User Manual for our example

(Book locations search through a network)

1- First apply PIM-to-PSM transformation rules on our PIM present in MAS/PIM_Level. This generates the corresponding psm in MAS/PSM_Level.
2- Then apply code generation rules on the resulting psm. This produces a non functional code in MAS/Code_Level/src/generatedCodeForMasWithMobileAgent
3- After complete manually the generated code to obtain a functional one. For example, the code can be completed as below (where added parts are indicated using a gray background):

NB : the remaining steps (4 and 5) are given bellow using a green background:

Interfaces.java

```
//****************** imports, added manually

import javact.util.ActorProfile;
import javact.util.BehaviorProfile;
import java.util.Vector;
import javact.lang.Actor;


//-------------- RoleEnding

interface RoleEndingBehaviorInterf extends BehaviorProfile {
	public Vector confirmEndOfRole();
}


//-------------- BooksListDeliver Role

interface BooksListDeliverRole extends BooksListDeliverRoleBehaviorInterf, RoleEndingBehaviorInterf, ActorProfile {
}

interface BooksListDeliverRoleBehaviorInterf extends BehaviorProfile {
	public Vector getBooksList();
	public void stopRole();
	void become(RoleEndingBehaviorInterf b);
}


//-------------- Librarian Agent

interface LibrarianAgent extends LibrarianAgentBehaviorInterf, ActorProfile {
}

interface LibrarianAgentBehaviorInterf extends BehaviorProfile {
	// Every interactive method in a role played by an agent, is also defined inside the agent. For more details, see the
	comment on getBooksList() given in the Implementations.java file
	public Vector getBooksList();
}


//-------------- BookChecker Role

interface BookCheckerRole extends BookCheckerRoleBehaviorInterf, RoleEndingBehaviorInterf, ActorProfile {
```

```
}

interface BookCheckerRoleBehaviorInterf extends BehaviorProfile {
        public void stopRole();
        void become(RoleEndingBehaviorInterf b);
}


//-------------- ResultsDeliver Role

interface ResultsDeliverRole extends ResultsDeliverRoleBehaviorInterf, RoleEndingBehaviorInterf, ActorProfile {
}

interface ResultsDeliverRoleBehaviorInterf extends BehaviorProfile {
        public void stopRole();
        void become(RoleEndingBehaviorInterf b);
}


//-------------- MobileBookSeeker Agent

interface MobileBookSeekerAgent extends MobileBookSeekerAgentBehaviorInterf, ActorProfile {
}

interface MobileBookSeekerAgentBehaviorInterf extends BehaviorProfile {
        void become(MobileBookSeekerAgentBehaviorInterf b);
}


//-------------- LibraryManagement Group

interface LibraryManagementGroup extends LibraryManagementGroupBehaviorInterf, ActorProfile {
}

interface LibraryManagementGroupBehaviorInterf extends BehaviorProfile {
        // Methods to join and leave the group
        public boolean joinGroup(Actor a);
        public void leaveGroup(Actor a);

        // Methods to ask for roles
        public boolean askForBookCheckerRole(Actor a);
        public boolean askForResultsDeliverRole(Actor a);
        public boolean askForBooksListDeliverRole(Actor a);
}
```

Implementations.java

```
//****************** imports, added manually

import java.io.Serializable;
import java.util.Vector;

import javact.lang.Actor;
import javact.lang.QuasiBehavior;

import javact.lang.HookInterface;
import javact.lang.GoException;

import javact.util.StandAlone;  // StandAlone allows to have an autonomous behavior, else it will be reactive

// local execution
//import javact.local.CreateCt;
//import javact.local.SendCt;

// network execution
import javact.net.rmi.CreateCt;
```

```
import javact.net.rmi.SendCt;
//------------------------------------------------------ End of imports section -------------------------------------------------------

//****************** Data Model

class SiteIdentifier implements Serializable { // it implements Serializable because, the tag value transfearbale=true

	private String siteID;

	public SiteIdentifier(String siteID) {
		this.siteID = siteID;
	}

	public String getSiteID() {
		return this.siteID;
	}
}

class BookRepository implements Serializable { // it implements Serializable because, the tag value transfearbale=true

	private boolean bookFound;
	private boolean siteVisited;
	private SiteIdentifier site;

	public BookRepository(boolean bookFound, boolean siteVisited, SiteIdentifier site) {
		this.bookFound = bookFound;
		this.siteVisited = siteVisited;
		this.site = site;
	}

	public boolean getBookFound() {
		return this.bookFound;
	}
	public boolean getSiteVisited() {
		return siteVisited;
	}
	public SiteIdentifier getSite() {
		return site;
	}
}

class Book implements Serializable { // it implements Serializable because, the tag value transfearbale=true
	private String isbn;
	private String title;
	private Vector authors;

	public Book(String isbn, String title, Vector authors) {
		this.isbn = isbn;
		this.title = title;
		this.authors = authors;
	}

	public String getIsbn() {
		return isbn;
	}
	public String getTitle() {
		return title;
	}
	public Vector getAuthors() {
		return authors;
	}
}
//------------------------------------------ End of code concerning the Data model ------------------------------------------------

//****************** RoleEnding
```

```
class RoleEndingBehaviorImpl extends RoleEndingBehaviorInterfQuasiBehavior {

        private Vector valuesToReturn;

        public RoleEndingBehaviorImpl (Vector valuesToReturn) {
                this.valuesToReturn = valuesToReturn;
        }

        public Vector confirmEndOfRole() {
                suicide();
                return valuesToReturn;
        }
}
//------------------------------------------- End of code concerning the RoleEnding -------------------------------------------
```

//**************** BooksListDeliver Role**

```
class BooksListDeliverRoleBehaviorImpl extends BooksListDeliverRoleBehaviorInterfQuasiBehavior {

        private Vector booksList;
        private boolean stop;
        public String itsGroup = "LibraryManagement";
        public Vector interactsWith={};
        private Vector valuesToReturn;

        private LibrarianAgentBehaviorImpl owner; // the type of owner is specified manually

        // Constructor
        public BooksListDeliverRoleBehaviorImpl(LibrarianAgentBehaviorImpl owner) {
        // The type of owner is specified manually
                this.owner = owner;
                this.stop = false;
                valuesToReturn = new Vector();
        }

        public Vector getBooksList() {

                booksList = owner.getGlobalVarBooksList(); /* added manually for test. In a real application it has to
                be initialized in the getBooksList() method and not in main program nor here */

                return booksList; /* added from the corresponding StateChart diagram: as send is preceded by
                receive */
        }
        public void stopRole() {
                stop = true;
                become(new RoleEndingBehaviorImpl(this.valuesToReturn));
                System.out.println("\nBooksListDeliver role ended");
        }
}
//------------------------------------- End of code concerning the BooksListDeliver Role -------------------------------------
```

//**************** Librarian Agent**

```
class LibrarianAgentBehaviorImpl extends LibrarianAgentBehaviorInterfQuasiBehavior  implements StandAlone { //
StandAlone allow to have an autonomous behavior; else it will be reactive

        private Vector booksList; /* added manually to serve only for test and will disappear in real applications because
        it has to be initiated inside the BooksListDeliver role and used there*/
        private Actor group; // to indicate the group to join

        private boolean groupJoined=false;
        private boolean roleObtained=false;

        private Actor booksListDeliverRoleActor;  // to point to the actor representing the played role (BooksListDeliver)

        public Vector mayPlay={BooksListDeliver}; // represents the adopts attribute in PIM
```

```
// Constructor
public LibrarianAgentBehaviorImpl(Actor libraryManagementGroup, Vector booksList) { /* arguments are
added manually for test and will disappear in real applications */
        this.group = libraryManagementGroup; /* added manually, for test only (so will disappear in a real
        application) */
        this.booksList = booksList; /* added manually for test only. it will disapear in real application because it
        has to be initiated inside the BooksListDeliver role and used there */
        System.out.println("a Librarian agent created");
}

// Every interactive method in a role played by an agent, is also defined inside the agent. This allows calling
the methods of roles played by this agent from external roles not played by this agent. This method is added
manually just for test and will disappear in real environment where a directory system describing the
services (methods) of roles exist and must be used.
public Vector getBooksList() { // to be called from external roles (not played by this agent)
        JSMgetBooksListVector getBooksListMsg = new JSMgetBooksListVector();
        send(getBooksListMsg, booksListDeliverRoleActor);
        return getBooksListMsg.getReply();
}

public Vector getGlobalVarBooksList() { // added manually. to be called from played roles
        return booksList;
}

public void run() {// the method run exist in the classes which implement StandAlone

        Vector returnedValues; // values returned by played roles

        // Determine the group containing the BooksListDeliver role.
        // For our test, this has been done in the constructor.

        // Join the group. ego() return the actual actor
        JSMjoinGroupboolean joinGroupRequest = new JSMjoinGroupboolean(ego());
        do {
                send(joinGroupRequest, group);
                groupJoined = joinGroupRequest.getReply();
        } while (!groupJoined);
        groupJoined = false;

        // Request the BooksListDeliver role
        JSMaskForBooksListDeliverRoleboolean playingRoleRequest = new
        JSMaskForBooksListDeliverRoleboolean(ego());
        do {
                send(playingRoleRequest, group);
                roleObtained = playingRoleRequest.getReply();
        } while (!roleObtained);
        roleObtained = false;

        // Play the BooksListDeliver role (create a local actor for it)
        booksListDeliverRoleActor = CreateCt.STD.create(myPlace(), new
        BooksListDeliverRoleBehaviorImpl(this)); /* the parameter (this) allow the role to know its owner, and
        myPlace() return the local place*/

/******* hidden section
        // This section illustrate how to stop a role. It is hidden here to allow for the continuation of the service

        // Stop the role because it loops and never stops alone
        JAMstopRole stopRoleMsg = new JAMstopRole(null);
        send(stopRoleMsg, booksListDeliverRoleActor);

        // Wait for the role to end
        JSMconfirmEndOfRoleVector confirmEndOfRoleMsg = new JSMconfirmEndOfRoleVector();
        send(confirmEndOfRoleMsg, booksListDeliverRoleActor);
        returnedValues = confirmEndOfRoleMsg.getReply();
```

```
                    // Extract necessary values from returnedValues
                    // none value to extract here

                    // Leave the group
                    JAMleaveGroup leaveGroupRequest = new JAMleaveGroup(ego());
                    send(leaveGroupRequest, group);

                    // Terminate
                    System.out.println("\nEnd of the the Librarian agent");
                    suicide();

End of hidden section********/
          }
}
//------------------------------------------ End of code concerning the librarian agent ------------------------------------------

//****************** BookChecker Role

class BookCheckerRoleBehaviorImpl extends BookCheckerRoleBehaviorInterfQuasiBehavior implements StandAlone {

          private Vector booksList;
          private boolean stop;
          public String itsGroup = "LibraryManagement";
          private Vector valuesToReturn;
          public Vector interactsWith={BoosListDeliver};
          private Actor playerOfBooksListDeliverRole; // the agent playing the role with which this role interacts

          private MobileBookSeekerAgentBehaviorImpl owner; // the type of owner is specified manually

          // Constructor
          public BookCheckerRoleBehaviorImpl(MobileBookSeekerAgentBehaviorImpl owner) {
          // The type of owner is specified manually
                    this.stop = false;
                    this.owner = owner;
                    valuesToReturn = new Vector();
          }

          private void booksFilter() {
                    // Complete the content manually
                    boolean exist = false;
                    for (int i=0; i<booksList.size(); i++)
                              if
                              (((Book)booksList.elementAt(i)).getIsbn().compareTo((owner.getGlobalVarSearchedBoo
                              k()).getIsbn()) == 0)
                                        exist = true;

                    if (exist)
                              System.out.println("\nthe searched book was founded on this site "+myPlace());
                    else
                              System.out.println("\nthe searched book was not founded on this site "+myPlace());

                    Vector bookRepositories = owner.getGlobalVarBookRepositories();
                    bookRepositories.addElement(new BookRepository(exist, true, new SiteIdentifier(myPlace())));


                    this.valuesToReturn.addElement(bookRepositories);
          }
          public void stopRole() {
                    stop = true;
                    System.out.println("\nBookChecker role ended");
          }


          public void run() {
                    // Initialize the playerOfBooksListDeliverRole attribute
                    playerOfBooksListDeliverRole = ((MobileBookSeekerAgentBehaviorImpl)owner).getLibrarian();
```

```
                JSMgetBooksListVector getBooksListMsg = new JSMgetBooksListVector(); /* because: send is not
                preceded by receive */
                send(getBooksListMsg, playerOfBooksListDeliverRole);
                booksList = getBooksListMsg.getReply(); // because receive is preceded by send

                // Filter the book
                booksFilter();

                // Terminate
                become(new RoleEndingBehaviorImpl(this.valuesToReturn));
        }
}
//---------------------------------------- End of code concerning the BookChecke Role ----------------------------------------

//****************** ResultsDeliver Role

class ResultsDeliverRoleBehaviorImpl extends ResultsDeliverRoleBehaviorInterfQuasiBehavior implements StandAlone {

        private Vector bookRepositories;
        private boolean stop;
        public String itsGroup = "LibraryManagement";
        public Vector interactsWith={};
        private Vector valuesToReturn;

        private MobileBookSeekerAgentBehaviorImpl owner; // the type of owner is specified manually

        // Constructor
        public ResultsDeliverRoleBehaviorImpl(MobileBookSeekerAgentBehaviorImpl owner) {
        // The argument type is specified manually
                this.stop = false;
                this.owner = owner;
                valuesToReturn = new Vector();
        }

        public void stopRole() {
                stop = true;
                System.out.println("\nResultsDeliver role ended");
        }

        private void displayBookRepositories() {

                // Complete the content manually
                String repositoriesList="";
                for (int i=0; i<this.bookRepositories.size(); i++)
                        if (((BookRepository) this.bookRepositories.elementAt(i)).getSiteVisited() &&
                        ((BookRepository) this.bookRepositories.elementAt(i)).getBookFound())
                                repositoriesList = repositoriesList +" "+(((BookRepository)
                                this.bookRepositories.elementAt(i)).getSite()).getSiteID();

                System.out.println("\nMission accomplished: the searched book was founded in "+repositoriesList);
        }

        public void run() {

                // Intialize the bookRepositories var
                bookRepositories = owner.getGlobalVarBookRepositories(); // added manually

                displayBookRepositories();

                // Terminate
                become(new RoleEndingBehaviorImpl(this.valuesToReturn));
        }
}
//---------------------------------------- End of code concerning the ResultsDeliver Role ----------------------------------------
```

```
//****************** MobileBookSeeker Agent

class GoFailed implements HookInterface {
// this class is added automatically to be used inside the migration method by mobiles agents
        Boolean moved;

        public GoFailed (Boolean moved) {
                this.moved = moved;
        }
        public void resume (GoException e) {
                moved = new Boolean(false);
                System.out.println("\nGo failed!!!!!!!!!!!!!! "); // message displayed if the go command failed
        }
}

class MobileBookSeekerAgentBehaviorImpl extends MobileBookSeekerAgentBehaviorInterfQuasiBehavior implements
StandAlone {

        private Vector libraryManagementGroupsList; /* added manually, used to test the program and will disappear
        in a real MAS */
        private Vector librariansList;  // added manually, used to test the program and will disappear in a real MAS

        private Vector itinerary;
        private SiteIdentifier finalSite;
        private SiteIdentifier nextSite;
        private Vector bookRepositories;
        private Book searchedBook;

        private boolean migrantAgent;
        private Actor group; // do not exist in PIM

        private boolean groupJoined=false;
        private boolean roleObtained=false;

        private Actor resultsDeliverRoleActor; // to point the actor representing the played role (ResultsDeliver)
        private Actor bookCheckerRoleActor; // to point the actor representing the played role (BookChecker)

        public Vector mayPlay; // represent the adopts attribute in PIM

        // Constructor
        public MobileBookSeekerAgentBehaviorImpl(Vector libraryManagementGroupsList, Vector librariansList, Vector
        itinerary, SiteIdentifier finalSite, Book searchedBook, Vector bookRepositories, boolean migrantAgent) {
        // arguments are specified manually

                // Any global variable must receive its value here. Indeed, as this agent move, this manner avoids to lost
                values of its global variables
                this.libraryManagementGroupsList = libraryManagementGroupsList; // added manually, used for
                the test and will desapear in a real MAS
                this.librariansList = librariansList; //added manually, used for the test and will desapear in a real MAS
                this.itinerary = itinerary; // added manually
                this.finalSite = finalSite; // added manually
                this.nextSite = (SiteIdentifier) itinerary.firstElement(); // added manually
                this.searchedBook = searchedBook; // added manually
                this.bookRepositories = bookRepositories; // added manually
                this.migrantAgent = migrantAgent; // added manually

                this.mayPlay = new Vector();
                this.mayPlay.addElement("BookChecker"); this.mayPlay.addElement("ResultsDeliver");

                System.out.println("a MobileBookSeeker agent created");
        }

        public Vector getGlobalVarBookRepositories(){ // added manually. to be called from played roles
                return this.bookRepositories;
        }
        public Book getGlobalVarSearchedBook() { // added manually. to be called from played roles
```

```
            return this.searchedBook;
        }
        public Actor getLibrarian() {
            // Added manually just for test. In a real application, the agent must search it, for example, in an
            directory.
            return (Actor) this.librariansList.firstElement();
        }

        private void afterMigration() {
            // Complete the content manually
            // None action in our example
            System.out.println("\nthe AfterMigration action was executed by the MobileBookSeeker agent");
        }
        private void firstSiteInItineraryBecomeNextSite() {
            // Complete the content manually
            this.nextSite = (SiteIdentifier)itinerary.firstElement(); // added manually
        }
        private void finalSiteBecomeNextSite() {
            // Complete the content manually
            this.nextSite = finalSite; // added manually
        }
        private void logTheRaison() {
            // Complete the content manually
            System.out.println("\nraison of move's failure was logged"); // added manually
        }
        private void deleteNextSiteFromItinerary() {
            // Complete the content manually
            this.itinerary.removeElementAt(0); // added manually
        }


        private void deleteLocalSiteFromItinerary() {
            // Complete the content manually
            this.itinerary.removeElementAt(0); // added manually
        }
        private boolean itineraryIsEmpty() {
            // Complete the content manually
            return this.itinerary.isEmpty(); // added manually
        }
        private void migration (Vector itinerary, Actor group, boolean clone) {
            // Update libraryManagementGroups List. This list exits for test and disappear in a real application
            libraryManagementGroupsList.removeElementAt(0);
            // Update librarians List. This list exists only for test and disappears in a real application
            librariansList.removeElementAt(0);

            // Delete localSite from itinerary
            deleteLocalSiteFromItinerary();

            // Leave the library management group
            JAMleaveGroup leaveGroupRequest = new JAMleaveGroup(ego());
            send(leaveGroupRequest, group);

            SiteIdentifier precedantSite;
            // Specify the value of the precedantSite
            precedantSite = nextSite;

            Boolean moved;

            do {        // Set nextSite
                        if (! itineraryIsEmpty())
                                    firstSiteInItineraryBecomeNextSite();
                        if (itineraryIsEmpty())
                                    finalSiteBecomeNextSite();

                        // Jump to nextSite
                        moved = new Boolean(true);
```

```
                /*
                 go(p, h) tries to move the actor to the place p. if a GoException is raized, the library returns
                 control to the resume(GoException e) method of the object h. Inside the resume method we
                 eventually update the value of the attribute "moved".
                 */
                go(nextSite.getSiteID(), new GoFailed(moved));

                if (!moved.booleanValue()) {
                        logTheRaison();
                        deleteNextSiteFromItinerary();
                }
        } while ( ! moved ); // jump failure

        this.migrantAgent =  true;

        if (! clone)
                become(this); // a mobile agent takealways its same behavior after moves
        else {  /*
                 De-comment the instruction below and the MobileBookSeekerAgentBehaviorImpl class
                 must implement the Cloneable interface and rewrite the clone() method.
                */
                //become(this.clone());
        }

        System.out.println("\nthe agent move to "+nextSite.getSiteID()); // added manually
}



public void run() {
                Vector returnedValues; // values returned by played roles

                if (migrantAgent) // not first launch
                        afterMigration(); // We do not consider the name of the action in PIM

                // Specify manually the group to join
                 group = (Actor) libraryManagementGroupsList.firstElement(); /* added manually for test
                 and will disappear in a real application */

                // Join the group
                JSMjoinGroupboolean joinGroupRequest = new JSMjoinGroupboolean(ego());
                do {
                        send(joinGroupRequest, group);
                        groupJoined = joinGroupRequest.getReply();
                } while (!groupJoined);
                groupJoined = false;

                if (itineraryIsEmpty()) { // missionTerminated

                        // Request the ResultsDeliver role
                         JSMaskForResultsDeliverRoleboolean playingRoleRequest = new
                         JSMaskForResultsDeliverRoleboolean(ego());
                        do {
                                send(playingRoleRequest, group);
                                roleObtained = playingRoleRequest.getReply();
                        } while (!roleObtained);
                        roleObtained = false;

                        // Play the ResultsDeliver role (create a local actor for it)
                         resultsDeliverRoleActor = CreateCt.STD.create(myPlace(), new
                         ResultsDeliverRoleBehaviorImpl(this)); // the parameter (this) allow the role to know
                         its owner, and myPlace() return the local place

                        // Wait the end of the role
                         JSMconfirmEndOfRoleVector confirmEndOfRoleMsg = new
```

```
                    JSMconfirmEndOfRoleVector();
                    send(confirmEndOfRoleMsg, resultsDeliverRoleActor);
                    returnedValues = confirmEndOfRoleMsg.getReply();

                    // Extract necessary values from returnedValues
                    // None value to extract here

                    // Leave the groupe
                    JAMleaveGroup leaveGroupRequest = new JAMleaveGroup(ego());
                    send(leaveGroupRequest, group);

                    /* Decomment this section to test the case where an agent move to an inexistant place

                    System.out.println("\nOur mobile agent has finished its mission; but before it ends,
                    we just test (the HookInterface) by trying to move the agent to an inexistant place");
                    System.out.println("\ntrying to move to 192.168.0.30:5000 .....");
                    go("192.168.0.30:5000", new GoFailed(true));
                    */

                    System.out.println("\nEnd of the MobileBookSeeker agent"); // added manually
                    suicide();
                } else {

                    // Request the BookChecker role
                    JSMaskForBookCheckerRoleboolean playingRoleRequest = new
                    JSMaskForBookCheckerRoleboolean(ego());
                    do {
                        send(playingRoleRequest, group);
                        roleObtained = playingRoleRequest.getReply();
                    } while (!roleObtained);
                    roleObtained = false;

                    // Play the BookChecker role (create a local actor for it)
                    bookCheckerRoleActor = CreateCt.STD.create(myPlace(), new
                    BookCheckerRoleBehaviorImpl(this)); // myPlace() return the actual place

                    // Wait the end of the role
                    JSMconfirmEndOfRoleVector confirmEndOfRoleMsg = new
                    JSMconfirmEndOfRoleVector();
                    send(confirmEndOfRoleMsg, bookCheckerRoleActor);
                    returnedValues = confirmEndOfRoleMsg.getReply();

                    // Extract necessary values from returnedValues
                    bookRepositories = (Vector) returnedValues.elementAt(0); // added manually

                    System.out.println("\nIt is time for the MobileBookSeeker to move"); // added
                    manually

                    migration(itinerary, group, false);
                    /*
                    The last parameter indicates if the migration is based on cloning or not. false means
                    no.
                    */
                }
        }
}
//----------------------------- End of code concerning the MobileSeeker agent and its roles ---------------------------------

//****************** LibraryManagement Group

class LibraryManagementGroupBehaviorImpl extends LibraryManagementGroupBehaviorInterfQuasiBehavior {

        private Vector agents; // to contain the list of agents that have joined the group
        private Vector roles= {BooksListDeliver,BookChecker,ResultsDeliver}; // to contains the list of roles in the group

        public LibraryManagementGroupBehaviorImpl() {
```

```
                this.agents = new Vector();
                System.out.println("\na LibraryManagementGroup created"); // added manually
        }

        // None check is expressed here, so all requests are accepted. Checks may be elaborated according to the business
        domain

        // Group methods
        public boolean joinGroup(Actor a) {
                // Add the agent to the group
                agents.addElement(a);
                System.out.print("\nan agent has JOINED the group: "); // added manually
                return true;
        }
        public void leaveGroup(Actor a) {
                // Delete the agent from the group
                agents.removeElementAt(agents.indexOf(a));
                System.out.println("\nthe agent LEAVE the group"); // added manually
        }


        // MobileSeeker roles
        public boolean askForBookCheckerRole(Actor a) {
                // Check if the agent may play the role
                System.out.println("the MobileBookSeeker agent has asked to PLAY the BookChecker role"); //
                added manually
                return true;
        }
        public boolean askForResultsDeliverRole(Actor a) {
                // Check if the agent may play the role
                System.out.println("the MobileBookSeeker agent has asked to PLAY the ResultsDeliver role"); //
                added manually
                return true;
        }

        // Librarian roles
        public boolean askForBooksListDeliverRole(Actor a) {
                // Check if the agent may play the role
                System.out.println("the librarian agent has asked to PLAY the BooksListDeliver role"); // added
                manually
                return true;
        }
}
//--------------------------------- End of code concerning the LibraryManagement group -----------------------------------
```

4- After the manual completeness of the code, we write a main class *BookSearchingApplication.java* to test it.

BookSearchingApplication.java (main program)

```
import javact.lang.Actor;

import java.util.Vector;

// local execution
//import javact.local.CreateCt;
//import javact.local.SendCt;

// network execution
import javact.net.rmi.CreateCt;
import javact.net.rmi.SendCt;
```

```
public class BookSearchingApplication {

	public static void main(String[] args) {

		if (args.length == 4) { // args 0, 1, 2 represents site1, site2 and site3. arg 3 represents the laptop

			// construct the itinerary
			Vector itinerary = new Vector();
			for (int i = 0; i < args.length-1; i++)
				itinerary.addElement(new SiteIdentifier(args[i]));

			// specify finalSite
			SiteIdentifier finalSite = new SiteIdentifier(args[3]);

			// create group managers
			System.out.println("\nCreation of a group manager on: "+args[0]+", "+args[1]+", "+args[2]+"
			and "+args[3]);

			Actor libraryManagementGroup1 = CreateCt.STD.create(args[0], new
			LibraryManagementGroupBehaviorImpl());
			Actor libraryManagementGroup2 = CreateCt.STD.create(args[1], new
			LibraryManagementGroupBehaviorImpl());
			Actor libraryManagementGroup3 = CreateCt.STD.create(args[2], new
			LibraryManagementGroupBehaviorImpl());
			Actor libraryManagementGroup4 = CreateCt.STD.create(args[3], new
			LibraryManagementGroupBehaviorImpl());

			// As javact do not manage groups, we place groups in a vector and we pass it to the mobile
			agent (for test only)
			Vector libraryManagementGroupsList = new Vector();
			libraryManagementGroupsList.addElement(libraryManagementGroup1); // on site1
			libraryManagementGroupsList.addElement(libraryManagementGroup2); // on site2
			libraryManagementGroupsList.addElement(libraryManagementGroup3); // on site3
			libraryManagementGroupsList.addElement(libraryManagementGroup4); // on the laptop

			// create books list
			Vector booksList1 = new Vector();
			Vector autors = new Vector();
			autors.addElement("Charles BAUDELAIRE");
			booksList1.addElement(new Book("9782266083263", "LES FLEURS DU MAL", autors)); //
			this book exist in booksList1 and booksList3
			autors = new Vector();
			autors.addElement("Robert BALDICK"); autors.addElement("J.K.HUYSMANS");
			autors.addElement("Patrick McGuinness");
			booksList1.addElement(new Book("9780140447637", "AGAINST NATURE", autors)); // this
			book exist in booksList1 and bookList2

			Vector booksList2 = new Vector();
			booksList2.addElement(new Book("9780140447637", "AGAINST NATURE", autors));
			autors = new Vector();
			autors.addElement("Arthur RIMBAUD");
			booksList2.addElement(new Book("9780811201841", "Illuminations", autors)); // this book
			exist only in booksList2

			Vector booksList3 = new Vector();
			autors = new Vector();
			autors.addElement("Jacques PREVERT"); // this book exist only in booksList3
			booksList3.addElement(new Book("9782070367627", "PAROLES", autors));
			autors = new Vector();
			autors.addElement("CHARLES BAUDELAIRE");
			booksList3.addElement(new Book("9782266083263", "LES FLEURS DU MAL", autors));

			// specify the searched book
			autors = new Vector();
			autors.addElement("Charles BAUDELAIRE");
			Book searchedBook = new Book("9782266083263", "LES FLEURS DU MAL", autors);
```

```
                    // create stationary agents (the three librarians), and we pass (just for test) to each one its
                    boosList
                    System.out.println("\nCreation of a librarian agent on: "+args[0]+", "+args[1]+" and "+args[2]);

                    Actor librarian1 = CreateCt.STD.create(args[0], new
                    LibrarianAgentBehaviorImpl(libraryManagementGroup1, booksList1));
                    Actor librarian2 = CreateCt.STD.create(args[1], new
                    LibrarianAgentBehaviorImpl(libraryManagementGroup2, booksList2));
                    Actor librarian3 = CreateCt.STD.create(args[2], new
                    LibrarianAgentBehaviorImpl(libraryManagementGroup3, booksList3));



                    // As javact do not manage agents' registration, we place librarians in a vector and we pass it to
                    the mobile agent (for test only)
                    Vector librariansList = new Vector();
                    librariansList.addElement(librarian1);
                    librariansList.addElement(librarian2);
                    librariansList.addElement(librarian3);

                    // create the mobile agent
                    System.out.println("\nCreation and launch of a MobileBookSeeker agent on "+args[0]+", to
                    search for the book("+searchedBook.getIsbn()+", "+searchedBook.getTitle()+",
                    "+searchedBook.getAuthors().toString()+")");

                    Vector bookRepositories = new Vector(); // empty at the begining
                    boolean migrantAgent = false;

                    Actor mobileBookSeeker = CreateCt.STD.create(args[0], new
                    MobileBookSeekerAgentBehaviorImpl(libraryManagementGroupsList, librariansList,
                    itinerary, finalSite, searchedBook, bookRepositories, migrantAgent));


            } else
                    System.out.println("\nSpecify 4 places for the execution of agents");

        }
}
```

5- Arriving to this point, we follow the steps below to install JavAct, generate intermediate classes (such as "... QuasiBehavior ..." and "JAM ..." or "... JSM"), compile all the resulting classes, launch JavAct home systems and finally launch our application.

/********** Installation, Compilation and Run commands

5.1- To install JavAct, JRE 1.3 (or higher) and JSDK 1.4 (or higher) are required
Download JavAct version 0.5.1 from http://www.irit.fr/JavAct
Run this command (only once): java installJavAct JavActv051.jar . // to install JavAct in the current directory

5.2- Generate automatically, the classes "...QuasiBehavior..." which implement the become methods and the classes corresponding to messages. Use the command: JavActv051/bin/javactgen *.java

5.3- Compile the set of classes: JavActv051/bin/javactc *.java

5.4- Launch a JavAct host system on 4 places; for example: 192.168.0.3:1200 (as site1), 192.168.0.3:1300 (as site2), 192.168.0.3:1400 (as site3) and 192.168.0.2:1200 (as laptop):

- On 192.168.0.3, execute:

```
JavActv051/bin/javactvm 1200
JavActv051/bin/javactvm 1300
JavActv051/bin/javactvm 1400
```

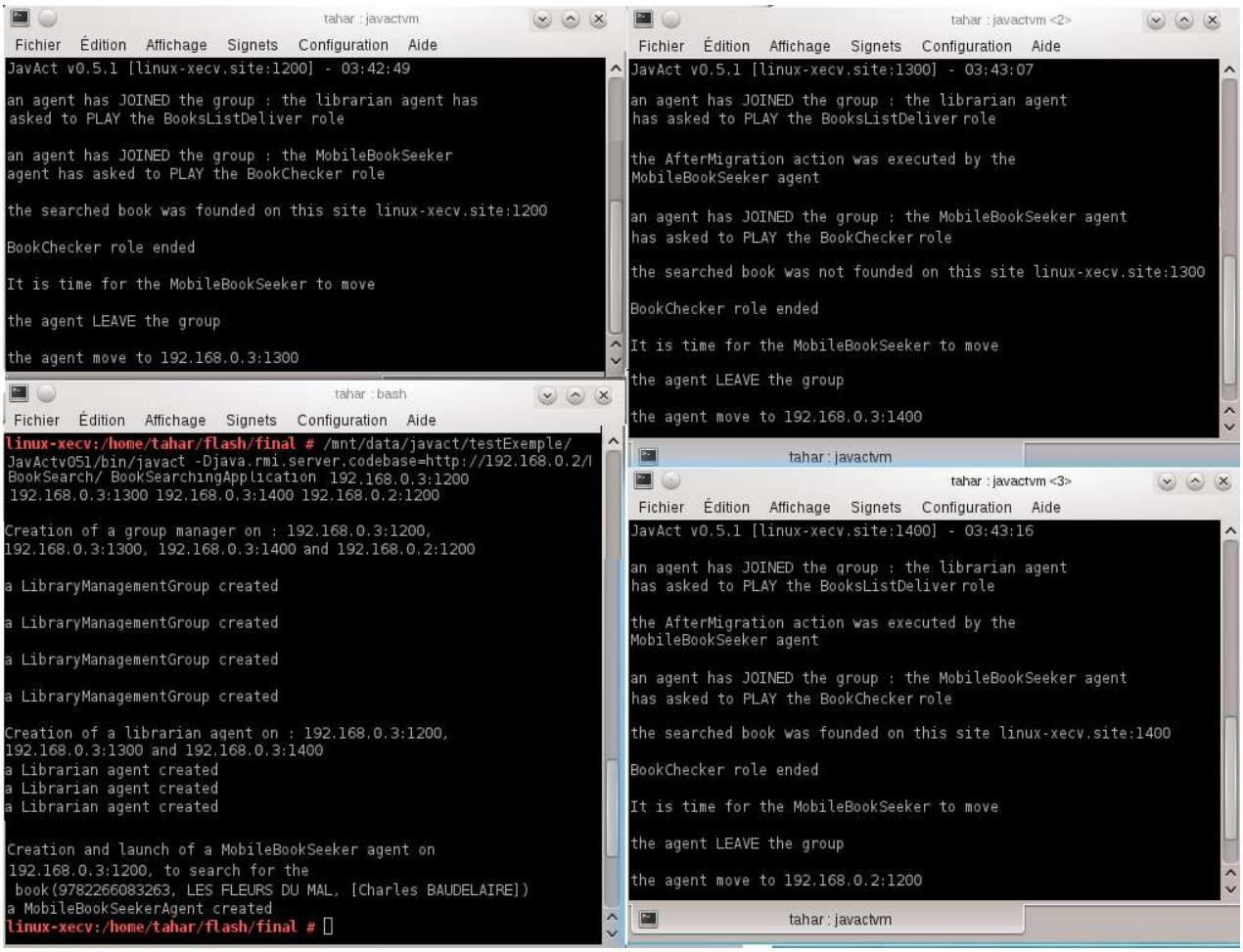
- On 192.168.0.2, execute:
  JavActv051/bin/javactvm 1200

The step 3 is done only once.

5.5- Make the classes available for a distributed execution; for example, by placing them into a BookSearch directory managed by a web server on 192.168.0.2

5.6- Launch the application:
JavActv051/bin/javact -Djava.rmi.server.codebase=http://192.168.0.2/BookSearch/ BookSearchingApplication 192.168.0.3:1200 192.168.0.3:1300 192.168.0.3:1400 192.168.0.2:1200

*/



The schell screen on bottom left of Figure 1 was used to launch the mobile agent. This later started running on the site 192.168.0.3:1200 (see schell screen on top left of Figure 1), then moved to the site 192.168.0.3:1300 (see schell screen on top right of Figure 1), then moved to the site 192.168.0.3:1400 (see schell screen on bottom right of Figure 1). Finally, the mobile agent moved to the site 192.168.0.2:1200 representing the laptop (see Figure 2).

Figure 1. Preview of the launch of the application of our example from a Linux machine

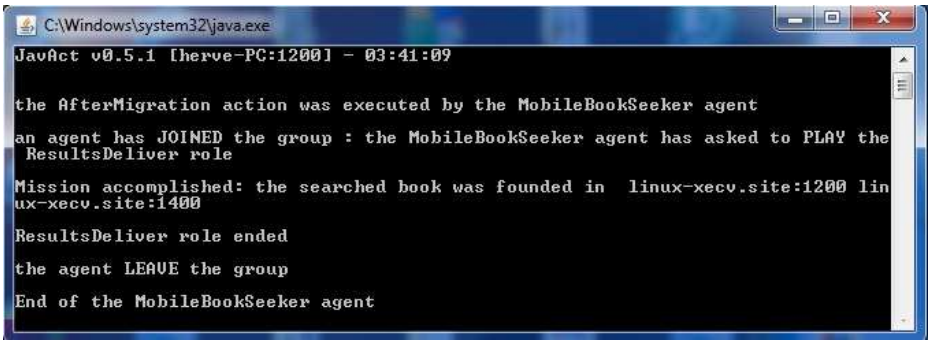


Figure 2. Preview of the end of the mobile agent execution on the laptop (under Windows XP)

The screens in figures 1 and 2 show the execution of our example. Four home JavAct systems are launched (Figure 1-a) on places (sites): 192.168.0.3:1200 (site1, Figure 1-b), 192.168.0.3:1300 (site2, Figure 1-c), 192.168.0.3:1400 (site3, Figure 1-d) and 192.168.0.2:1200 (laptop, Figure 2). Two stationary agents (representing a librarian and a group) are created on each site: site1, site2 and site3. Finally, a mobile agent is created on site1 to visit the three sites (site1, site2 and site3), and then return back on laptop to display its results (Figure 2).