社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

JMLを用いたホームネットワークシステムにおける連携サービスの安全性検証に関する提案

閻　　奔† 　中村　匡秀† 　リディドゥ ブスケ†† 　松本　健一†

† 奈良先端科学技術大学院大学・情報科学研究科 〒 630-0192 奈良県生駒市高山町 8916-5
†† ヨセフ・フーリエ大学（グルノーブル第 1 大学），フランス
E-mail: †{hon-e,masa-n,matumoto}@is.naist.jp, ††lydie.du-bousquet@imag.fr

あらまし　家電機器やセンサを家庭内ネットワークに接続し，様々なサービスを提供可能にするホームネットワークシステム（HNS）の研究・開発が盛んである． HNS 連携サービスの開発・提供にあたっては，そのサービスがユーザや家財に対して安全であることを保証しなくてはならない．しかしながら，連携サービスの開発・実装においては，従来家電を単体で使用する場合と比べ，より慎重かつ綿密な安全対策が必要となる．ソフトウェアのバグやロジック誤りが，深刻な事故や被害を引き起こす可能性がある．本稿では，JML（Java Modeling Language）を用いて、HNS における 3 種類の安全性性質を検証する方法を提案する. さらに、実際の連携サービスの通用実験を行う．
キーワード　ホームネットワーク，連携サービス，安全性性質，モデル，JML，JUnit, 契約による設計, テストケース

Describing and Verifying Safety in Integrated Services of Home Network System by JML Considering Safety

Ben YAN†, Masahide NAKAMURA†, Lydie DU BOUSQUET††, and Ken-ichi MATSUMOTO†

† Nara Institute of Science and Technology 8916-5 Takayama-cho, Ikoma-shi, Nara, 630-0192 Japan
†† LSR Laboratory, IMAG, Joseph Fourier University (Grenoble I), BP72 F-38402 Saint-Martin d'Heres Cedex France
E-mail: †{hon-e,masa-n,matumoto}@is.naist.jp, ††lydie.du-bousquet@imag.fr

Abstract　The recent ubiquitous technologies allow home electric appliances and sensors to be connected with local area network at home. While when we develop such HNS (Home Network System) integrated services and provide them to the users, we must assure that the integrated services are safe for the users, the appliances and the environments. In the normal operations of these appliances are very different from the operations of the HNS, so the development and the implementation of the HNS integrated service need more careful and exact safety measures for the safe operations of the appliances. There is a possibility that the bug and the logic mistake of software cause an accident and serious damage. In order to solute this problem, we present a framework to verify the safety of integrated service of HNS by using the DBC (Design by Contract) and JML (Java Modeling Language)
Key words　Home Network System, Integrated services, Safety, Model, JML, JUnit, Design by Contract, Test Case

1. はじめに

ユビキタス技術の進歩により，ホームネットワークシステム (HNS) の研究・開発が注目されている．HNS のひとつのアプリケーションとして，複数のネット家電やセンサを連携し，便利で付加価値のある機能を提供する **HNS 家電連携サービス** (以下，連携サービス) がある [6], [7]．連携サービスの開発・提供にあたっては，そのサービスがユーザや家財に対して**安全**であることを保証しなくてはならない．連携サービスの場合 (a) 操作者が人間のユーザではなくソフトウェアであること，(b) 1 つの連携サービスは複数の家電を組み合わせて使用すること，(c) 複数の連携サービスが同時に実行されうること，といった特徴がある．これらの理由から，連携サービスの開発・実装においては，従来家電を単体で使用する場合と比べ，より慎重かつ綿密な安全対策が必要となる．ソフトウェアのバグやロジック誤りが，深刻な事故や被害を引き起こす可能性がある．

この問題に対処すべく，我々は先行研究 [10] において，HNS および連携サービスの安全性検証の枠組みを提案している．この枠組みでは，連携サービスに対して 3 種類の安全性（ローカル安全性，グローバル安全性，環境安全性）を定式化している．また，HNS 内の家電，サービス，家屋をそれぞれオブジェクトとしてモデル化し，契約による設計 (Design by Contract, DbC) による安全性検証を行うガイドラインを提案している．

本稿では，Java プログラム上で DbC 検証を可能とする言語，JML (Java Modeling Language) を用いて，上記 3 種類の安全性を検証するための，具体的な手順の実装を目指す．さらに，実際の連携サービスの通用実験を行う．

2. 安全性検証提案の概要

本章では，我々が先行研究 [10] で提案した HNS 連携サービスの安全性検証のガイドラインを概観する．

2.1 ホームネットワークシステム安全性

HNS におけるサービスが安全であるとは，そのサービスが，住人，設備，環境，家財，近隣住人・環境に対して，死亡，けが，破損，損害を生じうるいかなる条件からも脱却していることである．高い安全性を確保するために，我々は [10] において，HNS およびサービスが遵守すべき 3 種類の**安全性性質** (safety properties) を定義した [2](表 1).

表 1　HNS 安全性性質分類

種類	定義
ローカル安全性性質	電機器を安全に使用するための注意事項．各家電で閉じた性質.
グローバル安全性性質	複数の機器をまたがった大域的な使用ルールや望ましい振舞い．サービスに依存する.
環境安全性性質	HNS が設置される住居における取り決めや運用ルール．家電やサービスに非依存.

2.2 安全性性質の抽出

2.1 で述べた各安全性性質は，表 2 に示すとおり抽出・決定される．各種類ごとに，参照する事項や決定する責任者が異なることに注意されたい．

表 2　安全性抽出基準

安全性種類	抽出基準
ローカル安全性性質	家電マニュアルから抽出する
グローバル安全性性質	サービス開発者によって慎重に決定
環境安全性性質	建物の使用説明書や非常用マニュアル，自治会や町内会で決定

2.3 安全性の定式化

与えられた連携サービス s の安全性検証問題は以下のように定式化される．

- $App(s) = \{d_1, d_2, ..., d_n\}$ を s が使用するネットワーク家電の集合,
- $LocalProp(di) = \{lp_{i1}, lp_{i2}, ..., lp_{im}\}$ を家電 $d_i \in App(s)$ のローカル安全性性質の集合,
- $LocalProp(s) = \cup_{d_i \in App(s)} LocalProp(d_i)$,
- $GlobalProp(s) = \{g_{p1}, g_{p2}, ..., g_{pk}\}$ を s で規定されたグローバル安全性性質の集合,
- $EnvProp(s) = \{e_{p1}, e_{p2}, \cdots, e_{pl}\}$ を s が提供される環境の安全性性質の集合とする．このとき，**安全性検証問題**とは,

入力：ホームネットワークシステム hns，連携サービス s, $LocalProp(s)$, $GlobalProp(s)$, $EnvProp(s)$.

出力：hns 上で s が安全かどうかの判定.

と定義される．ここで,

- s が $LocalProp(s)$ の全ての安全性性質を満たすとき，s は**ローカル安全**であるという.
- s が $GlobalProp(s)$ の全ての安全性性質を満たすとき，s は**グローバル安全**であるという.
- s が $EnvProp(s)$ の全ての安全性性質を満たすとき，s は**環境安全**であるという.

また，s がローカル安全かつグローバル安全かつ環境安全であるとき，s は**安全**であるという.

2.4 オブジェクト指向 HNS モデリング

安全性検証問題を実施するために，HNS および連携サービスのモデルを提案している [4], [10]（図 1）．本モデルは 3 つのオブジェクトから構成される.

- Appliance：家電機器オブジェクト.
- Service：連携サービスオブジェクト.
- Home：家（環境）オブジェクト.

さらに，これらオブジェクト間の関係は次のとおりである.

- 各 Home は，複数の Appliance を持つ.
- 各 Home は，複数の Service を持つ.
- 各 Service は，複数の Appliance を使う.

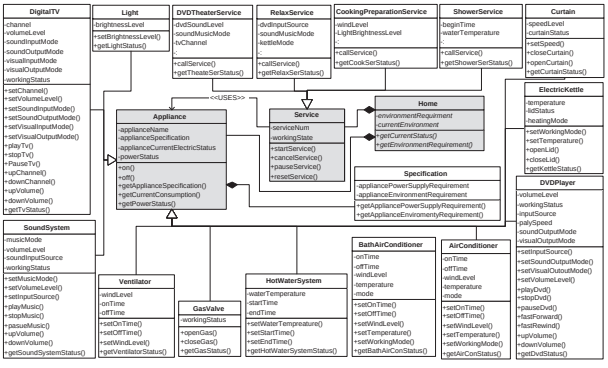

図 1　HNS のオブジェクト指向モデリング

2.5 DbC による安全性検証

安全性検証を実施するために，我々は**契約による設計** (Design by Contract, 以降 DbC) [3], [5]，と呼ばれる実装戦略を採る．DbC における契約には，以下の 3 種類が存在する.

- **事前条件（Pre-Condition）：**ある操作（メソッド）を実行するために，最低限必要な条件を定めたもの.
- **事後条件（Post-Condition）：**ある操作（メソッド）を

実行した後に，成立していなければならない条件を定めたもの．

- **クラス不変表明（**Class Invariant**）：**あるクラスについて，どのような操作を行っても常に満たされているべき条件を定めたもの．

我々のキーアイデアは，2.1 の各安全性性質を DbC の契約としてプログラムに埋め込み，プログラムの**テスト**によって安全性性質を検証することである．ローカル，グローバル，環境の各安全性性質は，それぞれ，2.4 で述べた Appliance, Service, Home オブジェクト内の契約としてエンコードされる．プログラムの実行中，もし契約が破られた場合には，例外が投げられるかエラーが報告される．

3. JML による安全性検証

本章では，JML (Java Modeling Language) [8] を用いた具体的な安全性検証を実施する．JML は，Java 言語で書かれたシステムを DbC に基づいて検証するための枠組みを提供する言語である．いま，2.3 で述べた hns および s が，2.4 で述べたモデルに基づいて，Java 言語で実装されていると仮定する．

安全性検証は，以下の 4 つのステップにしたがって実施した．

Step1: 検証対象の選定.
Step2: 安全性性質の抽出.
Step3: 安全性性質の JML へのエンコード.
Step4: 検証対象に対応したテストケースの生成.
Step5: テストの実施.

3.1 Step1:検証対象の選定

今回の検証実験では，調理準備サービスおよび調理準備サービスで用いる家電群を対象として，安全性の検証を行う．

検証対象とする連携サービス

サービス名： 調理準備サービス
クラス名： `CookingPreparationService.class`
Java コード行数： 61 行
JML コード行数： 15 行
サービスシナリオ： ガスバルブ，換気扇，キッチン照明及び電気ポットを連携して，炊事準備を行うサービス．ユーザがサービスを起動すると，キッチンの照明が点灯し，ガスバルブが開いて，換気扇が回る，さらに，電気ポットを沸騰モードにして，お湯を用意する．
検証メソッド： `begin()`, `end()`

検証対象とする家電機器

家電名： ガスバルブ，換気扇，キッチン照明，電気ポット
クラス名： `GasValve.class`, `Ventilator.class`, `Light.class`, `ElectricKettle.class`

紙面の制限のため，以降 `ElectricKettle` のみを対象に説明をすすめる．

Java コード行数： 113 行
JML コード行数： 25 行
検証メソッド： `setWorkingMode()` , `setTemperture()`, `openLid()`, `closeLid()`

3.2 Step2:安全性性質の抽出

表 1, 2 に基づき，調理準備サービス (および各家電機器) が満たすべき，グローバル (およびローカル安全性) 安全性性質を抽出した．表 3 に抽出された今回の検証で用いる安全性性質の概要を示す．

表 3 テスト対象安全性性質

対象機器	ローカル安全性性質
`Light`, `ElectricKettle`	$L1$：交流 100V(±10) 以外のコンセントに接続してはいけない
`GasValve`, `Ventilator`	$L2$：定格 15A 以下のコンセントに接続してはいけない
`ElectricKettle`	$L3$：沸騰モードにする前に上蓋を確実に閉める
	$L4$：湯わかし中は上蓋を開けてはいけない
対象サービス	**グローバル安全性性質**
`CookingPreparationService`	$G1$：ガスバルブを開ける際には，換気扇を付けなければならない

3.3 Step3:安全性性質の JML へのエンコード

具体的な安全性性質を抽出した後，各性質を関連メソッドの中に埋め込む．性質は，JML 言語に従った Java の注釈 (コメント) 形式でエンコードされる．

```
import java.io.IOException;
public class ElectricKettle extends Appliance{
        private /*@spec_public@*/int temperture=0;                    // 1~100;
        private /*@spec_public@*/boolean heatingMode=false;   // heating->true, warm->false
        private /*@spec_public@*/boolean lidStatus=true;      // true->opened; flase->closed
  ......

        /*@     requires lidStatus==false;
          @             assignable  lidStatus, heatingMode;
          @             ensures lidStatus==false && heatingMode ==mode && \result==true ;
          @*/
        public boolean setWorkingMode(boolean mode){
                this.heatingMode=mode;
                return true;}

        /*@     requires   true;
          @             assignable  lidStatus, temperture;
          @             ensures lidStatus==false && temperture==temper && \result==true ;
          @*/
        public boolean setTemperture(int temper){......}

        /*@     requires this.heatingMode==false;
          @             modifies lidStatus;
          @             ensures lidStatus==true && \result==true;
          @*/
        public boolean openLid(){
                this.lidStatus=false;
                return true;}

        /*@     requires true;
          @             modifies lidStatus;
          @             ensures this.lidStatus==false && \result==true;
          @*/
         public boolean closeLid(){......}
     ......
}
```

図 2 ElectricKettle.class コード

図 2 は, `ElectricKettle.class` の各メソッドに対して，ローカル安全性性質を JML で記述した例である．例えば，図中 1 つ目のメソッド `setWorkingMode()` の上部の注釈は，表 3 の L3 を JML の契約として表現したものである．

メソッド： `ElectricKettle.setWorkingMode()`
事前条件： `requires lidStatus==false`
事後条件： `ensures lidStatus==false && heatingMode == mode && \ result==true`

`requires` で始まる行はそのメソッドの前条件を表す, `ensures` で始まる行はそのメソッドの後条件を表す．lidStatus は電気ポットの上蓋の状態を記述するクラスの属性である．lidStatus の値が true の場合は蓋が開いている状態を表す．false の場合

は閉まっている状態を表す．この契約によって，前条件では，メソッド実行前には上蓋を開けてはならないことを規定している．また，後条件では，メソッド実行後には上蓋を確実に閉めることを規定している．

```
public class CookingPreparationService {
   ......
        private  /*@ spec_public @*/Light light;
        private  /*@ spec_public @*/Ventilator ventilator;
        private  /*@ spec_public @*/GasValve gasValve;
        private  /*@ spec_public @*/ElectricKettle kettle;

        /*@   requires light!=null && ven!=null && gasValve!=null;
         @             ensures  this.light==light && this.gasValve==gasValve && this.ven==ven;
          @*/
        public  CookingPreparationService(GasValve gasValve,Light light, Ventilator
ventilator){ ......}


        /*@   requires light!=null && ventilator!=null && gasValve!=null && kettle!=null;
          @             assignable this.gasValve, this.ven;
          @             ensures  this.gasValve.getGasStatus()==true
          @                      && this.ventilator.getVentilatorStatus()==true && \result==true;
          @*/
        public boolean begin(){
                boolean lightResult1=this.light.on("light");
                boolean lightResult2=this.light.setBrightnessLevel(2);

                boolean ventilatorResult1=this.ven.on("ventilator");
                boolean ventilatorResult3=this.ven.setWindLevel(2);

                boolean gasValveResult1=this.gasValve.on("gas");
                boolean gasValveResult2=this.gasValve.openGas();

                boolean kettleResult1=this.kettle.on("kettle");
                boolean kettleResult2=this.kettle.setTemperture(100);
                boolean kettleResult3=this.kettle.setWorkingMode(true);

                System.out.println(this.ven.getVentilatorStatus());
                System.out.println(this.gasValve.getGasStatus());

                return true;}

        /*@   requires true;
          @             assignable this.gasValve, this.ven;
          @             ensures this.gasValve.getGasStatus()==false
          @                      && this.ven.getVentilatorStatus()==false && \result==true;
         @*/
         public boolean end(){ ......}
  ......}
```

図 3　CookingService.class コード

同様に，調理準備サービスのグローバル安全性 $G1$ は，CookingPreparationService.class の内部に図 3 のようにエンコードできる．これは，表 3 のグローバル安全性 G1 を JML でエンコードした例である．

メソッド：　CookingPreparationService.begin()

事前条件：　requires light!=null&& ventilator!=null && gasValve!=null　&& kettle!=null

事後条件：　ensures this.gasValve.getGasStatus()==true && this.ventilator.getVentilatorStatus()==true && \ result ==true

注釈コード中で，各家電の現時点の動作状態を各家電の状態取得メソッドで取得する．例えば，gasValve オブジェクトの動作状態を getGasStatus() メソッドに通じて取得する．契約 G1 によって，前条件では，サービスを開始する前に，使う電器のオブジェクトを生成しなければならないことを規定している．また，後条件では，ガスバルブを開ける際には，換気扇をつけなければならないことを規定している．

3.4　Step4: 検証対象に対応したテストケースの生成

テストの効率化のため，提案検証法では Java のテスト用フレームワーク JUnit [9] を用いる．したがって，本ステップではテストケースを JUnit の書式にあわせて生成する．

通常，テストケースの作成方法は，テストしたいメソッドに対して，様々な入力パラメータを設定し，実行したメソッドの戻り値と期待した戻り値を assert 文で比較することで，テストを行う．

しかしながら，我々の手法では，JML コンパイラ (後述) が assert に相当するチェックルーチンを自動的に生成する．テストが失敗，すなわち，安全性の契約が破られたときには，このルーチンが例外を投げる．よって，その例外をキャッチして適当なメッセージを出力するようにテストケースのコーディングを行う．具体的には，以下の手順に従ってテストケースのコーディングを行う．

- JUnit の TestCase クラスを継承して，テスト対象のクラスを定義する．
- コンストラクタ，main() メソッドを定義する．
- テスト対象に対してのテストしたいメソッドを記述したメソッド及び対応 JML 例外処理を定義する．

図 4 および 図 5 はそれぞれ，ElectricKettle.class，CookingPreparationService.class をテストするためのテストコードである．

```
import junit.framework.*;
public class TestKettle extends TestCase{
      private CookingPreparationService cookingService;
      public static void main(String arg[]){
                  junit.textui.TestRunner.run(TestKettle.class);
      }

      protected void setUp(){kettle=new ElectricKettle("Kettle",100,90,5,1,60,50,40,0,65,0,1,1) }

      public void testOpenLidMethod(){
         try{     this.kettle.openLid();
            }catch(org.jmlspecs.jmlrac.runtime.JMLEntryPreconditionError e$)
                  {System.out.println(e$.getMessage());}
                  catch(org.jmlspecs.jmlrac.runtime.JMLAssertionError e$){
                        int l$ = org.jmlspecs.jmlrac.runtime.JMLChecker.getLevel();
                org.jmlspecs.jmlrac.runtime.JMLChecker.setLevel(org.jmlspecs.jmlrac.runtime.JMLOption.NONE);
                        try { junit.framework.Assert.fail("\n\t" + e$.getMessage()); }
                        finally {
                            org.jmlspecs.jmlrac.runtime.JMLChecker.setLevel(l$);
                        }
            }
      }

      public void testCloseLidMethod(){ ..... }
      public void testSetWorkingModeMethod(){ ..... }
      public void testSetTempertureMethod(){ ..... }
.....}
```

図 4　ElectricKettle.class のテストコード

図 4 のコード中，testOpenLidMethod()，testCloseLidMethod()，testSetTemperatureMethod()，testSetWorkingModeMethod() のメソッドは，ElectricKettle.class 中の openLid()，closeLid()，setTemperature() 及び setWorkingMode() に対するテストメソッドである．例えば，testOpenLid() メソッドは，内部で openLid() を呼び出す．もし，JML の契約（すなわち安全性属性）が侵害されれば，テストが失敗し例外が返される．catch 節では，例外が投げられた場合のエラーメッセージを規定している．

同様に，図 5 のような CookingPreparationService.class に対してのテストケースを生成した．このテストコード中にの testCookingServiceBeginMethod() と testCookingServiceEndMethod() は CookingPreparationService.class 中の begin() と end() メソッドのテストメソッドである．

3.5　Step5: テストの実施

各オブジェクトに埋め込まれた JML 契約 (＝安全性性質) およびテストケースに基づいて，実際にテストを実施する．図 6 にテストの流れを示す．

先ず検証者は，Step3 で得られた JML 注釈付のソースコードを **JML コンパイラ**でコンパイルする．JML コンパイラはソースコード中の JML 注釈をチェックするためのチェックルーチンを含んだ Java バイトコードを自動生成する [1]．次に，Step4 でテストケースを Java コンパイラでコンパイルする．最後に，

```
import junit.framework.*;
public class TestCookingService extends TestCase{
      private GasValve gasValve;
      private Light light;
      private Ventilator ven;
      private CookingPreparationService cookingService;
      private ElectricKettle kettle;

      public static void main(String arg[]){
                  junit.textui.TestRunner.run(TestCookingService.class);
      }

      protected void setUp(){
                  light=new Light("Light",100,90,5,1,60,50,40,0,65,0,1,1);
                  ven=new Ventilator("Ventilator",100,90,5,1,60,50,40,0,65,0,1,1);
                  gasValve= new GasValve("GasValve",100,90,5,1,60,50,40,0,65,0,1,1);
                  kettle=new ElectricKettle("Kettle",100,90,5,1,60,50,40,0,65,0,1,1)
                  cookingService=new CookingPreparationService(gasValve,light,ven,kettle);
      }

      public void testCookingServiceBeginMethod(){
            try{  this.cookingService.begin();
            }catch(org.jmlspecs.jmlrac.runtime.JMLEntryPreconditionError e$)
                  {System.out.println(e$.getMessage());}
                  catch(org.jmlspecs.jmlrac.runtime.JMLAssertionError e$){
                   int l$ = org.jmlspecs.jmlrac.runtime.JMLChecker.getLevel();
      org.jmlspecs.jmlrac.runtime.JMLChecker.setLevel(org.jmlspecs.jmlrac.runtime.JMLOption.NONE);
                        try { junit.framework.Assert.fail("\n\t" + e$.getMessage()); }
                        finally {
                              org.jmlspecs.jmlrac.runtime.JMLChecker.setLevel(l$);
                        }
            }
      }
      public void testCookingServiceEndMethod(){......}
......}
```

図 5　CookingPreparationService.class のテストコード

JUnit を用いて，生成したテストケースをバイトコードに与えてテストを行う．図 A·1 および図 A·2 にテスト実施例を示す．

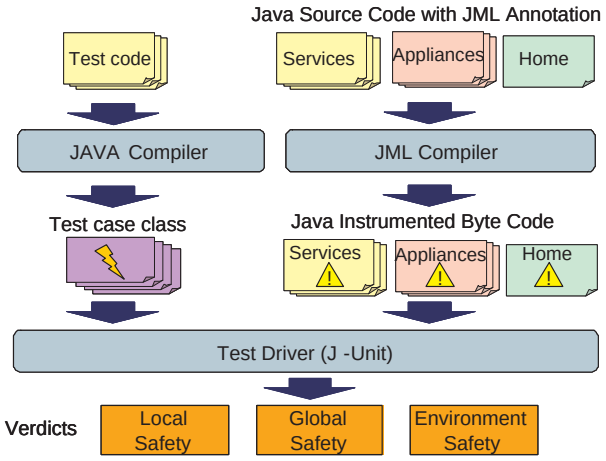

図 6　検 証 手 順

4. 検 証 結 果

今回のテストは以下の環境上で行った．

- CPU: 1GHZ
- RAM: 760MB
- OS: Microsoft Window XP Professional
- java: J2SDK 1.4.2.
- JML: JML tools release 5.3
- JUnit: JUnit3.8.1

4.1 ElectricKettle.class ローカル安全性検証実験結果

`ElectricKettle.class` 安全性検証実験は，クラス内の４つのメソッドが検証対象になった．結果を表 4 に示す．テストを通して二つの安全性性質の侵害が検出された．

失敗原因分析：

- `openLid()` 失敗原因:

`openLid()` メソッドを呼び出すと，電気ポットの上蓋を空け状態を開状態に設定しなければならない．しかしなから，コードを見ると上蓋の状態値の更新忘れのエラーが検出された．不具合修正後，再度テストすると成功した．

- `setWorkingMode()` 失敗原因:

「沸騰モードにする前に上蓋を確実に閉める」と言う安全性性質に違反した．つまり，`setWorkingMode()` メソッドを呼び出すとメソッド中に電気ポットの蓋の現状態をチェックしなければならない．今回のエラー原因は，本メソッド中にこのチェックルーチンの論理エラーから，上蓋が開いたまま電気ポットが沸騰状態になってしまうケースが存在した．

表 4　ElectricKettle.class ローカル安全性検証結果

項目名	対象名
テスト対象	ElectricKettle.class
テストケース	TestElectricKettle.class
テスト対象メソッド	openLid(), closeLid()
	setWorkingMode(), setTemperature()
テスト時間	0.665 seconds
openLid()	Failed
closeLid()	Passed
setWorkingMode()	Failed
setTemperture()	Passed

4.2 CookingPreparationService.class グローバル安全性検証テスト結果

`CookingPreparationService.class` 安全性検証実験では，クラス内の二つメソッドが検証対象となった．テストを通して，メソッド `begin()` に安全性性質の侵害が検出された．

表 5　CookingPreparationService.class グローバル安全性検証テスト結果

項目名	対象名
テスト対象	CookingPreparationService.class
テストケース	TestCookingService.class
テスト対象メソッド	begin(), end(),
テスト時間	0.48 seconds
begin()	Failed
end()	Passed

失敗原因分析：

- `begin()` 失敗原因:

本テストは「ガスバルブを開ける際には，換気扇を付けなければならない」と言う安全性性質違反が原因で失敗した．つまり，本サービスの begin() メソッドを呼び出しと「ガスバルブを開ける」と言う操作があるから，換気扇が起動しないと安全性性質に違反した．コードを見ると換気扇の主電源は ON したものの，ファンの起動メソッドが欠落していたことがわかった．

5. 終 わ り に

本稿では，[10] で提案したホームネットワークシステム（HNS）における連携サービスの安全性検証方法によって，安全性検証実験に関わる実験準備，実際の実験手順及び実験結果の分析についてが詳しく述べた．

まず，実験のために選定した HNS 連携サービス及びサービス使用する家電の安全性性質集合を抽出した．次に，JML を用いて各種類安全性性質集合を注釈した．そのうえで，テスト実験を行うために，選定したテスク対象におけるテストケースを作成した．最後に，JUnit を用いて，安全性検証テストの実験を行った．この実験テスト結果によって，HNS モデル CookingPreparationService.class 及び ElectricKettle.class 中に，幾つかの安全性に関わるバグを発見した．実験の結果を見ると，我々は提案した提案したホームネットワークシステム（HNS）における連携サービスの安全性検証方法が有効と判断された．

今後の課題は，本提案の有効性をさらに証明するために，より複雑な環境下での検証実験を行うこと，さらに，効率なテストケースの生成方案及び提案法の評価を行うことが挙げられる．また，HNS におけるサービス競合問題を考慮に入れた安全性検証手法の開発を行っていきたい．

謝辞

本研究は，科学技術研究費 (若手研究 B 18700062)，21 世紀 COE プログラム「NAIST-IS:ユビキタス統合メディアコンピューティング」の助成を受けて行われている．

文　献

[1] L. du Bousquet, Y. Ledru, O. Maury, and P. Bontron, A case study in JML-based software validation, Proceedings of 19th Int. IEEE Conf. on Automated Software Engineering (ASE'04), Linz, pages 294-297. IEEE Computer Society Press, Sep. 2004.

[2] International Electrotechnical Commission, Household and similar electrical appliances — Safety, IEC 60335-1, Sep. 2006.

[3] G. T. Leavens and Y. Cheon, Design by Contract with JML, Java Modeling Language Project, Internet: http://www.jmlspecs.org, 2003.

[4] P. Leelaprute, M. Nakamura, T. Tsuchiya, K. Matsumoto, and T. Kikuno, Describing and Verifying Integrated Services of Home Network Systems, In Proc of 12th Asia-Pacific Software Engineering Conference (APSEC 2005), pp.549-558, Dec. 2005.

[5] B. Meyer, Applying Design by Contract, IEEE Computer, vol. 25, no. 10, pp. 40-51, Oct. 1992.

[6] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, K. Matsumoto, Adapting Legacy Home Appliances to Home Network Systems Using Web Services, Proc. of Int'l Conf. on Web Services (ICWS 2006), pp.849-858, Sep. 2006.

[7] M. Nakamura, H. Igaki, and K. Matsumoto, Feature Interactions in Integrated Services of Networked Home Appliances -An Object-Oriented Approach-, Proc. of Int'l. Conf. on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFI'05), pp.236-251, Jul. 2005

[8] The Java Modeling Language - JML, http://www.cs.iastate.edu/ leavens/JML/

[9] JUnit, Testing Resources for Extreme Programming, http://www.junit.org/

[10] Ben Yan, Masahide Nakamura, Lydie du Bousquet and Kenichi Matsumoto, Considering Safety in Integrated Services in Home Network System, IEICE Technical Report IN2006-97(2006-11).

付　録

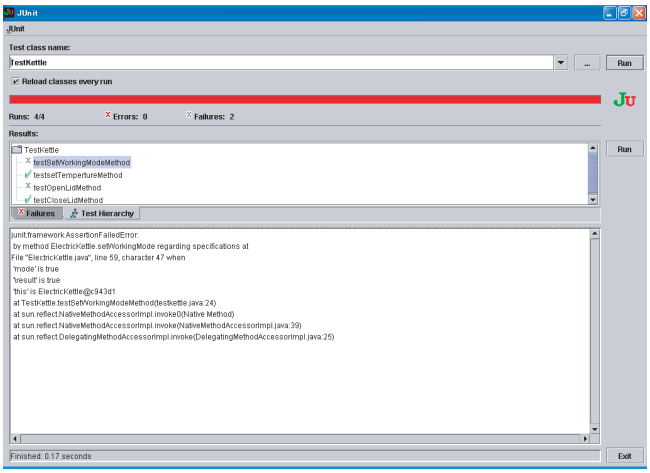

図 A·1　ElectricKettle.class の setWorkingMode() メソッドテスト画面

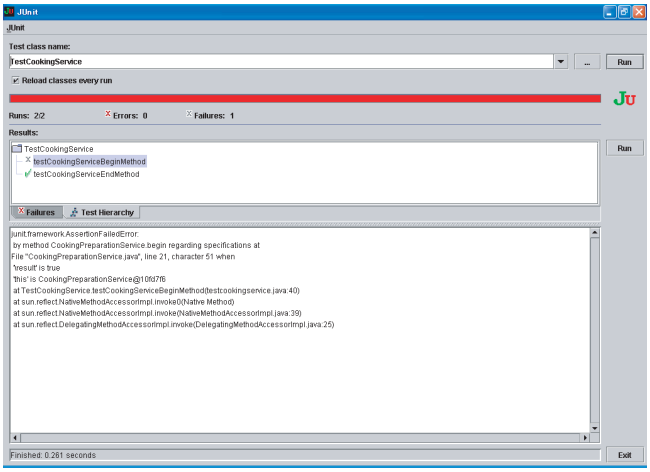

図 A·2　CookingPreparationService.class のテスト画面