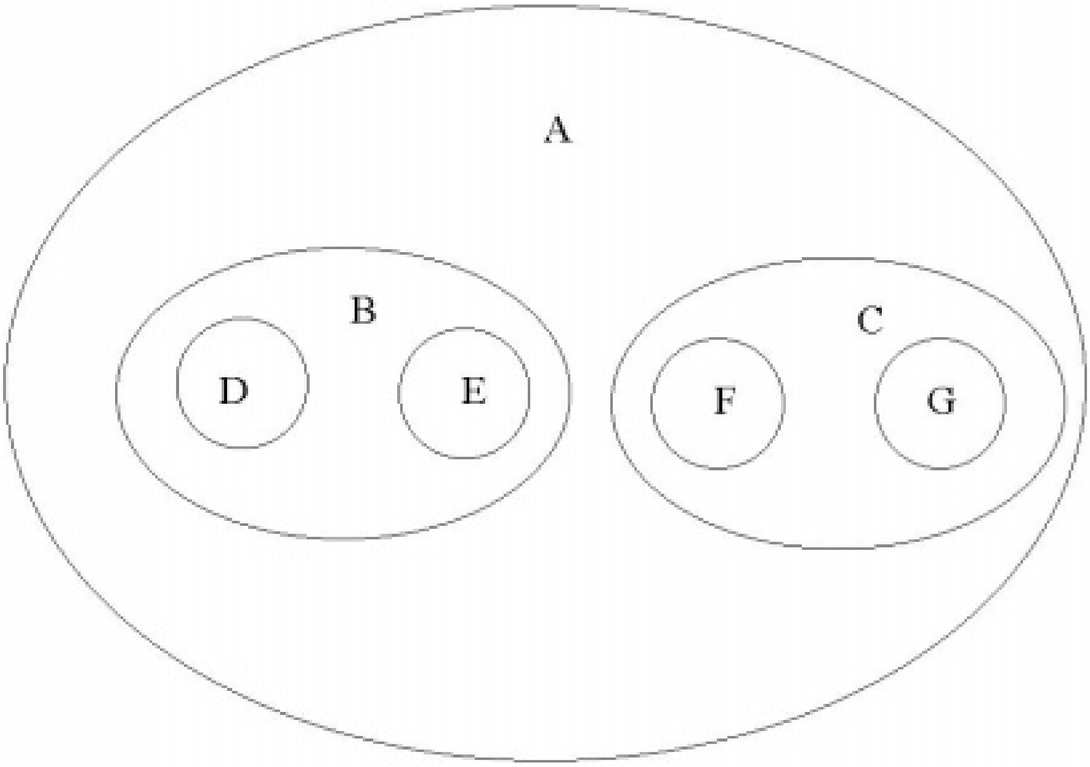
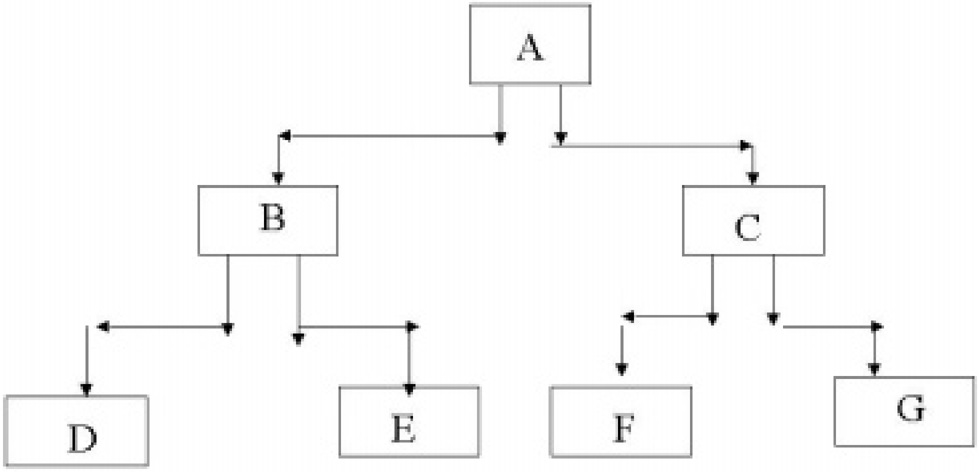


The P-Array System

A System for Developing Programs.



Hermann Schmitt
Dipl.Volkw., Dipl. Math
e-mail: 100042.1471@compuserve.com

July 12, 1998

The P-Array System
A System for Developing Programs.

Introduction.

In an article with the title "Structured Programming Reconsidered" the author has described the ideas, which are the foundation of a new kind of program development systems. Further the author has developed a software system, which implements these ideas. The System is called "The P-Array System": This article describes, how this software is to be applied.

The system is programmed in Java., Java 1.1 is required. The system may be used for programming in Java and C/C++. For Java it gives special advantages, which do not apply to programs in C/C++. The implementation in Java gives the advantage, that it can be used on all platforms. The programs of the system are not applets but normal Java programs. It could be possible to implement the programs as applets, but applets have different security restrictions on different platforms. Besides, the time to load the programs would be too long in most environments.

The Specification of P-Arrays.

In the article mentioned above the P-Array was defined. The P-Array is defined as a separetely stored part of a program, which specifies routines, which have to be executed for the different cases of one condition. It is comparable to a switch statement. Such a component of a program is conceived as a array with the different conditions as indices and the routines belonging to the indices as "values" of the indices i.e. a program array (P-Array)

In this article it was also mentioned, that the notion is expanded and so encompasses any source text with the restriction defined in the next paragraph. But these different P-Arrays are not treated equally. It is distinguished between "proper" and "improper" P-Arrays. The expansion of the notion is very important, because e.g. Java has no feature, which allows to insert source text at compile time.
Further, the notion of root element is used. A root element is the root of the tree of P-Arrays. It is an improper P-Array. The root element represents the program. The name of the program is equal to the name of the root element with an extension added.
The contents of the P-Array are normally stored in a separate file, but the contents may als be specified in the definition of the P-Array. Accordingly, it is distinguished between internal and external P-Arrays.

An notion important for the P-Array system is the nesting level. The nesting level at a certain point in a source text is defined as the difference between opening and closing braces, calculated from the beginning of the source text. The nesting level is assumed to be zero at the beginning of a source text. Further, the notion nesting-neutral is important for the P-Array System. A source text is called nesting-neutral, if the nesting level is zero at the end of the source text and nowhere negative in the source text.
One condition any P-Array must fulfill is that the P-Array must be nesting-neutral. If the components are nesting-neutral, the program composed of these components is nesting-neutral, too. The search for errors in the structure of a program can be restricted to the search for errors of the structure in the components. This is easier, because the components are smaller. Errors in the structure are especially troublesome, the compiler cannot assist in searching for errors in the structure, the compiler onliy notifies, if errors in the structure are present.

Syntax of P-Array Definition.

Proper P-Array

```
--> (program)  (??)name ([]) ------------------------------------------------------->
                              |
                              ----(file)--------------------------------------------------->  (;)
                                    |                              |
                                    ------- (=) directoy---------

--> ({) --------------------------------------------------------------------------------->
        |                                              |
        --- ({) definition of local variables (})----

--> ([)condition(]) (=) ({) routine (})---------------------------------------------------(; ) -> (})
 |                                             |                                          |
 |                                            (,)                                         |
 |                                             |                                          |
 <---------------------------------------------    -->([default]) (=) ({) routine (})--->
```

Inproper P-Array

```
--> (program)  (??)name  ----------------------------------( { ) routine (})--> (;)
                         |
                         --------(file)----------------------------------------> (;)
                                   |                            |
                                   ------ (=) directory------
```

Comments:
- Tokens enclosed in ( ) are invariable and have to be written as shown, the brackets are to be removed.
- If "file" is specified, the contents of the component are stored in a separate file.

Every P-Array has a name. In the statements the name is preceded by two "?". In the case of an external P-Array the file name has to be identical with name of the P-Array with the extension "par". But the root element has the extension "bse".

A P-Array has to defined in another P-Array, which precedes the invocation of the P-Array. In Java and C/C++ all datafields etc. have to be defined, and this rule is valid for P-Arrays, too. But this rule also make sense, especially for P-Arrays. Often, it is important to know, what components the program consists of. A good practice is, to define the P-Arrays in the root component,. but a P-Array may be defined in any component. A P-Array may be defined more than once (i.e. the same name), but only one definition is used in P-Array invocations. A warning message appears in this case.

By the example below, the syntax will surely get clearer. In this example only external P-Arrays are used, however.

Statement Types of the P-Array System.

By the P-Array System three types of statements are introduced into the programs (components).
These are:

1) The P-Array Definition.

The P-Array definition was already delt with in the preceding section. The P-Array definition prepares the P-Array for use in P-Array invocations. The P-Array definiton is different for proper an improper P-Arrays. Proper P-Arrays are transformed into normal source text of the programming language. Improper P-Arrays remain unchanged. When the P-Array definition is processed, it is examined, if the P-Array is nesting-neutral. A P-Array, that is not nesting-neutral, is rejected and an error message appears . The program cannot be composed.
The form of the P-Array definiton statement is:

program ??name..... (see the preceding section)

2) The P-Array Invocation.
The P-Array invocation has the effect, that the contents of the P-Array are inserted at that place, where the P-Array invocation statement il located.
The form of the P-Array invocation statement is:

??name

3) The P-Array Substitution.
The P-Array substitution assists in making P-Arrays reusable. Instead of the P-Array named in the P-Array invocation another P-Array is inserted or the insertion is suppressed. The text of the P-Array invocations is not changed in the components.
The form of the P-Array substitution statement is:

substitution (??name1 = [??name2] , ??name3 = [??name4]........)

If a P-Array invocation statement with name1 is encountered later on, the P-Array with name2 is inserted. If name2 is missing, no P-Array is inserted. In one substitution statement several substitutions may be defined.

The Development of Programs with the P-Array System.

The Development Environment.

Every developer has a development environment. The essential part of an development environment is a series of directories, in which files are stored according to there function in the system. Not all of the directories must be exclusively associated with one person. The directories must be available before using the system. Often the directories are automatically created, when the system is installed. But these directories then have a fixed structure and fixed names. All elements of the development environment are changeable. In order to do this, the class EntwUmg must be changed accordingly and compiled. This class is available in source and compiled form.

There are two directories one for root elements the other for P-Arrays, in which these elements are to be created and edited. As already mentioned the proper P-Arrays are transformed into normal source text of the programming language. The transformed P-Arrays are stored in a separate directory. This directory is normally not accessed by the developer. The composed programs are stored in a further directory, which is normally not accessed by the developer, either. But the source in this directory is the input for the compiler. The directories mentioned till now, will normally be situated on the private computer of the developer and exclusively associated with this developer. But it is also taken into consideration, that there may be common repositories for all or a group of developers, which normally can only be read by the developer. As a consequence of this reasoning there may be two additional directories one for root elements and one for P-Arrays. These directories must always be formally defined for the development environment, but the directories may be identical with the corresponding directories mentioned above.

The directory, where a P-Array to be used in the compositon of a program, is located may be also be specified individually in the definiton of the component (file parameter).

Further, there is a directory for message files. All messages of the system - which are mostly error messages - are stored in a file. There may be message files for different languages. A field in the development environment specifies, which message file is to be used, i.e. in which language the messages are to appear.

As already mentioned, the composed program should be transparent for the developer to a high degree. Therefore conversion tables for row numbers are stored in a further directoy. For each row in the composed program in must be determinable, from which component it originates and what the row number is in this component and vice versa.

In the developement environment the extension for the composed program is alse specified, because the development system may be used for different programming languages.

The Development Process.

The development of a program begins with the creation of P-Arrays. For this a normal editor is used. Data fields should not defined at the beginning of the program but in that P-Array, in which they are used in order to make the P-Arrays more reusable.

It may be advantageous to use the root element exclusively for the definition of P-Arrays and for the specification of substitution statements. In this way the composition of the program is determined by the root element. But P-Array definition statements and substitution statements may be in any P-Array.

When inputting a small part of a P-Array, identation my be used to show the structure. But later the invocation:

```
java Parray3 p1
```
p1: Name of the P-Array.

should be used to insert comments, which show the nesting level, into the P-rray and to remove the identation..

In developing a program, it may be useful to create an outline of the program, which shows the structure of the components including the indices of P-Arrays (conditions). This outline can often be made more understandable by inserting comments into the conditions.
Such an outline is created by invoking:

```
java Parray2 p1
```
p1: Name of the root element without extension.

Simultaneously with the creation of P-Arrays testing should be begun. The creation of P-Arrays should be done in such a way, that the composed program is alway executable. By the topdown method, which is also recommended for other reasons, this is accomplished to a high degree. This is also assisted by the possibility to exclude P-Arrays from the compostion by substitutin statements.

If the program is to be tested, the following invocation composes the program.

```
java Parray1 p1 [p2]
```
p1: Name of root element without extension.
p2: Name of directory, where the root element ist located. Often the default location specified by the development environment applies.

If the P-Arrays contains formal errors, error messages appear. It was already mentionend, that P-Arrays, which are not nesting neutral, are rejected.

If the composition produces no error messages, the compiler of the programming language may be invoked.

As already mentioned the composed program should be transparent for the develepor. Therefore the location of the error expressed by P-Array name and row in the P-Array is inserted before the error message of the compiler. In order to achieve this the error messages of the compiler must be redirected into a file, which has the name of the program with the extension "ems". Then the foolowing invocation is to be used:

```
java Parray4 p1
```
p1: name of the program without extension.

It is recommended, that a command file is created, which invocates thecompiler, and then invocates the program Parray4

An Example.

The purpose of the program can be derived from the two menues, which are shown on the screen with the statements:
System.out.println

How the P-Arrays have to be programmed is explained in the sections "The Specification of P-Arrays" and "Statement Types of the P-Array System".

```
  program ??parbsp01a file ;
  program ??dtzt file;
  program ??menue1[] file ;
  program ??menue2[] file ;
  program ??lesen file;
  ??parbsp01a;
```

Listing 1: Root Element parbsp01.

```
import java.io.IOException;
import java.io.*;
import java.util.*;
public class ParBsp01 {
   static ParBsp01 bsp;
   GregorianCalendar dtzt;
   public static String lesen()
   {
   ??lesen;
   }
   public static void main(String[] arg)
  {
    ??dtzt;
  }
    }
```

Listing 2: P-Array parbsp01a.

```
     int ci;
     char ch;
     String str;
     StringBuffer buf = new StringBuffer();
     try {
         while((ci= System.in.read()) != -1) {
              ch = (char)ci;
              if (ch=='\n') break;
              buf.append(ch);
         }
         str = new String(buf);
         return str.trim();
      } /* endtry */
      catch (IOException iex  ) {
         return "IOError";
      } /* endcatch */
```

Listing 3: P-Array lesen.

```
      String antwort;
      bsp = new ParBsp01();
      System.out.println("\nmenue");
      System.out.println("1: date");
      System.out.println("2: time");
      System.out.println("3: menue: time in some cities of the world");
      System.out.println("please, enter the number of the menue entry wan-
ted");
      System.out.flush();
      antwort = lesen();
      System.out.println("Answer: " + antwort);
      bsp.dtzt = new GregorianCalendar();
      ??menue1;
      System.out.println("end of program");
```

Listing 4: P-Array dtzt.

```
 {
```

```
[antwort.equals("1") /*date*/] =
 {
  System.out.println("The date is: "+
bsp.dtzt.get(Calendar.DAY_OF_MONTH)+"."+(1+bsp.dtzt.get(Calendar.MONTH))+".
"+bsp.dtzt.get(Calendar.YEAR));
 }
 ,
 [antwort.equals("2") /*time*/] =
 {
   System.out.println("the time is: "+
     bsp.dtzt.get(Calendar.HOUR_OF_DAY)+ "." +
bsp.dtzt.get(Calendar.MINUTE) + " hours");
 }
 ,
 [antwort.equals("3") /*time cities*/] =
 {
    System.out.println("\nmenue");
    System.out.println("1: time in Bombay");
    System.out.println("2: time in New York");
    System.out.println("3: time in San Francisco");
    System.out.println("timezone: time of a time zone");
    System.out.println("please, enter the number of the menue entry wan-
ted");
    System.out.flush();
    antwort = lesen();
    System.out.println("Answer: " + antwort);
   ??menue2 ;
   System.out.println("the time is: "+
     bsp.dtzt.get(Calendar.HOUR_OF_DAY)+ "." +
bsp.dtzt.get(Calendar.MINUTE) + " hours "+
          bsp.dtzt.getTimeZone().getID());
   }
 ,
 [antwort.equals("IOError") /*input error of operating system*/] =
 {
  System.out.println("input error of operating system");
 }
 ,
 [default /*wrong number for menue entry*/] =
 {
  System.out.println("error: the number of the menue entry is" + antwort);
  System.out.println("the number of the menue entry must be between 1 & 3
liegen");
 }
 ;
 }
```

Listing5: P-Array menue1.

```
 {
[antwort.equals("1") /*Bombay*/] =
 {
    bsp.dtzt = new GregorianCalendar(TimeZone.getTimeZone("IST"));
 }
 ,
 [antwort.equals("2") /*New York*/] =
 {
    bsp.dtzt = new GregorianCalendar(TimeZone.getTimeZone("CST"));
 }
 ,
 [antwort.equals("3") /*San Francisco*/] =
 {
    bsp.dtzt = new GregorianCalendar(TimeZone.getTimeZone("PST"));
  }
  ,
   [antwort.equals("IOError") /*input error of operating system*/] =
 {
  System.out.println("input error of operating system");
 }
 ,
 [default /*time of a time zone*/] =
 {
  bsp.dtzt = new GregorianCalendar(TimeZone.getTimeZone(antwort));
 }
 ;
 }
```

Listing 6: P-Array menue2.

```
Structure of Class: ParBsp01,       1998/6/8 : 7.53 hours.
------------------------------------------------------------

********
 1=Level
    P-Array Name: parbsp01a
            ??lesen
            ??dtzt

********
 2=Level
    P-Array Name: lesen
    P-Array Name: dtzt
            ??menue1

********
 3=Level
    P-Array Name: menue1
        [antwort.equals("1") /*date*/]
        [antwort.equals("2") /*time*/]
        [antwort.equals("3") /*time cities*/]
            ??menue2
        [antwort.equals("IOError") /*input error of operating system*/]
        [default /*wrong number for menue entry*/]

********
 4=Level
    P-Array Name: menue2
        [antwort.equals("1") /*Bombay*/]
        [antwort.equals("2") /*New York*/]
        [antwort.equals("3") /*San Francisco*/]
        [antwort.equals("IOFehler") /*input error of operating system*/]
        [default /*time of time zone*/]
 4=Level
********

 3=Level
********

 2=Level
********

 1=Level
********
```

Listing 7: Outline of Class ParBsp01.

Hermann Schmitt e-mail: 100042.1471@compuserve.com
Dipl.Volkw., Dipl. Math July 12, 1998

Special Advantages in the Case of Java Programs.

Class Definitions.

In the OO method it is distinguished between the interface of a method and its implementation. The interface of a method consists of the name of the method, the parameters with there types, and the type of the return value.

The developer, which uses the class, only needs to know the interface. Only the developer of the method must know, how it is implemented.

As a conseqence in C++ the class definiton only contains the interfaces of the methods. The source code of the methods is specified separately. In Java such a separation is not supported. By the P-Array system class definitons may be created, which essentially only contain the interfaces of the methods.

Try and Catch Blocks.

In Java many try- an catch blocks must be programmed. While the try blocks are necessarily application specific, there will be only a few developers, who want to program application specific catch blocks. The P-Array system offers the possibility to program catch blocks as P-Arrays, which can be invocated in the programs.

Similar Classes.

If similar classes have to be programmed, it may be advantageous to use the same class names for similar classes. The consequence is, that packages have to be used. In this clase this classes must have different package statements but may be otherwise (almost) equal.
The P-Array system makes it possible to use the same source code for similar classes with P-Array invocations for the package statements. By substitution different packages statements can be inserted.

Program Frames[1].

There are software systems which create program frames, into which individual code is be inserted by the developer. This is e.g. the case with CORBA. The P-Array system makes it possible to keep the program frame and the indiviual code separate by putting the individual code into a P-Array an a P-Array invocation into the program frame. This eliminates the danger, that the program frame is involuntarily modified, while inserting the individual code.In addition there are sometimes problems, when the page frame is updated and individual code is already present. It may be that in this case the individual code is deleted or not correctly taken account of.

Conclusion and Outlook.

The basic idea put forward in this article is, to better structure the programs. With this not the structuring by the control statements of the programming language is meant. This structuring is difficult to recognize. The identation, which is proposed in almost all programming manuals is no good solution in my opinion either. Instead a more physical structuring is proposed. This structuring is easier recognizable by the developer. Additionally this structuring makes programming more flexible. It is easier to add and remove code or move code to another position.

I think,that it would be further advantageous to put the-P Arrays into a database und to decompose the P-Arrays in the database into indices and values of indices. This would it make easier to add and remove indices and to edit idividual branches of the condition. A special editor would be needed too, to support these operations. The P-Array definition could be extended to encompass P-Arrays, which are stored in databases.

---

[1] Is also applicable to C/C++ programs.