The Petri Net Kernel: An INA-Pilot*

Ekkart Kindler Frank Oschmann
Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6
D-10099 Berlin†

1 Introduction

The *Petri Net Kernel* (*PNK*) [HHK+98] provides an infrastructure for easily building Petri net tools. The objective of the PNK [KD96] is to relieve the programmer of implementing graphical user interfaces, graphical editors and parsers. Altogether, the PNK should help the programmer to concentrate on programming the intended functionality such as analysis, verification, or simulation algorithms [Kin97]. What is more, the PNK is not restricted to a single Petri net type. Rather, the PNK can be used with any Petri net type. The Petri net type is given by a parameter which specifies the special nature of the used Petri net type. For each new Petri net type, the corresponding parameter can be easily implemented [Web97].

Originally, the Petri Net Kernel was designed for implementing new Petri net applications in a simple object oriented programming language called *Python* [Lut96] and for integrating these applications into a single Petri net tool. But, it can also be used for integrating existing Petri net tools or analysis modules—which will be demonstrated in this paper. As an example, we will discuss the integration of the well-known Petri net tool *INA* [RS98] of P. Starke into a PNK-application which we call *INA-pilot*.

We have chosen INA as an example for several reasons. First, INA provides a rich set of analysis techniques, which can hardly be found in any other existing Petri net tool. Second, INA as it stands has no graphical interface at all; the interface is only alpha-numerical. Therefore, the INA-Pilot makes the rich functionality of INA available to people who are not so fond of alpha-numerical interfaces. Last, we have chosen INA because it does not provide an explicit mechanism for integrating it into another program. In Sect. 4, you will see that the INA-pilot indeed simulates a real user; it watches the screen output of INA and simulates keyboard inputs. Therefore, a successful integration of INA proves that in principle every existing Petri net application could be integrated as a PNK-application.

Currently, we have implemented an INA-pilot which supports only a limited functionality of INA. Basically, it uses INA for calculating the place and transition invariants of a P/T-system. Then, the pilot gives a graphical representation of these invariants. This example will be used to explain the principles and the techniques for writing a pilot for a Petri net tool. An extension of the INA-pilot to support other INA-functionality is easily possible.

*Supported by the Deutsche Forschungsgemeinschaft within the research project 'Petri Net Technology' (DFG Forschergruppe 'Petrinetz-Technologie').

†e-mail: {kindler|oschmann}@informatik.hu-berlin.de

2 INA

Before introducing the INA-pilot, let us briefly introduce INA in order to provide a flavour of its interface. For more detailed information, we refer to the documentation of INA [RS98].
INA starts with a dialog which allows to choose between different activities. This dialog is shown in Fig. 1—with some omissions indicated by [...] and user interactions indicated by sans serif symbols. Let us assume that we want to analyse a net from file tmpnet.pnt. In this case, we

```
>>>>>>>>>>>>  Welcome to the Integrated Net Analyzer! <<<<<<<<<<<<
Version 2.0;                              Peter Starke, Berlin 1997
[...]
Do You want to
    edit ? ....................................E
    fire ? ....................................F
[...]
    quit ? ....................................Q
 choice > E
Editor Menue:
Quit the editing process.....................Q
Read a net from a File.......................RF
[...]
edit> RF
Netfiles:
tmpnet.pnt
Petri net input file > tmpnet.pnt
Editor Menue:
Quit the editing process.....................Q
[...]
edit> Q
[...]
Do You want to
    edit ? ....................................E
    fire ? ....................................F
    analyse ? .................................A
[...]
    quit ? ....................................Q
 choice > A
Information on elementary structural properties:
Current name options are:
     transition names to be written
     place names not to be written
.................................Reset options? Y/N N
.....................Print the static conflicts? Y/N N
The net is not statically conflict-free.
[...]
Analysis menue:
Decide structural boundedness.........................................B
[...]
Compute a base for all S/T-invariants [non-reachability test]..............I
[...]
choice > I
```

Figure 1: An INA dialog

type E as indicated in Fig. 1 in order to start the edit menu of INA. From this menu, we choose item 'read net from a file' by typing RF. INA displays some available file names, e.g. tmpnet.pnt; after entering this name, INA loads this net. Now, we quit the edit mode of INA by Q and we can enter the analysis mode by A. After two further interactions (concerning some options), INA outputs some structural properties of the chosen net. Again, INA offers some more analysis

routines. For example, we can choose I for computing place and transition invariants of the net. The above example session of INA should provide enough information on INA's navigation mechanisms by keyboard inputs. The INA-pilot, which will be explained in Sect. 4, must watch INA's screen output and simulate the users keyboard input. Watching INA's output and simulating input will be done by help of *expect*, which will be introduced in Sect. 3.
One of the main problems in navigating through the dialogs of INA is the 'intelligence' of INA: The structure of the dialog depends on the properties of the chosen net. Therefore, in some situations keyboard input is only required for some nets and is not required for some other nets. For example, the dialog differs from the above example, if the chosen net is fragmented: There will be an information that INA cannot analyse this net and some additional user interaction is required. So, navigating to the computation of place invariants may require different keyboard inputs for different nets. Therefore, the INA-pilot needs to carefully watch INA's output in order to cope with INA's intelligent behaviour. Even worse, navigation may vary between different versions of INA—in particular, it may change with forthcoming versions of INA. Since we cannot tackle the later problem, we have decided to implement a pilot for INA 2.0. An adaptation to other versions, however, should not be too difficult because the necessary expect scripts are written in a modular way.

3 expect

'expect' is a tool for starting and controlling interactive programs [Lib91] by scripts. It allows to react to a program's output and to simulate keyboard input for a program. The basic command is the expect command. For example, the following lines

```
expect{ "fragmented net? Y/N " { exit }
        "choice > " { send "A" } }
```

handle the above mentioned situation where the user interaction depends on INA's output. In case of a 'fragmented net' the expect script terminates; otherwise it sends an A to INA upon the next request for a 'choice'. By nesting these expect commands, scripts can control more complex situations.
Figure 2 shows an excerpt of the expect script which controls INA. This excerpt basically resembles the INA dialog shown in Fig. 1. First, INA (Vers. 2.0) is started. Then, expect navigates through INA's dialogs: It makes INA enter the edit mode, read the file 'tmpnet.pnt' and exit the edit mode again. After that, INA's behaviour depends on the structure of the net; if the net is fragmented, INA requires some user input. In that case, the script properly terminates INA (by some omitted expect code) and, then, terminates itself. If the the net is not fragmented, i.e. when INA request another 'choice', the script navigates to the calculation of the invariants.
In the rest of the expect script, which is omitted in this paper, the calculated invariants are written to some file and INA is terminated.

4 The INA-Pilot

In the previous sections, we have shown how INA can be controlled by an expect script. The script makes INA read a file 'tmpnet.pnt' and write the calculated invariants to the files 'tmpnetp.inv' and 'tmpnett.inv'. Now, the PNK-part of the INA-pilot only needs to generate a file 'tmpnet.pnt' in INA format, start the expect script, and parse the invariant files. Moreover, the PNK-application will provide a menu which displays all invariants calculated by INA. On selection, the chosen invariant will be graphically displayed in the net.

```
#!/usr/local/bin/expect -f
[...]
spawn INA2 [lindex $argv 0]
[...]
expect {
[...]
    "choice > " {
        send "E" } }
expect "edit> "
send "RF"
expect "file > "
send "tmpnet.pnt\n"
expect "edit> "
send "Q"
expect {
    "fragmented net? Y/N " {
        send "Y"
        expect "choice > "
        send "Q"
[...]
        exit }
    "choice > " {
        send "A" } }
```

```
expect {
    "There are isolated nodes." {
        expect "choice > "
        send "E" }
[...]
    "Reset options? Y/N " {
       send "N" } }
expect {
    "Print the static conflicts? Y/N " {
        send "N"
        exp_continue }
    "choice > " {
        send "I" } }
```

Figure 2: An excerpt from the expect script for INA: 'exp_ina'

We cannot present the code for the INA-pilot here. The general principle for writing applications for the PNK (and some information on Python) can be found in the documentation of the PNK [HHK+98]; a simple example can be found in [Kin97]. Here, we discuss some aspects special to the INA-Pilot.
In order to start a UNIX application, we use some functions of the Python module `os`, which provide functionality of the underlying operating system. Therefore, the INA-pilot is not platform-independent any more. But, there are no means for starting applications in a platform-independent way up to now. The INA-pilot uses `os.fork()` for creating a child process; from this child process, the expect script is started by `os.execlp("./exp_ina", "exp_ina")`. This process works independently from the father process of the PNK-application; the father process waits for the termination of the expect script by the statement

```
signal.signal(signal.SIGCHLD, handler_sigchld)
```

where `handler_sigchld` is a function which is started upon termination[1] of the expect script. From this function, a parser for the invariant files is started and the invariants are displayed in a menu; a chosen invariant can be graphically displayed in the net by the PNK operation `annotate_Places` or `annotate_Transitions`. The argument is a list of pairs where the left part of the pair is some place or some transition and the right part is some string. All places resp. transitions in this list will be highlighted and the corresponding string will be displayed for each place resp. for each transition.
People who are interested in the PNK and in particular in the INA-Pilot, may contact the authors to get free copies of the Petri Net Kernel as well as of the INA-Pilot. INA, however, is not for free (for more information on obtaining INA, please contact Peter Starke).

As mentioned above, the INA-Pilot was designed for INA version 2.0. For older or newer versions of INA, some changes to the expect scripts might be necessary. Since these scripts are rather

[1] Due to a problem with the Python module Tkinter, the function is only started after an additional Tkinter event such as a mouse click or a mouse move. In practice, this means that the user needs to move the mouse in some application window in order to continue.

straightforward, the necessary changes should not be too difficult—but need detailed knowledge of INA's dialog structure.
In the current version, the INA-pilot supports only the calculation of place and transition invariants, which could also be implemented as simple PNK-application without using INA at all. But, the INA-pilot can be easily extended to support other functionality of INA. This extension requires to change both, the expect script in order to navigate to the required INA functions, and the PNK-application code for generation INA's input files and for interpreting INA's output files. Some of the standard functions such as reading and writing INA net files are available in a separate module, which is part of the INA-Pilot distribution. These functions can be used for writing the PNK-application code for the extended INA-Pilot.

5 Conclusion

In this paper, we have sketched the idea and the implementation of an INA-pilot for the Petri Net Kernel. For more detailed information, you should have a look at the INA-pilot and its (forthcoming) documentation.
The successful integration of INA into the Petri Net Kernel shows that the Petri Net Kernel can be used for integrating several Petri net tools; in particular those which do not provide graphical interfaces. So, the Petri Net Kernel can not only be used for implementing and integrating new application functions; also already existing applications can be integrated.

Acknowledgments We would like to thank Peter Starke for his support concerning INA. Moreover, we would like to thank Ines Schwenzer for her parser of the INA net file format which we have used in the INA-pilot.

References

[HHK+98] J. Hauptmann, B. Hohberg, E. Kindler, I. Schwenzer, and M. Weber. Der Petrinetz-Kern – Dokumentation der Anwendungs-Schnittstelle. Informatik-Bericht 98, Humboldt-Universität zu Berlin, February 1998.

[KD96] Ekkart Kindler and Jörg Desel. Der Traum von einem universellen Petrinetz-Werkzeug — Der Petrinetz-Kern. In J. Desel, A. Oberweis, and E. Kindler, editors, *3. Workshop Algorithmen und Werkzeuge für Petrinetze*, Forschungsbericht 341. Institut AIFB, Universität Karlsruhe, October 1996.

[Kin97] Ekkart Kindler. Der Petrinetz-Kern: Ein einfaches Anwendungsbeispiel. In J. Desel, E. Kindler, and A. Oberweis, editors, *Algorithmen und Werkzeuge für Petrinetze, 4. Workshop*, Informatik-Bericht 85, pages 13–18. Humboldt-Universität, Institut für Informatik, October 1997.

[Lib91] Don Libes. expect: Scripts for controlling interactive processes. *Computing Systems*, 4(2), November 1991.

[Lut96] Mark Lutz. *Programming Python.* O' Reilly, October 1996.

[RS98] Stephan Roch and Peter Starke. INA: Integrierter Netzanalysator, Handbuch Version 2.1. Informatik-Bericht 101, Humboldt-Universität zu Berlin, 1998.

[Web97] Michael Weber. Der Petrinetz-Kern – Eine Aufteilung in Invariantes und Variables. In J. Desel, E. Kindler, and A. Oberweis, editors, *Algorithmen und Werkzeuge für Petrinetze, 4. Workshop*, Informatik-Bericht 85, pages 13–18. Humboldt-Universität, Institut für Informatik, October 1997.