Integrating BB1-Style Control into the Generic Blackboard System

Philip M. Johnson
Daniel D. Corkill
Kevin Q. Gallagher

June 1987
Coins Technical Report 87-59

Presented at the *Workshop on Blackboard Systems*
Seattle, Washington, July, 1987.

Abstract

This paper describes GBB1, a control shell written for the Generic Blackboard Development System (GBB) that emulates the BB1 blackboard model of control. GBB1 exploits the facilities of the GBB development system to provide both high-level constructs for knowledge source definition and efficiency in the generated control shell and applications code. After overviews of the BB1 control model and GBB, we describe the facilities of the GBB1 control shell and their use in a simple application.

1 Introduction

BB1 [1] and the Generic Blackboard Development System (GBB) [2] address significantly different research goals. The focus for BB1 is on blackboard-based control as well as the development and integration of diverse reasoning methods. This emphasis has motivated the development of facilities like language frameworks, explanation generators, and goal-directed reasoning. BB1 has been described as "fast enough" for these investigations, and thus contains few mechanisms for optimizing the efficiency of the generated control shell and applications code. In contrast, the focus for GBB is on providing a general blackboard development facility which produces efficient systems "tuned" to the specific application domain. GBB accomplishes this by providing a highly structured and efficient blackboard database subsystem and a choice of control shells.

This paper reports our work on GBB1 [3], a control shell for GBB based on the BB1 blackboard model of control. Its goal is to provide high level constructs for knowledge source definition and an agenda-based execution cycle facility without sacrificing execution efficiency in the application and control shell code. While GBB1 does not implement all of the facilities of BB1, it does demonstrate how the GBB database management facilities provide the substrate for the development of efficient high level control abstractions.

We begin by briefly reviewing the BB1 model of control and the GBB blackboard database management facilities. Next, we provide an overview of the GBB1 control shell, illustrating how the GBB facilities were used to implement the BB1 control model. We conclude by describing a simple application system, and demonstrating how greater efficiency can be achieved by exploiting the blackboard structuring and representation facilities of GBB.

2 The BB1 model of control

2.1 BB1 knowledge sources

BB1 has three types of knowledge sources: *learning*, *domain*, and *control*.

Learning KSs manipulate the information found on the *Knowledge Base* blackboard, which contains the basic facts about the problem domain and all the knowledge sources in the system. This blackboard provides the "long term memory" of the system. For organizing these facts, BB1 provides specialized linking mechanisms

This research was sponsored in part by the National Science Foundation under CER Grant DCR-8500332, by a donation from Texas Instruments, Inc., by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under Contract NR049-041, and by the National Science Foundation under Support and Maintenance Grant DCR-8318776.

to create a *generic-example-instantiation* hierarchy of knowledge. In this hierarchy, general knowledge about objects and their properties are stored at the generic level, examples of the object in a specific problem domain are stored at the example level, and instances of the object as used in a potential solution to a problem are stored at the instantiation level.

Domain KSs describe the actions to be taken in problem solving. These actions manipulate the objects on the *Domain* blackboard. This blackboard thus provides the "short term memory" of the system, where hypotheses about solutions are posted and manipulated.

Control KSs direct the problem solving activity by incrementally constructing a control plan on the *Control* blackboard. This control plan is used by the BB1 scheduler to decide which KS to execute next. BB1 Control Knowledge comes in three flavors:

- *Strategies* provide abstract, general descriptions of problem solving behavior to be performed during relatively long problem-solving time intervals.
- *Foci* are a part of one or more strategies, and prescribe more specific problem solving behavior as well as a goal that describes when the behavior should conclude.
- *Heuristics* implement a particular focus by defining the functions that describe the criteria to be used in evaluating how well the actions of a KS fit in to the goal of the focus.

2.2 The BB1 Execution Cycle

In general, the basic problem solving cycle in BB1 begins with the *execution* of a KS's *actions* which manipulate blackboard objects and generate *BB1 events*. These events may cause other KSs to become *triggered* by satisfaction of their triggering conditions. A triggered KS creates *instantiations* of itself called *knowledge source activation records*, or KSARs. Each KSAR waits for its *preconditions* to be satisfied, at which point it becomes *executable*. From the set of executable KSARs, the control shell selects one whose actions appear to best fit the current problem solving strategy and executes those actions. In addition, KSARs can be removed from the triggered and executable agendas by the satisfaction of their *obviation conditions*.

Here is a more explicit description of the BB1 execution cycle, which can be divided into interpretation, agenda maintenance, and execution:

1. *Interpretation* The interpreter executes the actions of 1 KSAR. On the first cycle the user supplies the KS whose actions are to be executed, while on the remaining cycles the scheduler chooses the KSAR. Interpretation consists of the following steps:
   (a) Confirm that the KSAR's preconditions are still satisfied.

   (b) Confirm that the KSAR's obviation conditions are not satisfied.
   (c) Evaluate and bind the internal variables of the KSAR.
   (d) Execute the actions of the KSAR.
   (e) Generate BB1 events caused by the KSAR actions.

2. *Agenda Maintenance* The agenda maintainer updates the agendas of triggered, obviated, and executable KSARs. This is done in the following way:
   (a) If the current cycle is a *precondition recheck cycle*, move all executable KSARs with unsatisfied preconditions back to the triggered agenda.
   (b) Move all triggered KSARs with satisfied preconditions to the executable agenda.
   (c) For each KS triggered by the BB1 events of the previous cycle, generate one or more KSARs depending upon the context variables, and make each KSAR triggered or executable depending upon whether its preconditions are satisfied.
   (d) If the current cycle is an *obviation check cycle*, move all triggered and executable KSARs with satisfied obviation conditions to the obviation agenda.

3. *Scheduling* The scheduler chooses the next KSAR to be executed from all those on the executable agenda in the following way:
   (a) Rate each KSAR against each heuristic currently active in problem solving. The rating of a KSAR against a heuristic is done in two steps:
      i. Rate each action of the KSAR using the heuristic's *action rating* function.
      ii. Rate the overall effect of the KSAR using the heuristic's *integration rating* function.
   (b) Determine the priority of each KSAR using the *priority* function, which orders the KSARs using the rankings they received from all the active heuristics.
   (c) Choose the KSAR to execute using the *recommendation* function. This selects from among the KSARs with the highest priority the single one to execute. It also normally gives the user the option to override the system's recommended KSAR with a user-selected KSAR.

This execution cycle ends when either the *termination function* returns a non-NIL value (indicating the problem has been solved), or when there are no longer any KSARs on the executable agenda.

3 The GBB database management facilities

3.1 The structure of the blackboard database

In GBB, the blackboard database consists of a set of *blackboards.* Blackboards are hierarchical structures composed of atomic blackboard pieces called *spaces.* In addition to being composed of spaces, a blackboard can also be composed of other blackboards (themselves eventually composed of spaces.)

Spaces are highly structured storage, organized by *dimensions.* Dimensions allow the location of objects on the space to be represented in terms natural to the application domain. For example, in the control shell, the space that stores BB1 events has the dimension *execution-cycle.*

GBB supports two types of dimensions: *ordered* and *enumerated.* Ordered dimensions use numeric ranges which support the concept of one object being nearby another object. Continuing the above example, specifying the execution-cycle dimension as ordered provides a natural manner of specifying the BB1 events which occurred during the "last" execution cycle.

In contrast, enumerated dimensions consist of a fixed set of labeled categories, called the *label set.* For example, the space storing KSs employs a labeled dimension *ks-type*, with the label set {*domain, control*}.

3.2 The structure of blackboard objects

In GBB, the objects stored in the blackboard database are termed *units.* A unit is an aggregate data type similar to those created using the Common Lisp **defstruct** macro, but only units can be placed onto blackboard spaces. All of the BB1 objects, such as KSs, KSARs, and events are implemented in the control shell as GBB units. Units are defined with five attributes: *slots*, *links*, *path-indexes*, *paths*, and *dimensional indexes.*

Slots are analogous to `defstruct` slots. Many of the BB1 KS attributes, such as `Trigger-conditions` and `Cost` are implemented as GBB slots.

Links are special purpose slots which contain links between units. Links can enforce one-to-one, many-to-one, and many-to-many mappings between units. The control shell uses links to connect KSs to the KSARs they generate.

Paths and path indexes are used to determine what spaces the unit should be stored on. One of their uses is to specify which agenda each KSAR should be located on.

Finally, dimensional indexes are used to determine where on its space(s) the unit should be located. For example, dimensional indexes are used to place events on the events space according to their execution cycle number.

4 Implementing the BB1 control model in GBB.

The purpose of the GBB1 is to provide a BB1-style control model to GBB users. GBB1 also illustrates how GBB's blackboard database machinery allows the implementation of efficient high-level control facilities. The following sections discuss the implementation of the GBB1 control shell. The first section describes how the full BB1 control model has been distributed into three layered subsystems in GBB1. In the second section, we describe how the blackboard database facilities are exploited in the implementation of the control shell.

4.1 Layering the BB1 control model implementation

GBB1 is not a monolithic control shell, but rather a layered set of three control shells which increasingly encompass the functionality of BB1. This provides two benefits to the application implementer. First, the additional overhead incurred by some parts of the BB1 control model can be eliminated if their functionality is not needed in the application context. Second, this approach allows the control model to be altered from the BB1 formulation if an alternative is more appropriate for the application.

The Execution Shell

The GBB1 Execution Shell is the core of the system, and provides the agenda-based execution cycle of BB1 as well as facilities for defining a single type of knowledge source. This layer does not provide the specific control knowledge sources and blackboard structures of BB1, or its knowledge representation constructs. The execution shell layer is appropriate for applications requiring a control model similar to the agenda-based facilities of the Hearsay-II Speech Understanding System [4], for example.

The KS Shell

The KS Shell incorporates the BB1 control knowledge sources and blackboard structures into the execution cycle facilities provided by the Execution Shell. This shell is useful to those who desire the BB1 model of control with different knowledge representation facilities.

For example, the Boeing Blackboard System (BBB) [5] implements the BB1 control model in KEE, which provides more extensive knowledge representation facilities than BB1's. However, there is a performance penalty for implementing the blackboard structures and objects in KEE. The KS Shell is designed to allow the application implementer to efficiently implement blackboard structures and objects, while allowing the use of other facilities to represent domain and world knowledge.

The GBB1 Control Shell

The top layer of the GBB1 system is the GBB1 Control Shell. This adds the inheritance facilities of BB1 to the KS Shell, thus providing the entire BB1 model of control.

4.2 Representing BB1 structures and objects in GBB

The GBB1 control shell implicitly defines a control blackboard structure to be added to the blackboard database along with any application blackboards defined by the implementer. The control shell blackboard is not defined until after all the knowledge sources have been defined, so that it may be structured for efficient retrieval of KSs and KSARs. The following sections describe the control shell blackboard and the objects stored on it.

4.3 The structure of the control shell blackboard

The KSARs, KSs, and events generated during problem solving are stored on the following spaces:

Triggered, Executable, Executed, Obviated These spaces implement the four BB1 agendas which store the KSARs generated during problem solving. Each of these spaces have a single enumerated dimension, *KSAR-phase*. Its label set consists of all the KSAR phases specified in the knowledge source definitions.

KSs This space stores the knowledge sources. In the KS shell as well as in the full control shell, this space is structured by an enumerated dimension *KS-type*, which has the label set {*domain*, *control*}.

Events This space stores the events generated during problem solving. This space is structured by an ordered dimension, *execution-cycle*, and an enumerated dimension, *triggering-space*.

The four agenda spaces are components of the blackboard **agendas**, and this blackboard, as well as the **KSs** and **Events** spaces are components of the topmost blackboard, **GBB1**. This is illustrated by Figure 1, where the solid boxes represent blackboards and the dotted boxes represent spaces.

4.4 The structure of the knowledge sources

Regardless of the "layer" of the GBB1 control shell selected, high-level constructs are provided for knowledge source definition[1]. The KS definition facility extends

[1]Unlike BB1, KSs are defined textually rather than through a specialized editor interface.

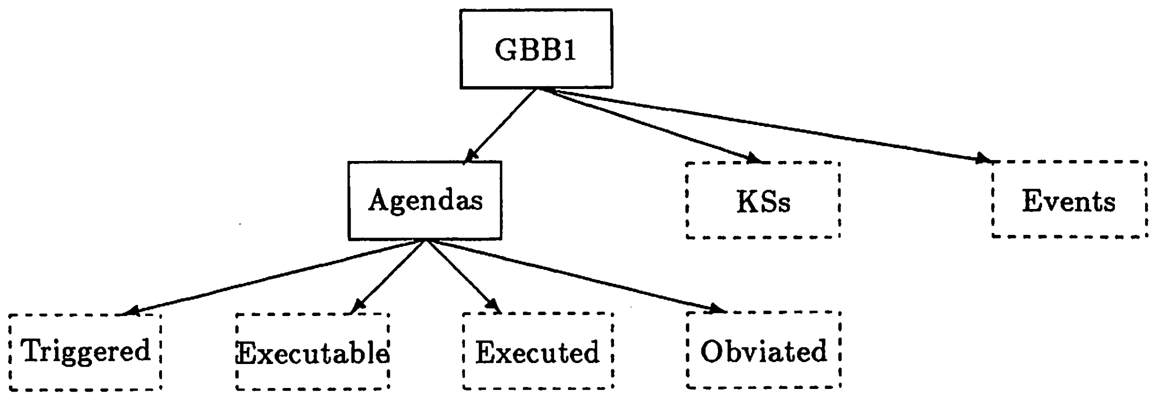

Figure 1: The GBB1 Control Blackboard.

the standard GBB syntax for blackboard unit definition with additional parameters useful to the BB1 control model.

Each KS definition normally defines two GBB unit types, one for the KS and one for the KSARs generated by this KS. However, it is sometimes desirable to "customize" the structure of the KSAR with additional slots or links. To allow this, an additional facility exists for explicit KSAR definition. When the KSAR is defined in this manner, the implicit definition of the KSAR structure by the KS definition facility is disabled.

Along with unit type definition, information useful to the structuring of the control shell spaces is collected during KS definition. For example, the KS and KSAR phase labels are accumulated, as well as information about the spaces of interest to the KS.

5 Efficiency considerations in GBB1

During each execution cycle, the BB1 control model requires every KS to be tested against every event generated, as well as testing every KSAR's preconditions and obviation conditions. This can "waste" significant amounts of problem solving time retrieving and checking KSs which won't be triggered or KSARs which won't become executable or obviated. The GBB1 control shell provides a number of facilities to reduce the total time spent retrieving blackboard objects and testing conditions.

Reducing the number of KSs to test against trigger events.

The BB1 control model requires that all of the KSs be tested against the trigger events. It is possible to "filter" the KSs by using the value of a global variable as the first triggering condition, but this still requires retrieving all of the KSs and performing one test per KS. The Boeing Blackboard System introduces "phases" of problem solving as a filtering method—only the trigger conditions of KSs active during the current phase are tested. GBB1 also provides problem solving phases for KSs, and structures the space storing KSs according to these phases. This allows efficient retrieval of only those KSs active during each execution cycle.

Reducing the number of events to test against KSs.

In BB1, all KSs are tested against all events generated during the previous execution cycle. Though KSs have a `From-BB` slot indicating the blackboard spaces of interest to it, this slot is only used as documentation. At blackboard definition time, GBB1 uses this information to structure the space storing events into dimensions corresponding to each of these spaces. At execution time, GBB1 will retrieve only those events generated on the spaces indicated by the `From-BB` slot for testing against the KS's trigger conditions.

Reducing the number and kinds of events generated.

BB1 provides two classes of events: object addition and object modification. The Generic Blackboard System provides seven classes of events: unit creation, unit deletion, slot update, link update, slot access, link access, and unlink events. This finer granularity allows more precise monitoring of changes to the blackboard, but can greatly increase system overhead if generated indiscriminately. To alleviate this problem while retaining the power of the smaller grain size, GBB1 allows the implementer to specify exactly which classes of events should be generated. This is accomplished by augmentation of GBB event specification and inheritance facilities by the control shell.

Reducing the number of KSARs to test against conditions.

Just as phases of problem solving are provided to reduce the number of KSs to test against triggering conditions, GBB1 also provides a separate set of problem solving phases for KSARs. This facility enables the application implementer to filter the set of KSARs to be retrieved and tested against preconditions and obviation conditions.

Reducing the number of preconditions and obviation conditions tested.

BB1 reduces the overhead associated with condition testing by the use of condition recheck intervals. If some but not all of a KS's preconditions (obviation conditions) succeed, only the failing preconditions (obviation conditions) will be checked until the next condition recheck cycle occurs. GBB1 extends this with a *stability* attribute for preconditions and obviation conditions. By default, conditions are *dynamic*, which results in the use of the precondition recheck mechanism described above. Specifying a condition to be *stable* indicates that once the precondition succeeds, it should never be tested again.

Reducing the calls to rating functions.

The functions used to rate KSARs also have a *stability* attribute indicating whether or not a KSAR previously rated by the function needs to be rerated. In addition to defining completely stable or completely dynamic rating functions, this facility also allows implementation of functions which are stable most of the time, but can become dynamic under exceptional conditions.

Compilation of conditions.

GBB1 compiles the list of trigger conditions, preconditions, and obviation conditions into normal lisp code. In contrast, BB1 employs a special interpreter for condition evaluation.

Reduction of consing.

Implementation of KS, event, and condition data structures as lists can lead to significant amounts of consing during problem solving. GBB1 employs the GBB space mapping and find facilities to minimize the creation of lists of KSs and events. In addition, the representation of preconditions and obviation conditions in GBB1 provides a non-consing implementation of recheck intervals and the stability attribute.

6 An Example: Building a "Hypothesis Pyramid"

The following simple example illustrates the style of system definition and some of the facilities available in GBB1. The structure of the hypothesis pyramid is a set of 10 blackboard spaces named `Space1...Space10`. At the beginning of execution, 100 hypotheses are created and placed on `Space1`, each with a `value` slot containing a random integer from 1 to 50. Whenever a hypothesis has "neighbors" (when there exists two hypotheses on the same space with values one greater and one less than its value) a new hypothesis is created with the same value on the next higher space. Thus three consecutively valued hypotheses on `Space1` will result in the creation of a new hypothesis on `Space2`, three on `Space2` will create one on `Space3`, and so forth. These actions slowly build a "pyramid" of supporting hypotheses until one is created on `Space10`, at which point execution ceases.

To illustrate the use of GBB1, we now describe two implementations of the hypothesis pyramid system. (For brevity, only the blackboard object definitions are actually included here.) The first implementation closely follows the BB1 control model. The second implementation employs several facilities for increasing the efficiency of the resulting system. Both systems use four blackboard objects: a Hyp object, which represents hypotheses and stores their values, an Initial-KS KS, which creates the initial 100 data points, a Pyr-builder KS, which looks for neighboring Hyps and creates new ones when this search succeeds, and Pyr-builder-ksar, the KSAR for the Pyr-builder KS.

Here is the definition of the Hyp blackboard object. In order to emulate the BB1 control model, we generate events whenever an instance of a Hyp is created or modified.

```
(define-gbb1-unit (HYP (:conc-name "HYP$")
                       (:print-function hyp-printer))

  "HYP (Hypothesis) Unit. The data for the pyramid."

  :EVENT-CLASSES
        (:creation-events :slot-update-events :link-update-events :unlink-events)
  :SLOTS ((value nil)
         (level 'space1)
         (response-frame nil))
  :LINKS ((supported-hyp (hyp supporting-hyps))
         (supporting-hyps (hyp supported-hyp))
         (triggered-ksar :singular (pyr-builder-ksar triggering-hyp :singular)))
  :DIMENSIONAL-INDEXES
        ((value value :TYPE :point))
  :PATH-INDEXES
        ((level level :type :label))
  :PATHS ((t ('blackboard level))))
```

The second object is the initial KS to be run by the control shell, which simply initializes the system by placing 100 Hyps on Space1:

```
(define-gbb1-ks INITIAL-KS

  "This KS simply creates some HYPs with random VALUEs on SPACE1."

  :ACTION-FUNCTION #'(lambda (ksar)
                       (declare (ignore ksar))
                         (dotimes (i 100)
                           (make-hyp :LEVEL 'space1
                                     :VALUE (random 50)))))
```

The following KS builds the pyramid. It is triggered by the creation of a Hyp, it becomes executable when there exists neighbors on either side of the triggering Hyp, and is executed if it is on the highest space. (This priority builds the pyramid faster, but with less of a "base." By reversing the priority so that the KSARs on lower spaces are rated higher, the pyramid will be built more slowly, but with a broader "foundation.")

```
(define-gbb1-ks (PYR-BUILDER (:conc-name "PYR-BUILDER$"))

  "This KS is triggered by the creation of a HYP, and generates one KSAR
   which will look for neighbors on either side of the triggering HYP."

  :TRIGGER-CONDITIONS
                     ((eq (gbb1-event$event-type *trigger-event*) :unit-creation)
                      (eq (type-of (gbb1-event$unit-instance *trigger-event*)) 'hyp))
  :PRECONDITIONS     ((lower-neighbor *this-ksar*)
                      (higher-neighbor *this-ksar*))
  :CONTEXT-SLOTS     ((triggering-hyp) (((gbb1-event$unit-instance *trigger-event*))))
  :KSAR-UNIT-TYPE  pyr-builder-ksar
  :ACTION-FUNCTION #'synthesize-hyp)
```

The KSAR for `Pyr-builder` is specified explicitly so that the slot `neighboring-hyps` and the link `triggering-hyp` can be defined.

```
(define-gbb1-ksar (PYR-BUILDER-KSAR (:conc-name "PYR-BUILDER-KSAR$"))

"This KSAR looks for the occurance of HYPs on both sides of the HYP
   which triggered it.  If both neighbors can be found, it creates a
   new HYP on the next higher space."

  :KS-UNIT-TYPE pyr-builder
  :SLOTS        ((neighboring-hyps nil))
  :LINKS        ((triggering-hyp :singular (hyp triggered-ksar :singular))))
```

This representation of the system reflects the standard BB1 control model. There are several inefficiencies in this representation, however. First, although `Initial-KS` is to be used only to initiate problem solving, it will be retrieved and tested throughout the execution of the system. Second, the preconditions of the `Pyr-builder` KSARs will be rechecked intermittently, even though they will always succeed after being satisfied once. Third, although the only event of interest is the creation of `Hyp` units, a great many extraneous events are produced and must be tested against the trigger conditions. Finally, if this application was a part of a larger system, the trigger conditions of the `Pyr-builder` KS would be checked against unrelated events generated on other blackboards.

Each of these problems can be addressed through the facilities of the GBB1 control shell. The following is a second, "optimized" representation of the blackboard objects in the system, which takes these issues into account (the `Pyr-builder-ksar` remains the same in each representation).

First, we modify the Hyp definition so that events are only generated when Hyps are created:

```
(define-gbb1-unit (HYP (:conc-name "HYP$")
                       (:print-function hyp-printer))

  "HYP (Hypothesis) Unit. The data for the pyramid."

  :EVENT-CLASSES
        (:creation-events)
  :SLOTS ((value nil)
          (level 'space1)
          (response-frame nil))
  :LINKS ((supported-hyp (hyp supporting-hyps))
          (supporting-hyps (hyp supported-hyp))
          (triggered-ksar :singular (pyr-builder-ksar triggering-hyp :singular)))
  :DIMENSIONAL-INDEXES
        ((value value :TYPE :point))
  :PATH-INDEXES
        ((level level :type :label))
  :PATHS ((t ('blackboard level))))
```

Next, we define two ks problem solving phases: :initial-phase and :build-phase. After creation of the initial Hyps, we reset the phase to avoid retrieval of irrelevent KSs during the remainder of problem solving:

```
(define-gbb1-ks INITIAL-KS

  "This KS simply creates some HYPs with random VALUEs on SPACE1."

  :ACTION-FUNCTION #'(lambda (ksar)
                       (declare (ignore ksar))
                       (dotimes (i 100)
                         (make-hyp :LEVEL 'space1
                                   :VALUE (random 50)))
                       (set-ks-phase :build-phase)))
```

The following definition of Pyr-builder is modified in three ways. First, we declare it to be relevent only during the :build-phase of problem solving. Second, we declare both preconditions to be stable. Lastly, we declare the blackboard of interest to this KS.

```
(define-gbb1-ks (PYR-BUILDER (:conc-name "PYR-BUILDER$"))

  "This KS is triggered by the creation of a HYP, and generates one KSAR
   which will look for neighbors on either side of the triggering HYP."
:TRIGGER-CONDITIONS
                    ((eq (gbb1-event$event-type *trigger-event*) :unit-creation)
                     (eq (type-of (gbb1-event$unit-instance *trigger-event*)) 'hyp))
:PRECONDITIONS      ((:stable (lower-neighbor *this-ksar*))
                     (:stable (higher-neighbor *this-ksar*))))
:CONTEXT-SLOTS      ((triggering-hyp) (((gbb1-event$unit-instance *trigger-event*))))
:KSAR-UNIT-TYPE     pyr-builder-ksar
:KS-PHASES          '(:build-phase)
:FROM-BB            (make-paths :paths '(hyp-blackboard))
:ACTION-FUNCTION #'synthesize-hyp)
```

These examples are certainly not representative of the large and complex systems intended for the GBB1 control shell. However, analysis of their run-time behavior does give a general idea of the kinds of execution time savings possible with the GBB1 control shell. In the 10 Space/100 Hyp experiment, the optimized version of the system generated only 30% of the events generated by the original version, and retrieved 80% fewer events from the blackboard database. The optimized version evaluated only 25% as many trigger conditions as the original. Finally, the optimized version evaluated only 20% as many preconditions.

Though the GBB1 control shell provides facilities for other types of reductions (such as for reducing the numbers of KSs and KSARs retrieved) this small example application does not serve to illustrate their benefits.

7 Conclusion

GBB1 illustrates one approach to the integration of high-level control constructs with mechanisms for the generation of efficient blackboard systems. While these constructs are based upon the BB1 model of control, the implementation remains true to the spirit of the Generic Blackboard System. For this reason, there are a few implementation features of BB1, such as the representation of blackboard objects as closures, that are not present in the implementation of the GBB1 control shell.

This initial work on GBB1 suggests several future directions for the integration of high-level control constructs with mechanisms for the generation of efficient blackboard systems. It would be useful to determine whether or not the current GBB1 facilities are sufficient for efficient implementation of the more "advanced" BB1 facilities, such as goal-directed reasoning and language frameworks. In addition, we plan to implement other control shells on top of GBB. The use of a common database support facility provides a powerful framework for evaluating different control models by factoring out differences in the underlying implementation of the blackboard structures and objects.

References

[1] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, July 1985.

[2] Philip M. Johnson, Kevin Q. Gallagher, and Daniel D. Corkill. *GBB Reference Manual.* Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, GBB Version 1.00 edition, March 1987. (Also published as Technical Report 87-36, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, March 1987.).

[3] Philip M. Johnson, Kevin Q. Gallagher, and Daniel D. Corkill. *The GBB1 Control Shell.* Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, GBB Version 1.10 edition, June 1987. (Published in the *GBB Reference Manual*, Technical Report 87-36, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, Revised June 1987.).

[4] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[5] L. Baum, R. Dodhiawala, and V. Jagannathan. *Boeing Blackboard System, Version 1.0.* Technical Report BCS-G2010-31, Boeing Computer Services, P.O. Box 24346, Seattle, Washington 98124, July 1986.