A Comparative Evaluation of Tests Generated from Different UML Diagrams: Diagrams and Data

Aynur Abdurazik and Jeff Offutt
Information and Software Engineering
George Mason University, USA
{aynur,ofut}@ise.gmu.edu

Andrea Baldini
Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
baldini@polito.it

April 2005

GMU Technical Report ISE-TR-05-04, April 2005

Abstract

This paper presents a single project experiment on the fault revealing capabilities of model-based test sets. The tests are generated from UML statecharts and UML sequence diagrams. This experiment found that the statechart test sets did better at revealing unit level faults than the sequence diagram test sets, and the sequence diagram test sets did better at revealing integration level faults than the statechart test sets. The statecharts also resulted in more test cases than the sequence diagrams. The experiment showed that model-based testing can be used to systematically generate test data and indicates that different UML models can play different roles in testing.

This technical report version includes appendices with data that does not fit in a published paper.

1 Introduction

Traditional testing has often generated tests from program source code, usually by abstracting the program into control flow diagrams, data flow graphs, call graphs, or other high level representations. Techniques to derive tests from formal specifications have also been published. A perhaps more general term is that of *model-based testing*, which generally creates tests from some model of the software, including formal specifications and semi-formal design descriptions such as UML diagrams.

When deriving tests from model-based descriptions of the software, the test engineer must choose among a variety of models, and also decide when during testing is it best to apply the various models. These are questions that can only be answered with empirical data. This paper presents data on these questions, comparing the use of statecharts and sequence diagrams for unit and model software testing. Specifically, this study asked whether tests that are designed from one type of software model to (theoretically) target one type of faults will find the faults that are targeted, and furthermore, whether the same tests can find other types of faults. This experiment is based on a non-trivial model of a cell phone, and tests are derived from UML statechart and sequence diagrams, and faults seeded into the software by hand.

In this paper, *unit and module testing* (or just unit testing) is the testing of program units and modules (including classes and packages) independently from the rest of the program. *Integration testing* refers to testing interfaces between units and modules to assure that they have consistent assumptions and communicate correctly [4]. This is in contrast to *system testing* where the objective is test the entire integrated system as a whole. A *test criterion* is a collection of rules that lead to *test requirements*, or specific elements in a program or model that must be covered during testing. Test suites are measured by the percent of requirements that they satisfy.

2 The Unified Modeling Language

The Unified Modeling Language (UML) is a collection of languages for specifying, visualizing, constructing, and documenting the artifacts of software systems [11]. In the UML, complex systems are designed and modeled through a collection of views of a model. The UML defines nine separate graphical diagrams to specify and design software.

Sequence diagrams capture time dependent (temporal) sequences of interactions between objects. Sequence diagrams can be transformed to equivalent collaboration diagrams. Message sequence descriptions are provided in sequence diagrams to elicit meanings of the messages passed between objects. Sequence diagrams describe interactions among software components, and thus are naturally considered to be a good source for integration testing.

Sequence diagrams include flows of events during interactions, with primary flows and *alternative* flows. Alternative flows represent conditional branches in the processing. For example, we describe the normal flow of events for "make phone call" as a flow of events that result in a successful call. Alternatives for this interaction include other event flows that cause "make phone call" to fail, including "callee busy," "network unavailable," and "caller aborts the call before connection is made."

Statechart diagrams describe software behaviors with states and state transitions. They define the dynamic behavior of software in terms of how it responds to external stimuli. Statechart diagrams are especially useful for modeling reactive objects whose states are triggered by specific events. Statechart diagrams describe behavior of individual software components, and thus are naturally considered to be a good source for unit testing.

3 Description of the Experiment

To be consistent with previous experiments, we choose to use the experimental framework by Basili et al. [3]. Their suggested framework contains four phases: (1) definition, (2) planning, (3) operation, and (4) interpretation. Table 1 shows the definition phase of this experiment. The motivation of this experiment is to understand the roles of different UML diagrams in test case generation. To achieve this goal, test cases that are generated from UML statecharts and sequence diagrams are used both at unit and integration level testing, and their fault revealing capabilities are compared. This experiment is designed from the perspective of a researcher, and is carried out as a case study (single project).

Motivation	Understand the roles of different UML diagrams in test case generation
Object	Theory
Purposes	Characterize the test cases that were generated from different UML diagrams, and compare their fault revealing capability
Perspective	Researcher
Domain	Project
Scope	Single project

Table 1: **Study Definition**

3.1 Hypotheses

A number of papers in the literature have assumed that effective tests at several levels (unit/module, integration, and system) can be created by basing the tests on specification and design model artifacts that describe aspects of the software at those levels [1, 5, 10, 12, 17]. One of our goals is to evaluate that assumption. As a beginning, we compare the fault revealing ability of tests cases generated from artifacts at different levels.

We are also interested in how many test cases are required for different specification and design artifacts. We are looking for a correlation between the number of test cases in a test set and the number of faults revealed by that test set. As a beginning, we empirically compare tests derived from UML statecharts, which are used to describe units and modules, and sequence diagrams, which are used to define module integration.

The null hypotheses for our experiment are:

- H_{01}. There is no difference between the number of faults revealed by statechart and sequence diagram test sets at unit and integration testing levels.
- H_{02}. There is no difference between the number of test cases generated from statecharts and sequence diagrams.

3.2 Independent and Dependent Variables

Independent variables in this experiment are the types of UML diagrams, the testing levels used, the criteria used to create tests, and the faults that are used at each testing level. Statecharts and sequence diagrams are used because they are intended to help developers describe software at different levels of abstraction and because criteria for generating tests have previously been defined that easily be applied to these diagrams. The criteria based on statecharts are designed to be applied during unit and module testing, and the criteria based on sequence diagrams are designed to be applied during integration testing. These criteria are defined in Section 3.4. The faults are inserted by hand.

The dependent variables of the experiment are the two sets of test cases that are generated and the number of faults found at each level using these test sets.

3.3 Experimental Subjects

We modeled software for a typical cell phone and developed five types of experimental materials.

1. Specification and design documents of the cell phone handset system. This includes a class diagram of six classes, five statechart diagrams, and six sequence diagrams with 37 alternatives.
2. The implementation of the above specification and design, including eight classes of about 600 lines of Java code.
3. Test cases that are generated from the statecharts and sequence diagrams. There were **81** tests for the statecharts and **43** from the sequence diagrams.
4. A collection of **49** unit and integration level faults, each of which was placed into a separate copy of the program.
5. Unix shell scripts that run the test cases on each faulty version of the implementation and record the results.

Space prohibits including all the UML diagrams in this paper. The diagrams, implementation, and other experimental subject material are provided in appendices. The use case diagram is shown in Figure 1. The cell phone functions include system initialization, making a call, answering a call, turning off the cell phone, notification of an incoming call, and notification of an incoming text message. Their behaviors are described with six sequence diagrams with a total of 37 alternatives. These diagrams were drawn with Rational Rose and range in size from seven states with two levels of abstraction to two states.

The class diagram is shown in Figure 2. The cell phone software has eight classes, five of which use state dependent design. So there are five state charts: *UserInterface*, *HandsetController*, *NetworkInterface*, *Transmitter*, and *Receiver*. Their statecharts are shown in Figures 3 through 7.

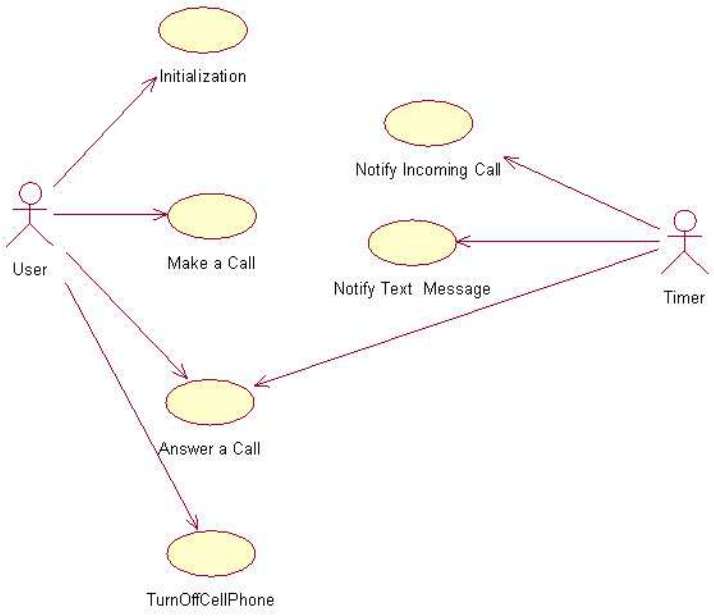

Figure 1: Cell Phone Use Case Diagram

3.4 Generating Test Cases

Test cases were generated by hand. To eliminate bias, the first author wrote the software and inserted the faults, and the third author generated the tests. The test criteria used and process followed for each diagram are described below.

Sequence diagrams: In the UML, a *message* is a request for a service from one UML actor to another. These are typically implemented as method calls. Each sequence diagram represents a complete trace of messages during the execution of a user-level operation. We form *message sequence paths* by using the messages and their sequence numbers. *Message sequence paths* can be traces of system level interactions or component (object) level interactions. The following coverage criteria for generating tests from sequence diagrams was defined in our previous paper [1].

Message sequence path coverage: *For each sequence diagram in the specification, there must be at least one test case T such that when the software is executed using T, the software that implements the message sequence path of the sequence diagram is executed.*

Statecharts: UML statecharts are based on finite state machines using an extended Harel state chart notation, and are used to represent the behavior of an object. The *state* of an object is the combination of all values of attributes and objects the object contains. Objects' behaviors are modeled through transitions among states.

We use the *full predicate (FP)* test case generation criterion defined by Offutt et al. [12, 13]. FP is similar to the masking form of MCDC [6, 7] and CACC [2]. Statecharts represent *guards* and *actions* on transitions using predicates. The guards are conditions that must be true for the transition to be taken, and the actions represent what happens when the transition is taken. Full predicate coverage requires that for every transition, every predicate and every clause within the predicate has taken every outcome at least once, and every clause has been shown to independently affect its predicate.

Full Predicate Coverage: *For each predicate P on each transition, the test set T includes tests that cause each clause c in P to result in a pair of outcomes where the value of P is directly correlated with the value of c.*

In this definition, "directly correlated" means that c controls the value of P, that is, one of two situations

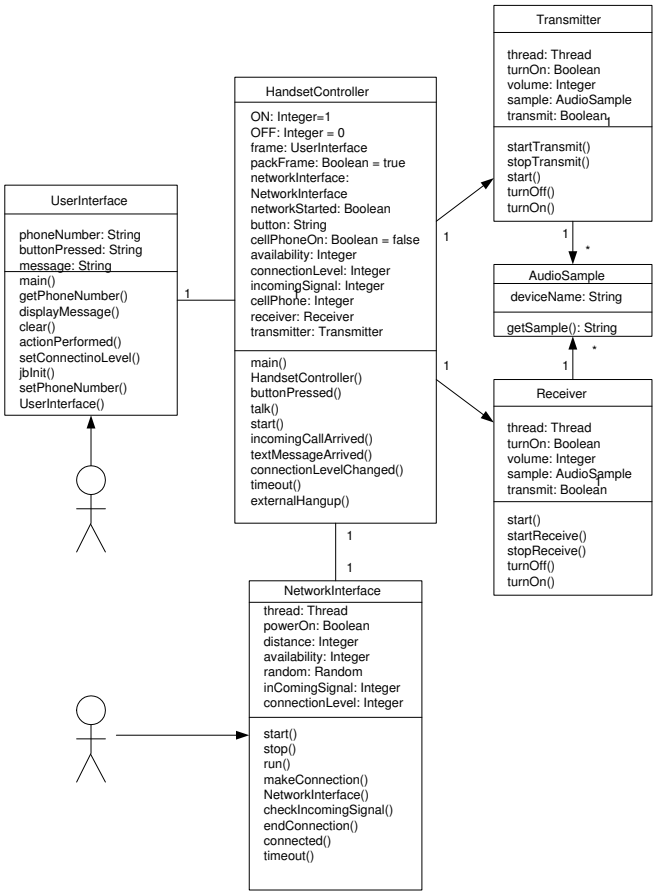

Figure 2: Cell Phone Class Diagram

occurs. Either c and P have the same value (c is true implies P is true and c is false implies P is false), or c and P have opposite values (c is true implies P is false and c is false implies P is true).

To satisfy the requirement that the *test clause* controls the value of the predicate, other clauses in the predicate must have specific values. For example, if the predicate is $(X \wedge Y)$, and the test clause is X, then Y must be **True**. If the predicate is $(X \vee Y)$, Y must be **False**.

At the complete sequence level, test engineers must use their experience and judgment to develop sequences of states that should be tested.

As stated above, the full predicate and complete sequence coverage criteria are used with modifications in generating tests from statecharts. The original full predicate coverage criterion was based on the notion of a predicate. The criterion considers transitions that are triggered by change events with or without other conditions that can be expressed in boolean expressions. However, UML statecharts have other types of events, *call events* and *signal events*. These events cannot be mapped directly into the existing full predicate testing method. Also, instead of considering each transition at a time, a complete sequence of transitions is considered for test case generation. To generate test cases, we first find out what event can trigger the starting transition of the statechart and under what conditions the event can be triggered. We then choose values to cause that event to occur and to satisfy the conditions. Next, we assign other values as necessary for the statechart to have the chosen complete sequence.

A total of 81 test cases were generated from the statecharts and 43 from the sequence diagrams. As an example, consider the transition from "*waiting for phone number*" to "*making call*" in the state diagram for

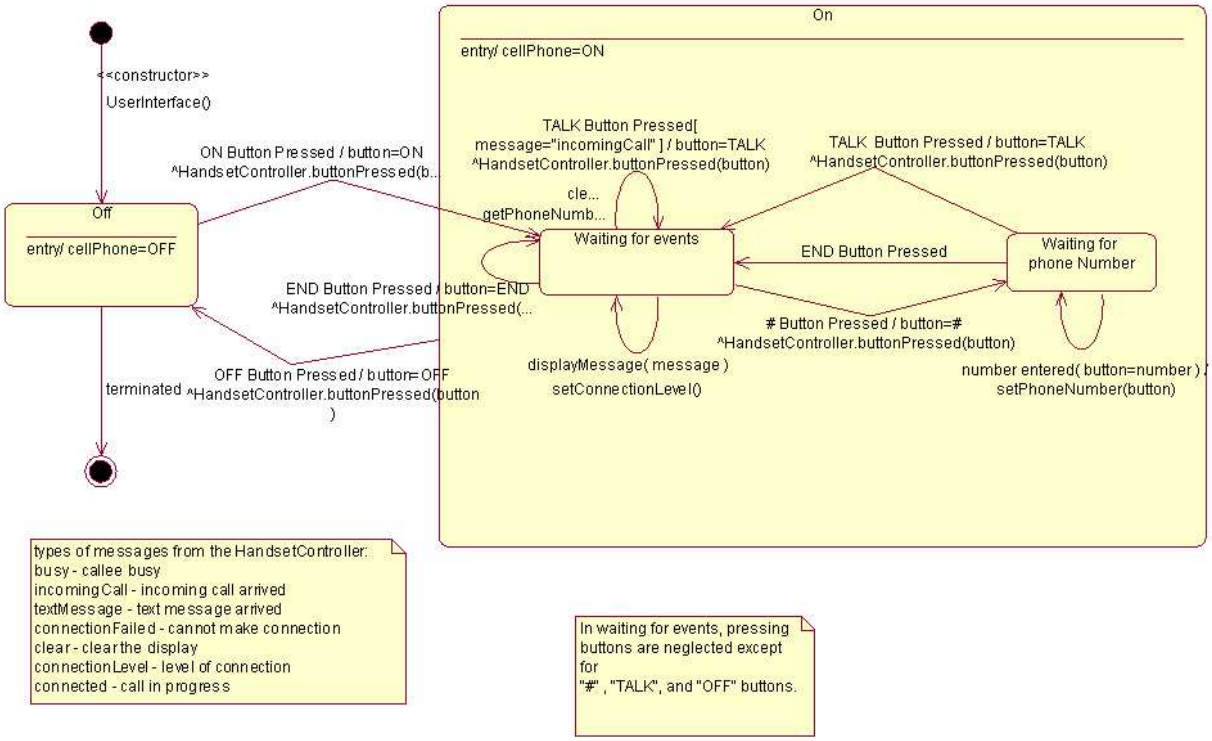

Figure 3: User Interface State Chart

the handset controller, shown in Figure 7. The predicate on the transition is:

$buttonPressed\ (button = TALK)\ [cl > 0]\ /\ getPhoneNumber() \wedge NI.makeConnection\ (phoneNumber)$

The guard, $cl > 0$, refers to connection level, and takes an integer from 0 to 5. This guard indicates that the user can only make a call if the phone has connection. This guard results in two tests, one where $cl > 0$ and one where $cl = 0$.

Tests were created as sequences of method calls to the associated object. The mapping of values to method calls was done by hand. The tests for this transition are:

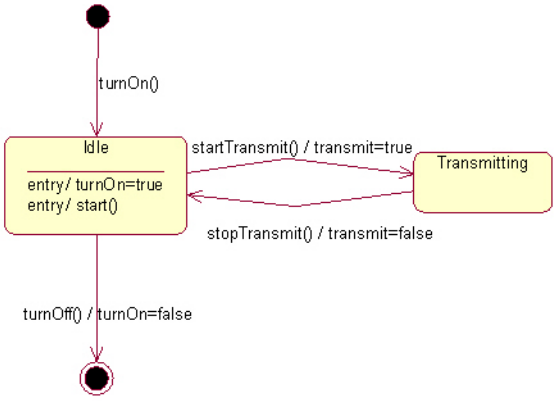

Figure 4: Transmitter State Chart

Test 1

```
HandsetController.printStateInfo();
HandsetController.printStateInfo();
HandsetController.buttonPressed ("ON");
HandsetController.printStateInfo();
HandsetController.buttonPressed ("#");
HandsetController.printStateInfo();
HandsetController.connectionLevelChanged (0);
HandsetController.printStateInfo();
HandsetController.buttonPressed ("TALK");
HandsetController.printStateInfo();
HandsetController.buttonPressed ("OFF");
HandsetController.printStateInfo();
HandsetController.kill();
HandsetController.printStateInfo();
```

Test 2

```
HandsetController.printStateInfo();
HandsetController.printStateInfo();
HandsetController.buttonPressed ("ON");
HandsetController.printStateInfo();
HandsetController.buttonPressed ("#");
HandsetController.printStateInfo();
HandsetController.connectionLevelChanged (3);
HandsetController.printStateInfo();
HandsetController.buttonPressed ("TALK");
HandsetController.printStateInfo();
HandsetController.buttonPressed ("OFF");
HandsetController.printStateInfo();
HandsetController.kill();
HandsetController.printStateInfo();
```

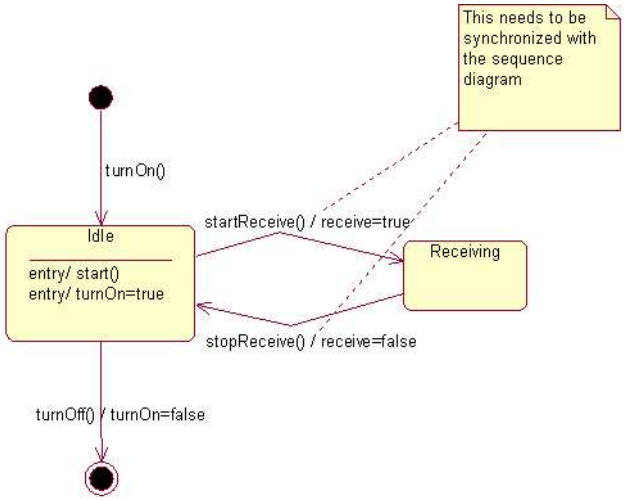

Figure 5: Receiver State Chart

3.5 Program Faults

Unit level and integration level faults were inserted into the implementation by hand. We define a *unit level fault* to cause incorrect behavior of a unit when executed in isolation. This includes most of the traditional mutation operators, including variable reference faults, operator reference faults, associative shift faults, variable negation faults, and expression negation faults [8, 9]. We define an *integration fault* to cause two or more units to interact together incorrectly, even if they are correct when tested in isolation. This includes faults such as incorrect method call, incorrect parameter passing, and incorrect synchronization. For this experiment, 30 unit level faults and 20 integration level faults were designed. We found one existing unit level fault in the implementation, and three integration faults turned out to be similar and all failed under the same conditions. These were combined, resulting in 31 unit level faults and 18 integration faults.

The faults were inserted and tested in the following manner: one faulty version of the program was created at a time, and then ran against each test cases in turn until either the fault was revealed or all test cases are executed. A fault is considered to be *revealed* if the output of the faulty version of the program is different from that of the original program on the same input. That is, we used the original program as the "oracle." The faults were kept in separate versions of the program to make bookkeeping easier (when a failure occurred, it was clear which fault was found) and to avoid interactions between faults such as masking.

3.6 Experimental Procedure

The experiment followed the following five steps.

1. Analyze and specify the cell phone handset system using UML diagrams. This step resulted in class, statecharts, collaboration/sequence, and use case diagrams (first author).

2. Implement the system in Java (first author).

3. Generate test cases by hand from statecharts (81) and sequence diagrams (43) to satisfy the testing criteria (third author).

4. Design faults by hand for unit and integration testing level and insert them into the implementation. The second author designed the faults and the first author implemented them.

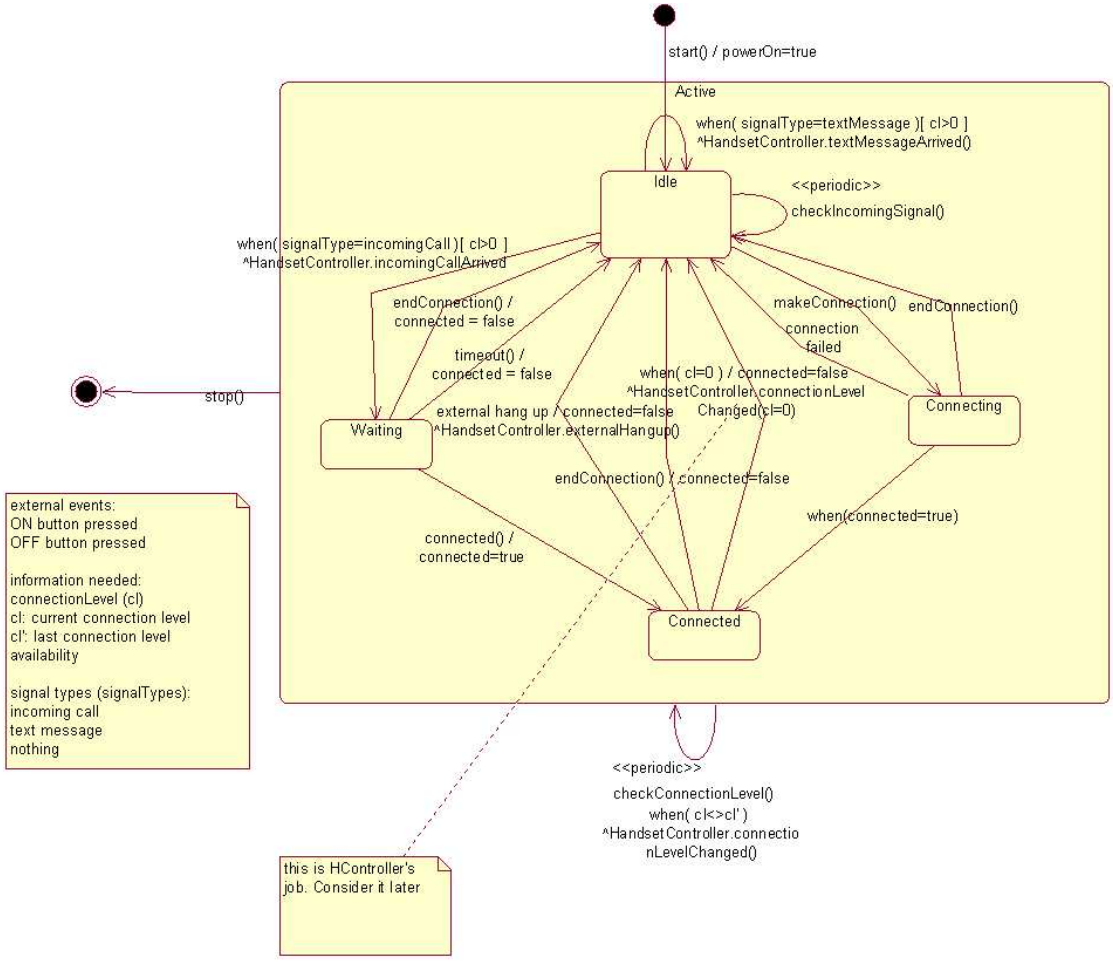

Figure 6: Network Interface State Chart

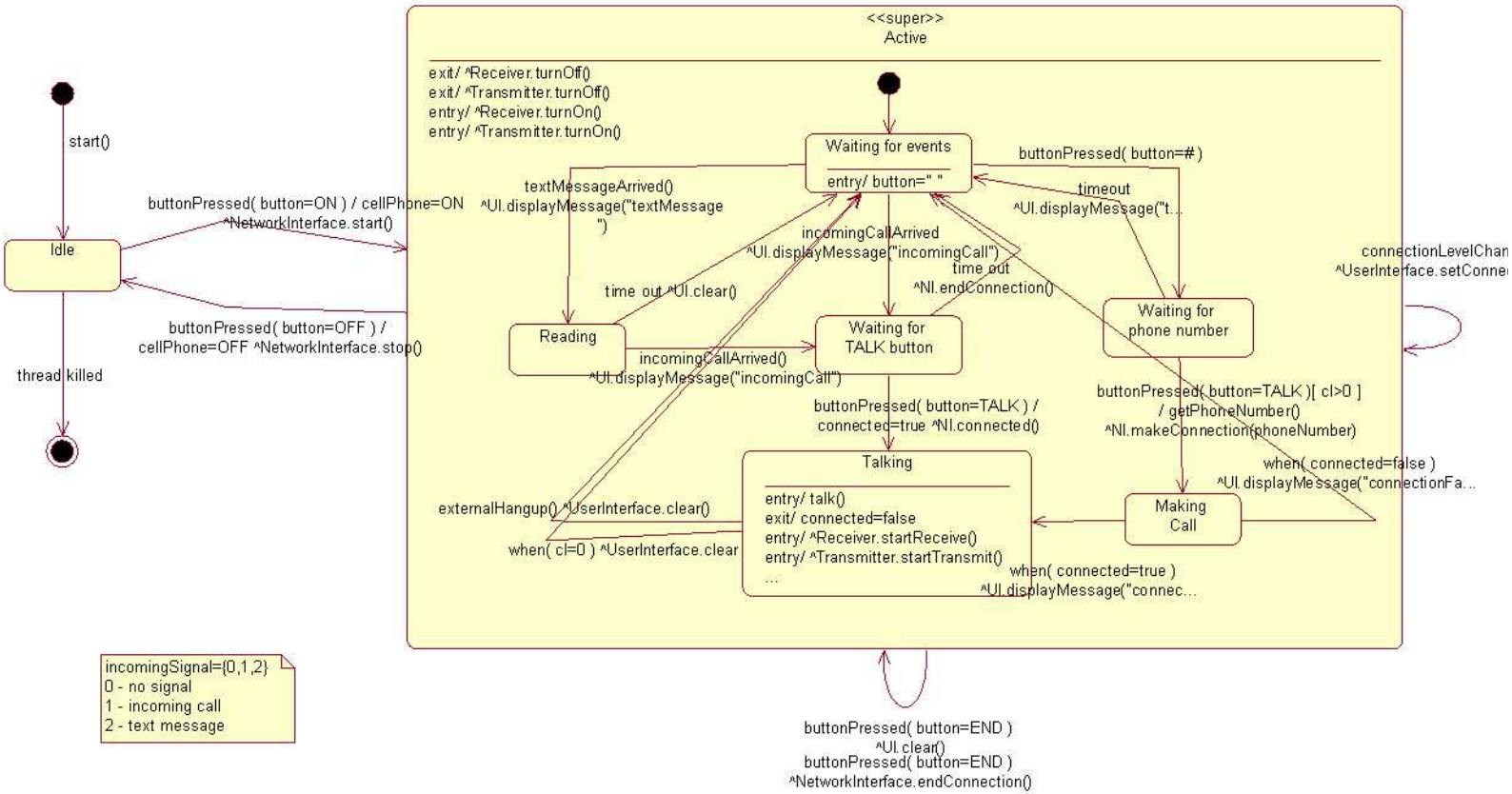

Figure 7: Handset Controller State Chart

5. Run each set of test cases from each diagram type on the implementation, and record faults found by their types (first author).

The TogetherControl Center software (*www.togethersoft.com/*) was used to specify the cell phone system, and the JBuilder tool (*www.borland.com/jbuilder/*) was used to implement it in Java.

4 Experimental Results and Analysis

The numbers of faults found during this experiment are given in Table 2. The rows represent the two types of faults, and the columns represent the number and percentage of faults found by the two groups of tests.

Fault Levels	**Number Faults**	**Faults Found**	
		Statechart Tests	**Sequence Tests**
Unit	31	**77%** (24)	65% (20)
Integration	18	56% (10)	**83%** (15)

Table 2: **Experimental Results**

We can see from the data in the table that the statechart tests revealed 12% more unit level faults than the sequence diagram tests, and the sequence diagram tests revealed 27% more integration level faults than the statechart tests. Also, there are 88% more statechart tests (81) than sequence diagram tests (43). Hence, both null hypotheses are rejected for these data.

An analysis of the faults revealed some insights into the techniques. The statechart tests found all of the 20 unit faults that the sequence diagram tests found, plus four more. Likewise, the sequence diagram tests found all of the 10 integration faults that the statechart tests found, plus five more. While it is tempting to conclude that sequence diagram tests are redundant for unit testing and vice versa, there is not enough data to make that general conclusion. The four unit faults missed by the sequence diagram tests were all related to functionality that was not used in the integrated cell phone system; so integration tests could not find them. Similarly, the five integration faults missed by the statechart tests were all related to functionality that could only be used when two classes were integrated together; so unit tests could not find them.

The primary threat to the validity of the experimental data is external. This is only one application and one set of tests. Repetition of these results are needed in order to generalize the results.

There were also several lessons learned during this experiment. One thing that became apparent is that UML statecharts are not always sufficient for specifying low level details, particularly when great precision is required. The Object Constraint Language [16] can play a supplementary role for this purpose, and the integration of the OCL into the UML will provide better information for testing.

Another problem encountered during this experiment was with concurrency. The expected execution trace that was developed from the sequence diagram sometimes turned out to be different from the actual execution trace because of concurrency interactions. This could be a potential problem in automating the testing process, and we probably need to incorporate some concurrent testing approaches [14, 15].

5 Conclusions and Future Work

This paper has presented a single project experiment on the fault revealing capabilities of test sets that are generated from UML statecharts and sequence diagrams. In this experiment, the statechart tests found more unit faults than the sequence diagram tests, and the sequence diagram tests found more integration faults than the statechart tests. There are also almost twice as many statechart tests as sequence diagram tests.

Although the fact that we used only one project limits the general conclusions that can be drawn from this study, some preliminary conclusions can safely be made. First, it is reasonably effective to use UML diagrams as either a source for generating tests or as a means for evaluating tests generated elsewhere. Second, the data matches intuition, specifically that statecharts should be used for unit level testing and

sequence diagrams should be used for integration level testing. This is evidence not only of the proper use of the UML diagrams for testing, but also that both unit and integration testing should be done.

The fact that more tests were generated to satisfy the statechart criterion (full predicates) than the sequence diagram criterion (message sequence paths) should not be surprising. Designers will usually include more detail and more decision points in statecharts than they will message sequence paths. This difference **cannot be quantified**, however, because the numbers can vary by developer and project.

This is but one step in an ongoing research project. In the future, we plan to apply similar experiments to other projects to evaluate the consistency of these results. We also plan to integrate UML specifications and OCL to generate stronger tests, and to develop new testing criteria to test concurrency aspects.

References

[1] Aynur Abdurazik and Jeff Offutt. Using UML collaboration diagrams for static checking and test generation. In *Proceedings of the Third International Conference on the Unified Modeling Language (UML '00)*, pages 383–395, York, England, October 2000.

[2] Paul Ammann, Jeff Offutt, and Hong Huang. Coverage criteria for logical expressions. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, pages 99–107, Denver, CO, November 2003. IEEE Computer Society Press.

[3] Victor R. Basili, Richard W. Selby, and David H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, SE-12(7):733–743, July 1986.

[4] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, Inc, New York NY, 2nd edition, 1990. ISBN 0-442-20672-0.

[5] Philippe Chevalley and Pascale Thévenod-Fosse. Automated generation of statistical test cases from UML state diagrams. In *Proc. of IEEE 25th Annual International Computer Software and Applications Conference (COMPSAC2001)*, Chicago IL, October 2001.

[6] J. J. Chilenski and S. P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, September 1994.

[7] John Chilenski and L. A. Richey. Definition for a masking form of modified condition decision coverage (MCDC). Technical report, Boeing, Seattle, WA, 1997. http://www.boeing.com/nosearch/mcdc/.

[8] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, April 1978.

[9] R. A. DeMillo and A. J. Offutt. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, 17(9):900–910, September 1991.

[10] Y. G. Kim, H. S. Hong, S. M. Cho, D. H. Bae, and S. D. Cha. Test cases generation from UML state diagrams. *IEE Proceedings – Software*, 146(4):187–192, August 1999.

[11] Object Management Group. *OMG UML Specification Version 1.3*, June 1999. Available at http://www.omg.org/uml/.

[12] Jeff Offutt and Aynur Abdurazik. Generating tests from UML specifications. In *Proceedings of the Second International Conference on the Unified Modeling Language (UML '99)*, pages 416–429, Fort Collins, CO, October 1999.

[13] Jeff Offutt and Shaoying Liu. Generating test data from SOFL specifications. *The Journal of Systems and Software*, 49(1):49–62, December 1999.

[14] K. C. Tai, R. H. Carver, and E. E. Obaid. Debugging concurrent Ada programs by deterministic execution. *IEEE Transactions on Software Engineering*, 17(1):45–63, January 1991.

[15] R. N. Taylor and L. J. Osterweil. Anomaly detection in concurrent software by static data flow analysis. *IEEE Transactions on Software Engineering*, 6(3):265–277, May 1980.

[16] Jos Warmer and Anneke Kleppe. *The Object Constraint Language.* Addison-Wesley, 1999. ISBN 0-201-37940-6.

[17] H. Yoon and B. Choi. Effective testing technique for the component customization in EJB. In *Proceedings of 8th Asia-Pacific Software Engineering Conference (APSEC 2001)*, Macau SAR, China, December 2001.

A Appendix: Sequence Diagrams With Alternatives

This appendix contains all the sequence diagrams with their various alternatives. The order is the Initialization Use case, Answer Call, Make Call, Notify Incoming Call, Notify Text Message, and finally Turn Off Cell Phone.

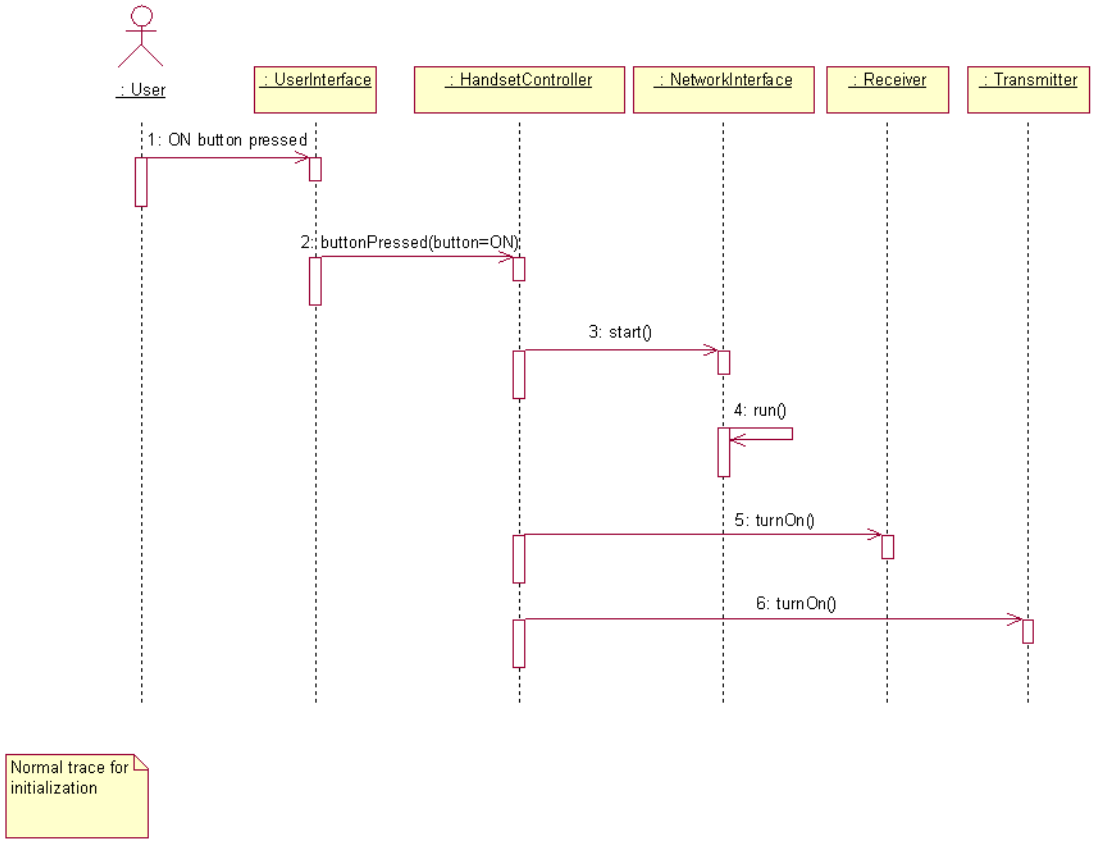

Figure 8: Sequence Diagram for Initialization Use Case

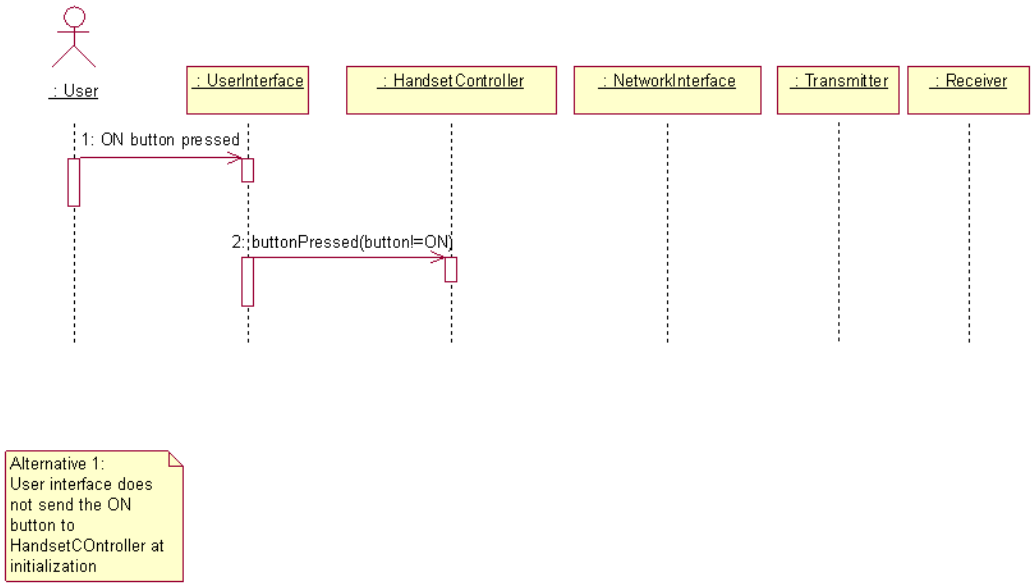

Figure 9: Sequence Diagram for Initialization Use Case: Alternative 1

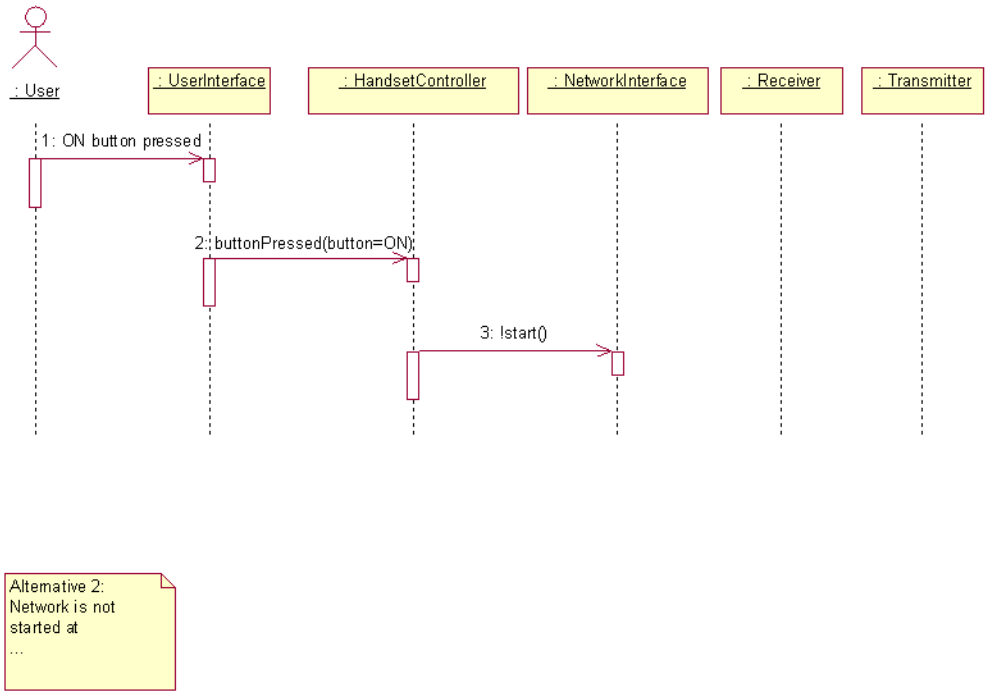

Figure 10: Sequence Diagram for Initialization Use Case: Alternative 2

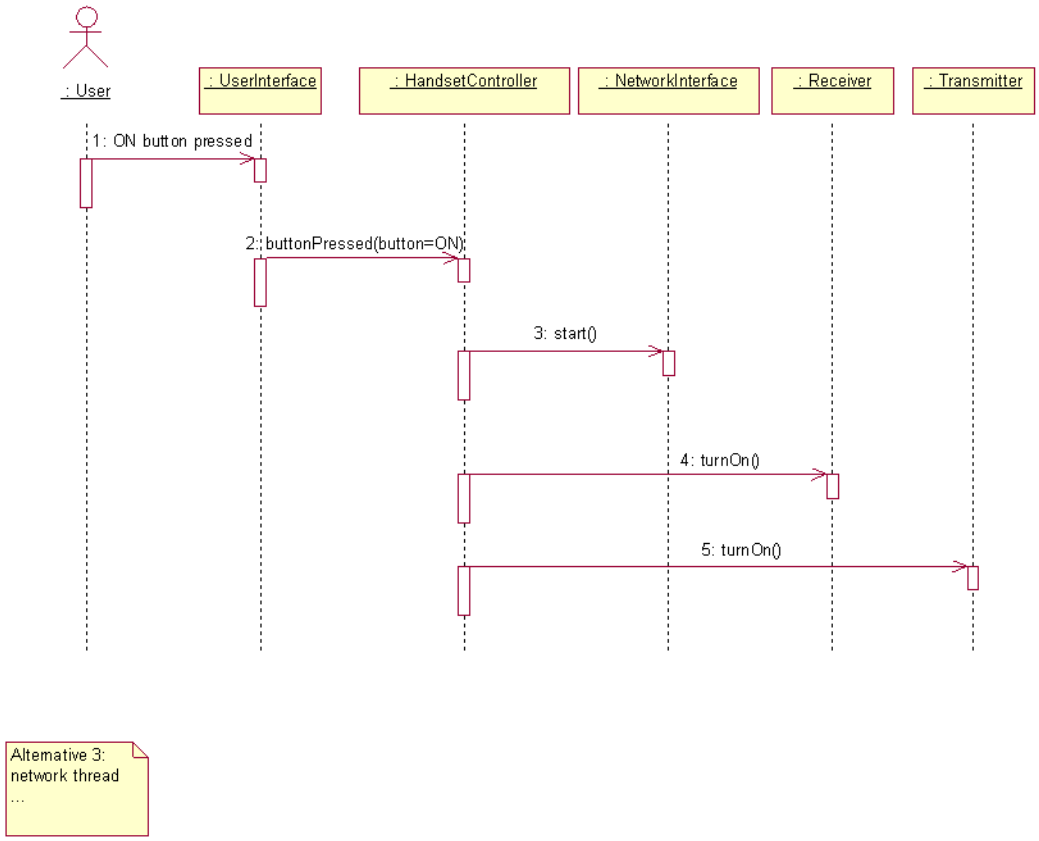

Figure 11: Sequence Diagram for Initialization Use Case: Alternative 3

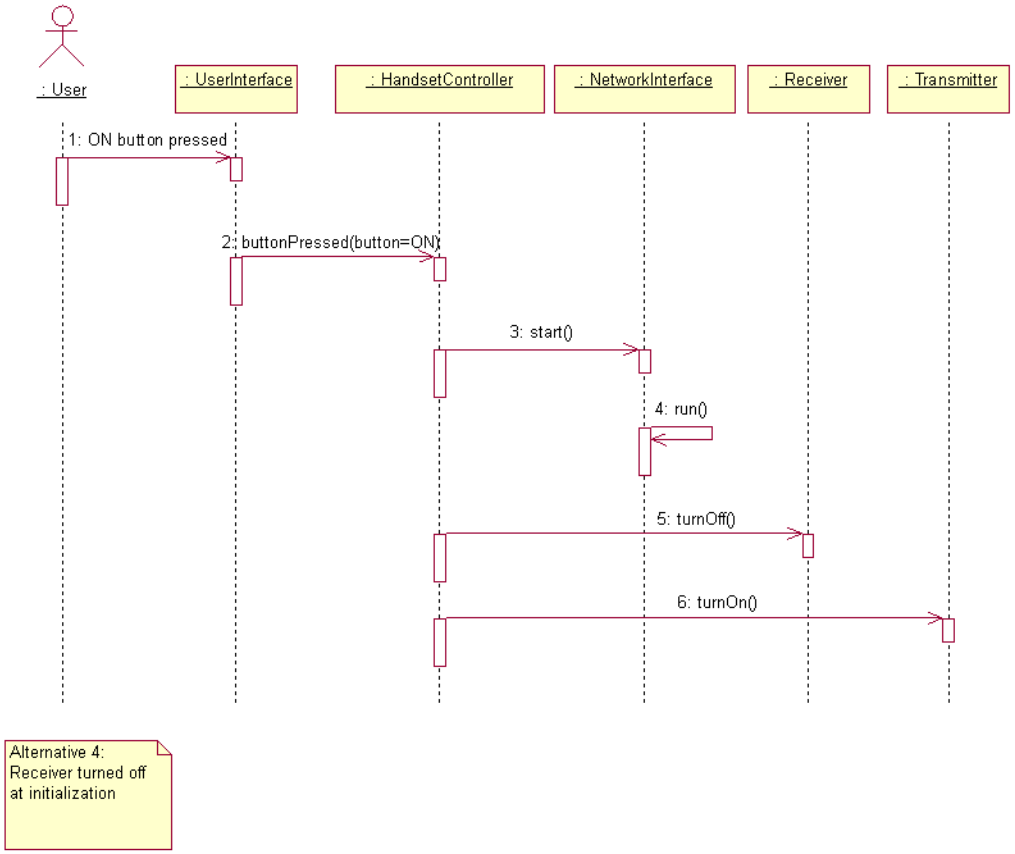

Figure 12: Sequence Diagram for Initialization Use Case: Alternative 4

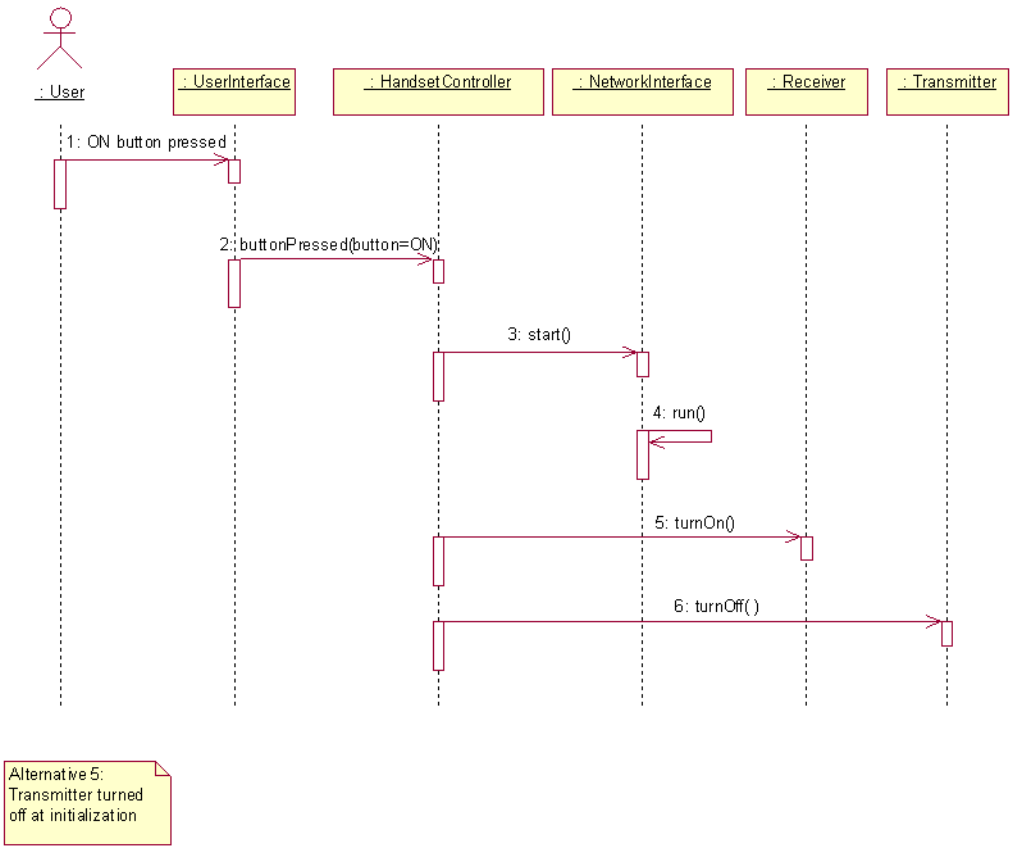

Figure 13: Sequence Diagram for Initialization Use Case: Alternative 5

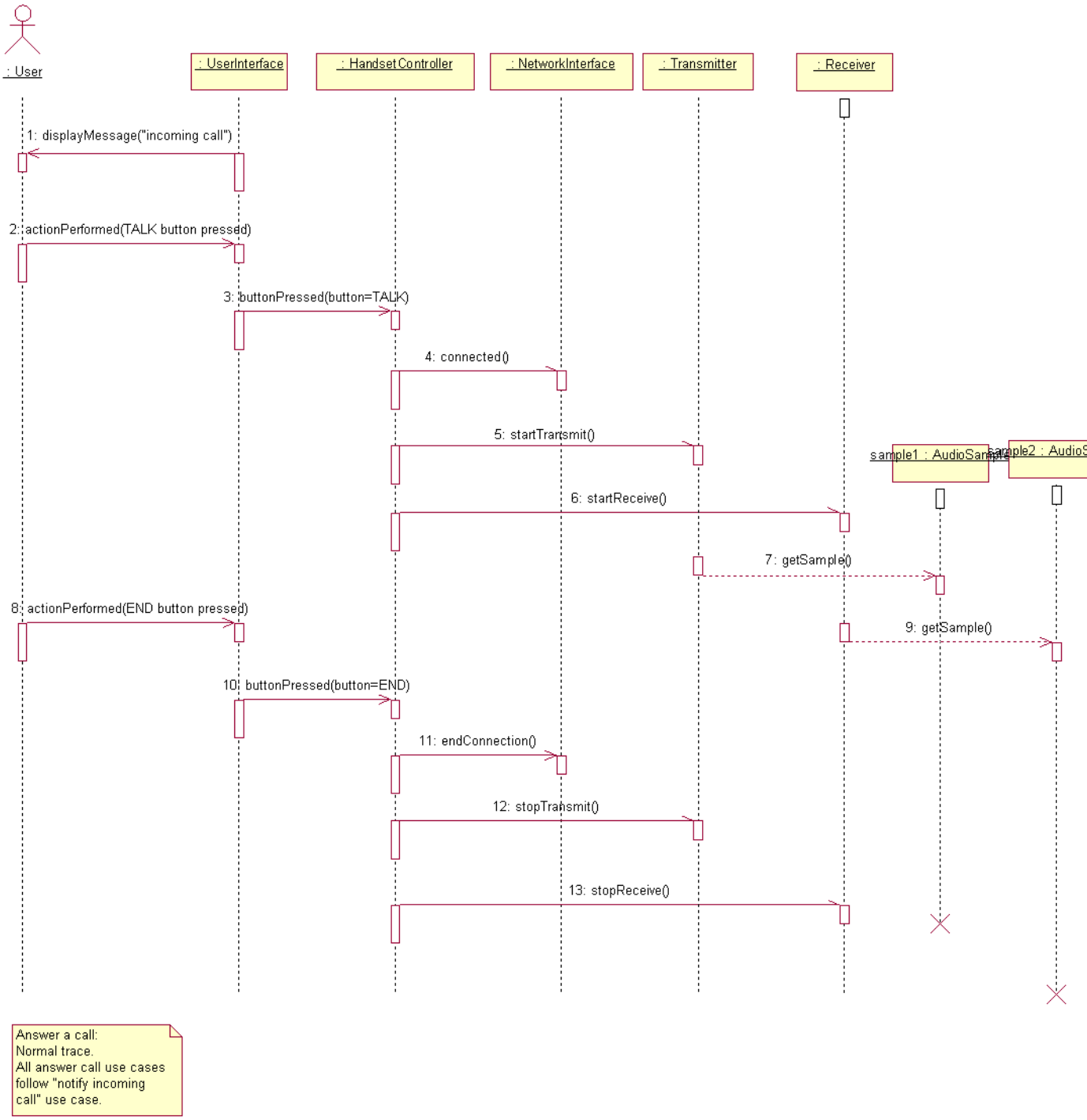

Figure 14: Sequence Diagram for Answer Call Use Case

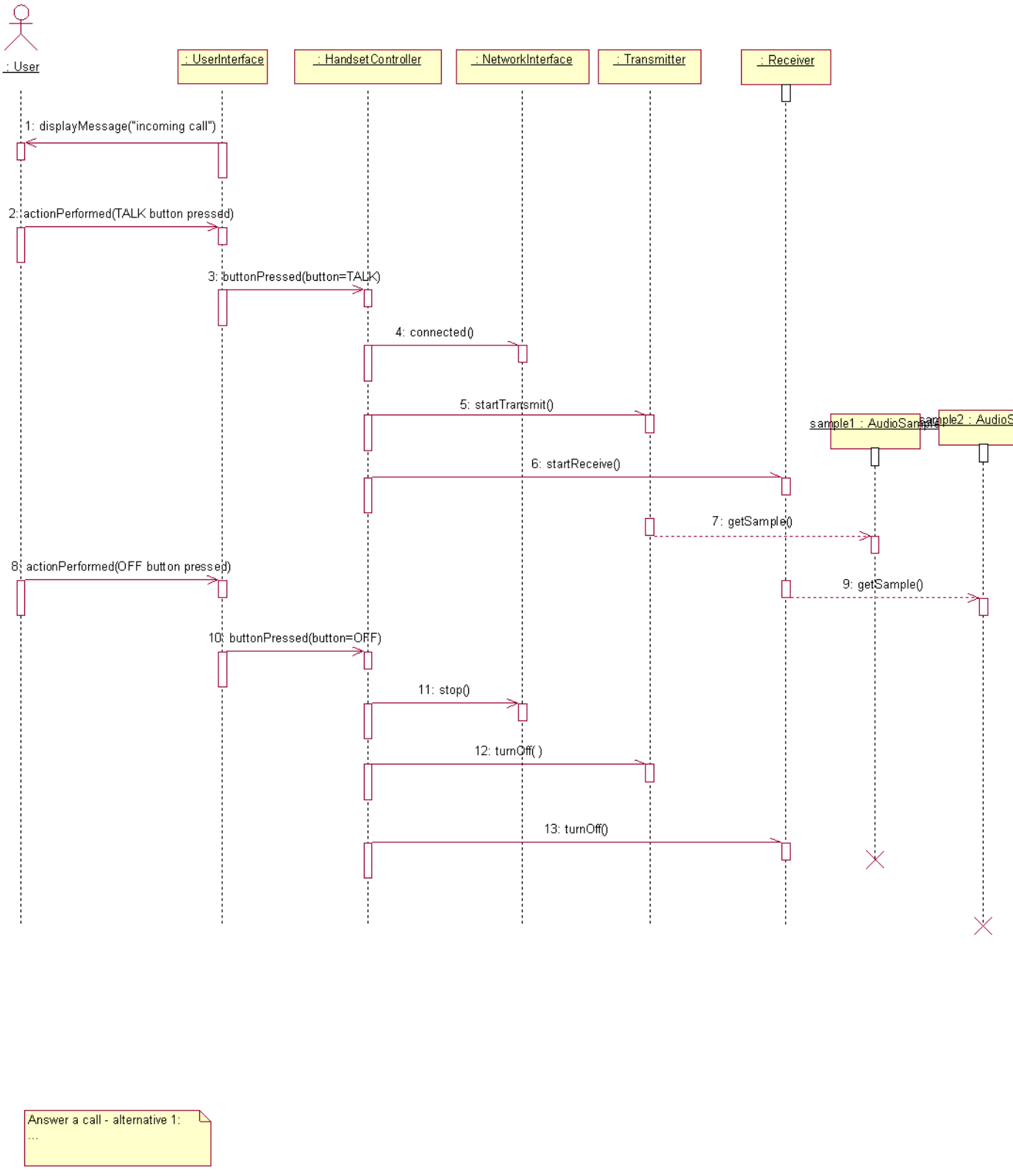

Figure 15: Sequence Diagram for Answer Call Use Case: Alternative 1

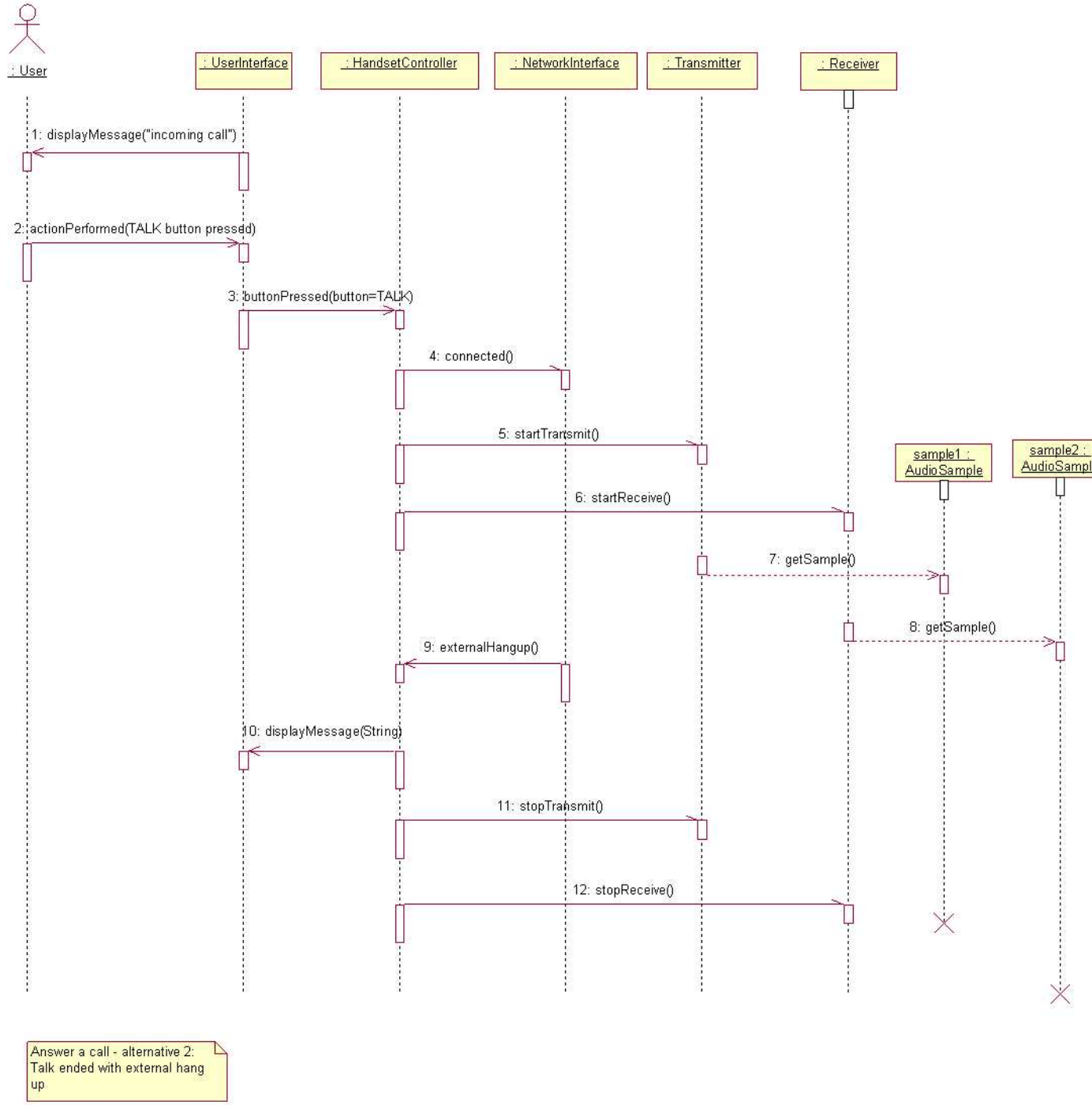

Figure 16: Sequence Diagram for Answer Call Use Case: Alternative 2

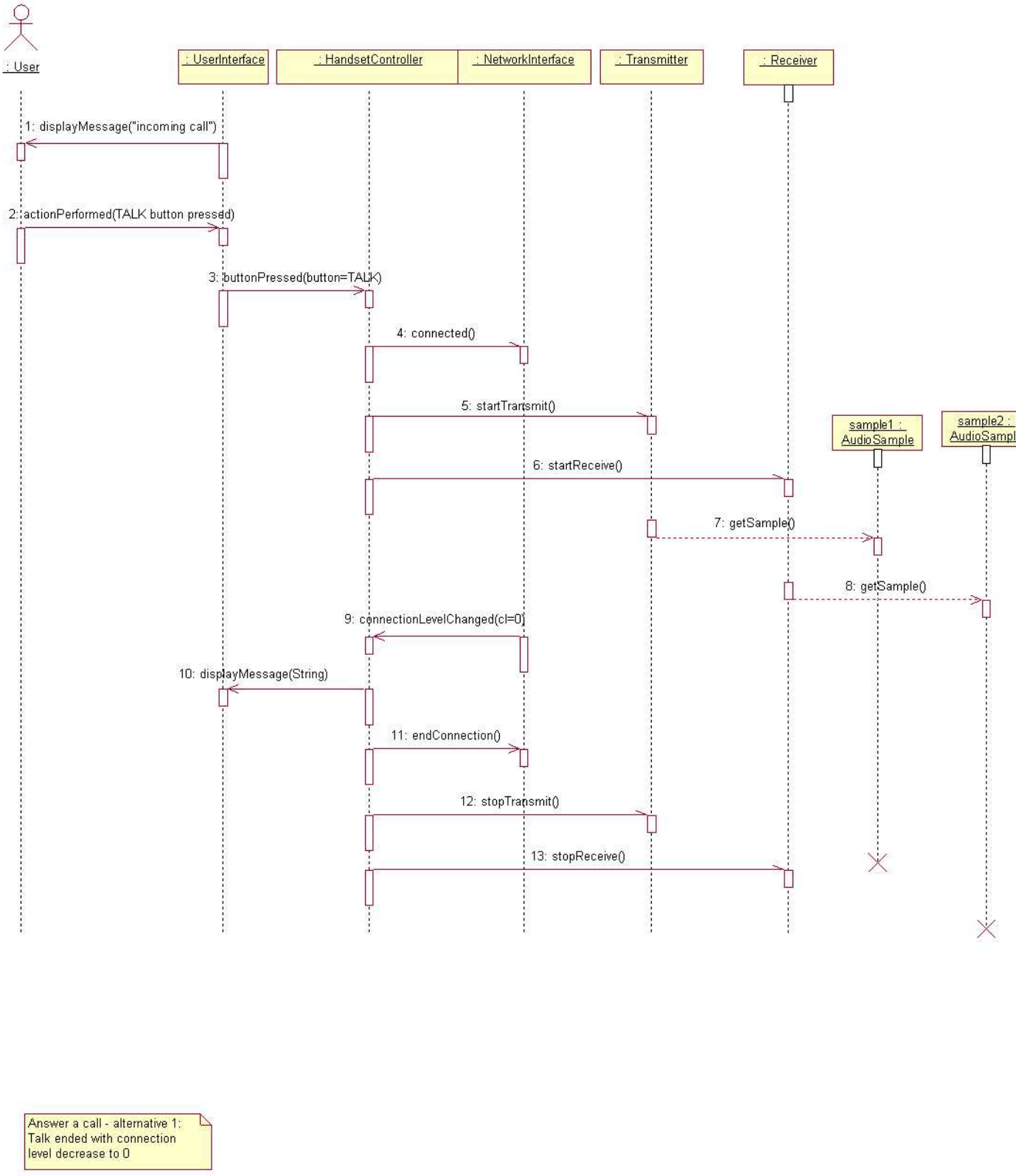

Figure 17: Sequence Diagram for Answer Call Use Case: Alternative 3

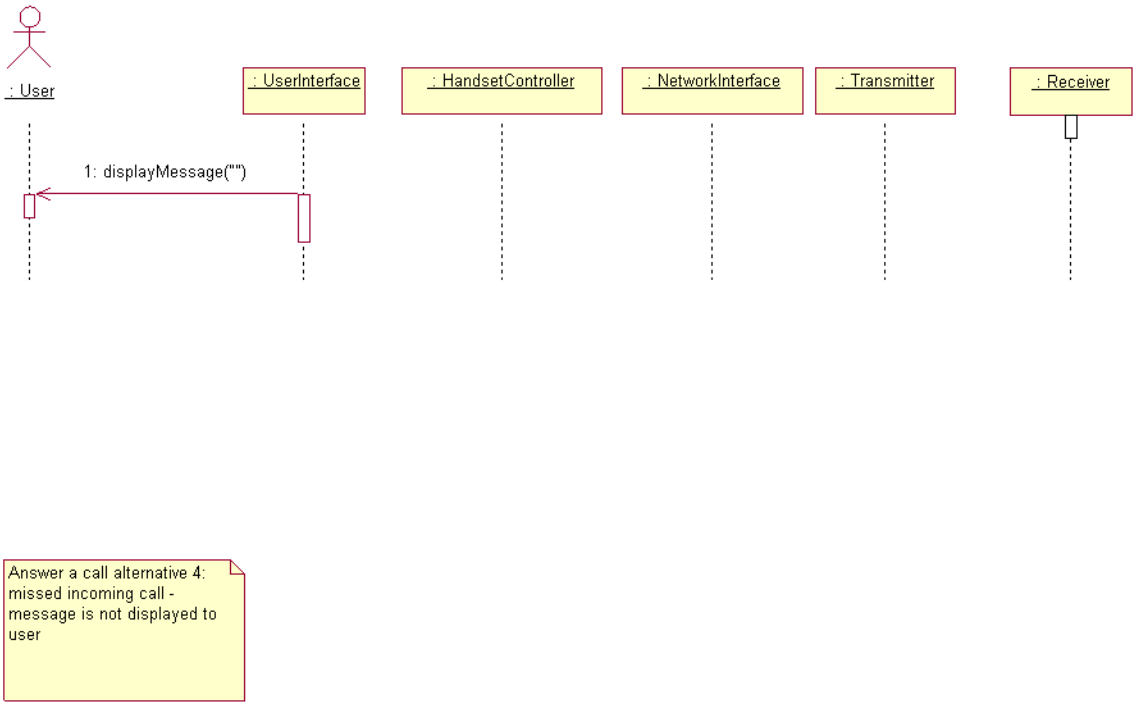

Figure 18: Sequence Diagram for Answer Call Use Case: Alternative 4

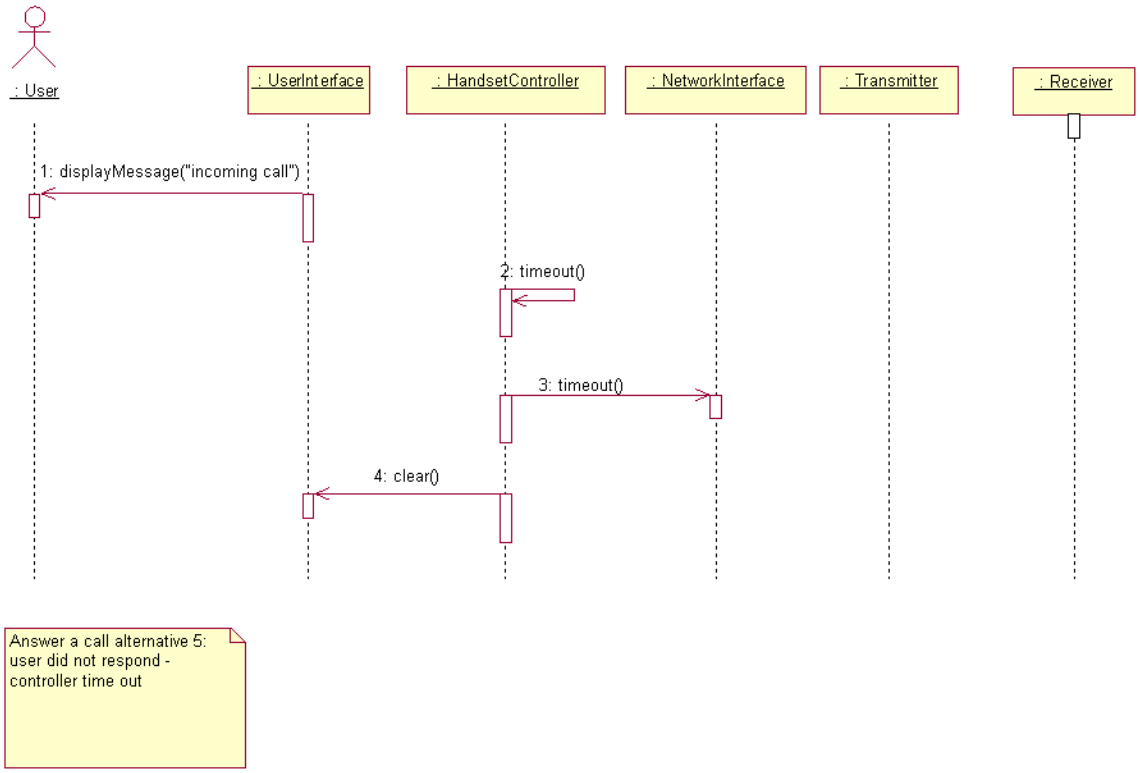

Figure 19: Sequence Diagram for Answer Call Use Case: Alternative 5

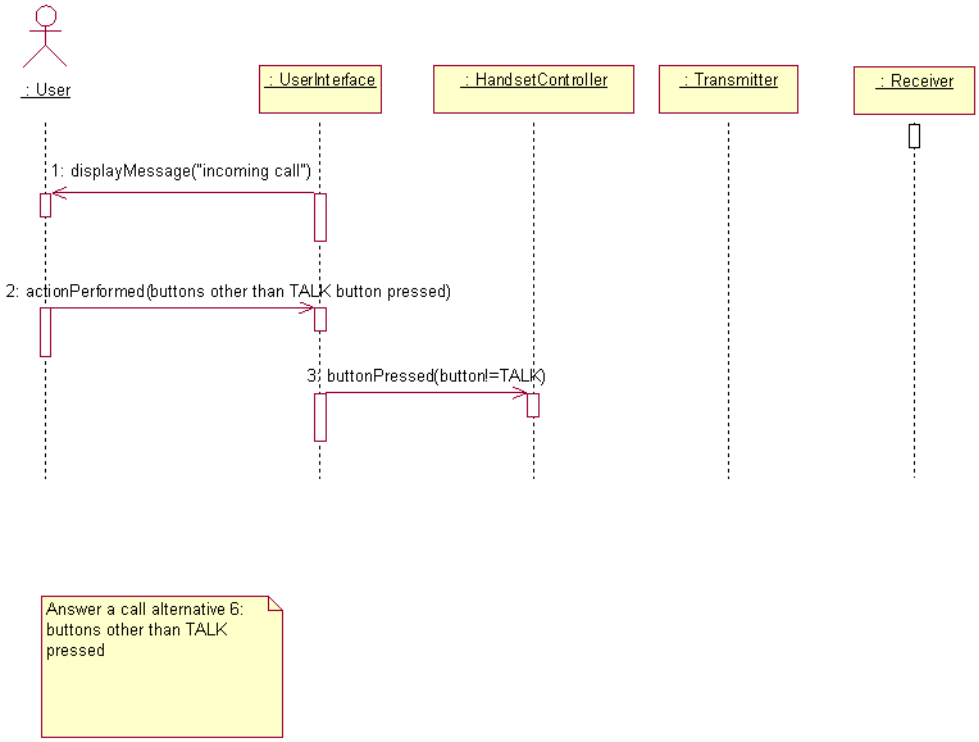

Figure 20: Sequence Diagram for Answer Call Use Case: Alternative 6

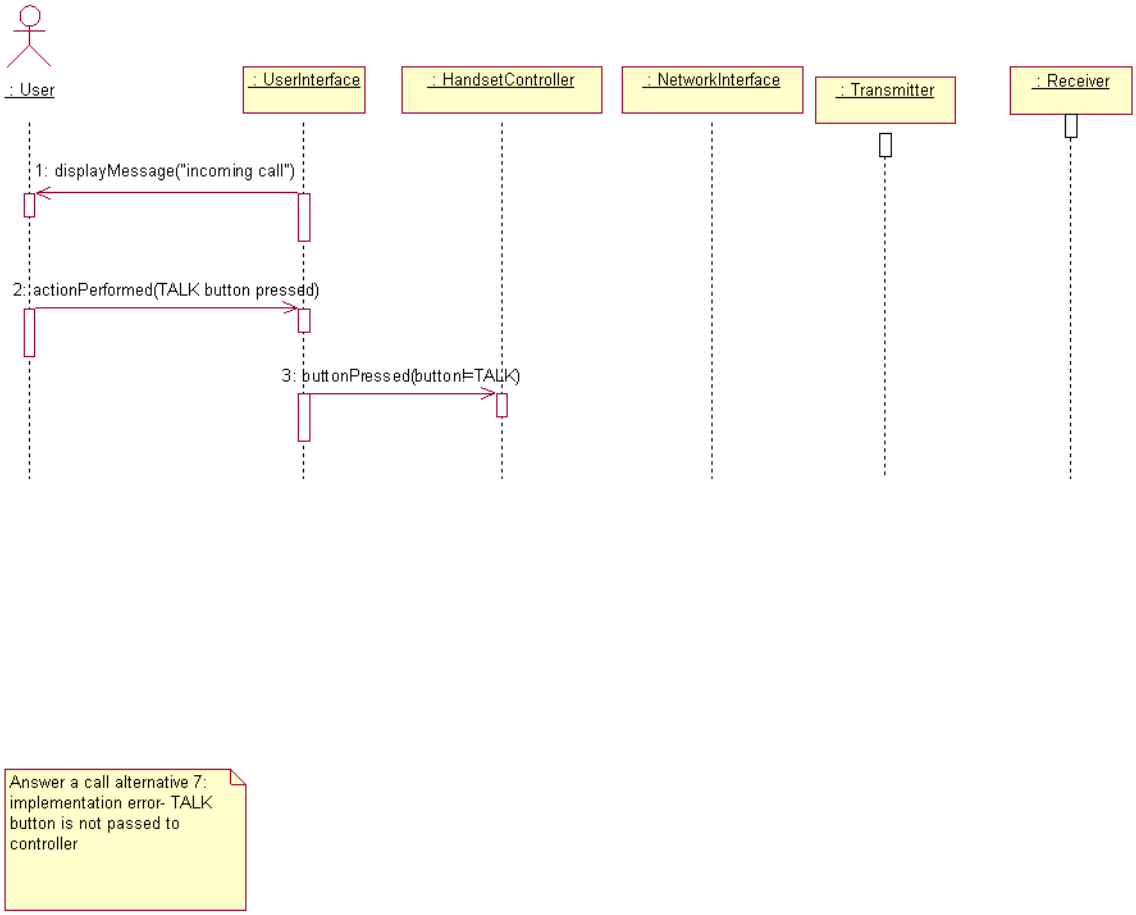

Figure 21: Sequence Diagram for Answer Call Use Case: Alternative 7

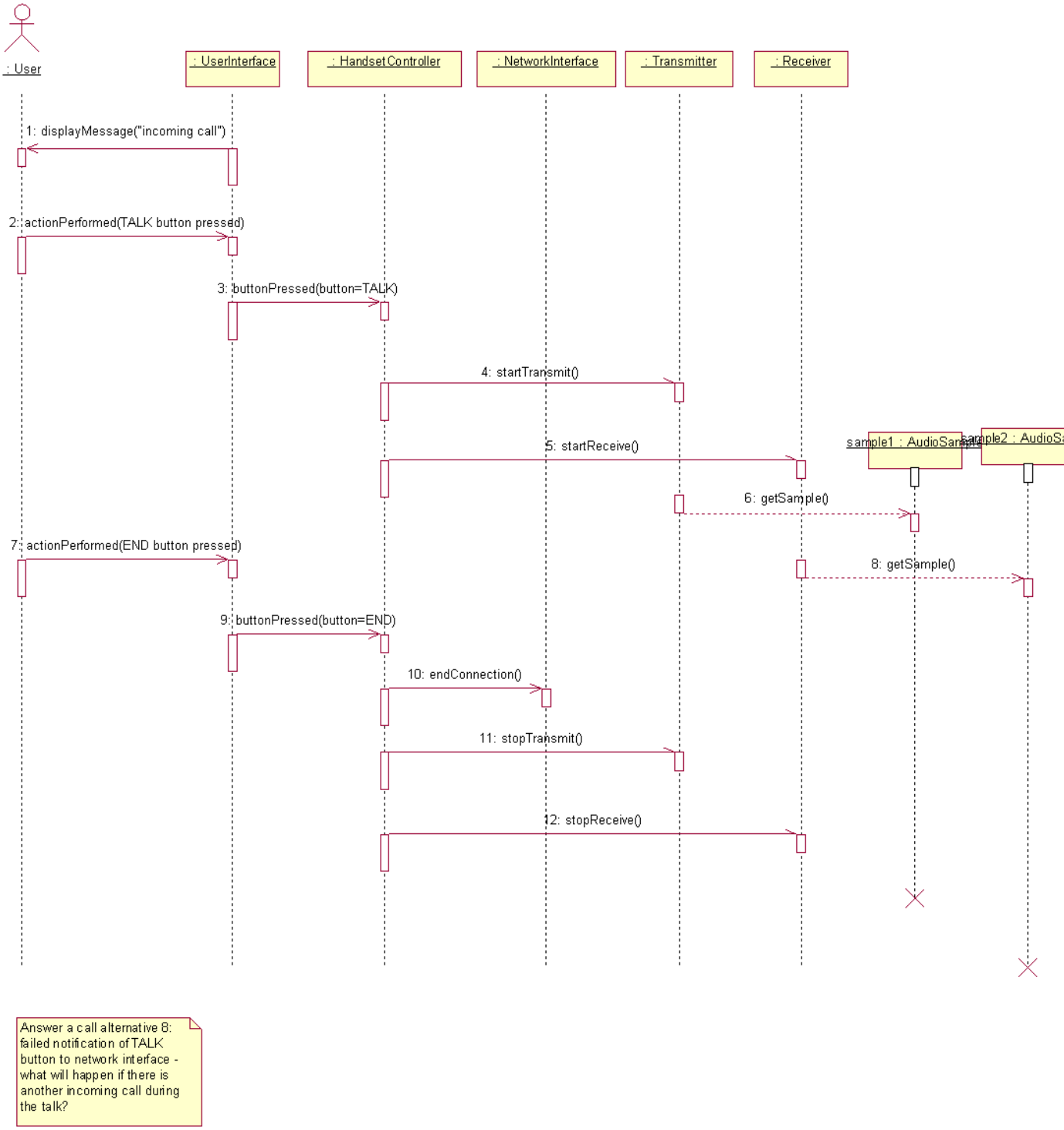

Figure 22: Sequence Diagram for Answer Call Use Case: Alternative 8

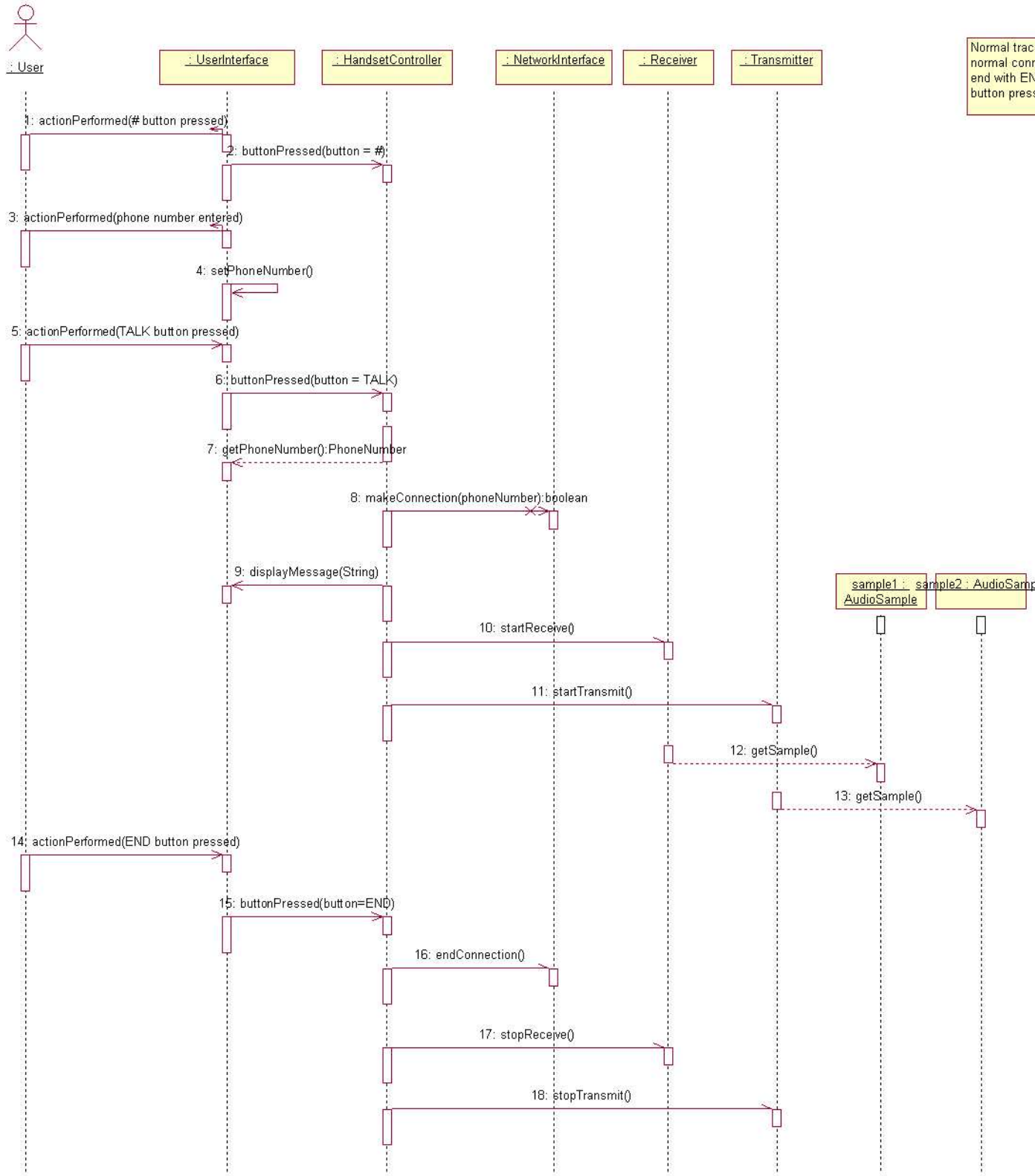

Figure 23: Sequence Diagram for Make a Call Use Case

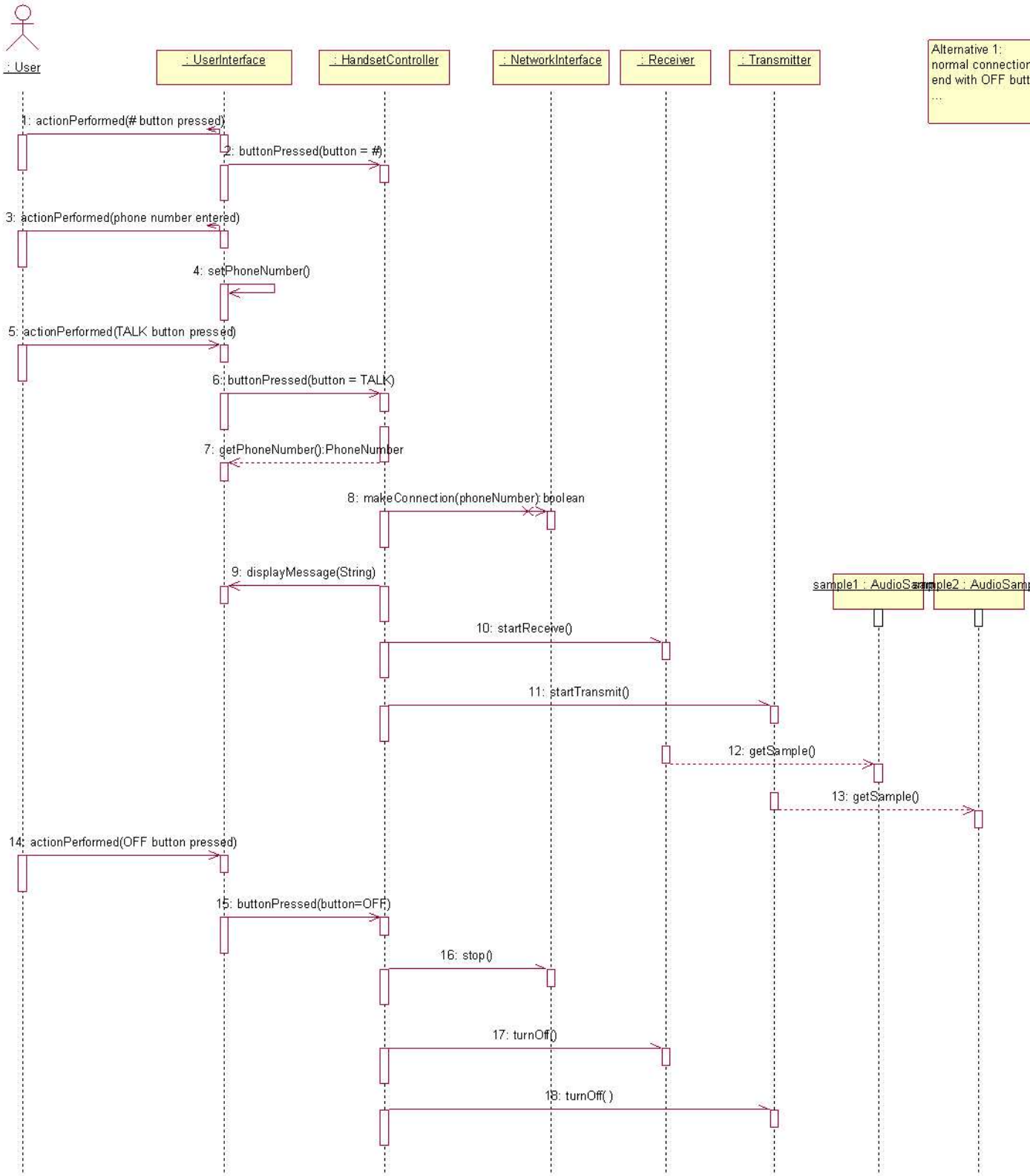

Figure 24: Sequence Diagram for Make a Call Use Case: Alternative 1

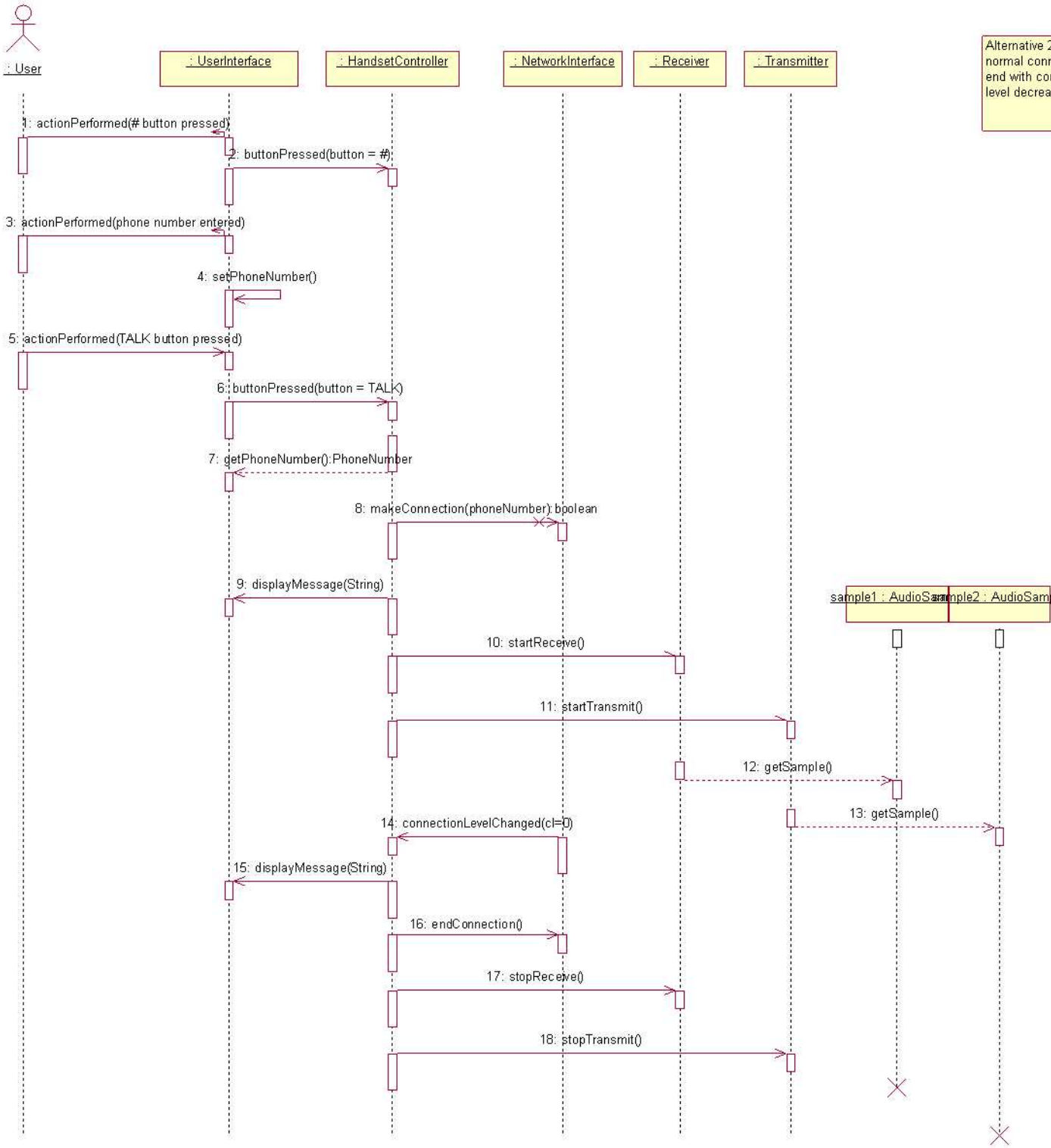

Figure 25: Sequence Diagram for Make a Call Use Case: Alternative 2

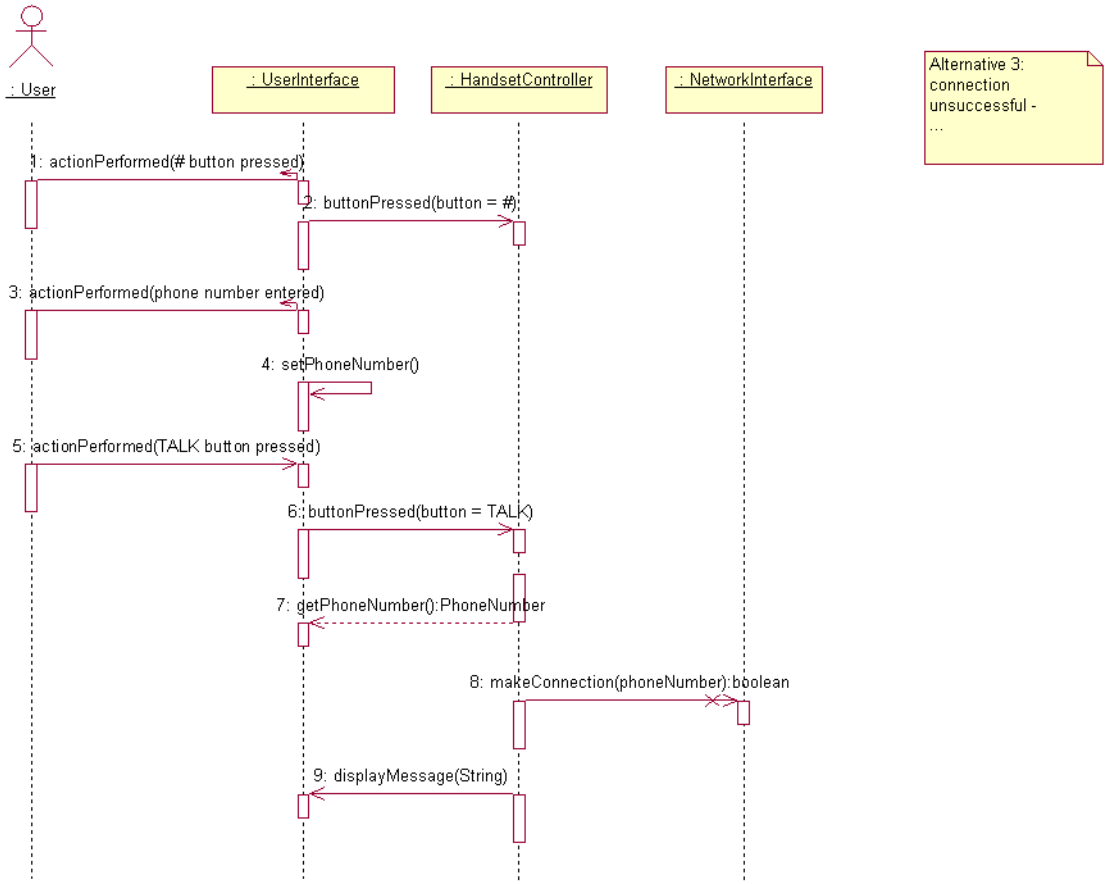

Figure 26: Sequence Diagram for Make a Call Use Case: Alternative 3

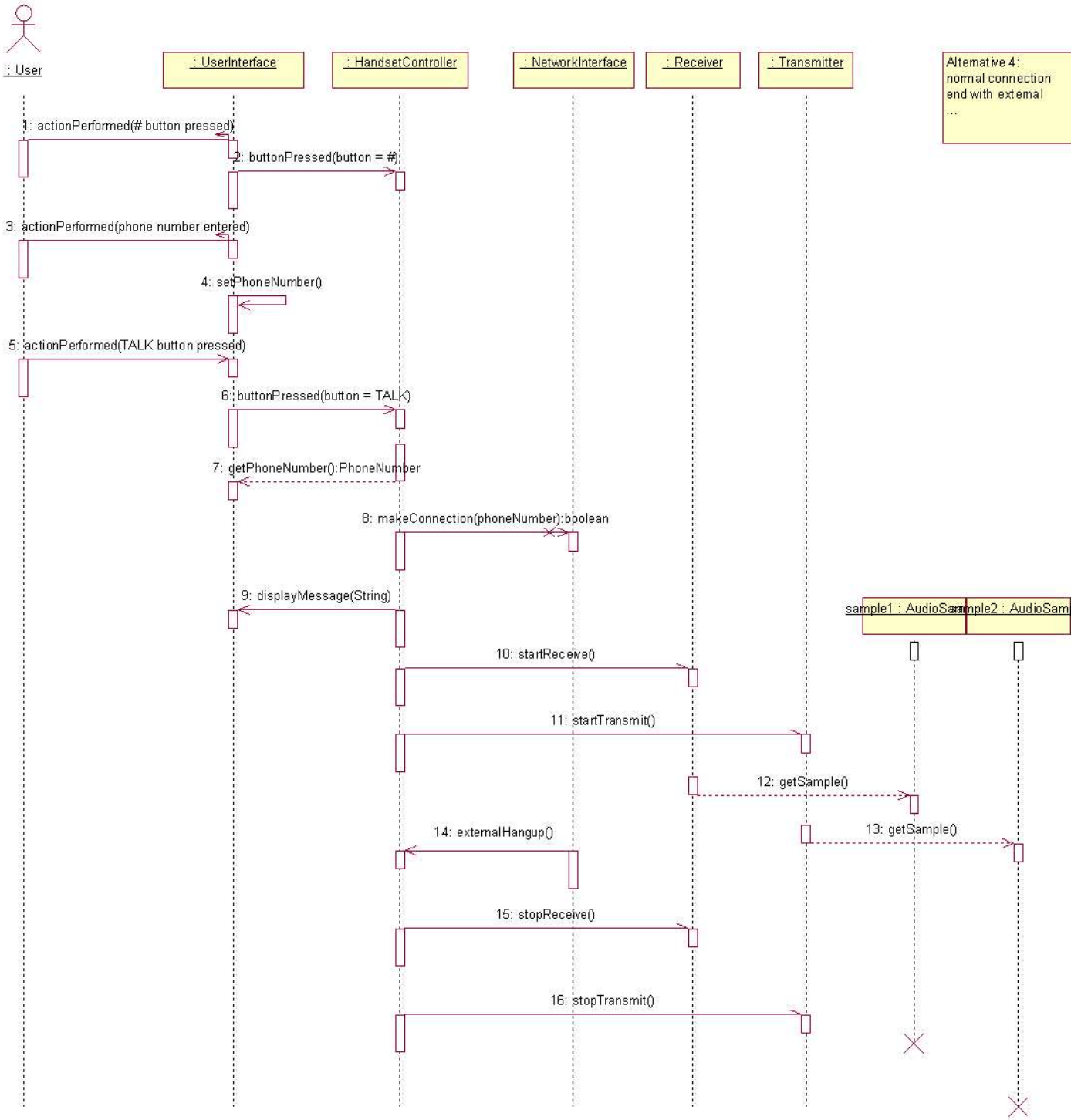

Figure 27: Sequence Diagram for Make a Call Use Case: Alternative 4

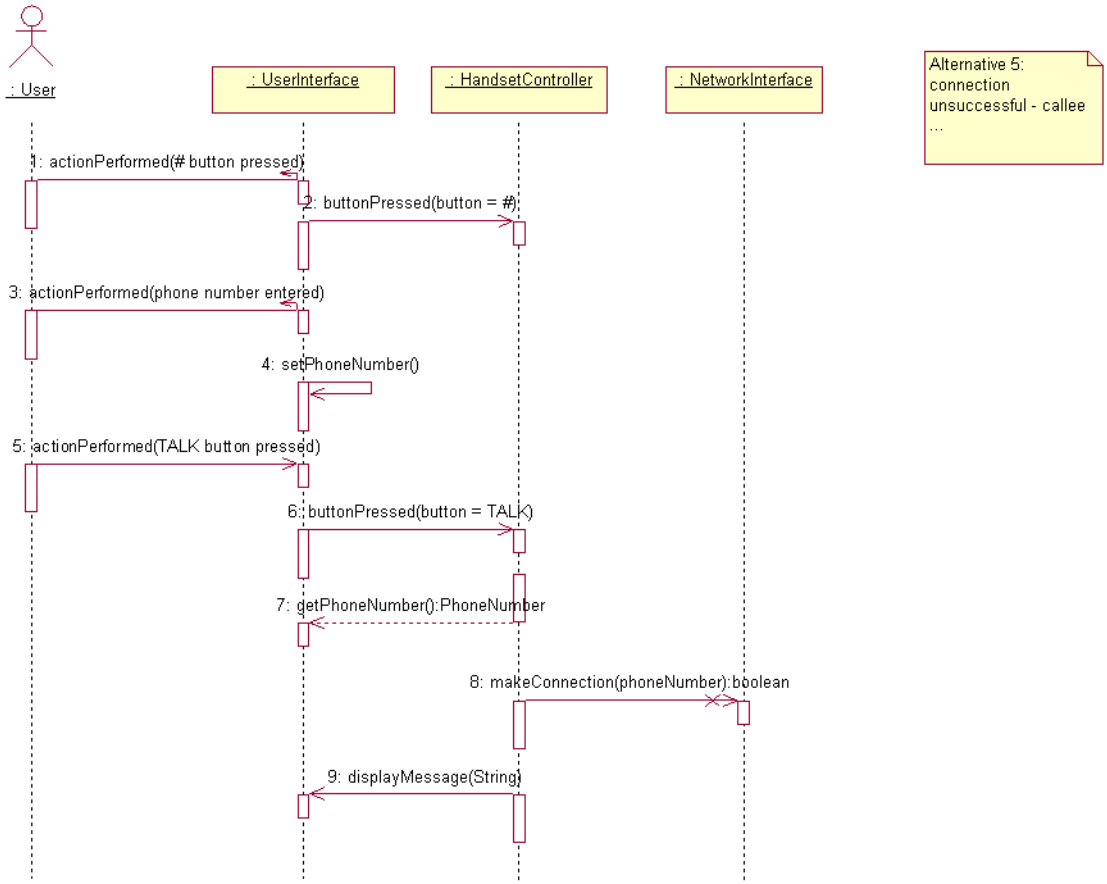

Figure 28: Sequence Diagram for Make a Call Use Case: Alternative 5

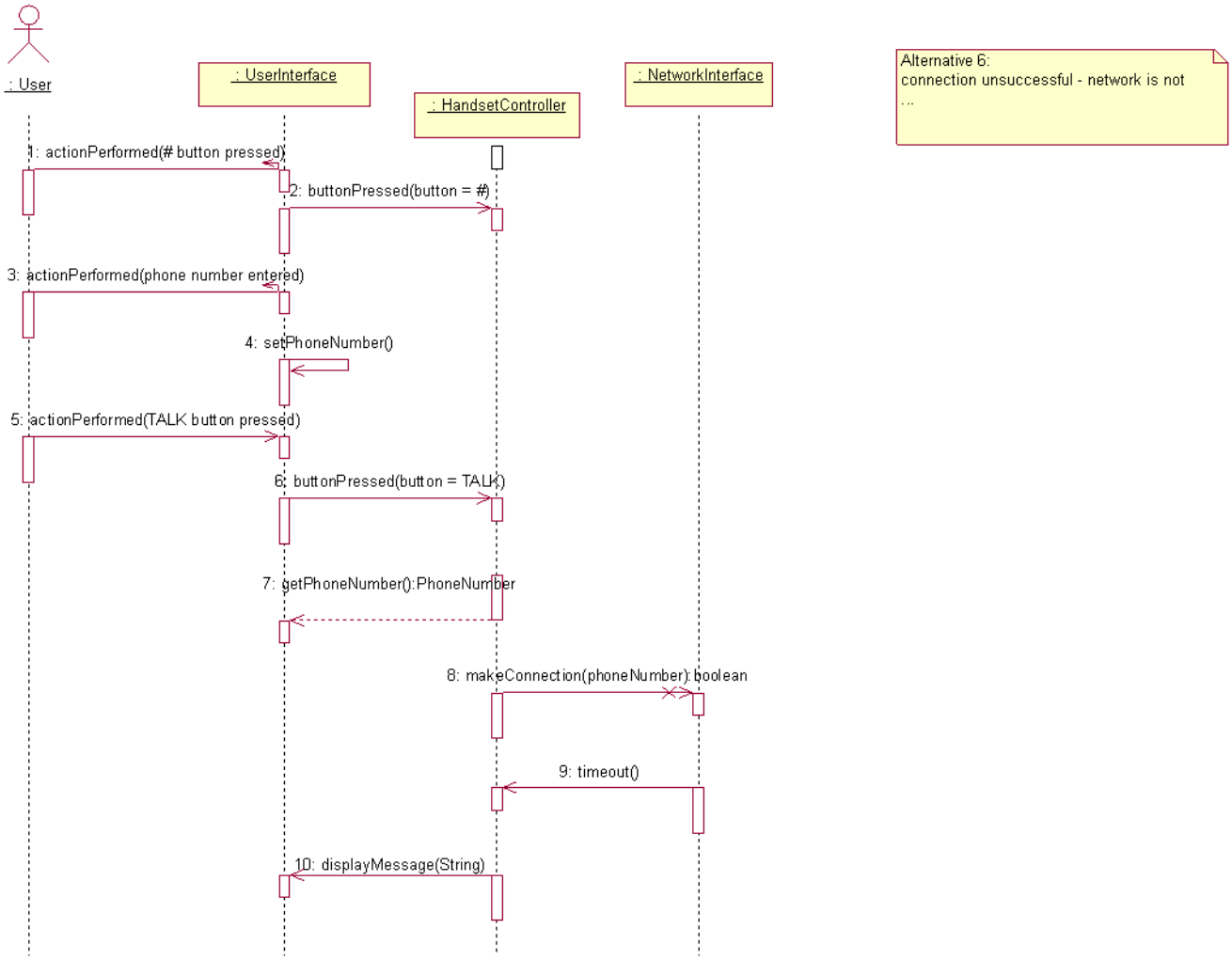

Figure 29: Sequence Diagram for Make a Call Use Case: Alternative 6

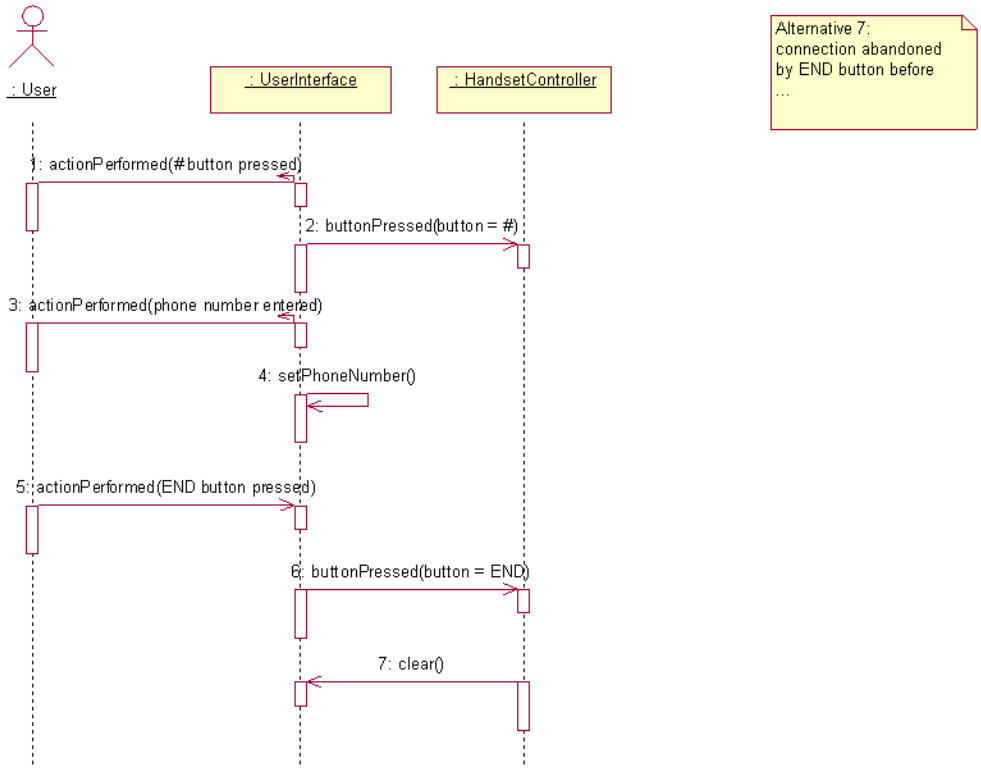

Figure 30: Sequence Diagram for Make a Call Use Case: Alternative 7

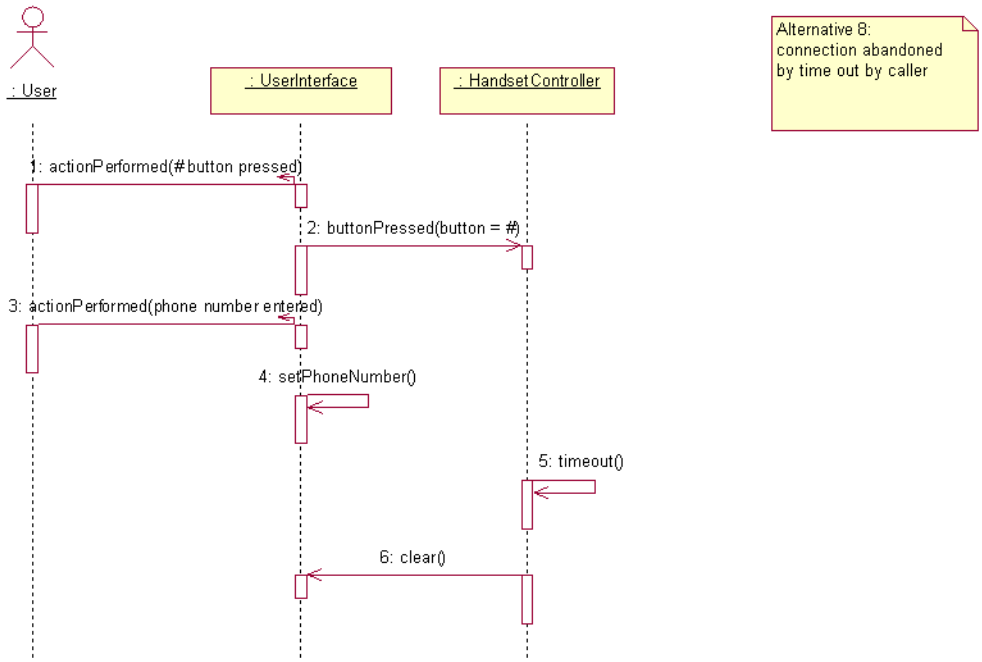

Figure 31: Sequence Diagram for Make a Call Use Case: Alternative 8

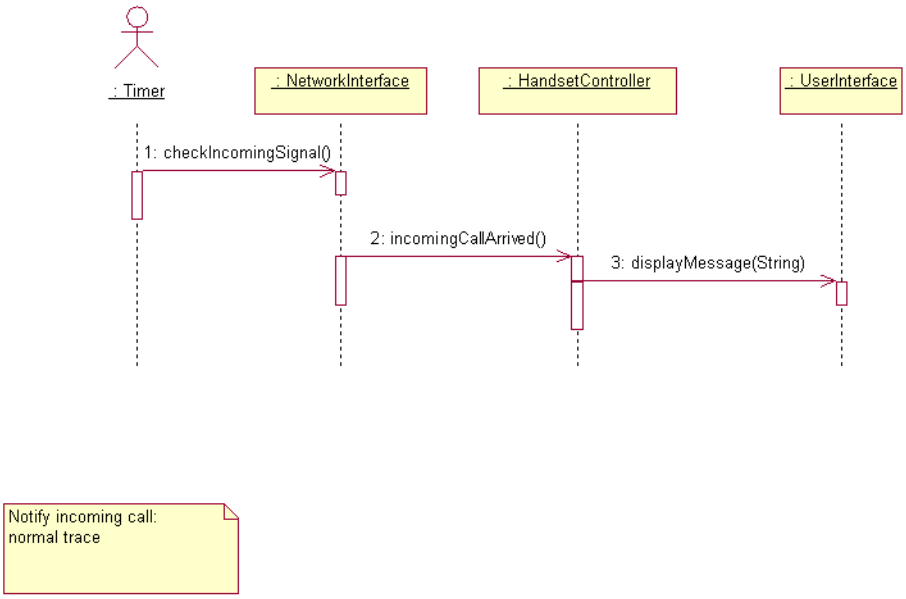

Figure 32: Sequence Diagram for Notify Incoming Call Use Case

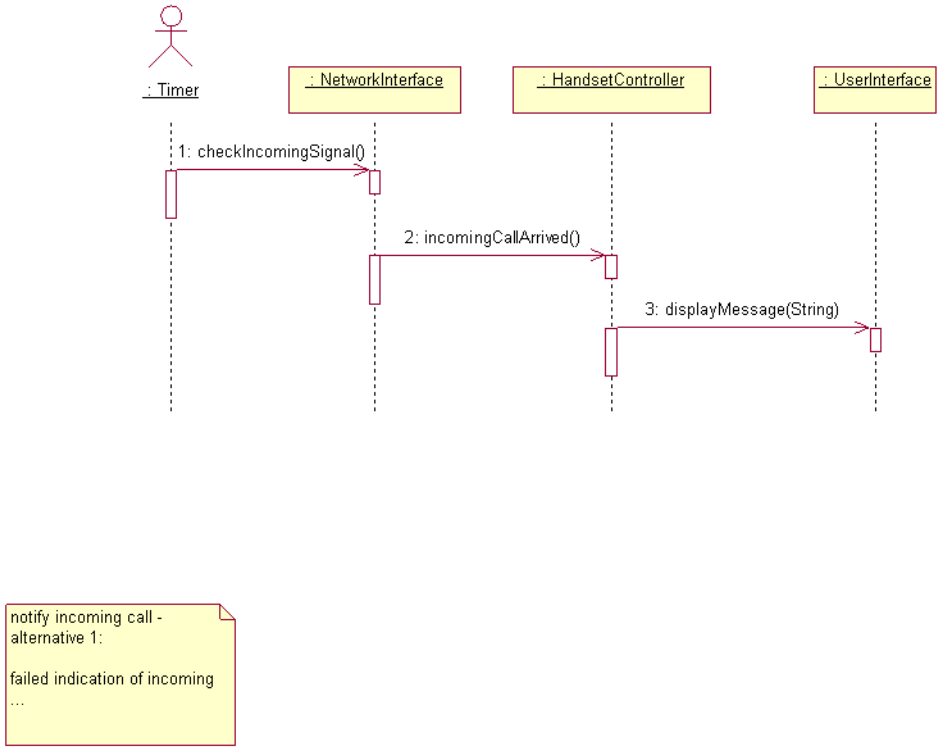

Figure 33: Sequence Diagram for Notify Incoming Call Use Case: Alternative 1

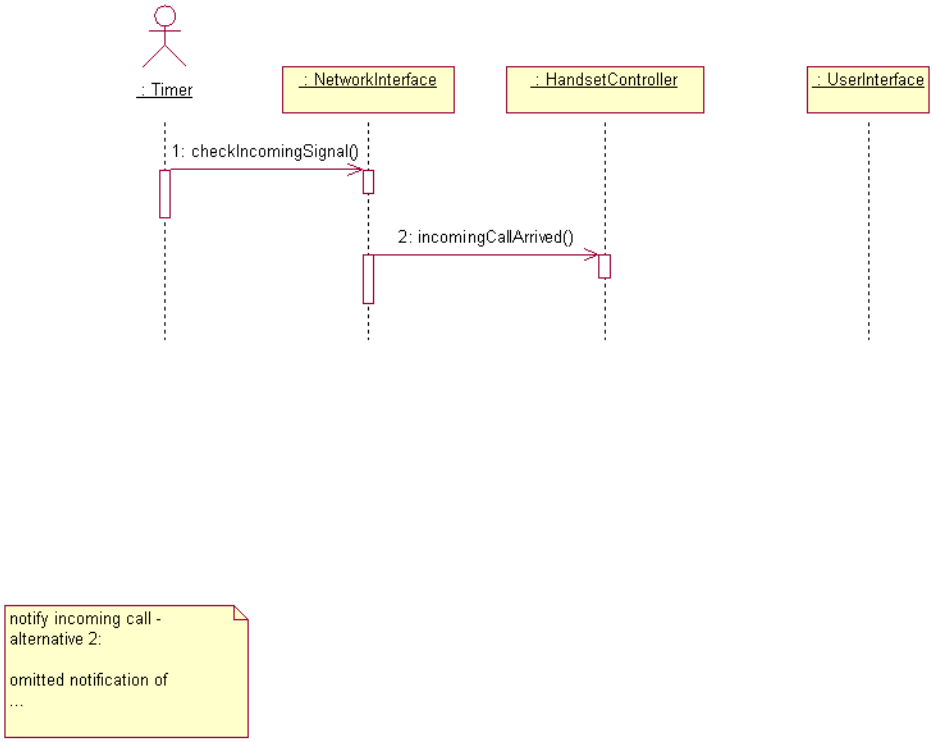

Figure 34: Sequence Diagram for Notify Incoming Call Use Case: Alternative 2

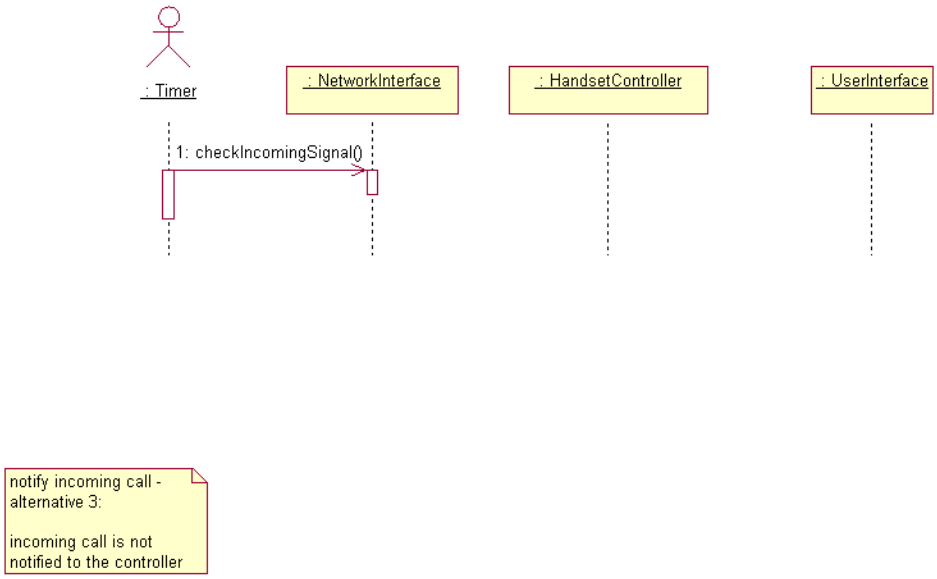

Figure 35: Sequence Diagram for Notify Incoming Call Use Case: Alternative 3

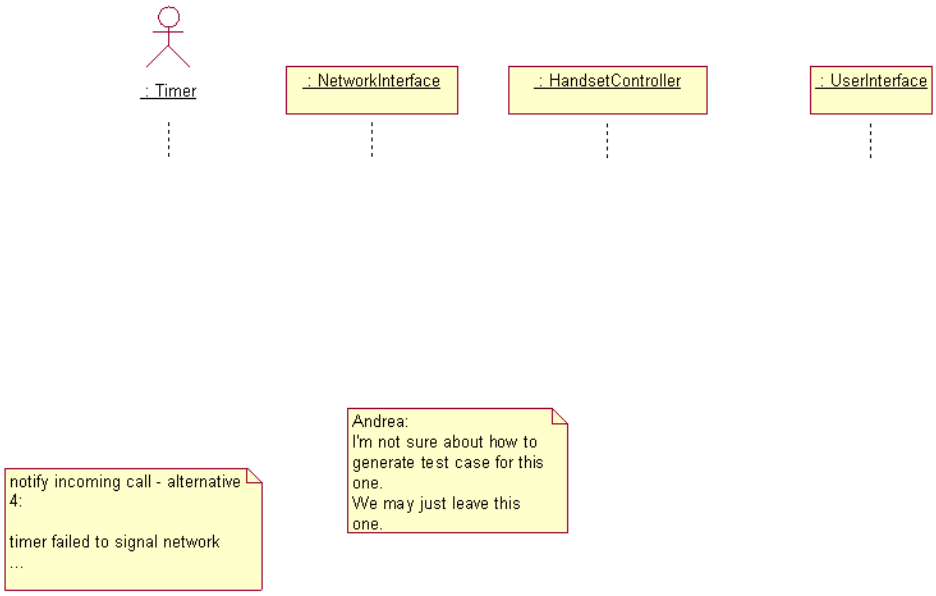

Figure 36: Sequence Diagram for Notify Incoming Call Use Case: Alternative 4

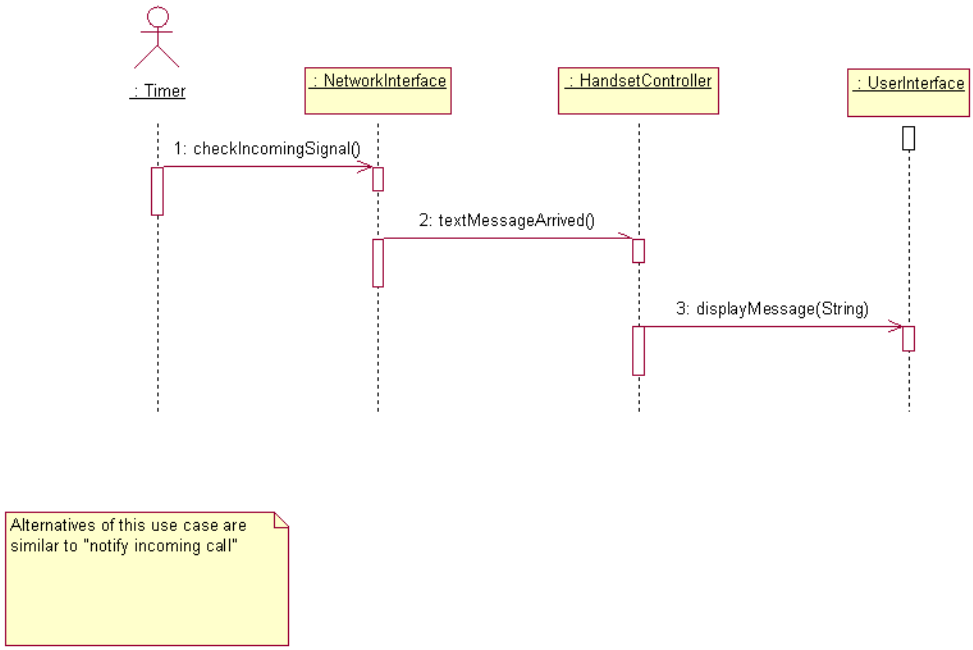

Figure 37: Sequence Diagram for Notify Incoming Text Message Use Case

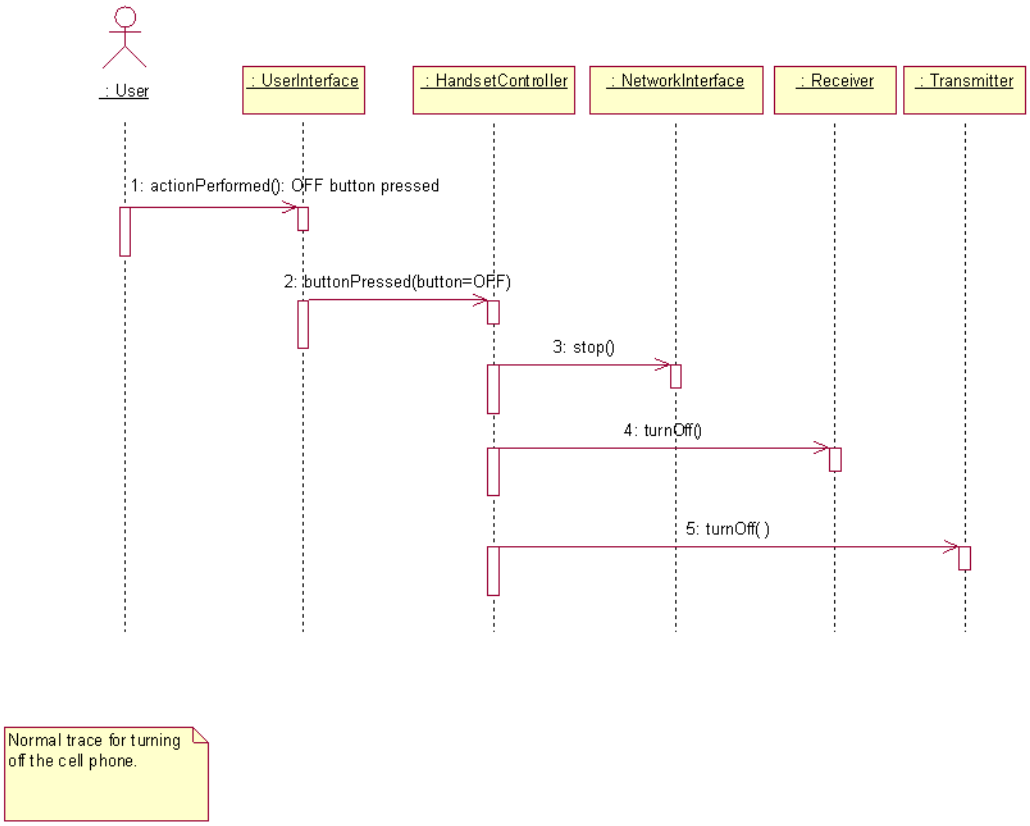

Figure 38: Sequence Diagram for Turn Off Cell Phone Use Case

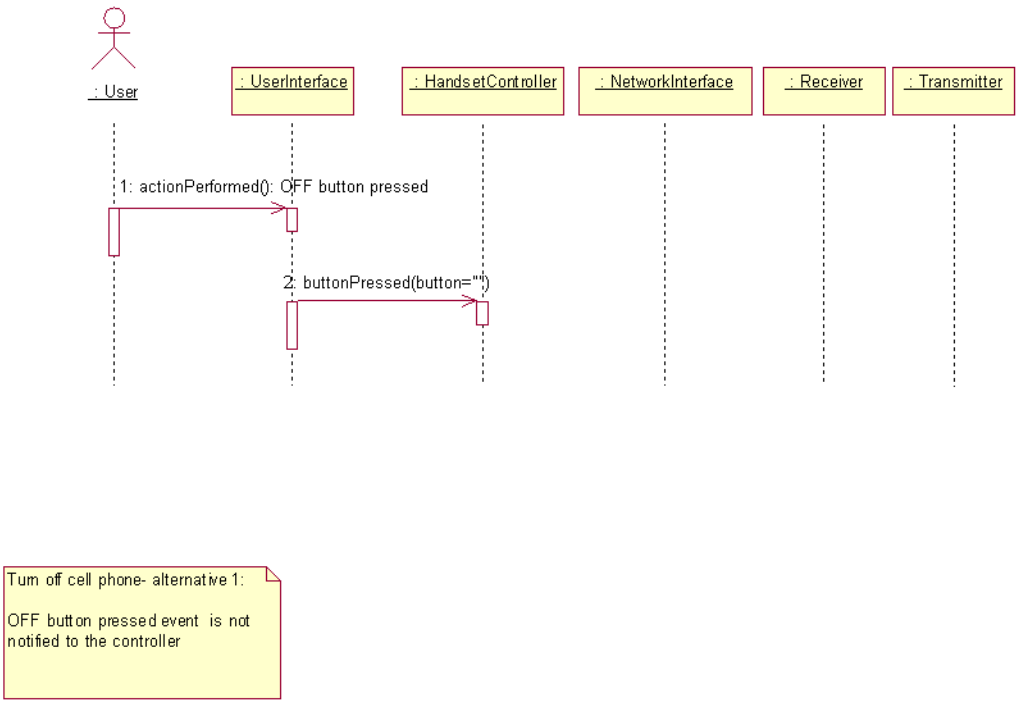

Figure 39: Sequence Diagram for Turn Off Cell Phone Use Case: Alternative 1

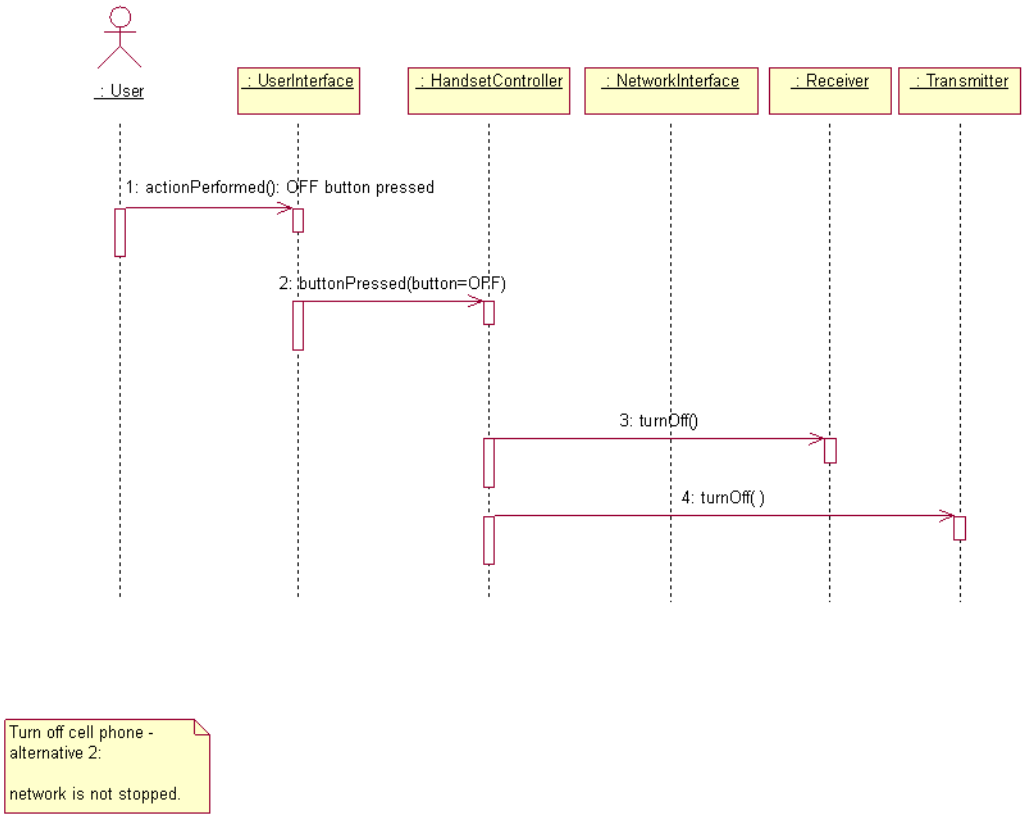

Figure 40: Sequence Diagram for Turn Off Cell Phone Use Case: Alternative 2

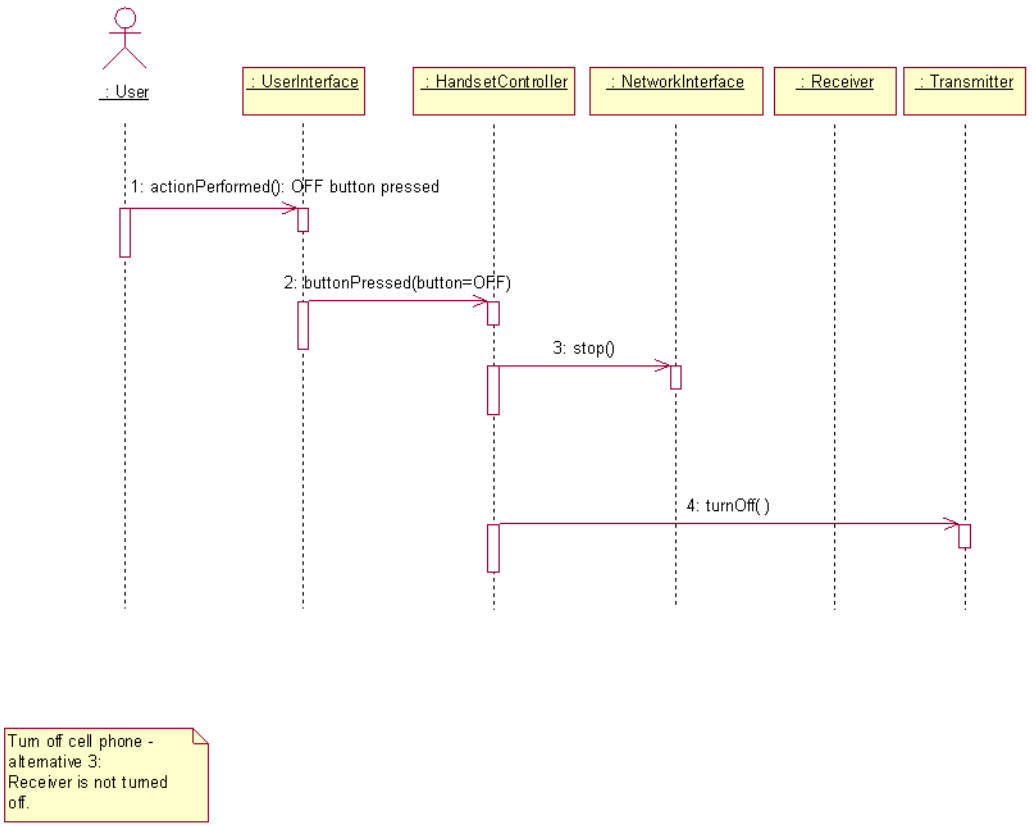

Figure 41: Sequence Diagram for Turn Off Cell Phone Use Case: Alternative 3

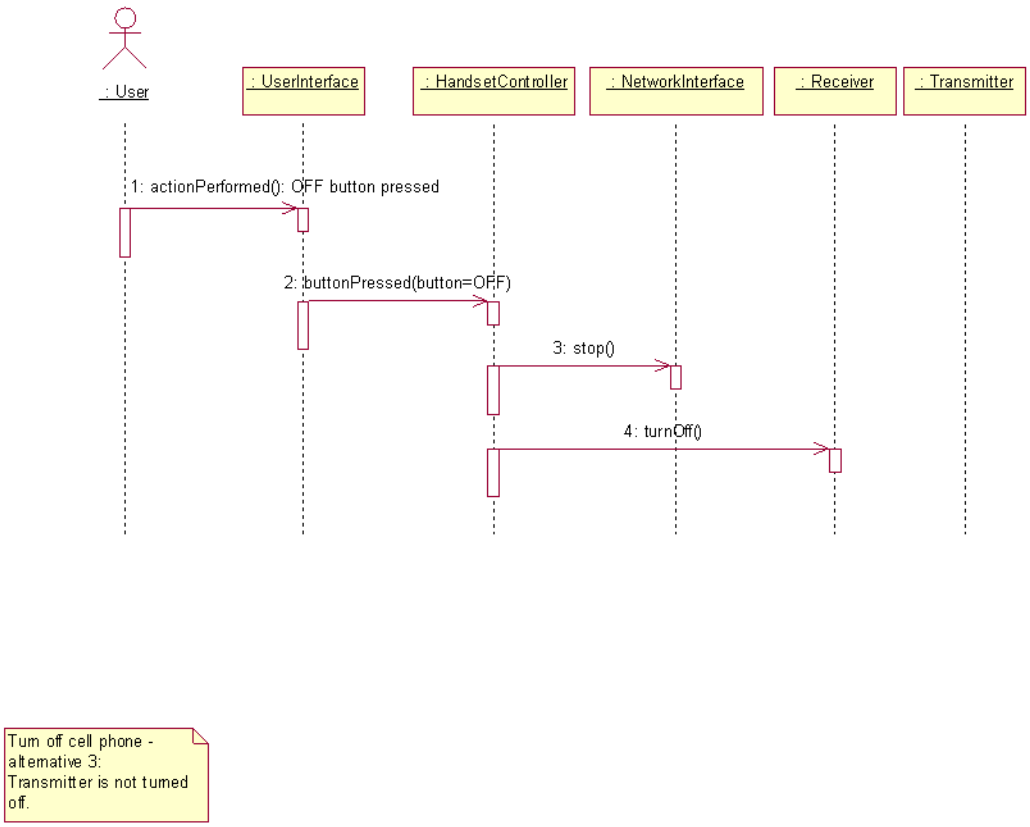

Figure 42: Sequence Diagram for Turn Off Cell Phone Use Case: Alternative 4

B Appendix: Test Scripts

Each test is implemented as a separate method. All tests for each class are implemented inside a single class. This document lists 39 tests for the `HandsetController` class, 5 tests for the `Initialization` class, 21 tests for the `NetworkInterface` class, 2 tests for the `Receiver` class, 2 tests for the `Transmitter` class, and 25 tests for the UserInterface class.

```
package cellphone;

/**
 * Title:        Tests for HandsetController class.
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class HandsetControllerTC implements Runnable
{

   static HandsetController cut; // cut -- class under test
   Thread t;

   public HandsetControllerTC()
   {
      //cut = new HandsetController( "testing HandsetController" );
   }

   public void start()
   {
      t = new Thread( this );
      t.setName( "Handset Controller Test Cases" );
      t.start();
   }

   public void run()
   {
      tc1();
      tc2();
      tc3();
      tc4();
      tc5();
      tc6();
      tc7();
      tc8();
      tc9();
      tc10();
      tc11();
      tc12();
      tc13();
      tc14();
      tc15();
      tc16();
      tc17();
      tc18();
      tc19();
      tc20();
      tc21();
      tc22();
      tc23();
      tc24();
      tc25();
      tc26();
      tc27();
      tc28();
      tc29();
      tc30();
      tc31();
      tc32();
      tc33();
      tc34();
      tc35();
      tc36();
```

```
   tc37();
   tc38();
   tc39();
}

private static void tc1()
{
  System.out.println( "running tc1 for HandsetController.java" );
  //cut = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  //HandsetController.printStateInfo();

  //HandsetController.kill();
  //HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc2()
{
  System.out.println( "running tc2 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  //HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  sleep();
  HandsetController.printStateInfo();


  //this.wait( 1000 );
  HandsetController.buttonPressed( "OFF" );
  sleep();
  HandsetController.printStateInfo();

  //HandsetController.kill();
  //HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc3()
{
  System.out.println( "running tc3 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  //HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  sleep();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "END" );
  sleep();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  sleep();
  HandsetController.printStateInfo();

  //HandsetController.kill();
  //HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}
```

```
private static void tc4()
{
  System.out.println( "running tc4 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  //HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  sleep();
  HandsetController.printStateInfo();

  HandsetController.textMessageArrived();
  sleep();
  HandsetController.printStateInfo();

  //timeout
  try
  {
     Thread.sleep( 5000 );
  }
  catch ( Exception e )
  {
     e.printStackTrace();
  }
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  sleep();
  HandsetController.printStateInfo();

  //HandsetController.kill();
  //HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc5()
{
  System.out.println( "running tc5 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.textMessageArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "END" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  //HandsetController.kill();
  //HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc6()
{
  System.out.println( "running tc6 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();
```

```
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.textMessageArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc7()
{
  System.out.println( "running tc7 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.textMessageArrived();
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc8()
{
  System.out.println( "running tc8 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.textMessageArrived();
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "END" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
```

```
  System.out.println( "====================================================" );
}

private static void tc9()
{
  System.out.println( "running tc9 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.textMessageArrived();
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  //timeout
  try
  {
     Thread.sleep( 50000 );
  }
  catch ( Exception e )
  {
     e.printStackTrace();
  }
  HandsetController.printStateInfo();


  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc10()
{
  System.out.println( "running tc10 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc11()
{
  System.out.println( "running tc11 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();
```

```
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "END" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc12()
{
  System.out.println( "running tc12 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  //timeout
  try
  {
     Thread.sleep( 50000 );
  }
  catch ( Exception e )
  {
     e.printStackTrace();
  }
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc13()
{
  System.out.println( "running tc13 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();
```

```
  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc14()
{
  System.out.println( "running tc14 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "END" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc15()
{
  System.out.println( "running tc15 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.externalHangup();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc16()
{
```

```
  System.out.println( "running tc16 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 3 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc17()
{
  System.out.println( "running tc17 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 4 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc18()
{
  System.out.println( "running tc18 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();
```

```
  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 3 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 4 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc19()
{
  System.out.println( "running tc19 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 3 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc20()
{
  System.out.println( "running tc20 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();
```

```
  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc21()
{
  System.out.println( "running tc21 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 4 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc22()
{
  System.out.println( "running tc22 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.incomingCallArrived();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 3 );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
```

```
  System.out.println( "====================================================" );
}

private static void tc23()
{
  System.out.println( "running tc23 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "#" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc24()
{
  System.out.println( "running tc24 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "#" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "END" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc25()
{
  System.out.println( "running tc25 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.timeout();
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();
```

```
  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc26()
{
  System.out.println( "running tc26 for HandsetController.java" );
  HandsetController.printStateInfo();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "#" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc27()
{
  System.out.println( "running tc27 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc28()
{
  System.out.println( "running tc28 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();
```

```
  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "#" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 3 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc29()
{
  System.out.println( "running tc29 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "#" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 0 );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "TALK" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "OFF" );
  HandsetController.printStateInfo();

  HandsetController.kill();
  HandsetController.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc30()
{
  System.out.println( "running tc30 for HandsetController.java" );
  //cut  = new HandsetController( "testing HandsetController" );
  HandsetController.printStateInfo();

  //cut.start();

  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "ON" );
  HandsetController.printStateInfo();

  HandsetController.buttonPressed( "#" );
  HandsetController.printStateInfo();

  HandsetController.connectionLevelChanged( 3 );
  HandsetController.printStateInfo();
```

```
      HandsetController.buttonPressed( "TALK" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "OFF" );
      HandsetController.printStateInfo();

      HandsetController.kill();
      HandsetController.printStateInfo();

      System.out.println();
      System.out.println( "=====================================================" );
    }

    private static void tc31()
    {
      System.out.println( "running tc31 for HandsetController.java" );
      //cut  = new HandsetController( "testing HandsetController" );
      HandsetController.printStateInfo();

      //cut.start();

      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "ON" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "#" );
      HandsetController.printStateInfo();

      HandsetController.connectionLevelChanged( 3 );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "TALK" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "END" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "OFF" );
      HandsetController.printStateInfo();

      HandsetController.kill();
      HandsetController.printStateInfo();

      System.out.println();
      System.out.println( "=====================================================" );
    }

    private static void tc32()
    {
      System.out.println( "running tc32 for HandsetController.java" );
      //cut  = new HandsetController( "testing HandsetController" );
      HandsetController.printStateInfo();

      //cut.start();

      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "ON" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "#" );
      HandsetController.printStateInfo();

      HandsetController.connectionLevelChanged( 3 );
      HandsetController.printStateInfo();

      HandsetController.setTestValues( false, true );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "TALK" );
      HandsetController.printStateInfo();

//      HandsetController.setTestValues( false, true );
 //     HandsetController.printStateInfo();

      HandsetController.buttonPressed( "OFF" );
```

```
      HandsetController.printStateInfo();

      HandsetController.kill();
      HandsetController.printStateInfo();

      System.out.println();
      System.out.println( "=====================================================" );
    }

    private static void tc33()
    {
      System.out.println( "running tc33 for HandsetController.java" );
      //cut  = new HandsetController( "testing HandsetController" );
      HandsetController.printStateInfo();

      //cut.start();

      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "ON" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "#" );
      HandsetController.printStateInfo();

      HandsetController.connectionLevelChanged( 3 );
      HandsetController.printStateInfo();

      HandsetController.setTestValues( true, false );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "TALK" );
      HandsetController.printStateInfo();

//      HandsetController.setTestValues( false, true );
 //     HandsetController.printStateInfo();

      HandsetController.buttonPressed( "OFF" );
      HandsetController.printStateInfo();

      HandsetController.kill();
      HandsetController.printStateInfo();

      System.out.println();
      System.out.println( "=====================================================" );
    }

    private static void tc34()
    {
      System.out.println( "running tc34 for HandsetController.java" );
      //cut  = new HandsetController( "testing HandsetController" );
      HandsetController.printStateInfo();

      //cut.start();

      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "ON" );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "#" );
      HandsetController.printStateInfo();

      HandsetController.connectionLevelChanged( 0 );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "TALK" );
      HandsetController.printStateInfo();

      HandsetController.connectionLevelChanged( 3 );
      HandsetController.printStateInfo();

      HandsetController.setTestValues( true, false );
      HandsetController.printStateInfo();

      HandsetController.buttonPressed( "TALK" );
      HandsetController.printStateInfo();
```

```
    //     HandsetController.setTestValues( false, true );
//    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "OFF" );
    HandsetController.printStateInfo();

    HandsetController.kill();
    HandsetController.printStateInfo();

    System.out.println();
    System.out.println( "=====================================================" );
  }

  private static void tc35()
  {
    System.out.println( "running tc35 for HandsetController.java" );
    //cut  = new HandsetController( "testing HandsetController" );
    HandsetController.printStateInfo();

    //cut.start();

    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "ON" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "#" );
    HandsetController.printStateInfo();

    HandsetController.connectionLevelChanged( 3 );
    HandsetController.printStateInfo();

    HandsetController.setTestValues( true, false );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "TALK" );
    HandsetController.printStateInfo();

    //     HandsetController.setTestValues( false, true );
//    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "END" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "OFF" );
    HandsetController.printStateInfo();

    HandsetController.kill();
    HandsetController.printStateInfo();

    System.out.println();
    System.out.println( "=====================================================" );
  }

  private static void tc36()
  {
    System.out.println( "running tc36 for HandsetController.java" );
    //cut  = new HandsetController( "testing HandsetController" );
    HandsetController.printStateInfo();

    //cut.start();

    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "ON" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "#" );
    HandsetController.printStateInfo();

    HandsetController.connectionLevelChanged( 3 );
    HandsetController.printStateInfo();

    HandsetController.setTestValues( true, false );
    HandsetController.printStateInfo();
```

```
    HandsetController.buttonPressed( "TALK" );
    HandsetController.printStateInfo();

    //     HandsetController.setTestValues( false, true );
//    HandsetController.printStateInfo();

    HandsetController.externalHangup();
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "OFF" );
    HandsetController.printStateInfo();

    HandsetController.kill();
    HandsetController.printStateInfo();

    System.out.println();
    System.out.println( "====================================================" );
  }

  private static void tc37()
  {
    System.out.println( "running tc37 for HandsetController.java" );
    //cut  = new HandsetController( "testing HandsetController" );
    HandsetController.printStateInfo();

    //cut.start();

    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "ON" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "#" );
    HandsetController.printStateInfo();

    HandsetController.connectionLevelChanged( 3 );
    HandsetController.printStateInfo();

    HandsetController.setTestValues( true, false );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "TALK" );
    HandsetController.printStateInfo();

    //     HandsetController.setTestValues( false, true );
//    HandsetController.printStateInfo();

    HandsetController.connectionLevelChanged( 0 );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "OFF" );
    HandsetController.printStateInfo();

    HandsetController.kill();
    HandsetController.printStateInfo();

    System.out.println();
    System.out.println( "====================================================" );
  }


  private static void tc38()
  {
    System.out.println( "running tc38 for HandsetController.java" );
    //cut  = new HandsetController( "testing HandsetController" );
    HandsetController.printStateInfo();

    //cut.start();

    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "ON" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "#" );
    HandsetController.printStateInfo();
```

```
    HandsetController.connectionLevelChanged( 3 );
    HandsetController.printStateInfo();

    HandsetController.setTestValues( true, false );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "TALK" );
    HandsetController.printStateInfo();

    //     HandsetController.setTestValues( false, true );
//    HandsetController.printStateInfo();

    HandsetController.connectionLevelChanged( 0 );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "END" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "OFF" );
    HandsetController.printStateInfo();

    HandsetController.kill();
    HandsetController.printStateInfo();

    System.out.println();
    System.out.println( "====================================================" );
  }

  private static void tc39()
  {
    System.out.println( "running tc39 for HandsetController.java" );
    //cut  = new HandsetController( "testing HandsetController" );
    HandsetController.printStateInfo();

    //cut.start();
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "ON" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "#" );
    HandsetController.printStateInfo();

    HandsetController.connectionLevelChanged( 3 );
    HandsetController.printStateInfo();

    HandsetController.setTestValues( true, false );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "TALK" );
    HandsetController.printStateInfo();

    //     HandsetController.setTestValues( false, true );
//    HandsetController.printStateInfo();

    HandsetController.externalHangup();
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "END" );
    HandsetController.printStateInfo();

    HandsetController.buttonPressed( "OFF" );
    HandsetController.printStateInfo();

    HandsetController.kill();
    HandsetController.printStateInfo();

    System.out.println();
    System.out.println( "====================================================" );
  }
/**
  public static void main(String[] args)
  {
     /**
     tc1();
     tc2();
     tc3();
```

```
      tc4();
      tc5();
      tc6();
      tc7();
      tc8();
      tc9();
      tc10();
      tc11();
      tc12();
      tc13();
      tc14();
      tc15();
      tc16();
      tc17();
      tc18();
      tc19();
      tc20();
      tc21();
      tc22();
      tc23();
      tc24();
      tc25();
      tc26();
      tc27();
      tc28();
      tc29();
      tc30();
      tc31();
      tc32();
      tc33();
      tc34();
      tc35();
      tc36();
      tc37();

      tc38();
      tc39();
   }
  **/

   private static void sleep()
   {
      try
      {
         Thread.sleep( 1000 );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }
   }
}

package cellphone;

/**
 * Title:        Tests for Initialization class.
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class InitializationTCs implements Runnable
{

   static HandsetController cut; // cut -- class under test
   Thread t;

   public InitializationTCs()
   {

   }

   public void start()
   {
```

```
      t = new Thread( this );
      t.setName( "Initialization Test Cases" );
      t.start();
   }

   public void run()
   {
      tc1();
      tc2();
      tc3();
      tc4();
      tc5();
   }

   private static void tc1()
   {
     System.out.println( "running tc1 for Initialization Sequence" );
     HandsetController.printStateInfo();

     //cut.start();

     //HandsetController.printStateInfo();

     //HandsetController.kill();
     //HandsetController.printStateInfo();

     System.out.println();
     System.out.println( "=====================================================" );
   }

   private static void tc2()
   {
     System.out.println( "running tc2 for HandsetController.java" );
     //cut  = new HandsetController( "testing HandsetController" );
     HandsetController.printStateInfo();

     //cut.start();

     //HandsetController.printStateInfo();

     HandsetController.buttonPressed( "ON" );
     sleep();
     HandsetController.printStateInfo();


     HandsetController.buttonPressed( "OFF" );
     sleep();
     HandsetController.printStateInfo();

     //HandsetController.kill();
     //HandsetController.printStateInfo();

     System.out.println();
     System.out.println( "=====================================================" );
   }

}

package cellphone;

/**
 * Title:        Tests for NetworkInterface class.
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class NetworkInterfaceTCs
{
   static NetworkInterface cut; // cut -- class under test

   public NetworkInterfaceTCs()
   {
      cut = new NetworkInterface();
   }
```

```
private static void tc1()
{
  System.out.println( "running tc1 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc2()
{
  System.out.println( "running tc2 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 10 );
  cut.printStateInfo();

  cut.checkConnectionLevel( 20 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc3()
{
  System.out.println( "running tc3 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 10 );
  cut.printStateInfo();

  cut.checkConnectionLevel( 10 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc4()
{
  System.out.println( "running tc4 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();
```

```
  cut.checkConnectionLevel( 10 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc5()
{
  System.out.println( "running tc5 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 60 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 2 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc6()
{
  System.out.println( "running tc6 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 30 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 2 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc7()
{
  System.out.println( "running tc7 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 60 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.stop();
```

```
  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc8()
{
  System.out.println( "running tc8 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 5 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.endConnection();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc9()
{
  System.out.println( "running tc9 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.timeout();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc10()
{
  System.out.println( "running tc10 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.connected();
```

```
  cut.printStateInfo();

  cut.checkConnectionLevel( 55 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc11()
{
  System.out.println( "running tc11 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.connected();
  cut.printStateInfo();

  cut.endConnection();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc12()
{
  System.out.println( "running tc12 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.connected();
  cut.printStateInfo();

  cut.endConnection();
  cut.printStateInfo();

  cut.checkConnectionLevel( 55 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc13()
```

```
{
  System.out.println( "running tc13 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.connected();
  cut.printStateInfo();

  cut.externalHangup();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc14()
{
  System.out.println( "running tc14 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 5 );
  cut.printStateInfo();

  cut.makeConnection( "7039931632", 1 );
  cut.printStateInfo();

  cut.timeout();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "====================================================" );
}

private static void tc15()
{
  System.out.println( "running tc15 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 5 );
  cut.printStateInfo();

  cut.makeConnection( "7039931632", 1 );
  cut.printStateInfo();

  cut.endConnection();
  cut.printStateInfo();

  cut.stop();
```

```
  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc16()
{
  System.out.println( "running tc16 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.makeConnection( "7039931632", 1 );
  cut.printStateInfo();

  cut.endConnection();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc17()
{
  System.out.println( "running tc17 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.makeConnection( "7039931632", 1 );
  cut.printStateInfo();

  cut.checkConnectionLevel( 55 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "======================================================" );
}

private static void tc18()
{
  System.out.println( "running tc18 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.makeConnection( "7039931632", 1 );
  cut.printStateInfo();

  cut.externalHangup();
```

```
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc19()
{
  System.out.println( "running tc19 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.makeConnection( "7039931632", 1 );
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc20()
{
  System.out.println( "running tc20 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkConnectionLevel( 25 );
  cut.printStateInfo();

  cut.checkIncomingSignal( 1 );
  cut.printStateInfo();

  cut.connected();
  cut.printStateInfo();

  cut.stop();

  cut.printStateInfo();

  System.out.println();
  System.out.println( "=====================================================" );
}

private static void tc21()
{
  System.out.println( "running tc21 for NetworkInterface.java" );
  cut = new NetworkInterface();
  cut.printStateInfo();

  cut.start();

  cut.printStateInfo();

  cut.checkIncomingSignal( );
  cut.printStateInfo();

  cut.stop();
  cut.printStateInfo();
```

```
      System.out.println();
      System.out.println( "=====================================================" );
   }


   public static void main(String[] args)
   {
      tc1();
      tc2();
      tc3();
      tc4();
      tc5();
      tc6();
      tc7();
      tc8();
      tc9();
      tc10();
      tc11();
      tc12();
      tc13();
      tc14();
      tc15();
      tc16();
      tc17();
      tc18();
      tc19();
      tc20();
      tc21();
   }
}

package cellphone;

/**
 * Title:        Tests for Receiver class.
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author       Aynur Abdurazik
 * @version 1.0
 */

public class ReceiverTCs
{
   static Receiver cut; // cut -- class under test

   public ReceiverTCs()
   {
      cut = new Receiver();
   }

   public static void main(String[] args)
   {
      tc1();
      tc2();
   }

   private static void tc1()
   {
     System.out.println( "running tc1 for Receiver.java" );
     cut = new Receiver( new AudioSample( "Receiver" ) );
     cut.printStateInfo();
     cut.turnOn();

     cut.printStateInfo();

     cut.turnOff();

     cut.printStateInfo();

     System.out.println();
     System.out.println( "=====================================================" );
   }

   private static void tc2()
   {
     System.out.println( "running tc2 for Receiver.java" );
```

```
     cut = new Receiver( new AudioSample( "Receiver" ) );
     cut.printStateInfo();
     cut.turnOn();

     cut.printStateInfo();

     cut.startReceive();

     cut.printStateInfo();

     cut.stopReceive();

     cut.printStateInfo();

     cut.turnOff();

     cut.printStateInfo();

     System.out.println();
     System.out.println( "======================================================" );
   }
}

package cellphone;

/**
 * Title:        Tests for Transmitter class.
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author       Aynur Abdurazik
 * @version 1.0
 */

public class TransmitterTCs
{
   static Transmitter cut; // cut -- class under test

   public TransmitterTCs()
   {
      cut = new Transmitter();
   }

   public static void main(String[] args)
   {
      tc1();
      tc2();
   }

   private static void tc1()
   {
     System.out.println( "running tc1 for Transmitter.java" );
     cut = new Transmitter( new AudioSample( "Transmitter" ) );
     cut.printStateInfo();
     cut.turnOn();

     cut.printStateInfo();

     cut.turnOff();

     cut.printStateInfo();

     System.out.println();
     System.out.println( "======================================================" );
   }

   private static void tc2()
   {
     System.out.println( "running tc2 for Transmitter.java" );
     cut = new Transmitter( new AudioSample( "Transmitter" ) );
     cut.printStateInfo();
     cut.turnOn();

     cut.printStateInfo();

     cut.startTransmit();
```

```
      cut.printStateInfo();

      cut.stopTransmit();

      cut.printStateInfo();

      cut.turnOff();

      cut.printStateInfo();

      System.out.println();
      System.out.println( "======================================================" );
   }
}

//package testcases;
package cellphone;

//import cellphone.UserInterface;
import java.awt.event.ActionEvent;

public class UserInterfaceTCs
{
   static UserInterface cut; // cut -- class under test

   public UserInterfaceTCs()
   {
      cut = new UserInterface( "for testing" );
   }

   private static void tc1()
   {
     System.out.println( "running tc1 for UserInterface.java" );
     cut = new UserInterface( "for testing" );
     cut.printStateInfo();
     cut.printStateInfo();

     System.out.println();
     System.out.println( "======================================================" );
   }

   private static void tc2()
   {
      System.out.println( "running tc2 for UserInterface.java" );
      cut = new UserInterface( "for testing" );
      cut.printStateInfo();
      cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
      cut.printStateInfo();
      cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
      cut.printStateInfo();

      System.out.println();
      System.out.println( "======================================================" );

   }

   private static void tc3()
   {
      System.out.println( "running tc3 for UserInterface.java" );
      cut = new UserInterface( "for testing" );
      cut.printStateInfo();
      cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
      cut.printStateInfo();
      cut.getPhoneNumber();
      cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
      cut.printStateInfo();

      System.out.println();
      System.out.println( "======================================================" );
   }

   private static void tc4()
   {
      System.out.println( "running tc4 for UserInterface.java" );
      cut = new UserInterface( "for testing" );
      cut.printStateInfo();
      cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
```

```
   cut.printStateInfo();
   cut.clear();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}

private static void tc5()
{
   System.out.println( "running tc5 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "END") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}

private static void tc6()
{
   System.out.println( "running tc6 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.displayMessage("busy");
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc7()
{
   System.out.println( "running tc7 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.displayMessage("incoming call");
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc8()
{
   System.out.println( "running tc8 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.displayMessage("text message");
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc9()
{
   System.out.println( "running tc9 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
```

```
   cut.printStateInfo();
   cut.setConnectionLevel(0);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc10()
{
   System.out.println( "running tc10 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.setConnectionLevel(1);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc11()
{
   System.out.println( "running tc11 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.setConnectionLevel(2);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc12()
{
   System.out.println( "running tc12 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.setConnectionLevel(3);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc13()
{
   System.out.println( "running tc13 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.setConnectionLevel(4);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc14()
{
   System.out.println( "running tc14 for UserInterface.java" );
```

```
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.setConnectionLevel(5);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc15()
{
   System.out.println( "running tc15 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.setConnectionLevel(6);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc16()
{
   System.out.println( "running tc16 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#" ) );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc17()
{
   System.out.println( "running tc17 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "END") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc18()
{
   System.out.println( "running tc18 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "TALK") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
```

```
   System.out.println( "=====================================================" );
}


private static void tc19()
{
   System.out.println( "running tc19 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "1") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "TALK") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}



private static void tc20()
{
   System.out.println( "running tc20 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "1") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc21()
{
   System.out.println( "running tc21 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "1") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "END") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc22()
{
   System.out.println( "running tc22 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "1") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "TALK") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "END") );
```

```
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc23()
{
   System.out.println( "running tc23 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "#") );
   cut.printStateInfo();
   cut.setConnectionLevel(0);
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "1") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "TALK") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "END") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}

private static void tc24()
{
   System.out.println( "running tc24 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.displayMessage( "incoming call" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "TALK") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}


private static void tc25()
{
   System.out.println( "running tc25 for UserInterface.java" );
   cut = new UserInterface( "for testing" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "ON") );
   cut.printStateInfo();
   cut.displayMessage( "incoming call" );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "TALK") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "END") );
   cut.printStateInfo();
   cut.actionPerformed( new ActionEvent( new Object(), 1, "OFF") );
   cut.printStateInfo();
   System.out.println();
   System.out.println( "=====================================================" );
}

public static void main( String[] arg )
{
   tc1();
   tc2();
   tc3();
   tc4();
   tc5();
   tc6();
   tc7();
```

```
        tc8();
        tc9();
        tc10();
        tc11();
        tc12();
        tc13();
        tc14();
        tc15();
        tc16();
        tc17();
        tc18();
        tc19();
        tc20();
        tc21();
        tc22();
        tc23();
        tc24();
        tc25();
    }
}
```

C Appendix: Source implementation

```
package cellphone;

import java.awt.*;
import javax.swing.*;
import testcases.*;

public class HandsetController //implements Runnable
{
   final static int ON = 1;
   final static int OFF = 0;

   final static int IDLE = 0;
   final static int ACTIVE = 1;

   final static int WAITING_FOR_EVENTS = 0;
   final static int READING = 1;
   final static int WAITING_FOR_TALK_BUTTON = 2;
   final static int WAITING_FOR_PHONENUMBER = 3;
   final static int MAKING_CALL = 4;
   final static int TALKING = 5;

   static boolean activated;

   static UserInterface frame;
   boolean packFrame = true;

   static NetworkInterface networkInterface;
   static boolean networkStarted;
   static boolean testingForTruth, testingForFalse;

   static volatile String button;
///   static boolean cellPhoneOn;

   static int availability;
   static int connectionLevel;
   static int incomingSignal;

   static int cellPhone;
   static int superState;
   static int subState;

   static Receiver receiver;
   static Transmitter transmitter;

   static Thread t;

   static boolean testing;

   public HandsetController()
   {
      System.out.println( " Handset Controller constructor" );
      networkInterface = new NetworkInterface();
      receiver = new Receiver();
      transmitter = new Transmitter();
      networkStarted = false;
      button = new String(" ");
      activated = true;

      frame = new UserInterface();
      //Pack frames that have useful preferred size info, e.g. from their layout
      //Validate frames that have preset sizes
      if ( packFrame )
         frame.pack();
      else
         frame.validate();

      // Center the frame
      Dimension screenSize =  Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = frame.getSize();
      if ( frameSize.height > screenSize.height )
         frameSize.height = screenSize.height;
      if ( frameSize.width > screenSize.width )
         frameSize.width = screenSize.width;
      frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height - frameSize.height) / 2);
```

```
      frame.setSize( 500, 600 ) ;
      frame.setVisible(true);
      frame.repaint() ;
//      t = new Thread( this );
//      t.setName( "HandsetController" );

//      testing = false;

  }

   public HandsetController( String testVersion )
   {
      System.out.println( testVersion );
      networkInterface = new NetworkInterface();
      receiver = new Receiver();
      transmitter = new Transmitter();
      networkStarted = false;
      button = new String(" ");
      activated = true;

      HandsetControllerTC hctc = new HandsetControllerTC();
      hctc.start();

//      t = new Thread( this );
//      t.setName( "HandsetController--testing" );

      testing = true;


  }


  // Main method
  static public void main( String[] args )
  {
     System.out.println( " Handset Controller main method " );

     try
     {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
     }

     catch(Exception e)
     {
        e.printStackTrace();
     }

     new HandsetController( "testing " );
     // new HandsetController(); // no testing
     start();

  }

//  static void start()
//  {
//     t.start();
//  }

  static void start()
  {
     System.out.println( " Handset Controller start()" );
     String connectionMessage, generalMessage = new String();
     String phoneNumber;
     boolean connected = false;
     int i;

     superState = IDLE;


      while ( activated )
      {
         //button = frame.getButtonPressed();

         //frame.clear();

         //System.out.println( "button: " + button );
```

```
switch ( superState )
{

   case IDLE:

      if ( button.equals( "ON" ) )
      {
         cellPhone = ON;


         if ( ! networkStarted )
         {
            //receiver = new Receiver();
            //transmitter = new Transmitter();
            receiver.turnOn();
            transmitter.turnOn();
            networkInterface.start();
            networkStarted = true;
         }

         superState = ACTIVE;

      }
      break;

   case ACTIVE:

      if ( button.equals( "OFF" ) )
      {
         cellPhone = OFF;
         superState = IDLE;

         if ( networkStarted )
         {

            receiver.turnOff();
            transmitter.turnOff();
            networkInterface.stop();
            networkStarted = false;
         }

         //button = " ";
      }

      else if ( button.equals( "END" ) )
      {
         if ( ! testing )
         {
            frame.clear();
         }

         networkInterface.endConnection();
         subState = WAITING_FOR_EVENTS;
         connected = false;
         //button =  " ";
      }

      else
      {
         switch ( subState )
         {
            case WAITING_FOR_EVENTS:

               if ( incomingSignal == 1 )
               {
                  subState = WAITING_FOR_TALK_BUTTON;
               }

               else if ( incomingSignal == 2 )
               {
                  subState = READING;
               }

               else if ( button.equals( "#" ) )
               {
                  subState = WAITING_FOR_PHONENUMBER;
```

```
      button = " ";
   }

   break;

case WAITING_FOR_TALK_BUTTON:

   i = 0;

   while ( ! button.equals( "TALK" ) && ! button.equals( "END" ) &&
           ! button.equals( "OFF" ) && i++ < 500 )
   {
      try
      {
         Thread.sleep( 100 );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }
   }

   if ( button.equals( "TALK" ) )
   {
      connected = true;
      networkInterface.connected();
      subState = TALKING;
   }

   else if ( button.equals( "END" ) || button.equals( "OFF" ) )
   {
       System.out.println("Button = " + button );
      break;
   }

   else //timeout
   {
      networkInterface.timeout();

      if ( ! testing )
      {
         frame.displayMessage( "timeout" );
      }

      else
      {
         System.out.println( "timeout" );
      }

      subState = WAITING_FOR_EVENTS;
      connected = false;
   }
   break;

case WAITING_FOR_PHONENUMBER:

   i = 0;

   phoneNumber = new String(" ");

   while ( ! button.equals( "TALK" ) && ! button.equals( "END" ) &&
           ! button.equals( "OFF" ) && i++ < 500 )
   {
      try
      {
         Thread.sleep(100);
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }
   }

   if ( button.equals( "TALK" ) && connectionLevel > 0 )
   {
      subState = MAKING_CALL;
```

```
            if ( ! testing )
            {
               phoneNumber = frame.getPhoneNumber();
            }

            if ( testingForTruth )
            {
               connected = true;
            }
            else if ( testingForFalse )
            {
               connected = false;
            }

            else
            {
               connected = networkInterface.makeConnection( phoneNumber );
            }
         }

         else if ( connectionLevel == 0 )
         {
            if ( ! testing )
            {
               frame.displayMessage( "connection failed - callee is busy" );
            }

            else
            {
                System.out.println( "connection failed = callee is busy" );
            }

            subState = WAITING_FOR_EVENTS;
            connected = false;
         }

         else if ( button.equals( "OFF" ) || button.equals( "END" ) )
         {
            break;
         }

         else // timeout
         {
            if ( ! testing )
            {
               frame.displayMessage( "timeout" );
            }

            else
            {
               System.out.println( "timeout" );
            }

            subState = WAITING_FOR_EVENTS;
            connected = false;
         }

         break;

      case READING:

         System.out.println( "READING state" );
         i = 0;

         while ( ! button.equals( "END" ) && ! button.equals( "OFF" ) &&
                  i++ < 500 && incomingSignal != 1 )
         {
            try
            {
               Thread.sleep( 100 );
            }
            catch ( Exception e )
            {
               e.printStackTrace();
            }
         }
```

```
      if ( incomingSignal == 1 )
      {
         if ( ! testing )
         {
            frame.displayMessage( "incoming call" );
         }

         else
         {
            System.out.println( "incoming call" );
         }

         subState = WAITING_FOR_TALK_BUTTON;
      }

      else if ( button.equals( "END" ) || button.equals( "OFF" ) )
      {
         incomingSignal = 0;
         break;
      }

      else //timeout
      {
         if ( ! testing )
         {
            frame.clear();
         }

         subState = WAITING_FOR_EVENTS;
      }

      break;

   case TALKING:

      connectionMessage = "connected";

      if ( ! testing )
      {
         frame.displayMessage( connectionMessage );
      }

      else
      {
         System.out.println( connectionMessage );
      }

      generalMessage = talk();

      if ( ! testing )
      {
         frame.displayMessage( generalMessage );
      }

      else
      {
         System.out.println( generalMessage );
      }

      break;

   case MAKING_CALL:

      if ( button.equals( "OFF" ) || button.equals( "END" ) )
      {
         break;
      }

      if ( connected )
      {
         if ( ! testing )
         {
            frame.displayMessage( "connected" );
         }
         else
         {
            System.out.println( "connected" );
```

```
                        }

                        subState = TALKING;
                    }

                    else
                    {
                        if ( !testing )
                        {
                            frame.displayMessage( "connection failed" );
                        }
                        else
                        {
                            System.out.println( "connection failed" );
                        }
                        subState = WAITING_FOR_EVENTS;
                    }

                    break;
                }
            }

            break;
        }
    }
}

private static String talk()
{
   System.out.println( " Handset Controller talk()" );


   AudioSample sample;

   sample = new AudioSample( "Transmitter" );
   transmitter.setSample( sample );
   transmitter.startTransmit();

   sample = new AudioSample( "Receiver" );
   receiver.setSample( sample );
   receiver.startReceive();




    while ( ! button.equals( "END" ) && connectionLevel > 0
                                    && ! button.equals( "OFF" ) ) //potential bug cellPhone =OFF?
    {
        try
        {
            Thread.sleep( 1000 );
        }

        catch ( Exception e )
        {
           e.printStackTrace();
        }
    }

    transmitter.stopTransmit();
    receiver.stopReceive();

    if ( ! testing )
    {
       frame.clear();
    }

    if ( button.equals( "OFF" ) )
    {
       networkInterface.stop();
    }

    else
    {
       networkInterface.endConnection();
    }
```

```
   return "talk done!";

}

public static void buttonPressed( String b )
{
   System.out.println( " Handset Controller buttonPressed(" + b + ")" );
   button = b;
}


public static void incomingCallArrived()
{
   if ( cellPhone == ON )
   {
      System.out.println( " Handset Controller incomingCallArrived()" );
      incomingSignal = 1;

      if ( ! testing )
      {
         frame.displayMessage( " incoming call " );
      }
      else
      {
         System.out.println( "incoming call" );
      }
   }
}

public static void textMessageArrived()
{
   if ( cellPhone == ON )
   {
      System.out.println( " Handset Controller textMessageArrived()" );
      incomingSignal = 2;
      //frame.setIncomingSignal( incomingSignal );

      if ( ! testing )
      {
         frame.displayMessage( " text message " );
      }
      else
      {
         System.out.println( "text message" );
      }
   }
}

public static void connectionLevelChanged( int conLevel )
{
   if ( cellPhone == ON )
   {

      System.out.println( " Handset Controller connectionLevelChanged()" );
      connectionLevel = conLevel;

      if ( ! testing )
      {
         frame.setConnectionLevel( connectionLevel );
      }
      else
      {
          System.out.println( connectionLevel );
      }
   }
}

public static void kill()
{
   activated = false;
}

public static void printStateInfo()
{
   System.out.print( "HandsetController state = " );

   //System.out.print(  " " );
```

```
      //System.out.println( superState +  " " + subState );
/**
      try
      {
         Thread.sleep( 500 );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }
**/
      if ( superState == IDLE )
      {
         System.out.println( "IDLE" );
      }

      else if ( subState == WAITING_FOR_EVENTS )
      {
         System.out.println( "ACTIVE: WAITING_FOR_EVENTS" );
      }

      else if ( subState == READING )
      {
         System.out.println( "ACTIVE: READING" );
      }

      else if ( subState == WAITING_FOR_TALK_BUTTON )
      {
         System.out.println( "ACTIVE: WAITING_FOR_TALK_BUTTON" );
      }

      else if ( subState == WAITING_FOR_PHONENUMBER )
      {
         System.out.println( "ACTIVE: WAITING_FOR_PHONENUMBER" );
      }

      else if ( subState == MAKING_CALL )
      {
         System.out.println( "ACTIVE: MAKING_CALL" );
      }

      else if ( subState == TALKING )
      {
         System.out.println( "ACTIVE: TALKING" );
      }

      System.out.println( "activated = " + activated );
   }

   static void setTestValues( boolean forTruth, boolean forFalse )
   {
      testingForTruth = forTruth;
      testingForFalse = forFalse;
   }

   static void timeout()
   {
        //timeout
      try
      {
         Thread.sleep( 50000 );
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }
   }

   static void externalHangup()
   {
      superState = ACTIVE;
      subState = WAITING_FOR_EVENTS;
      networkInterface.endConnection();

      if ( ! testing )
      {
         frame.clear();
```

```
      }

      button = " ";
   }
}

/**
         while ( cellPhone == ON )
         {

            try
            {
               Thread.sleep( 50 );
            }
            catch ( Exception e )
            {
               e.printStackTrace();
            }

            if ( incomingSignal == 1 )
            {

               System.out.println(" waiting for TALK button" );

               i = 0;
               while ( ! button.equals( "TALK" ) && i++ < 50 )
               {
                  try
                  {
                     Thread.sleep( 100 );
                  }
                  catch ( Exception e )
                  {
                     e.printStackTrace();
                  }
               }

               //button = frame.getButtonPressed();

               System.out.println( " button = " + button + button.length() );
               if ( button.equals( "TALK" ) )
               {
                    connected = true;
                    networkInterface.connected();
                    generalMessage = talk();
                    frame.displayMessage( generalMessage );
               }

               else  if ( button.equals( "END" ) )
               {
                  frame.clear();
               }

               else //timeout
               {
                  networkInterface.timeout();
                  frame.displayMessage( "timeout" );
               }

               incomingSignal = 0;
            }


            else if ( incomingSignal == 2 ) // text message
            {
               i = 0;

               while ( !button.equals( "END" ) && !button.equals( "OFF" ) &&
                         i++ < 50 && ! ( incomingSignal == 1 ) )
               {
                  try
                  {
                     Thread.sleep(100);
                  }
                  catch ( Exception e )
                  {
                     e.printStackTrace();
```

```
      }
   }

   if ( !( incomingSignal == 1 ) )
   {
      frame.clear();
      incomingSignal = 0;
   }


}

//else
{
   if ( button.equals( "#" ) )
   {

      i = 0;
      while ( ! button.equals( "TALK" ) && i++ < 500 )
      {
         try
         {
            Thread.sleep(100);
         }
         catch ( Exception e )
         {
            e.printStackTrace();
         }
      }


      if ( button.equals( "TALK" ) && connectionLevel > 0 )
      {
         phoneNumber = frame.getPhoneNumber();

         System.out.println( " getting phone number " );
         connected = networkInterface.makeConnection( phoneNumber );

         if ( connected )
         {
            connectionMessage = "connected";
            //frame.setConnectionMessage( connectionMessage );
            frame.displayMessage( connectionMessage );
            generalMessage = talk();
            frame.displayMessage( generalMessage );
         }

         else
         {
            connectionMessage = "connection failed - callee is busy";
         }
      }

      else if ( connectionLevel == 0 )
      {
         connectionMessage = "no connection - cannot connect";
      }

      else
      {
         connectionMessage = " timeout - connection aborted ";
      }

      frame.displayMessage( connectionMessage );

      try
      {
          Thread.sleep(2000);
      }
      catch ( Exception e )
      {
         e.printStackTrace();
      }

      frame.clear();
      button = " ";
```

```
            }
         }

         if ( button.equals( "OFF" ) )
         {
            networkInterface.stop();
            networkStarted = false;

            cellPhone = OFF;
         }

         if ( button.equals( "END" ) )
         {
            frame.clear();
            button = " ";
            //networkInterface.endConnection();
         }
      }
**/

package cellphone;

/**
 * Title:        Source file for class NetworkInterface
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

import java.util.Random;
import java.util.Timer;

 public class NetworkInterface implements Runnable
{

  private final static int IDLE       = 0;
  private final static int CONNECTING = 1;
  private final static int WAITING    = 2;
  private final static int CONNECTED  = 3;

  Thread t;
  Timer timer;
  PeriodicTasks pt;

  private static volatile boolean powerOn;

  private static volatile int connectionLevel;
  static volatile int distance, pre_conLevel;
  private static volatile int availability; // 1 -- available; 0 -- not available (callee)
  private static volatile int incomingSignal; // 0 -- no signal, 1 -- incoming call, 2 -- text message
  private static volatile int state; // 0 - idle, 1 - waiting, 2 - connecting, 3 - connected

//  private volatile boolean  connected, connecting, waiting;
  //private volatile int state;

  boolean instrumented;

  int status; // 1 -- talking, 2 - idle

  public NetworkInterface()
  {
     System.out.println( " NetworkInterface constructor" );

     connectionLevel = 0;
     powerOn = false;
     availability = 0;
     incomingSignal = 0;

     instrumented = true;
  }

  /**
   * The distance to the nearest cell tower should be less than 50 miles.
   * The distance is generated through random number generation in this
   * program.
```

```
   * level 5: 0-9   -- excellent connection |||||
   * level 4: 10-19  -- good connection ||||
   * level 3: 20-29  --  connection |||
   * level 2: 30-39  --  weak ||
   * level 1: 40-49  --  very weak |
   * level 0: >=50    -- out of area
   *
   * External signals:
   * incoming call - 1
   * text message  - 2
   * nothing       - 0
   */

  public void run()
  {

    System.out.println( " NetworkInterface run()" );

    //first set to non-exist values

    int temp_availability = 100;

    int temp_inSignal = 100;

    while ( powerOn ) //original
    {
        //System.out.println( powerOn );
        //System.out.println( "checking connection ... " );

        switch ( state )
        {
           case IDLE:

              if ( temp_inSignal != incomingSignal ) //original
              //if ( temp_inSignal == incomingSignal ) //fault5 -found
              {
                 temp_inSignal = incomingSignal;

                 if ( incomingSignal == 1)
                 {
                     System.out.println( "incoming call at connection level " + connectionLevel );

                     HandsetController.incomingCallArrived();
                     try
                     {
                        Thread.sleep(50);
                     }
                     catch ( Exception e )
                     {
                        e.printStackTrace();
                     }
                     //waiting = true;
                     state = WAITING;
                     System.out.println( " network interface waiting" );
                 }

                 else if ( incomingSignal == 2)
                 {
                  HandsetController.textMessageArrived();
                 }
              }

              //System.out.println( "In network: " + connectionLevel );
              break;

           case WAITING:

              //System.out.println( " network interface waiting" );
              while ( state == WAITING )
              {
                 //System.out.println( " network interface: waiting state" );
              }

/**
              if ( connected )
              {
                 state = 3;
```

```
               }
               else
               {
                  state = 0;
               }
**/
            case CONNECTING:

               //System.out.println( " network interface connecting" );
               while ( state == CONNECTING )
               {
               }
/**
               if ( connected )
               {
                  state = CONNECTED;
               }
               else
               {
                  state = IDLE;
               }
**/
               break;
            case CONNECTED:

               //System.out.println( " network interface connected" );
               while ( state == CONNECTED )
               {
                  if ( connectionLevel == 0 )
                  {
                     state = IDLE;
                  }
               }
               //state = 0;
               break;

         }

     // System.out.println(" running");

     }

  }

  public int getAvailability()
  {
     System.out.println( " NetworkInterface getAvailability()" );
     return availability;
  }

  public int getIncomingSignal()
  {
     if ( instrumented )
     {
        System.out.println( " NetworkInterface getIncomingSignal()" );
     }
     return incomingSignal;
  }

  public int getConnectionLevel()
  {
     if ( instrumented )
     {
        System.out.println( " NetworkInterface getConnectionLevel()" );
     }

     return connectionLevel;
  }

  public void stop()
  {
     if ( instrumented )
     {
        System.out.println( " NetworkInterface stop()" );
     }
```

```
/**
     try
     {
        timer.t.join();
     }
     catch ( InterruptedException e )
     {
        e.printStackTrace();
     }
**/
     state = IDLE;
     powerOn = false;
     pt.cancel();
     timer.cancel();

  }


  public void start()
  {
     if ( instrumented )
     {
        System.out.println( " NetworkInterface start()" );
     }

     t = new Thread( this );
     t.setName( "Network" );
     powerOn = true; //original
     state = IDLE;
     //connected = false;
     //waiting = false;
     //connecting = false;
     pre_conLevel = 100;
     //powerOn = false; //fault2  -- not found (concurrency)
     //t.start(); //fault3 -- missing statement -found
     timer = new Timer();
     pt = new PeriodicTasks();
     timer.schedule( pt, 0, 15000 );
     t.start();

  }


  public boolean makeConnection( String phoneNumber )
  {
     Random random = new Random();

     if ( state == IDLE )
     {
        state = CONNECTING;
     }

     System.out.println( " network interface connecting" );

     if ( instrumented )
     {
        System.out.println( " NetworkInterface makeConnection()" );
     }

     boolean connected;

     System.out.println( "dialing " + phoneNumber );
     System.out.println( "connection level: " + connectionLevel );

     availability = random.nextInt( 2 );

     if ( availability == 1 && connectionLevel > 0 )
     {
         connected = true;
         state = CONNECTED;
         System.out.println( " network interface connected" );
     }

     else
     {
        connected = false;
        state = IDLE;
```

```
         System.out.println( " network interface idle" );
      }

      try
      {
         Thread.sleep( 1000 );
      }
      catch ( InterruptedException e )
      {
         e.printStackTrace();
      }

      return connected;
   }

   /**
    * This method is written for the testing purpose
    */

   public boolean makeConnection( String phoneNumber, int availability )
   {
//     Random random = new Random();

      if ( state == IDLE )
      {
         state = CONNECTING;
      }

      System.out.println( " network interface connecting" );

      if ( instrumented )
      {
         System.out.println( " NetworkInterface makeConnection()" );
      }

      boolean connected;

      System.out.println( "dialing " + phoneNumber );
      System.out.println( "connection level: " + connectionLevel );

 //    availability = random.nextInt( 2 );

      if ( availability == 1 && connectionLevel > 0 )
      {
          connected = true;
          state = CONNECTED;
          System.out.println( " network interface connected" );
      }

      else
      {
         connected = false;
         state = IDLE;
         System.out.println( " network interface idle" );
      }

      try
      {
         Thread.sleep( 1000 );
      }
      catch ( InterruptedException e )
      {
         e.printStackTrace();
      }
      return connected;
   }


   static void checkIncomingSignal()
   {
      Random random = new Random();

      if ( state == 0 )
      {
         if ( connectionLevel > 0 )
         {
             incomingSignal = random.nextInt( 3 ); //original
```

```
            //incomingSignal = 1;
        }
        else
        {
           //availability = 0;
           incomingSignal = 0;
        }
     }

     System.out.println( "incoming signal is: " + state + " " + incomingSignal );
  }

/**
 * This method is for testing
 */

  static void checkIncomingSignal( int signal )
  {

     if ( state == 0 )
     {
        if ( connectionLevel > 0 )
        {
            incomingSignal = signal;
        }

        else
        {
           //availability = 0;
           incomingSignal = 0;
        }
     }
  }


  static void checkConnectionLevel()
  {
     Random random = new Random();

     distance = random.nextInt( 60 );

     System.out.println( " current distance from cell tower: " + distance );

     if ( distance < 10 )
     {
         connectionLevel = 5;
     }

     else if ( distance < 20 )
     {
        connectionLevel = 4;
     }

     else if ( distance < 30 )
     {
        connectionLevel = 3;
     }

     else if ( distance < 40 )
     {
        connectionLevel = 2;
     }

     else if ( distance < 50 )
     {
        connectionLevel = 1;
     }

     else
     {
        connectionLevel = 0;
     }

      if ( pre_conLevel != connectionLevel )
      {
          pre_conLevel = connectionLevel;
          HandsetController.connectionLevelChanged( connectionLevel );
```

```
    }
}

static void checkConnectionLevel( int distance )
{

   System.out.println( " current distance from cell tower: " + distance );

   if ( distance < 10 )
   {
       connectionLevel = 5;
   }

   else if ( distance < 20 )
   {
      connectionLevel = 4;
   }

   else if ( distance < 30 )
   {
      connectionLevel = 3;
   }

   else if ( distance < 40 )
   {
      connectionLevel = 2;
   }

   else if ( distance < 50 )
   {
      connectionLevel = 1;
   }

   else
   {
      connectionLevel = 0;
   }

    if ( pre_conLevel != connectionLevel )
    {
        pre_conLevel = connectionLevel;
        HandsetController.connectionLevelChanged( connectionLevel );
    }
}

public void endConnection()
{
   state = IDLE;
   //connected = false;
   //waiting = false;
   //connecting = false;
}

public void connected()
{

   state = CONNECTED;
   //connected = true;
   //waiting = false;
}

public void timeout()
{
   state = IDLE;
   //connected = false;
   //waiting = false;
   //connecting = false;
}

void   externalHangup()
{
   state = IDLE;
   //connected = false;
}

 public void printStateInfo()
 {
```

```
      System.out.print( "networkInterface state = " );

      if ( state == IDLE )
      {
         System.out.println( "IDLE" );
      }

      else if ( state == WAITING )
      {
         System.out.println( "WAITING" );
      }

      else if ( state == CONNECTING )
      {
         System.out.println( "CONNECTING" );
      }

      else if ( state == CONNECTED )
      {
         System.out.println( "CONNECTED" );
      }

   }

  /**
  public static void main( String[] args )
  {
     new HandsetController();
     NetworkInterface ni = new NetworkInterface();
     ni.start();
      System.out.println("test started");
     powerOn=false;
     powerOn=true;
     //powerOn=false;
     distance=0;
     distance=0;
     System.out.println("test is done");
  }
  */
}

package cellphone;

/**
 * Title:        Source file for class Receiver
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class Receiver implements Runnable
{
  Thread t;
  private volatile boolean turnOn;
  AudioSample sample;

  boolean receive, instrumented;

  public Receiver( AudioSample sample )
  {
     this.sample = sample;
     t = new Thread( this, "Receiver" );
     //new Thread( this, "Receiver" ).start();

     instrumented = true;

     if ( instrumented )
     {
        System.out.println( " Receiver constructor" );
     }
  }

  public Receiver()
  {
     t = new Thread( this, "Receiver" );
```

```
   instrumented = true;

   if ( instrumented )
   {
      System.out.println( " Receiver constructor" );
   }
}

public void run()
{
   if ( instrumented )
   {
      System.out.println( " Receiver run()" );
   }

   int i = 0;

   while ( turnOn )
   {
      while ( receive )
      {
         System.out.println( "Receiver receiving: " + sample.get() );
      }

      try
      {
         t.sleep( 10000 );
      }

      catch ( Exception e )
      {
         e.printStackTrace();
      }
   }
}


public void turnOn()
{
   if ( instrumented )
   {
      System.out.println( " Receiver start()" );
   }

   turnOn = true;
   t.start();
}

public void turnOff()
{
   if ( instrumented )
   {
      System.out.println( " Receiver stop()" );
   }

   turnOn = false;
   try
   {
      t.join();
   }
   catch ( InterruptedException e )
   {
      e.printStackTrace();
   }
}

public void startReceive()
{
   if ( instrumented )
   {
      System.out.println( " Receiver startReceive()" );

   }

   receive = true;
}
```

```
  public void stopReceive()
  {
     if ( instrumented )
     {
        System.out.println( " Receiver stopReceive()" );
     }

     receive = false;
  }

  public void setSample( AudioSample sample )
  {
     this.sample = sample;
  }

   public void printStateInfo()
   {
      System.out.println( "turnOn = " + turnOn );
      System.out.println( "receive = " + receive );
   }

}

package cellphone;

/**
 * Title:        Source file for class Transmitter
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class Transmitter implements Runnable
{
  Thread t;
  private volatile boolean turnOn;
  AudioSample sample;

  private volatile boolean transmit, instrumented;

  public Transmitter( AudioSample sample )
  {
     this.sample = sample;
     t = new Thread( this, "Transmitter" );
     //new Thread( this, "Transmitter" ).start();

     instrumented = true;

     if ( instrumented )
     {
        System.out.println( " Transmitter constructor" );
     }
  }

  public Transmitter()
  {
     t = new Thread( this, "Transmitter" );

     instrumented = true;

     if ( instrumented )
     {
        System.out.println( " Transmitter constructor" );
     }
  }

  public void run()
  {

     if ( instrumented )
     {
        System.out.println( " Transmitter run()" );
     }

     while ( turnOn )
```

```
      {
         while ( transmit )
         {
            System.out.println( "Transmitter transmitting: " + sample.get() );
         }

         try
         {
            t.sleep( 10000 );
         }

         catch ( Exception e )
         {
            e.printStackTrace();
         }
      }
   }

   public void turnOn()
   {
      if ( instrumented )
      {
         System.out.println( " Transmitter start()" );
      }

      turnOn = true;
      t.start();
   }

   public void turnOff()
   {
      if ( instrumented )
      {
         System.out.println( " Transmitter stop()" );
      }

      turnOn = false;
   }

   public void startTransmit()
   {
      if ( instrumented )
      {
         System.out.println( " Transmitter startTransmit()" );
      }

      transmit = true;
   }

   public void stopTransmit()
   {
      if ( instrumented )
      {
         System.out.println( " Transmitter stopTransmit()" );
      }

      transmit = false;
   }

   public void setSample( AudioSample sample )
   {
      this.sample = sample;
   }

    public void printStateInfo()
    {
       System.out.println( "turnOn = " + turnOn );
       System.out.println( "transmit = " + transmit );
    }
}

package cellphone;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
```

```
/**
 * Title:        Source file for class UserInterface
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class UserInterface extends JFrame implements  ActionListener
{
  //Thread t;
  String phoneNumber;
  String message1 = new String( "Connection Status:  " );
  String centerSpace = new String( "                " );

  static final int ON = 1;
  static final int OFF = 0;
  int cellPhone;
  int connectionLevel;
  String command;
  String button;

  boolean waitingForEvents;

  JPanel contentPane;
  JButton number4 = new JButton();
  JButton number6 = new JButton();
  JButton number3 = new JButton();
  JButton number1 = new JButton();
  JButton number2 = new JButton();
  JButton number5 = new JButton();
  TitledBorder titledBorder1;
  Border border1;
  Border border2;
  JButton poundButton = new JButton();
  JButton number7 = new JButton();
  JButton number8 = new JButton();
  JButton starButton = new JButton();
  JButton number9 = new JButton();
  JButton number0 = new JButton();
  JButton talkButton = new JButton();
  JButton endButton = new JButton();
  JButton menuButton = new JButton();
  JButton okButton = new JButton();
  JButton upButton = new JButton();
  JButton leftButton = new JButton();
  JButton rightButton = new JButton();
  JButton onButton = new JButton();
  JButton offButton = new JButton();
  JSlider volumeSlider = new JSlider();
  JButton downButton = new JButton();
  JTextArea connectionStatus = new JTextArea();
  JTextArea message = new JTextArea();
  JTextArea empty = new JTextArea();
  JToggleButton onOffToggleB = new JToggleButton();
  Border border3;
  Border border4;


  public UserInterface( String consType )
  {
     System.out.println( " UserInterface constructor for testing " );

     cellPhone = OFF;
     phoneNumber = new String();
     waitingForEvents = false;
  }

  public UserInterface()
  {
     System.out.println( " UserInterface constructor" );

     cellPhone = OFF;
     phoneNumber = new String();
     waitingForEvents = false;
```

```
   enableEvents(AWTEvent.WINDOW_EVENT_MASK);
   try
   {
     jbInit();
     //t = Thread.currentThread();
   }
   catch(Exception e)
   {
     e.printStackTrace();
   }
}

public static void main(String[] args)
{
  System.out.println( " UserInterface main()" );
  UserInterface ui = new UserInterface();
}
private void jbInit() throws Exception
{

  System.out.println( " UserInterface jbInit()" );

  contentPane = ( JPanel )this.getContentPane();
  //contentPane = new JPanel();
  titledBorder1 = new TitledBorder(BorderFactory.createEtchedBorder(Color.white,new Color(148, 145, 140)),"Cell Phone Handset");
  border1 = new TitledBorder(BorderFactory.createMatteBorder(3,0,0,0,Color.pink),"Cell Phone Handset");
  border2 = new TitledBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED,Color.red,Color.red,new Color(178, 0, 0),new Color(124, 0,
  border3 = new EtchedBorder(EtchedBorder.RAISED,Color.white,Color.pink);
  border4 = BorderFactory.createEtchedBorder(Color.white,Color.pink);
 // contentPane.setBorder(titledBorder1);
  //contentPane.setLayout( new BorderLayout() );
  number4.setToolTipText("");
  number4.setText("4");
  number4.setBounds(new Rectangle(93, 219, 42, 43));
  number4.addActionListener( this );
  number6.setBounds(new Rectangle(179, 219, 42, 43));
  number6.setToolTipText("");
  number6.setText("6");
  number6.addActionListener( this );
  number3.setBounds(new Rectangle(179, 176, 42, 43));
  number3.setText("3");
  number3.addActionListener( this );
  number1.setBounds(new Rectangle(93, 176, 42, 43));
  number1.setToolTipText("press for number 1");
  number1.setText("1");
  number1.addActionListener( this );
  number2.setBounds(new Rectangle(136, 176, 42, 43));
  number2.setText("2");
  number2.addActionListener( this );
  number5.setBounds(new Rectangle(136, 219, 42, 43));
  number5.setToolTipText("");
  number5.setText("5");
  number5.addActionListener( this );
  this.setState(Frame.ICONIFIED);
  this.setTitle("Cell Phone");
  this.setBackground(UIManager.getColor("ToolTip.background"));
  poundButton.setToolTipText("press to make a call");
  poundButton.setText("#");
  poundButton.setBounds(new Rectangle(179, 307, 42, 43));
  poundButton.addActionListener( this );
  number7.setToolTipText("");
  number7.setText("7");
  number7.setBounds(new Rectangle(93, 263, 42, 43));
  number7.addActionListener( this );
  number8.setText("8");
  number8.setBounds(new Rectangle(136, 263, 42, 43));
  number8.addActionListener( this );
  starButton.setText("*");
  starButton.setBounds(new Rectangle(93, 307, 42, 43));
  starButton.addActionListener( this );
  number0.setText("0");
  number0.setBounds(new Rectangle(136, 307, 42, 43));
  number0.addActionListener( this );
  number9.setBounds(new Rectangle(179, 263, 42, 43));
  number9.setText("9");
  number9.addActionListener( this );
```

```
 talkButton.setText("TALK");
 talkButton.setBounds(new Rectangle(93, 125, 64, 29));
 talkButton.addActionListener( this );
 endButton.setBounds(new Rectangle(158, 125, 64, 29));
 endButton.setText("END");
 endButton.addActionListener( this );
 menuButton.setText("MENU");
 menuButton.setBounds(new Rectangle(259, 207, 73, 29));
 menuButton.addActionListener( this );
 okButton.setToolTipText("");
 okButton.setText("OK");
 okButton.setBounds(new Rectangle(336, 207, 64, 29));
 okButton.addActionListener( this );
 upButton.setToolTipText("");
 upButton.setText("UP");
 upButton.setBounds(new Rectangle(298, 244, 64, 29));
 upButton.addActionListener( this );
 leftButton.setText("LEFT");
 leftButton.setBounds(new Rectangle(260, 278, 64, 29));
 leftButton.addActionListener( this );
 rightButton.setToolTipText("");
 rightButton.setText("RIGHT");
 rightButton.setBounds(new Rectangle(331, 277, 73, 29));
 rightButton.addActionListener( this );
 volumeSlider.setBounds(new Rectangle(137, 374, 200, 16));
 //volumeSlider.addActionListener( this );
 downButton.setBounds(new Rectangle(299, 314, 73, 29));
 downButton.setText("DOWN");
 downButton.addActionListener( this );

 connectionStatus.setBorder(border3);
 connectionStatus.setText(message1);
 connectionStatus.setBounds(new Rectangle(102, 50, 258, 26));
 message.setBorder(border3);
 message.setText("");
 message.setBounds(new Rectangle(102, 74, 258, 26));

 onButton.setText("ON");
 onButton.setBounds(new Rectangle(278, 125, 52, 29));
 onButton.addActionListener( this );
 offButton.setText("OFF");
 offButton.setBounds(new Rectangle(330, 125, 58, 29));
 offButton.addActionListener( this );

 empty.setEnabled(false);
 empty.setEditable(false);
 contentPane.add(number3, null);
 contentPane.add(number2, null);
 contentPane.add(number1, null);
 contentPane.add(number4, null);
 contentPane.add(number7, null);
 contentPane.add(number8, null);
 contentPane.add(number5, null);
 contentPane.add(number9, null);
 contentPane.add(number6, null);
 contentPane.add(number0, null);
 contentPane.add(starButton, null);
 contentPane.add(poundButton, null);
 contentPane.add(talkButton, null);
 contentPane.add(endButton, null);
 contentPane.add(volumeSlider, null);
 contentPane.add(connectionStatus, null);
 contentPane.add(message, null);
 contentPane.add(downButton, null);
 contentPane.add(leftButton, null);
 contentPane.add(rightButton, null);
 contentPane.add(upButton, null);
 contentPane.add(menuButton, null);
 contentPane.add(okButton, null);
 contentPane.add(onButton, null);
 contentPane.add(offButton, null );
 contentPane.add(empty, null );
 this.repaint();
}

 private void setPhoneNumber( String number )
 {
```

```
      System.out.println( " UserInterface setPhoneNumber()" );
      phoneNumber = phoneNumber + number;
      message.setText( phoneNumber );
      this.repaint();
   }

   public String getPhoneNumber()
   {
      System.out.println( " UserInterface getPhoneNumber()" );
      message.setText( "            dialing " + phoneNumber );
      this.repaint();
      return phoneNumber;

   }

/**
   public void setIncomingSignal( int inSignal )
   {
      System.out.println( " UserInterface setIncomingSignals()" );
      String incomingSignal;

      if (inSignal == 0 )
      {
         incomingSignal = new String("");
      }

      else if ( inSignal == 1 )
      {
         incomingSignal = new String( " incoming call ");

      }

      else
      {
         incomingSignal = new String( " text message " );
      }

      message.setText( incomingSignal );
      this.repaint();
   }


   public void setConnectionMessage( String conMessage )
   {
      System.out.println( " UserInterface setConnectionMessage()" );
      message.setText( conMessage );
      this.repaint();
   }
**/

   public void setConnectionLevel( int conLevel )
   {
      System.out.println( " UserInterface setConnectionLevel()" );
      String displayMessage;
      String externalSignalMessage = new String(" ");
      int connectionLevel = conLevel;

      try
      {
            if ( connectionLevel > 0 )
            {
               System.out.println( connectionLevel );

               displayMessage = message1;
               for ( int i = 1; i <= connectionLevel; i++ )
               {
                  displayMessage = displayMessage + "|";
               }
           }

            else
            {
               displayMessage = message1 + "out of area";
            }

//            System.out.println( displayMessage );
            connectionStatus.setText( displayMessage );
```

```
         this.repaint();

   }
   catch(Exception e)
   {
     e.printStackTrace();
   }
}

public void actionPerformed( ActionEvent ae )
{
   System.out.println( " UserInterface actionPerformed( ActionEvent e)" );

   String command = ae.getActionCommand();
   System.out.println( command );

   if ( cellPhone == OFF && command.equals( "ON" ) )
   {
      cellPhone = ON;
      waitingForEvents = true;
      button = "ON";
      //HandsetController.buttonPressed( button );
   }

   else if ( cellPhone == ON && command.equals( "OFF" ) )
   {
      cellPhone = OFF;
      waitingForEvents = false;
      button = "OFF";
      connectionStatus.setText( message1 );
      message.setText( "" );
      this.repaint();
      //HandsetController.buttonPressed( button );
   }

   else if ( cellPhone == ON && waitingForEvents && command.equals( "#" ) )
   {
      phoneNumber = new String();
      waitingForEvents = false;
      button = "#";
      message.setText( "       Enter phone number"  );
      this.repaint();
   }

   else if ( cellPhone == ON && waitingForEvents && command.equals( "END" ) )
   {
      waitingForEvents = true;
      button = "END";
      message.setText( "              " + button );
      this.repaint();
   }

   else if ( cellPhone == ON && waitingForEvents && command.equals( "TALK" ) )
   {
      waitingForEvents = true;
      System.out.println( message.getText() + message.getText().length() ) ;
      if ( ( message.getText().trim() ).equals( "incoming call" ) )
      {
         button = "TALK";
      }

      else
      {
         button = " ";
      }

      message.setText( "              " + button );
      this.repaint();
   }

   else if ( cellPhone == ON && !waitingForEvents && command.equals( "TALK" ) )
   {
      button = "TALK";
      waitingForEvents = true;
   }

   else if ( cellPhone == ON && !waitingForEvents && command.equals( "END" ) )
```

```
      {
         waitingForEvents = true;
         button = " ";
         message.setText( "              " + button );
         this.repaint();
      }

      else if ( cellPhone == ON && command.equals( "MENU" ) )
      {
         button = "MENU";
      }

      else if ( cellPhone == ON && command.equals( "OK" ) )
      {
         button = "OK";
      }

      else if ( cellPhone == ON && command.equals( "UP" ) )
      {
         button = "UP";
      }

      else if ( cellPhone == ON && command.equals( "DOWN" ) )
      {
         button = "DOWN";
      }

      else if ( cellPhone == ON && command.equals( "RIGHT" ) )
      {
         button = "RIGHT";
      }

      else if ( cellPhone == ON && command.equals( "LEFT" ) )
      {
         button = "LEFT";
      }

      else if ( cellPhone == ON && command.equals( "*" ) )
      {
         button = "*";
      }

      else if ( ( cellPhone == ON ) && ! waitingForEvents )
      {
         setPhoneNumber( command );
      }

      if ( cellPhone == ON )
      {
         HandsetController.buttonPressed( button );
      }

/**
      //      if ( ( cellPhone == OFF & button.equals("ON") ) ||
//           cellPhone == ON & ( button.equals( "OFF" ) ||
//                               ( waitingForEvents & ( message.equals("incomingCall") || button.equals("END") || button.equals("#"))) ||
//                                !waitingForEvents & button.equals("TALK") ))
//      {
//         HandsetController.buttonPressed( button );
//      }

      if ( cellPhone == OFF )
      {
         if ( button.equals( "ON" ) )
         {
            HandsetController.buttonPressed( button );
         }
      }

      else
      {
         if ( button.equals( "OFF" ) )
         {
            HandsetController.buttonPressed( button );
         }

         if ( waitingForEvents )
```

```
         {
            if ( button.equals( "#" ) || button.equals(" END" ) ||
            button.equals( "TALK" ) && message.equals( "incomingCall" ) )
            {
                HandsetController.buttonPressed( button );
            }
         }

         else
         {
            if ( button.equals( "TALK" )
            {
               HandsetController.buttonPressed( button );
            }
         }

      }

**/
   }


   public void clear()
   {
      message.setText( "" );
      button = " ";
      this.repaint();
   }

   public void displayMessage( String controllerMessage )
   {
      message.setText( controllerMessage );
      this.repaint();
   }

   public void printStateInfo()
   {
      System.out.print( "cellPhone=" );

      if ( cellPhone == OFF )
      {
         System.out.println( "OFF" );
      }

      else
      {
         System.out.println( "ON" );
      }

      System.out.println( "waitingForEvents=" + waitingForEvents );
   }
}

package cellphone;

/**
 * Title:        Source file for class AudioSample
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class AudioSample
{
  int i, j;
  boolean valueSet = false;
  String deviceName;

  boolean instrumented;

  public AudioSample( String dname )
  {
     if ( dname.equals( "Receiver" ) )
     {
        deviceName = new String( "Receiver" );
```

```
     }

     else
     {
        deviceName = new String( "Transmitter" );
     }

     instrumented = true;

     if ( instrumented )
     {
        System.out.println( " AudioSample constructor" );
     }

     i = 0;
     j = 0;
  }

  int get()
  {
     if ( instrumented )
     {
        System.out.println( " AudioSample get()" );
     }

     System.out.println( deviceName + " getting sample ... " );
     valueSet = false;
     return ( deviceName.equals( "Receiver" ) )?i++:j++;
  }
}

package cellphone;

import java.util.*;


/**
 * Title:        Source file for class Periodic Tasks
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      George Mason University
 * @author Aynur Abdurazik
 * @version 1.0
 */

public class PeriodicTasks extends TimerTask
{

  public PeriodicTasks()
  {
  }
  public void run()
  {
    /**@todo: implement this java.util.TimerTask abstract method*/
     NetworkInterface.checkConnectionLevel();
     NetworkInterface.checkIncomingSignal();
  }
  public static void main(String[] args)
  {
    PeriodicTasks periodicTasks1 = new PeriodicTasks();
  }
}
```

D Appendix: Faulty versions

Not yet available electronically.