EXHIBIT 10

Pseudo code implementation of Context and Tracking Components

This report describes the implementation in pseudo code of a context and a context tracking functionality based on the descriptions provided in the provisional patent application. The design approach was to articulate into pseudo code a generalization of the workflow example provided in ATTACHMENT 2 of the provisional patent application. In that regards, the pseudo code contains two basic elements:

(a) a generic application skeleton that allows the user to navigate through contexts as defined in a particular workflow (referred to as Webslice in ATTACHMENT 2) and where the change of context is identified automatically
(b) an implementation of a context (referred to as Boards in ATTACHMENT 2)

Those two elements also utilize the source code described in the Web and WebSlice classes as described in ATTACHMENT 2 of the provisional patent application.

Class: WebApp

```
import java.awt.AWTEvent;
import java.awt.Menu;
import java.awt.MenuItem;
import java.awt.event.ActionListener;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

/**
* This class represents the general skeleton of an application
* (e.g. web based application) that would provide an end-user
* with the basic interface to create contexts, navigate across
* them based on the workflows relevant to the user and track
* changes to contexts.
*/
public class WebApp {
  /* the basic elements to keep track of */
  private Collection<Web> webs;
  private Web currentUserWeb; // the specific web. It is possible to have multiple webs
                         // and the user selects one at any particular time
  private Board currentUserCtx;  // the specific context within a particular web
  private WebSlice currentUserWorkFlow;
  private String userID;

  /*
   * UI elements
   */
  private Menu menuWebs;
  private Menu menuWebSlices;
  private Menu menuBoards;

  /**
  * Constructor
  * @param the User that is using the particular instance of the application
  */
  public WebApp(String userName) {
    this.userID = userName;
    this.currentUserWeb = null;
    this.currentUserCtx = null;
    this.currentUserWorkFlow = null;
    setupWebApp();
  }

  /*
   * We assume that there is some persistent repository where the definitions of Webs
   * Webslices and Boards are stored. This method would access such data and present it
   * to the user. Upon selection of the relevant UI elements on the part of the user,
   * the basic elements (e.g. currentUserWeb, currentUserCtx, currentUserWorkflow) would
   * be defined
   */
  public void setupWebApp() {
          /*
           * For simplicity sake, we assume that the persistent storage interface referred to
           * in page 11 of the provisional patent application (CollectionFactory)
           */
          this.webs = CollectionFactory.getPersistentCapableCollectionOfWebs();
          /*
           * At this point we would have a collection of instantiated Web objects.
           * The next step would be to present the list of available Webs to the user
           * through some UI element (e.g. a menu) so the user select the web of interest.
           */
          menuWebs = new Menu("Webs");
          menuWebSlices = new Menu("Workflows");
          menuBoards = new Menu("Contexts");
          for(Iterator iter = webs.iterator(); iter.hasNext(); ) {
                Web w = (Web) iter.next();
                menuWebs.add(new MenuItem(w.getName()));
                // getName method is defined in ATTACHMENT 2, page 12
          }
          /*
           * The UI element would have event-based mechanism such as a listener that
           * automatically detect the users' selection. Upon reception of the event,
           * the application can automatically update its internal state. Specifically,
           * once a use selects a particular Web, the variable currentUserWeb can be
           * updated and the list of available Contexts (or Boards) in the Web can be
           * retrieved and displayed.
           */
```

```
        menuWebs.addActionListener(new ActionListener() {
                public void actionPerformed(AWTEvent e) {
                        MenuItem m = (MenuItem)e.getSource();
                        String webName = m.getLabel();
                        // the getWebByName method would iterate over the webs collection
                       // and return the right object
                        currentUserWeb = this.getWebByName(webName);
                        currentUserCtx = null;
                        currentUserWorkFlow = null;
                        /*
                         * We now have the a Web object. The first step is to update the
                         * menus with the webslices
                         */
                        menuWebSlices.removeAll();
                        menuBoards.removeAll();
                        Collection<WebSlice> slices =
CollectionFactory.getPersistentCapableCollectionOfWebSlices();
                        for(Iterator iter = slices.iterator(); iter.hasNext(); ) {
                                WebSlice ws = (WebSlice) iter.next();
                                menuBoards.add(new MenuItem(ws.getName()));
                        }
                }
        });

        /*
         * The next step is to add a listener to the menu of Workflows (webslices)
         */
        menuWebSlices.addActionListener(new ActionListener() {
                public void actionPerformed(AWTEvent e) {
                        MenuItem m = (MenuItem)e.getSource();
                        String webSliceName = m.getLabel();
                        Collection<WebSlice> slices =
CollectionFactory.getPersistentCapableCollectionOfWebSlices();
                        for(Iterator iter = slices.iterator(); iter.hasNext(); ) {
                                WebSlice ws = (WebSlice) iter.next();
                                if (webSliceName.equals(ws.getName())) {
                                        currentUserWorkFlow = ws;
                                }
                        }
                        currentUserCtx = null;
                        menuBoards.removeAll();
                        /*
                         * We now update the menu with the appropriate list of contexts (boards)
                         */
                        List boards = currentUserWeb.getBoardsList();
                               // getBoardsList method is defined in ATTACHMENT 2, page 14
                        for(Iterator iter = boards.iterator(); iter.hasNext(); ) {
                                Board b = (Board) iter.next();
                                menuBoards.add(new MenuItem(b.getName()));
                        }
                }
        });

        /*
         * Finally, we add a listener to the menu of Contexts (Boards)
         */
        menuBoards.addActionListener(new ActionListener() {
                public void actionPerformed(AWTEvent e) {
                        MenuItem m = (MenuItem)e.getSource();
                        String boardName = m.getLabel();
                        /*
                         * the getBoardByName method would iterate over the
                         * currentUserWeb's list of boards and return the right
                         * Board object corresponding to the boardName string
                         */
                        Board oldCtx = currentUserCtx;
                        currentUserCtx = getBoardByName(boardName);
                        if (oldCtx != null && oldCtx.getName() != currentUserCtx.getName()) {
                                //The user changed contexts
                                currentUserCtx.importDataFromParent(currentUserWeb,
                                                                    currentUserWorkFlow);
                        }
                        /*
                         * At this point a particular UI element that articulates the context
                         * should be updated. For instance, a list of applications as well as
                         * the list of data elements that are available could be displayed
                         * using the data provided by the accessor methods from the Board Class
                         * such getAllDataItems(),getAllUpstreamDataItems() and getAllAppItems()
                         */
                }
```

```
        });

  }

  /*
   *
   */
  public Web getWebByName(String name) {
        for (Iterator iter = webs.iterator(); iter.hasNext(); ) {
              Web w = (Web)iter.next();
              if (name.equals(w.geName())) {
                    return w;
              }
        }
        return null;
  }

  /*
   *
   */
  public Board getBoardByName(String name) {
        List boards = currentUserWeb.getBoardsList();
           // getBoardsList method is defined in ATTACHMENT 2, page 14
        for(Iterator iter = boards.iterator(); iter.hasNext(); ) {
              Board b = (Board) iter.next();
              if (name.equals(b.geName())) {
                    return b;
              }
        }
        return null;
 }

} //END-OF-CLASS
```

Class: Board

```
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;

/**
* A Board represents a particular context that consists of a collection of data
* and applications. Building the example of workflow in ATTACHMENT 2, we also
* assume that a Board contains a collection of individuals associated with the context
*/
public class Board {
  /* the basic constituent elements of a Board */
  HashMap<String,DataItem> data; // a hash table of data objects (e.g. files, emails, etc)
  HashMap<String,AppItem>  apps;    // a hash table of applications
  HashSet<String> users;     // a hash table of users associated with the context
                                  // (following the workflow example)
  String name;

  /* another set of elements that might be useful are those data items from upstream
   * contexts in the workflow that that might be relevant to the current context
   */
  HashMap<String,DataItem> upstreamData;

  /**
  * Constructor
  * @param the name of the Board
  */
  public Board(String name) {
    this.name = name;
    data = new HashMap<String,DataItem>();
    apps = new HashMap<String,AppItem>();
    users = new HashSet<String>();

    upstreamData = new HashMap<String,DataItem>();
  }

  /*
   * As users move across contexts, they might find the need to access data items from
   * upstream context in the workflow. This method imports the data items from the
   * Board's parent nodes
   * ASSUMPTION: error would generate some appropriate exception
   */
  public void importDataFromParent(Web w, WebSlice ws) {
         if (w.contains(this)) { // we only do work if this Board belongs to the Web
                                 // method defined in ATTACHMENT 2, Web class, page 13
                 /*
                  * get the set of boards that are part of the workflow of interest as
                  * represented by the webslice
                  */
                 Board[] boardsInWS = ws.getBoards();
                 // method defined in ATTACHMENT 2, WebSlice class, page 18
                 /*
                  * get the list of the parents of the current board.
                  */
                 Set<Boards> parents = w.getParents(this);
                 // method defined in ATTACHMENT 2, Web class, page 14
                 /*
                  * we import data from the parents that are in the webslice
                  */
                 for(int i=0; i < boardsInWS.lentgh()) {
                        if (parents.contains(boardsInWS[i])) {
                          Collection dataToImport = boardsInWS[i].getAllDataItems();
                          for(Iterator iter = dataToImport.iterator(); iter.hasNext();) {
                                DataItem ditem = (DataItem)iter.next();
                                if (!upstreamData.containsKey(ditem.id)) {
                                     upstreamData.put(ditem.id,ditem);
                                }
                          }
                        }
                 }
         }
  }

  /*
   * A particular data item might move from context to context as the workflow
   * progresses. This method transfer an imported data item into the permanent set of
   * data items of the board.
   * ASSUMPTION: error would generate some appropriate exception
```

```
 */
public void transferDataItem(DataItem d) {
	if (!data.containsKey(d.id)) { data.put(d.id,d); }
}

/*
 * As described in ATTACHMENT 2, a particular workflow could be modified such that
 * two interrelated Boards (e.g. A->B->C->D) are merged into a combined context
 * (e.g. A->B/C->D). This method accomplishes such operation.
 * ASSUMPTION: error would generate some appropriate exception
 */
public void merge(Board src) {
	/*
	 * we start by merging the data items
	 */
	Collection dataToMerge = src.getAllDataItems();
	for(Iterator iter = dataToMerge.iterator(); iter.hasNext();) {
		DataItem ditem = (DataItem)iter.next();
		if (!data.containsKey(ditem.id)) { data.put(ditem.id,ditem); }
	}
	/*
	 * 2nd, we merge the list of applications available in this context
	 */
	dataToMerge = src.getAllAppItems();
	for(Iterator iter = dataToMerge.iterator(); iter.hasNext();) {
		AppItem ditem = (AppItem)iter.next();
		if (!apps.containsKey(ditem.id)) { apps.put(ditem.id,ditem); }
	}
	/*
	 * Finally, we merge the set of users associated with the context
	 */
	for(Iterator iter = src.getAllUsers(); iter.hasNext();) {
		String user = (String)iter.next();
		users.add(user);
	}
}

/*
 * accessor methods
 */
public String getName() {
	return this.name;
}
public void addDataItem(DataItem d) {
	data.put(d.id,d);
}
public void removeDataItem(String did) {
	data.remove(did);
}
public void removeDataItem(DataItem d) {
	data.remove(d.id);
}
public Collection<DataItem> getAllDataItems() {
	return data.values();
}
public DataItem getDataItem(String did) {
	return data.get(did);
}
public boolean hasDataItem(String did) {
	return data.containsKey(did);
}
public boolean hasDataItem(DataItem d) {
	return data.containsKey(d.id);
}
public Collection<DataItem> getAllUpstreamDataItems() {
	return upstreamData.values();
}
public void addAppItem(AppItem d) {
	apps.put(d.id,d);
}
public void removeAppItem(String did) {
	apps.remove(did);
}
public void removeAppItem(AppItem d) {
	apps.remove(d.id);
}
public Collection<AppItem> getAllAppItems() {
	return apps.values();
}
public AppItem getAppItem(String did) {
```

```
            return apps.get(did);
    }
    public boolean hasAppItem(String did) {
            return apps.containsKey(did);
    }
    public boolean hasAppItem(AppItem d) {
            return apps.containsKey(d.id);
    }
    public void addUser(String uid) {
            users.add(uid);
    }
    public void removeUser(String uid) {
            users.remove(uid);
    }
    public Iterator<String> getAllUsers() {
            return users.iterator();
    }
    public boolean hasUser(String uid) {
            return users.contains(uid);
    }

} //END-OF-CLASS
```