ROMAI J., **3**, 1(2007), 65–76

TENDENCIES IN VERIFYING OBJECT-ORIENTED SOFTWARE

Pompiliea Cozma

University of Piteşti

pcosma99@hotmail.com

Abstract

The goal of this paper is to investigate the main approaches to the concept of object-oriented systems verification, namely:

- the adaptation of the classic technique to the context of new technologies of analysis, design and programming (object-oriented)

- applying the specific characteristics of the context of the new technologies (analysis, design and programming) in order to get new techniques.

Some considerations regarding the checking approaches to the aspect-oriented programming (AOP) whose defining characteristic is the moment, depending on the woven program, are also taken into account, when the systems verification is completed.

1. PRELIMINARY

Systems verification was developed at the same time with the projection and programming techniques which became in a short time a concept.

The complexity of systems verification determined a new checking technique based on new data. Throughout the development of a project steps were established (analysis, projection and programming) and for all these the methodology was settled. Among these methodologies the object oriented software has the capability to resolve systems complexity (the main problem of the new systems).

When resolving complexity [12] object-oriented software has some benefits like the abstract, encapsulation, the heritage which ensures the capacity of a new reality dimension report, seen from a certain point of view, but also two important facilities like reusing the code and the libraries.

The paper is structured as follows: we define the concept of systems verification and its classic techniques. Then we examine the verification techniques that are used in the context of object-oriented systems, namely those developed after the classic techniques of system verification - model checking - and the techniques specific to the object-oriented context - checking the consistency of class of collaboration diagrams.

Close attention is paid to the checking approaches to the aspect-oriented programming (AOP) whose defining characteristic is the moment, depending on the woven program, when the systems verification is completed.

In the first approach that we develop the aspects are considered as independent components which can be woven together with the other software system components. In this case the verification is completed before the software system starts its functionalities.

In the last part of the paper we consider another approach that is based on the use of classic techniques (model checking) to verify aspect-oriented programming system after the woven program, that is it verifies the programming code (Java) with familiar instruments (JPF/Java).

The techniques of system verification are exemplified by checking the deadlock freedom propriety.

2. THE CONCEPT OF SYSTEMS VERIFICATION

The concept of systems verification correctness was developed after applying (within program systems) formal language and methodologies, which rely on

well-developed mathematical procedures. This enabled the verification of the correctness of these specifications.

In the formal language of temporal logistic [1], formalization permitted the right investigation of fundamental system program properties:

- safety proprieties (deadlock freedom, partial correctness, global invariant, generalized invariant);

- liveness proprieties (total correctness and termination, accessibility property);

The development of programming systems and their increasing complexity necessitate an adaptation of the systems verification or a new technique.

The paper [8] presents a verification of the relevant properties for the program systems like "deadlock freedom" property in the case of concurrent systems.

Another important technique is "model checking" which means the systematic verification of a given property, in all the phases the system is passing through [5].

3. SYSTEMS VERIFICATION OF OBJECT ORIENTED SOFTWARE

In object oriented software context the first method of systems verification consists in adapting the classic methods mentioned in the previous section.

At this point a good example is checking the context of the deadlock freedom properties approached in [8].

Another example is the adaptation of the model checking technique to the context of object oriented software, discussed in [17].

The second method, specific to the context, is the diagrams consistency checking [15] which could start with the first steps of analysis and projection.

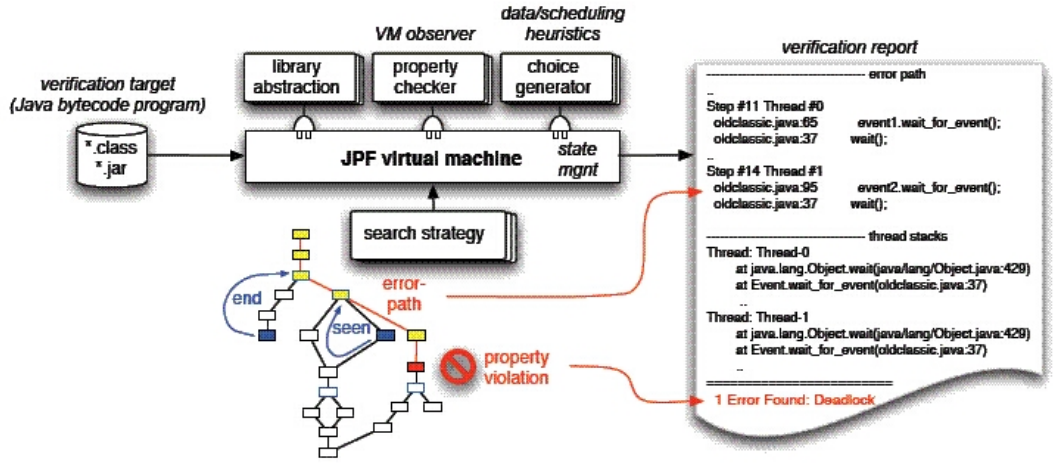


Fig.1. JPF usage.

This method includes the verification of some properties by type of preconditions, post conditions or invariant states.

The implementation of these perspectives on the system verification in the case of Java object-oriented software is done through the Java Path Finder instrument, which is in progress.

The way of using this tool is shown in [17] from where we choose the suggestive fig.1 in which the "deadlock freedom" property is shown.

4. SYSTEMS VERIFICATION OF ASPECT-ORIENTED SOFTWARE

The perspectives on the systems verification of aspect-oriented software are defined by the moment, according to the woven process, in which the way of achieving the verification is foreseen.

For example, it takes the generic problem producer/consumer (illustrated in fig. 2) defined as in [1]:

-the producer - for each going through the curl - generates an object which is stored in a limited buffer (which cannot contain more than N such objects)

-the consumer : from time to time it drags out such an object from the buffer and eats it

-stocking up the object in the buffer (on being made) means that it is not full and dragging out the same objects from the buffer (on consumption) means that it is not empty.

For this problem we will analyze the section with the buffer functioning, whose description is illustrated in fig. 3 by means of object-oriented software.

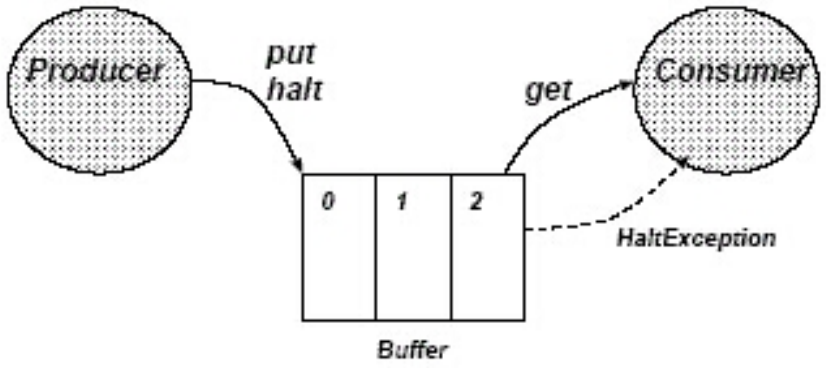


Fig.2.

The generic problem of producer/consumer.

4.1. THE VERIFICATION PRIOR TO THE WOVEN PROCESS

Within an aspect-oriented program verification aspects are considered as independent components which can be woven together with the other entities of an object oriented system. Its verification can be achieved before the system accomplishing its functions.

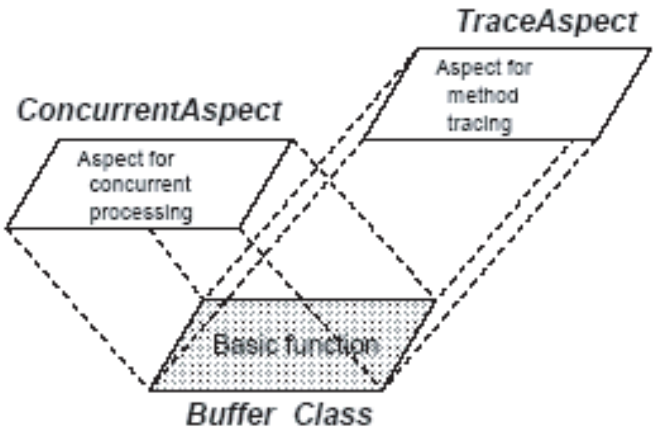


Fig.3.

Buffer description in AOP.

In the paper [8] the following methodology is suggested (the approach is recommended for safety properties verification defining the aspect oriented programs).

1. For the beginning, safety proprieties are selected, which are relevant for the soft system (such as deadlock freedom, liveness of state).

To illustrate this method [8] analyzes the deadlock freedom property in the case of a hypothetical concurrent system rendered in aspect oriented language (AspectJ - shown in fig. 4).

Similarly, the deadlock freedom propriety is selected for analysis in the case of the generic problem producer/consumer (illustrated in fig. 2).

2. The formal verification technique is selected according to the above established properties.

In the case of deadlock freedom property such a technique can be model checking.

3. The aspects which encapsulate the code, relevant to the context of established properties are identified.

```
public class User1 {
  public void removeUseless(File f){
    if(f.isUseless()){
      Directory d = f.getDir();
      d.removeFile(f);
      System.out.println("Removed: " + f.getName());
    }
  }
}
```

(a)

```
public class User2 {
  public void updateDir(Directory d){
    Enumeration e = d.getFiles();
    while(e.hasMoreElements()){
      File f = (File)e.nextElement();
      f.update();
      System.out.println("Updated: " +
                         f.getDir().getName()
                         + f.getName());
      }
    }
}
```

(b)

Fig.4. Hypothetical concurrent system.

For the example of a hypothetical concurrent system the relevant aspects are those which encapsulate the synchronized policy - shown in fig. 5.

For the example of the generic problem producer/consumer the relevant aspects are those which encapsulate the buffer whose description, by means of aspect-oriented method, is shown in fig. 3.

4. Subsequently, the formal verification technique is used; this was selected for the identified aspects.

In the case of the aspects which encapsulate the synchronized procedure as in the example of hypothetical concurrent system, modeled in PROMELA, SPIN was used for the verification process.

```
1  aspect ConcurrencyAspect{
2
3   /* pointcut declarations */
4   pointcut file(File f):
5     (instanceof(f) && executions(void File.update()))
6      || executions(void User1.removeUseless(f));
7   pointcut dir(Directory d):
8     (instanceof(d)
9      && executions(boolean Directory.removeFile(File))
10     || executions(void User2.updateDir(d));
11
12  /* code to be woven when pointcuts occur */
13  static around(File f) returns void: file(f){
14      synchronized(f){
15        System.out.println("locked: "  + f.getName());
16        proceed(f);
17        System.out.println("unlocked: " + f.getName());
18      }
19  }
20  static around(Directory d) returns boolean: dir(d){
21      synchronized(d){
22        boolean b;
23        System.out.println("locked: " + d.getName());
24        b = proceed(d);
25        System.out.println("unlocked: " + d.getName());
26        return b;
27      }
28  }
29 }
```

Fig.5.

Aspects which encapsulate synchronized policy.

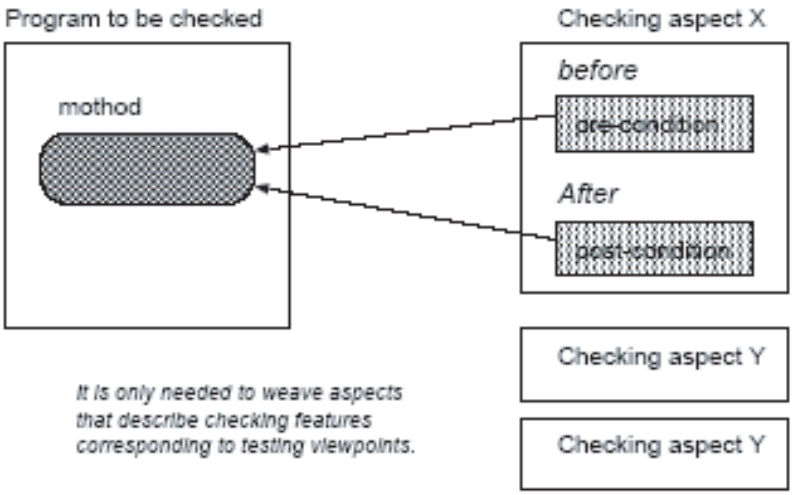
Similarly, one can approach the buffer described by means of aspect-oriented method as in the example of the generic problem producer/consumer.

4.2. THE SUBSEQUENT VERIFICATION OF THE WOVEN PROCESS

Another approach is focused on using the model checking techniques to verify the aspect-oriented programs after the woven process, shown in the

paper [11] and to verify the code which has resulted (Java) with the already known instruments (JPF/Java) respectively.

In the paper [11] both a procedure and a working flow are presented - shown in fig. 6.




```
aspect Sample {
   // specify a join point
   pointcut fooPointcut(Foo o):
     call(public void bar()) && target(o);
   before(Foo o): fooPointcut(o){
     // specify a pre-condition
   }
   after(Foo o): fooPointcut(o){
     // specify a post-condition
   }
}
```

Fig.6.

The procedure and the working flow for verification whit the help of JPF.

This approach is exemplified in the paper regarding the generic problem producer/consumer (described in figs. 2 and 3) by the model checking verification procedure, illustrated in fig. 7.

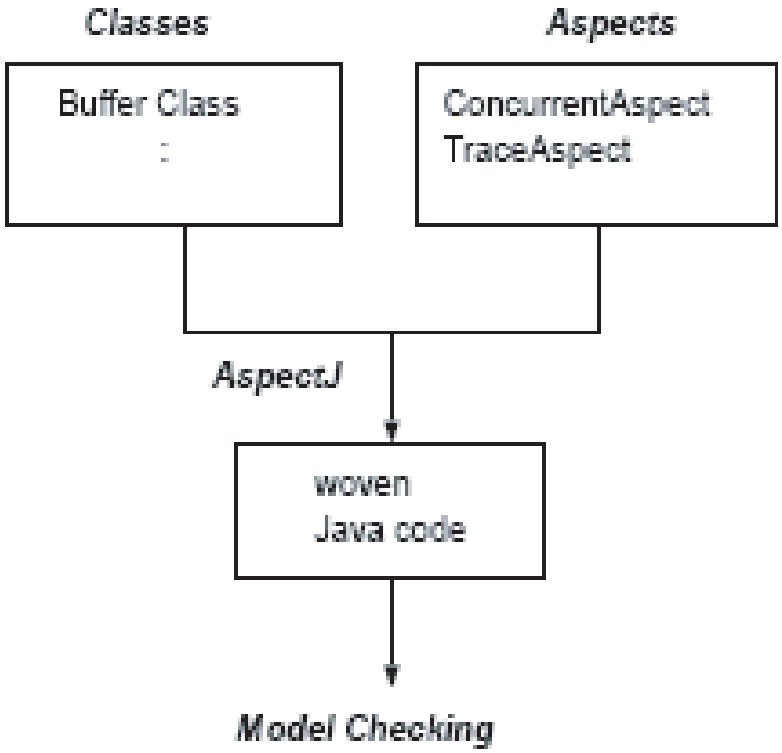


Fig.7.

Model checking procedure for the generic problem producer/consumer.

5. CONCLUSIONS

The recent results regarding the concept of programs systems verifications have led to two main tendencies.

The first tendency - the adaptation of the classic technique to the context of new technologies of analysis, design and programming (object-oriented).

The second tendency - applying the specific characteristics of the context of the new technologies (analysis, design and programming) in order to get new techniques.

References

[1] F. Kroger, *Temporal logic of programs*, Springer, Berlin, 1986.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, J. Irwin, *Aspect-oriented programming*, in ECOOP‘ 97 - Object-Oriented Programming, 11th European Conference, LNCS 1241, 220-242, 1997.

[3] D. Parsons, *Introductory Java*, Letts Educational, 1998.

[4] F. Ipate, M. Holcombe, *Correct system: Building a business process solution*, Springer, Berlin, 1998.

[5] J. P. Katoen, *Concepts, algorithms, and tools for model checking Lecture Notes of the Course "Mechanised Validation of Parallel Systems"* (course number 10359) Semester 1998/1999.

[6] G. J. Holzmann, Software model checking, Engineering Theories of Software Construction, July 25-August 6, 2000.

[7] M. Fowler, *UML Distilled*, Addison-Wesley, New York, 2000.

[8] G. Denaro, M. Monga, *An experience on verification of aspect properties*, 2001.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. G. Griswold, *An overview of AspectJ*, Proceedings of the European Conference on Object-Oriented Programming, Budapest, Hungary, June 18-22, 2001.

[10] R. C. Martin, *The principles, patterns, and practices of agile software development*,Prentice Hall, 2002.

[11] N. Ubayashi, T. Tamai, *Aspect-oriented programming with model checking*, Proceedings of AOSD 2002 (1st International Conference on Aspect-Oriented Software Development) Enschede, The Netherlands, April 22-26, 2002, 148-154.

[12] R. C. Martin, *UML for Java programmers*, Prentice Hall, 2002.

[13] R. Laddad, *AspectJ in action*, Manning Publications Co., 2003.

[14] I. Jacobson, *Use cases and aspects - working seamlessly together*, Journal of Object Technology, **2**, 4, (2003).

[15] R. Paige, J. Ostroff, P. Brooke, *A test - based agile approach to checking the consistency of class of collaboration diagrams*, UK Software Testing Research **2**, 4-5 September 2003.

[16] R. Pawlak, H. Younessi, *On getting use cases and aspects to work together*, Journal of Object Technology, **3**, 1, January-February 2004, 15- 26.

[17] P. C. Mehlitz, W. Visser, J. Penix, The JPF runtime verification system, 2005.

[Internet] : http://www.objectmentor.com/resources/articles.