R., Honsell, F. and Plotkin, G., *A Framework for Defining Logics*, Report ECS-LFCS-87-23, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1987.

[18] Harper, R., *An Equational Formulation of LF*, Report ECS-LFCS-88-67, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1988.

[19] Hayes, P.J., The Logic of Frames, in Metzing, D. (ed.), *Frame Conceptions and Text Understanding*, pp. 46-61, Walter de Gruyter and Co., Berlin, 1979.

[20] Hayes, P.J., The Second Naive Physics Manifesto, in Hobbes, J. and Moore, R.C. (eds.), *Formal Theories of the Commonsense World*, pp. 1-36, Ablex Publishing Corp., Norwood, NJ, 1985.

[21] Hayes-Roth, B. and Hayes-Roth, F., A Cognitive Model of Planning, *Cognitive Science*, Vol. 3, pp. 275-310, 1979.

[22] Hayes-Roth, B., *The Blackboard Architecture: A General Framework for Problem Solving?*, Report HPP-83-30, Heuristic Programming Project, Computer Science Department, Stanford University, Palo Alto CA, 1983.

[23] Hayes-Roth, B., A Blackboard Model for Control, *Artificial Intelligence Journal*, Vol. 26, pp. 251-322, 1985.

[24] Hayes-Roth, B., Garvey, A., Johnson, M.V. and Hewett, M., *BB*: A Layered Environment for Reasoning About Action*, Technical Report No. KSL 86-38, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1986.

[25] Hayes-Roth, B., Hewett, M., Washington, R., Hewett, R. and Seiver, R., Distributing Intelligence within an Individual, in Gasser, L. and Huhns, M. (eds.), *Distributed Artificial Intelligence Volume Two*, pp. 385-412, Pitman, London, 1989.

[26] Hobbes, J. and Moore, R.C. (eds.), *Formal Theories of the Commonsense World*, Ablex Publishing Corp., Norwood, NJ, 1985.

[27] Jones, C.B., *Systematic Software Development using VDM*, Prentice Hall, England, 1986.

[28] Lakin, W.L. and Miles, J.A.H., *A Blackboard System for Multi-Sensor Fusion*, Technical Report, ASWE, Portsdown, England, 1984.

[29] Morgan, C., *Programming from Specifications*, Prentice Hall, Hemel Hempstead, England, 1990.

[30] Nii, H.P., The Blackboard Model of Problem Solving, *AI Magazine*, Vol. 7, pp. 38-53, 1986.

[31] Nii, H.P., Blackboard Systems Part Two: Blackboard Application Systems, *AI Magazine*, Vol. 7, pp. 82-106, 1986.

[32] Paulson, L., *Logic and Computation*, Cambridge University Press, 1987.

[33] Randell, D.A., *Analysing the Familiar*, Ph.D. Thesis, Department of Computer Science, University of Warwick, 1991.

[34] Spivey, J.M., *The Z Notation: A Reference Manual*, Prentice Hall, Hemel Hempstead, England, 1989.