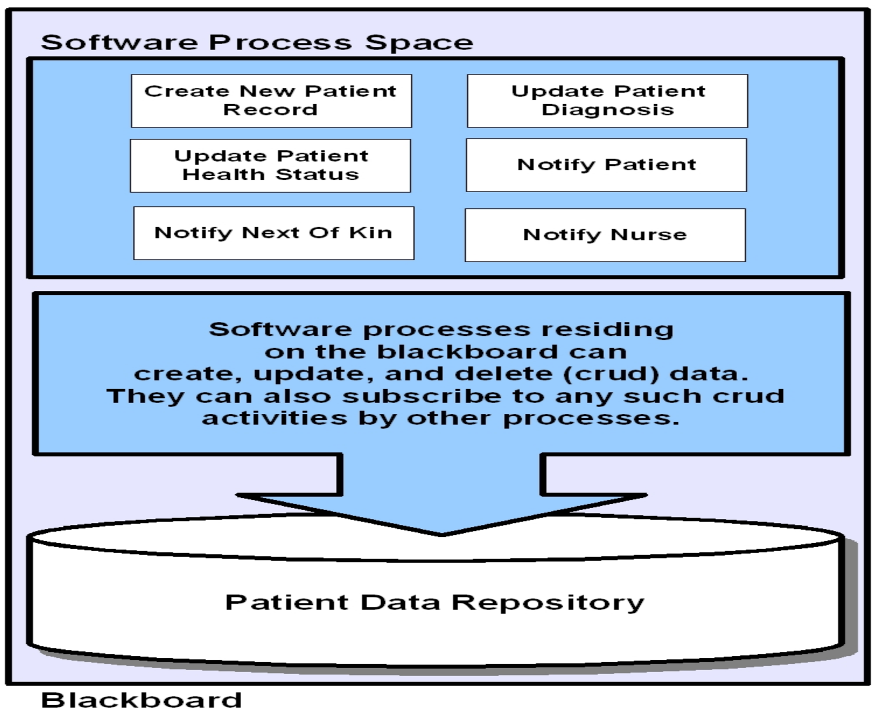
Blackboard User's Guide

Bediako George

December 13th, 2006

Introduction

The Blackboard Workspace Server is a Java based implementation of the blackboard design paradigm. The blackboard design paradigm facilitates the organization of chunks of information for processing by loosely coupled software processes. These chunks of information are stored in the blackboard's central data repository. This central data repository is accessible by all participating software processes. Software processes can add, update, and remove chunks of information in the repository. The blackboard design paradigm provides for data change notification services to the participating software processes, and provides for the scheduling of those software processes that wish to effect changes on the chunks of information in the repository. The diagram below illustrates the relationship among the data, software processes and the blackboard in a hypothetical patient management application.


Software Process Space
Create New Patient Record
Update Patient Diagnosis
Update Patient Health Status
Notify Patient
Notify Next Of Kin
Notify Nurse
Software processes residing on the blackboard can create, update, and delete (crud) data. They can also subscribe to any such crud activities by other processes.
Patient Data Repository
Blackboard

The advantage of this design pattern is that it encourages the development of a loosely coupled programming structure. In addition, the blackboard design pattern also promotes contractual software process design. That is to say, software processes have the choice to subscribe to only the data changes they find interesting, or they may choose to proclaim disinterest in the current state of the blackboard, at which point their execution ends.

Consider the example patient application. Some of the activities supported by this application include “creating a new patient record”, “updating a patients diagnosis”, and “updating a patient's health status”. These activities result in changes to the patient's data that is stored in the blackboard's data repository.

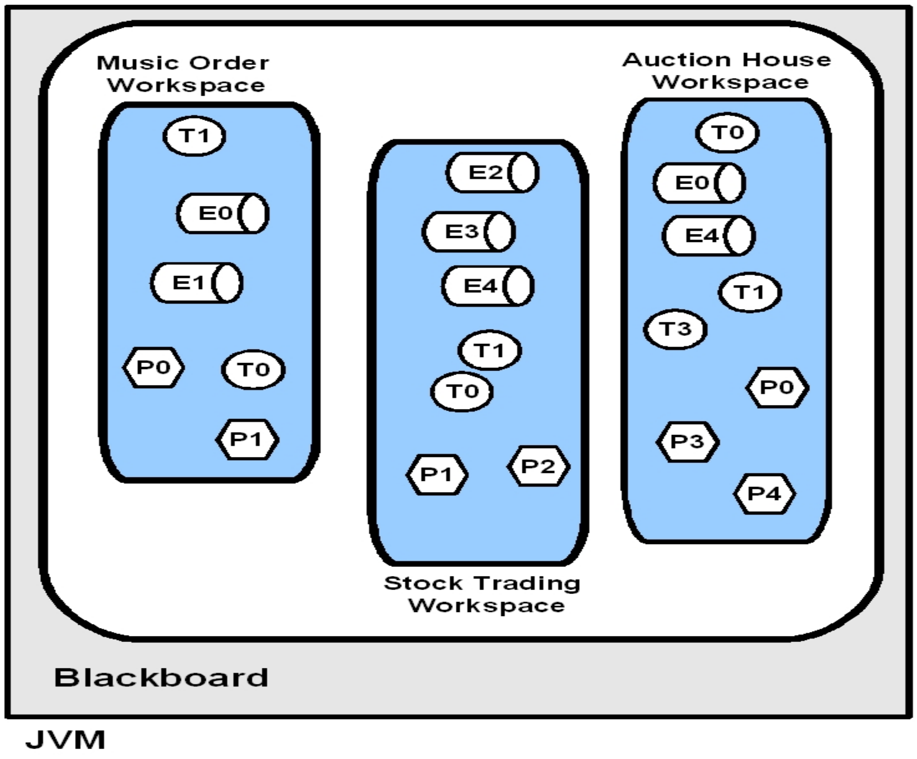
Other processes like “notify nurse” for instance, can subscribe to the patient data changes committed by the other software processes. As an example, if the prescription of the patient changes, both the “notify patient” and the “notify nurse” processes would be interested in that, whereas if the patient slips into a coma, “notify nurse” and “notify next of kin” would be interested instead.
Using the blackboard model, programmers developing software focus on creating the following logic for each of the software processes they develop:

1. Logic that dictates the conditions for which the software process is interested in executing.
2. Logic that dictates the software process activity.
3. Logic that dictates the conditions for which the software process is no longer interested in executing.

The Blackboard Workspace Server implements many of the ideas from the blackboard programming model. In Blackboard Workspace Server, chunks of information are called "targets", and the software processes interested in these targets are called "plans". There are also special types of targets called "events". Events differ from ordinary targets in that they are typically created outside the blackboard environment. Finally, targets, plans, and events are all organized on the blackboard in "workspaces". A workspace is a logical division of the blackboard. The relationship amongst an event, its workspace, its associated plans, and the blackboard is similar to the relationship between a JMS message, a J2EE container, its associated message driven beans, and a J2EE application server, in the sense that many different types of workspaces can run on a single blackboard, and that the arrival of an event starts the execution plans on that workspace. The difference lies in the fact that workspaces are specifically geared to handle event based processing.

High Level Blackboard Workspace Server Design

The following diagram illustrates the logical relationship of targets, events, plans, and the blackboard.



In this diagram the large white oval represents the single blackboard. This blackboard supports three types of workspaces. Each type of workspace can support multiple instances, but for the purpose of this illustration only one instance of each workspace is shown. Targets are represented by circles, events by cylinders[1], and plans by hexagons. Targets, events, and plans are all named, and can be addressed by their names on the workspace[2]. For instance, in the “Music Order Workspace”, there are named targets, “T0” and “T1”, and named events “E0” and “E1”, and named plans “P0” and “P1”. Whenever a target is retrieved by its name, the last instance of the target that was added to the workspace is retrieved, and the previous instance if lost to the workspace[3]. This differs from event retrieval where all versions of the named event added to the workspace are stored. This is why events are represented as a cylinder the diagram above.

Blackboard

The Blackboard Workspace Server consists of a blackboard, workspaces, and plans. The blackboard is responsible for managing the resources used by the server, and it serves as the gateway through which all events must pass through in order to arrive in workspace. In the general blackboard design paradigm, events are placed directly on the blackboard, however, the Blackboard Workspace Server implementation of the blackboard promotes the separation of the blackboard into smaller parts called workspaces. This allows for performance improvements, and it lends well to programmer organization of the software processes by attaching plans to workspaces of a particular name.

After instantiating a blackboard, placing an event on the blackboard is as easy as invoking the placeOnBlackboard(Object _object) method. The blackboard consults workspace configuration information to figure out what workspaces should be created for the object type placed[4]. So conceivable one could defined workspaces to respond to the any of the types a particular class may have. For instance, if Customer extends the java.util.Object class, one could create workspaces in response to the Customer type and the Object type. This means the blackboard will create two workspaces every time it receives an object of class Person.

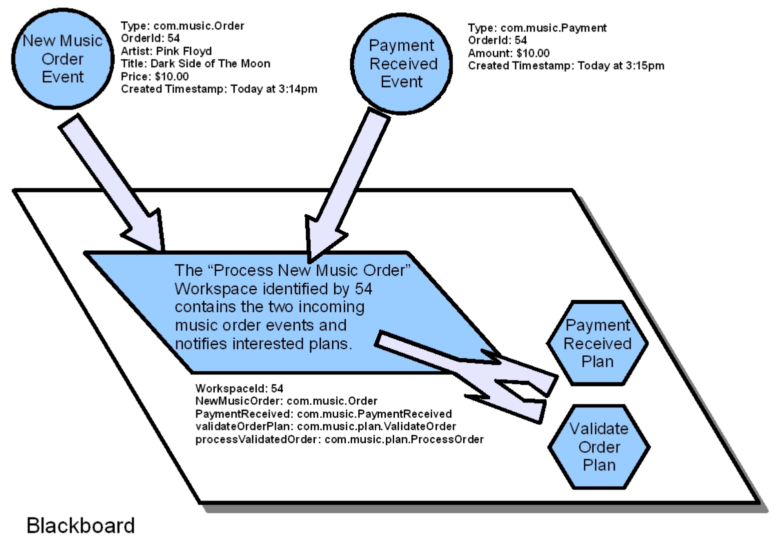
Additionally, the blackboard is responsible for persisting workspaces whenever they are retired, or whenever resource constraints called for temporary removal from memory.

Events

In general, an event can be thought of as meta data describing an unique occurrence of an action of some kind. With the Blackboard Workspace Server, events are represented as plain Javabeans that can be placed on the blackboard at anytime. It is the action of placing an event on a blackboard that causes the creation of a new workspace. Every event is expected to have an attribute that will be used to identify a particular workspace instance. As an example, in the “Music Order Processing” workspace, if there exists a “New Order Submitted” event, the uniquely identifying attribute may be the “orderId”. Whenever an event of the type “New Order Submitted” is placed on the blackboard, the blackboard first checks to see if an instance of the “Music Order Processing” workspace with a workspace identifier with the same value of the “orderId” attribute exists. If such a workspace instance exists, the event is placed in the workspace. If not, a new workspace is created, its workspace identifier attribute is set to the same value as the event's “orderid” attribute, and the event is placed there. Consequently, all events arriving after the first event arrived, will be ferried to that workspace, so long as the value of that order's orderId attribute is the same as the previous event. All interested plans attached to that workspace are automatically notified of the specific event arrivals. In the diagram below note that even though the “Payment Received” event arrives a full minute after the “New Music Order” Event it still

1 Think of a cylinder as a stack of circles. This represents the fact that many events of the same type may be directed to the blackboard.
2 `Plans are not accessible in this way as yet.
3 Actually the target instance is still available as part of the workspaces change history.
4 It is important to note that events are defined by Type.

gets routed to the correct workspace.


New Music Order Event
Type: com.music.Order
OrderId: 54
Artist: Pink Floyd
Title: Dark Side of The Moon
Price: $10.00
Created Timestamp: Today at 3:14pm
Payment Received Event
Type: com.music.Payment
OrderId: 54
Amount: $10.00
Created Timestamp: Today at 3:15pm
The “Process New Music Order” Workspace identified by 54 contains the two incoming music order events and notifies interested plans.
WorkspaceId: 54
NewMusicOrder: com.music.Order
PaymentReceived: com.music.PaymentReceived
validateOrderPlan: com.music.plan.ValidateOrder
processValidatedOrder: com.music.plan.ProcessOrder
Payment Received Plan
Validate Order Plan
Blackboard

It important to note that there is no requirement for the “New Music Order” event to be the first to arrive. Indeed, it is perfectly acceptable for the “Payment Received” event to arrive first. The Blackboard Workspace Server will still create the appropriate workspace, and the appropriate plans will be executed as they express interest.

This is a very powerful concept, and it forms the basis for much of the functionality of the Blackboard application. It also has implications regarding the performance and design of the software infrastructure that is creating the events in the first place. This flexibility in the arrival of events makes it easier to develop complicated applications.

Consider a stock market order processing application. The unique workspace identifier can be a stock's ticker symbol, and buy and sell orders can arrive in any order and at any time. Using the Blackboard Workspace Server, this means that all buy and sell orders for a particular ticker symbol can be automatically ferried to the same workspace instance. Within that instance, plans can operate on the buy and sell orders, performing order validation, execution, and fulfillment, without having to bother with grouping orders together. It also means that order enhancements, corrections, and cancellations will automatically get pushed to the right workspace as well, as they are all organized by the stock's ticker symbol. And since all instances of a particular event are kept on the workspace, version information for all of the events is provided for free.

Workspaces

The workspace is the central organizing entity on the blackboard. It forms the basis for grouping events, and targets, and it defines the types of plans that are allowed to act upon that grouping. Workspaces are also responsible for notifying plans of changes to events and targets on the name and attribute level.[5] They check on plans for interest of execution, and schedule plans for execution on the blackboard. Finally, they are responsible for determining when a workspace can be retired, with the persistence of results produced by the plans a natural consequence.

Instances of workspaces are created whenever an event with a workspace identifier is placed on the blackboard. These workspaces are managed on the blackboard. The blackboard will automatically

5 A workspace keeps track of changes to adds, updates, and removals of targets and events at the name level. It will also keep track of changes on those targets and events attributes as well.

manage the number of active workspaces that can be held in memory at any time, and will temporarily persist workspaces to a workspace repository whenever that maximum workspace limit is exceeded[6].

Workspaces can have several states. They are described below.

1. Active: Workspaces in this state have plans that are still interested in execution.
2. Completed: Workspaces in this state no longer have plans interested in execution.
3. Terminated: Workspaces in this state have reached an unrecoverable error condition.
4. Executing: Workspaces in this condition have plans that are currently executing on the blackboard.
5. Persisted: Workspaces in this state are no longer in memory, but are persisted in the workspace repository.

Plans

Plans are reentrant[7] Java objects that implement the Blackboard's "Plan" interface. They are expected to provide a method that will execute its operating logic when provided with a picture of the workspace, and a method that provides an object that implements the Blackboard's "PlanPredicate" interface. This PlanPredicate object in turn should provide an additional two methods. The definitions of these methods are listed below:

1. void execute(Workspace): This method contains the operating logic the plan wishes to execute.
2. boolean IsInterested(Workspace): This method returns true if the plan is interested in operating on the workspace.
3. boolean isFinished(Workspace): This method returns true if the plan is no longer interested in operating on the workspace.

Plans can also request at configuration that the workspace is not allowed to change once the plan begins execution. As such any new event arrivals will be blocked from entering the workspace until the plan finishes its scheduled execution run.

During a execution run[8], plans may alter the state of the workspace. They may add, remove, or update targets and events, and change target and events attributes. All such activities by the plan are recorded by the workspace, and stored in a change history object.

Targets

Targets are Javabeans that are placed on a workspace by Plans. They are mainly used to store new state generated by an executing plan, or to relabel Javabeans as necessary. Consequently, two different targets may point to the same underlying Javabean. For instance, a newly required event, called "New Music Order" may be operated on by the plan "Validate New Music Order". Upon completion of the validation of the "New Music Order", that plan may choose to add to the workspace a target called "Validated Music Order". The underlying object will still be the original "New Music Order" event, but with the new label "Validated Music Order" the plan "Fulfill Music Order" may suddenly express interest. Using this technique, a workspaces work flow may be dictated by active plan's expression of interest to execute. This represents the basis of the loose coupling, and it allows otherwise complicated programming logic to be simplified.

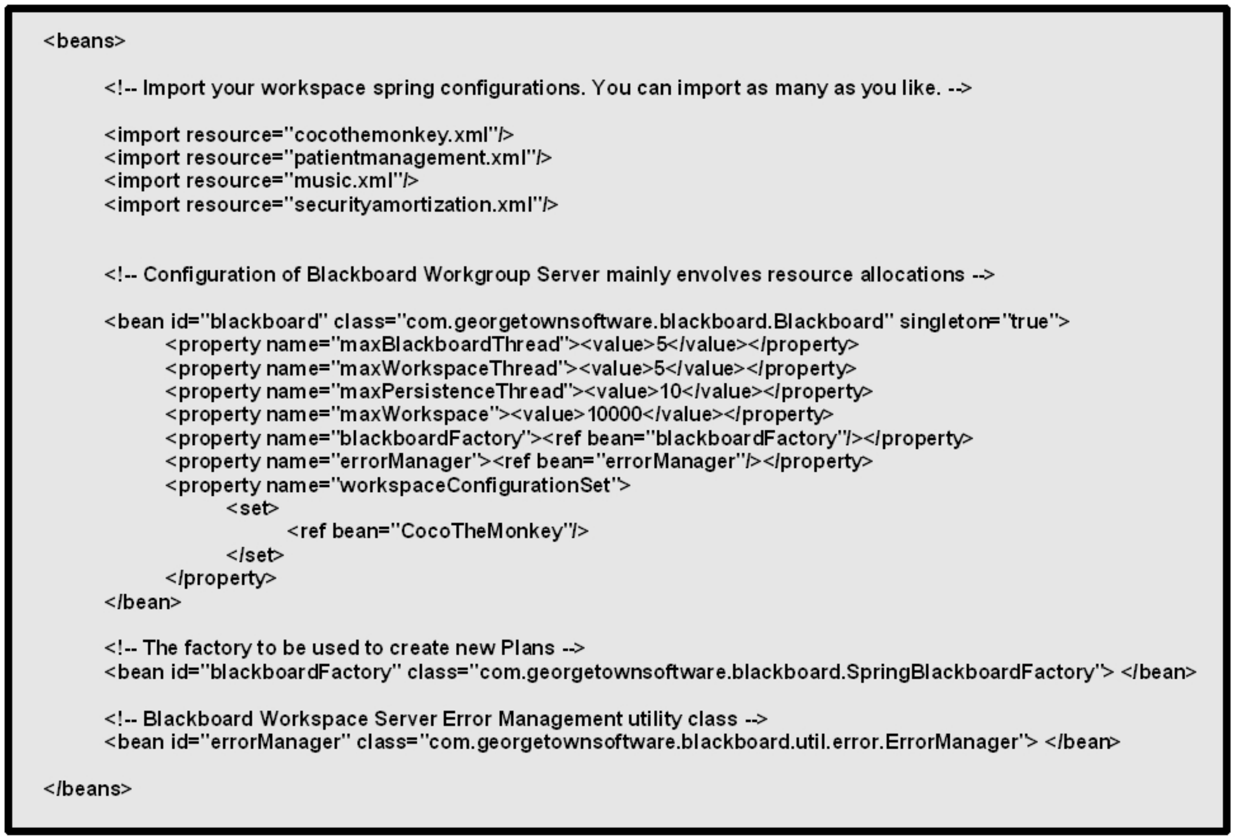
Configuration

The Blackboard Workspace Server is designed to be configured via dependency injection. It has been tested to work with the Spring Framework, but it should also work with any other framework that

6 Least recently used algorithm is used to select workspaces for temporary persistence.
7 Since a single instance of a Plan is created on the blackboard and that instance is shared by multiple workspaces and can be executed by multiple threads concurrently, it is imperative that a Plan be reentrant. Any Plan that wishes to keep state should use the workspace to do so.
8 Note that a single plan may express interest to run against a workspace several times before proclaiming it is finished.

provides dependency injection.

First you configure your blackboard application:



```
<beans>

        <!-- Import your workspace spring configurations. You can import as many as you like. -->

        <import resource="cocothemonkey.xml"/>
        <import resource="patientmanagement.xml"/>
        <import resource="music.xml"/>
        <import resource="securityamortization.xml"/>


        <!-- Configuration of Blackboard Workgroup Server mainly envolves resource allocations -->

        <bean id="blackboard" class="com.georgetownsoftware.blackboard.Blackboard" singleton="true">
                <property name="maxBlackboardThread"><value>5</value></property>
                <property name="maxWorkspaceThread"><value>5</value></property>
                <property name="maxPersistenceThread"><value>10</value></property>
                <property name="maxWorkspace"><value>10000</value></property>
                <property name="blackboardFactory"><ref bean="blackboardFactory"/></property>
                <property name="errorManager"><ref bean="errorManager"/></property>
                <property name="workspaceConfigurationSet">
                        <set>
                                <ref bean="CocoTheMonkey"/>
                        </set>
                </property>
        </bean>

        <!-- The factory to be used to create new Plans -->
        <bean id="blackboardFactory" class="com.georgetownsoftware.blackboard.SpringBlackboardFactory"> </bean>

        <!-- Blackboard Workspace Server Error Management utility class -->
        <bean id="errorManager" class="com.georgetownsoftware.blackboard.util.error.ErrorManager"> </bean>

</beans>
```

In the diagram above, the Blackboard Workspace Server is configured as a Spring bean. First you import other Spring configuration XML files that describe your workspaces. You can import as many as you like.

Next you simply specify values that the Blackboard Workspace Server uses to manage resource allocations.

1. maxBlackboardThread – Blackboard threads are used to process incoming events. Set this attribute to the number of threads that should be allocated to processing incoming events.
2. maxWorkspaceThread – Workspace threads are used to process plans whenever they are scheduled to execute. Set this attribute to the number of threads that should be allocated to run scheduled plans.
3. maxPersistenceThread – Persistence threads are used to persist retired workspaces[9]. Set this attribute to the number of threads that should be allocated to persist retired workspaces.
4. maxWorkspace – This attribute governs how many workspaces are allowed to exist in memory at once. Once the number of workspaces exceeds this amount a background thread begins persisting the workspaces to a temporary repository. It is expected that this implementation will change in the future.
5. BlackboardFactory – This attribute is expected to be a bean defined in you Spring configuration. It must be a factory that implements "com.georgetownsoftware.blackboard.BlackboardFactory". As a convenience, the SpringBlackboardFactory is provided.

9 Workspaces that are persisted as part of resource management are handled by a continually running background thread. It is expected that this implementation will change to one that is pool based instead.

6. ErrorManager - Utility class used by the Blackboard Workspace Server to process errors. You should not need to change this.

Here is a example of a Spring configuration file for the workspace application "Coco The Monkey". First the events are to be defined. This section contains data describing the types of events the "Coco The Monkey" workspace cares about.

```
<bean id="CocoTheMonkey.monkey" class="com.georgetownsoftware.blackboard.config.EventConfiguration" singleton="true">
        <property name="name"><value>monkey</value></property>
        <property name="eventClass"><value>com.georgetownsoftware.blackboard.examples.cocothemonkey.Monkey</value></property>
        <property name="workspaceIdentifierName"><value>name</value></property>
</bean>


<bean id="CocoTheMonkey.fruit" class="com.georgetownsoftware.blackboard.config.EventConfiguration" singleton="true">
        <property name="name"><value>fruit</value></property>
        <property name="eventClass"><value>com.georgetownsoftware.blackboard.examples.cocothemonkey.Fruit</value></property>
        <property name="workspaceIdentifierName"><value>fruitFor</value></property>
</bean>


<bean id="CocoTheMonkey.eagle" class="com.georgetownsoftware.blackboard.config.EventConfiguration" singleton="true">
        <property name="name"><value>eagle</value></property>
        <property name="eventClass"><value>com.georgetownsoftware.blackboard.examples.cocothemonkey.Eagle</value></property>
        <property name="workspaceIdentifierName"><value>lookingFor</value></property>
</bean>


<bean id="CocoTheMonkey.hunter" class="com.georgetownsoftware.blackboard.config.EventConfiguration" singleton="true">
        <property name="name"><value>hunter</value></property>
        <property name="eventClass"><value>com.georgetownsoftware.blackboard.examples.cocothemonkey.Hunter</value></property>
        <property name="workspaceIdentifierName"><value>inForestOf</value></property>
</bean>
```

Text 2: Event Configuration.

The following defines the attributes of the event configuration XML fragment:

1. name - The name of the event. When an event arrives it is placed the blackboard with this as the identifier.
2. eventClass[10] - The Java type representing this event. It can be an interface, abstract, or concrete class.
3. workspaceIdentifierName - This is a Java object that uniquely identifies this workspace. It is typically a String, but it can be any object[11]. Care must be taken to make sure that the hashcode and equals methods of these objects behave according to the expectations of a java.util.HashMap.

Here are the plan definitions that will be added to the workspace whenever it is instantiated. Plan are to be stateless. All state generated by a plan is to be stored on the workspace. Plans must implement the "com.georgetownsoftware.blackboard.Plan" interface.

10 This is an unfortunate misnomer. It is really the object type and not the class that is important here. As it will be revealed in the example application, the event "CocoTheMonkey.fruit" is actually an abstract class, and the concrete class Mango that extends the Fruit abstract is recognized as the event "CocoTheMonkey.fruit". This attribute name will be changed in the near future.

11 As long as the hashcode() and equals contract is honored, any object can be used.

```
<bean id="CocoTheMonkey.playPlan" class="com.georgetownsoftware.blackboard.examples.cocothemonkey.Play" singleton="true">
        <property name="planName"><value>play</value></property>
</bean>


<bean id="CocoTheMonkey.sleepPlan" class="com.georgetownsoftware.blackboard.examples.cocothemonkey.Sleep" singleton="true">
        <property name="planName"><value>sleep</value></property>
</bean>


<bean id="CocoTheMonkey.eatPlan" class="com.georgetownsoftware.blackboard.examples.cocothemonkey.Eat" singleton="true">
        <property name="planName"><value>eat</value></property>
</bean>


<bean id="CocoTheMonkey.huntPlan" class="com.georgetownsoftware.blackboard.examples.cocothemonkey.Hunt" singleton="true">
        <property name="planName"><value>hunt</value></property>
</bean>
```

Text 3: Plan configuration

1. planName - The name of this plan. Used for debugging only.

Note that the plans are all deployed as singletons. This is done to minimize memory resource consumption. As a consequence, plans are expected to be reentrant, as the same plan may be executed by several threads concurrently.

```
<property name="name"><value>CocoTheMonkey</value></property>

<property name="eventConfigurationSet">
        <set>
                <ref bean="CocoTheMonkey.monkey"/>
                <ref bean="CocoTheMonkey.fruit"/>
                <ref bean="CocoTheMonkey.eagle"/>
                <ref bean="CocoTheMonkey.hunter"/>
        </set>
</property>

<property name="planSet">
        <set>
                <ref bean="CocoTheMonkey.sleepPlan"/>
                <ref bean="CocoTheMonkey.playPlan"/>
                <ref bean="CocoTheMonkey.eatPlan"/>
                <ref bean="CocoTheMonkey.huntPlan"/>
        </set>
</property>


<property name="doNotPersistSet">
        <set>
          <value>hunter</value>
          <value>eagle</value>
          <value>tree</value>
        </set>
</property>


<property name="exclusivePlanSet">
        <set>
                <ref bean="CocoTheMonkey.huntPlan"/>
        </set>
</property>


<property name="persistChangeInfoHistory"><value>false</value></property>
```

Text 4: Workspace Configuration

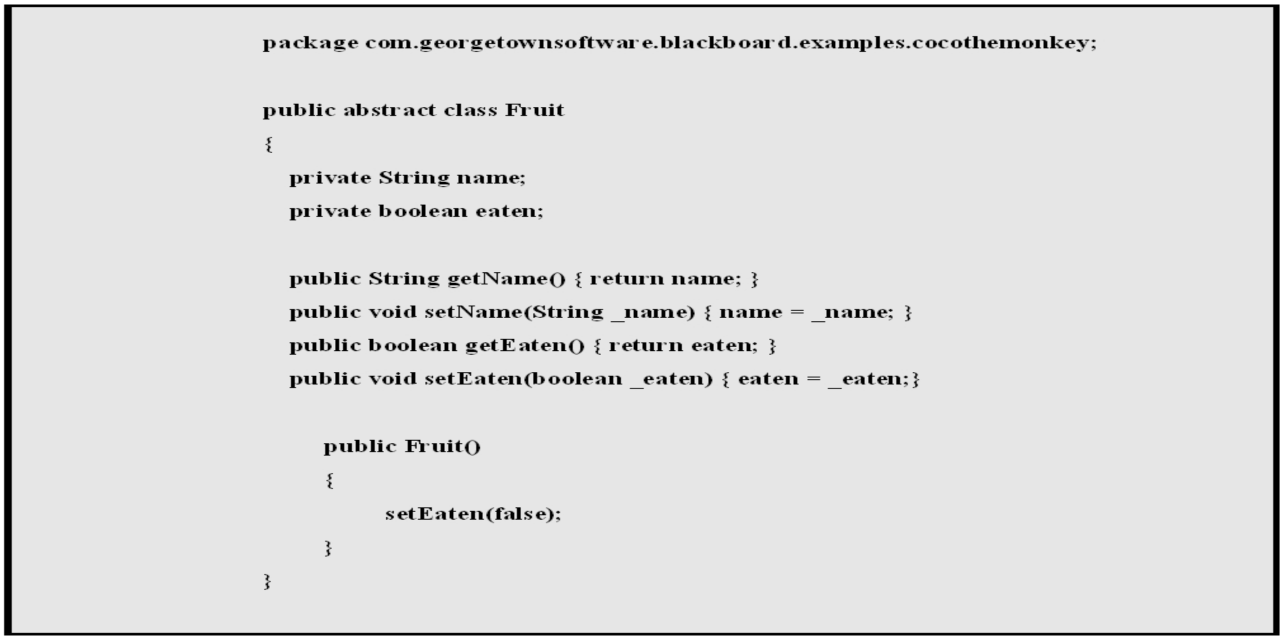
Finally the workspace itself is configured. The following describes what each of the properties of the workspace configuration represent:

1. name - The name of this workspace.
2. eventConfigurationSet - This defines the set of events that will cause this workspace to be instantiated.
3. planSet - Represents the set of plans that will be assigned to the workspace defined by this configuration.
4. doNotPersistSet - This represents the set of targets that should not be persisted when a workspace is retired. These targets will be persisted during any intermediary persistence due to a need to maintain blackboard state.
5. exclusivePlanSet - This represents the set of plans that should not be run as exclusive on this workspace. This means incoming events slated for this workspace will be blocked from entering this workspace until any of these plans current execution run is completed. This is useful when incoming events can change the state of the workspace in a way that could be detrimental to any of these plans operation.
6. persistChangeInfoHistory - Determines whether or not the change information that is collected for each target on the workspace will be persisted whenever the workspace is persisted. This means that even during persistence to maintain blackboard state change information will NOT be persisted.

The Coco The Monkey Example

Now that configuration has been explained the code for this example can be investigated. Let's take a look at a part of the whole app. To see the whole application altogether, visit the sourceforge repository at http://sourceforge.net/projects/blkbrd/. You can browse release 0.1 alpha on the SVN repository at this URL: http://blkbrd.svn.sourceforge.net/viewvc/blkbrd/branches/release0.1/. Here you will find the Java source code for the "Coco the Monkey" application, and also the source code for the Blackboard Workspace Server.

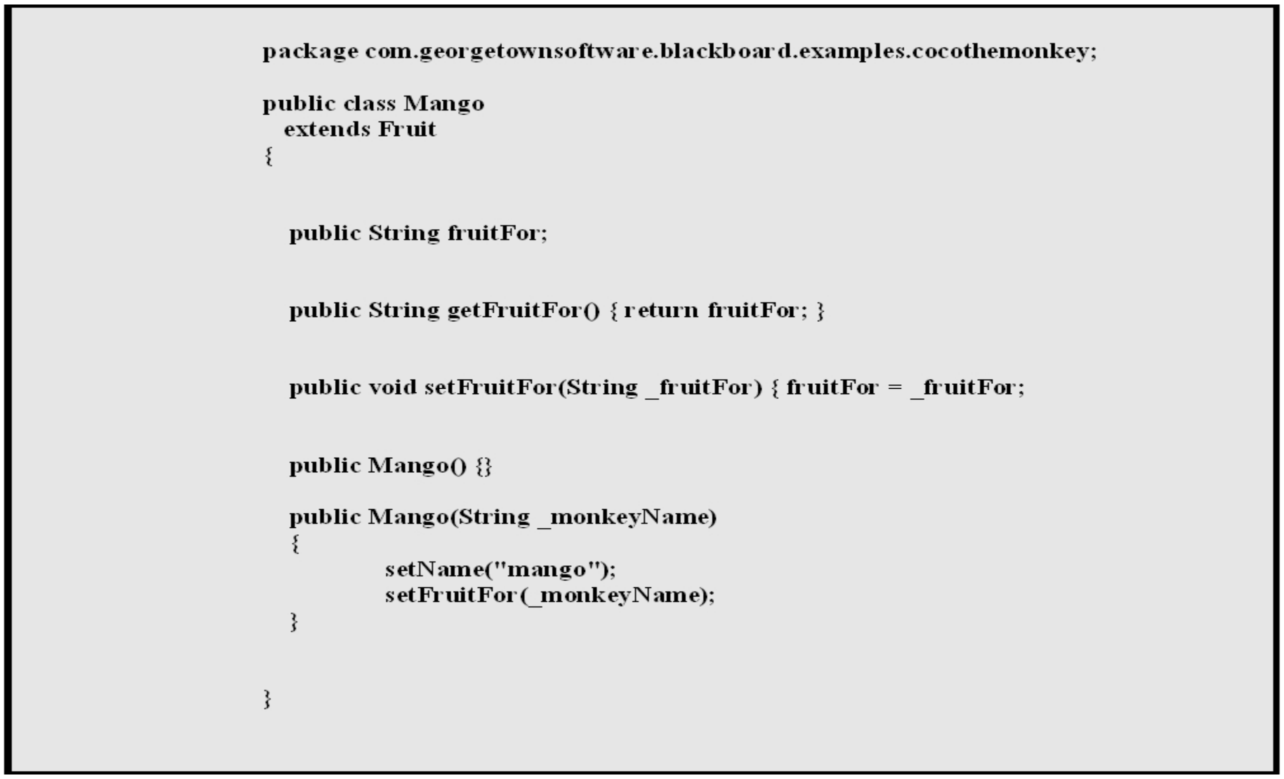
First lets look at a sample event. Here is definition of the abstract class Fruit.



```
package com.georgetownsoftware.blackboard.examples.cocothemonkey;

public abstract class Fruit
{
   private String name;
   private boolean eaten;

   public String getName() { return name; }
   public void setName(String _name) { name = _name; }
   public boolean getEaten() { return eaten; }
   public void setEaten(boolean _eaten) { eaten = _eaten;}

      public Fruit()
      {
           setEaten(false);
      }
}
```

Text 5: Fruit class definition

As it is apparent, there is no need to implement any interfaces. The following describes the class Mango that extends the Fruit class:



```
package com.georgetownsoftware.blackboard.examples.cocothemonkey;

public class Mango
  extends Fruit
{


   public String fruitFor;


   public String getFruitFor() { return fruitFor; }


   public void setFruitFor(String _fruitFor) { fruitFor = _fruitFor;


   public Mango() {}

   public Mango(String _monkeyName)
   {
            setName("mango");
            setFruitFor(_monkeyName);
   }


}
```

Text 6: Mango Class Definition

Here there are no surprises. Mango simply extends the Fruit class and implements additional methods. Of note here is that a workspace will be created whenever a class of of type Fruit is placed on the Blackboard. The only out of the ordinary thing here is that even though the blackboard will react to the type Fruit, it also expects that the object will have an attribute called “fruitFor”, and that attribute will be used to identify the active workspace. Looking at the class definitions, “fruitFor” is not an attribute that belongs to a Fruit type, but instead it belongs to a Mango. Since the blackboard only cares about the type with regards to events, this is acceptable.

Finally, here is the Eat plan class definition:

```
package com.georgetownsoftware.blackboard.examples.cocothemonkey;

import com.georgetownsoftware.blackboard.*;

public class Eat
  implements Plan
{
        private String planName;

        public String getPlanName() { return planName; }
        public void setPlanName(String _planName) { planName = _planName; }

        public Eat() {}

        public String getName() { return getPlanName(); }

        public void execute(Workspace _workspace)
        {
                Fruit fruit = (Fruit) _workspace.get("fruit");
                Monkey monkey = (Monkey) _workspace.get("monkey");

                fruit.setEaten(true);
                monkey.setSleeping(false);
                monkey.setPlaying(false);
                monkey.setEating(true);
        }

        public PlanPredicate getPlanPredicate()
        {
                return new PlanPredicate()
                {
                        public boolean isInterested(Workspace _workspace)
                        {
                                Fruit fruit = (Fruit) _workspace.get("fruit");

                                return          (_workspace.has("fruit") == true) &&
                                                (_workspace.has("monkey") == true) &&
                                                ((fruit instanceof Mango) == true);
                        }

                        public boolean isFinished(Workspace _workspace)
                        {
                                //This means you will be executed once and never again.
                                return true;
                        }
                };
        }
}
```

Text 7: Eat plan definition.

Here the plan Eat is expected to implement the Plan interface. There are only two methods needed to be implemented, “void execute(Workspace)” and “PlanPredicate getPlanPredicate()”. The PlanPredicate is itself an interface. Consequently, the plan Eat returns the anonymous PlanPredicate object. This object must itself implement two methods. The first is the isInterested(Workspace). This method inspects the workspace for the presence fruit and monkey target. If those targets are present it returns true as its result. The workspace then schedules this plan for execution. When a thread is allocated for this plans execution, The Eat plan's execute(Workspace) method is invoked. The Eat plan then sets the “eaten” attribute on the fruit to “true”. It also sets several attributes on the monkey as well. In the background, the workspace is recording all the changes to the targets “fruit” and “monkey”, and storing this history. The programmer need not be aware of this activity. After execution is completed the PlanPredicates isFinished(Workspace) method is called. This PlanPredicate isFinished method always returns true, which simply guarantees that the plan Eat will run once and only once. A similar approach to the one taken by the isInterested method could be applied here are well. This would be necessary if the goal state of this plan was more involved.

When this plans execution is completed, and all the other plans have been successfully executed, the workspace will be retired by the blackboard. To see all the code associated with the CocoTheMonkey application, please visit this URL:
http://blkbrd.svn.sourceforge.net/viewvc/blkbrd/branches/release0.1/src/java/com/georgetownsoftware/blackboard/examples/cocothemonkey/.

Conclusion

The Blackboard Workspace Server offers a different way of processing data. It relies on modern programming techniques to address the difficulties envolved in processing event based applications. Because persistence is handled for the developer, a lot of programming time is save. Finally, since programmer's are expected to clearly and concisely define start and stop states for each software plan, the resulting code is cleaner and easier to understand.