IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITB12847

This paper appears in the book, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2* edited by Mehdi Khosrow-Pour © 2006, Idea Group Inc.

Design Interactive Applications Using Object-Oriented Petri Nets in Software Components

Jaime Muñoz Arteaga & Francisco Álvarez Rodríguez
Univer. Autónoma de Aguascalientes Av. Universidad # 940, CP 20100, Aguascalientes, Mexico, {jmunozar, fjalvar}@correo.uaa.mx

Gustavo Rodríguez Gómez, Coordinación de Ciencias Computacionales del INAOE, Calle Luis Enrique Erro No. 1, Tonantzintla, Puebla, 72840, Mexico, grodrig@inaoep.mx

Héctor Perez González, Facultad de Ingeniería de la Univer. Autónoma de San Luis Potosí, Av. Dr. Manuel Nava No. 8, 78290 Zona universitaria SLP, Mexico, hectorgerardo@acm.org

ABSTRACT

An interactive application requires a high rate of maintenance and reutilization of software in order to guide the large diversity of user tasks. Software components have proven to be effective covering these requirements for the interactive applications at implementation level. However the current specification techniques for the software components are limited to design the dynamics aspects of an interactive application, such as the dialogue of user-driven interfaces. This work proposes a formalism based on object-oriented Petri nets which it is possible to specify the behavior and the structural aspects of the component based interactive application. The goal is offer to designer a formal specification supporting the functionality and the usability factors independently of any language programming or any graphic toolkit. In addition, the designer can make an easy maintenance and reuse of software component models, improving in this way the communication with the people involved with the software development.

1. INTRODUCTION

An interactive application requires a high rate of maintenance and reutilization of software in order to aid the user in accomplishments throughout the user interface. Software components have proven to be effective to develop interactive application at implementation level. For example a software component can represent the software abstractions the application and the support to user task throughout the graphical user interface. Here we will adopt the Szyperski´s definition [2]: *¨A software component is a unit of composition with contractually specified interfaces and explicit context dependencies. A software component can be deployed independently and is subject to composition by third parties¨.*

Although a lot of work has recently been devoted to the formal specification for software components, the interactive applications have received much less consideration. Currently a designer doesn't have a specification technique for design of dynamics aspects for interactive applications using software components, such as the dialogue of user-driven interfaces [3] and [4]. We found that it is not all clear the user interface specification of a software component what are the services to support the user actions and the order in which these actions invoke enable(disable) other services.

This work proposes the use of the object oriented and the Petri nets in order to specify respectively the structure and the behavior of user interfaces. The goal is to extend the specification of software component approach enable to capture the user needs independently of any language programming or any graphic toolkit. We first present the architectural model used in general for an interactive application. We asses the rational of the Petri net in such perspective. We then present a case study illustrating the usability and utility factor are taken into account in our approach. Lastly, the current work is compared with related work using a set of criteria relatives to the formal methods of interactive applications.

2 PROBLEM STATEMENT

A software component is a software unit for a third-party composition; the composition is realized through the set of specified interfaces. A software component can represent the software abstractions of an interactive application and it supports the user services throughout the graphical user interface. Software components are like black boxes where the programmer generally defines the internal part by a set of resources, data and object-oriented classes and the external part is represented by a set of services enables to user's actions (see figure 1).

In addition, a software component offers a mechanism to get information about its structure. This self-descriptiveness is achieved by introspection, this is a low level mechanism and the information provided for the services is not useful to support the user task. The user task is facilitated by the graphical user interface considered as a dynamic part of an interactive application. However, a large number of specification techniques for the software components don´t support the design the dynamics aspects of an interactive application, such as the dialogue of user-driven interfaces [1] [3] [4]. We found that it is not all clear in the interface of a software component what are the services to support the user actions and the order in which these actions invoke enable(disable) other services. This is a common weak point of the software component technologies such as the ActiveX, COM and Java Beans. For example the interface of a JavaBeans is represented by the archive manifest which offers only a list of object-oriented classes containing the component. It is necessary to be an experienced programmer in order to apply the introspection mechanism and at the same time it is necessary to interpret the information about the events, services and properties of the underlying component. In addition, the designer have a lot of difficulties to reuse the information related to user interactions such as the services, and the data offered to user throughout the graphical user interface.

Figure 1. Graphical representation of a software component

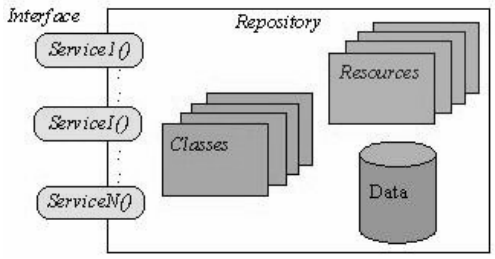
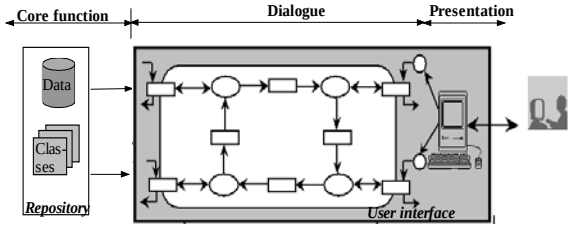


Figure 2. Architectural model for interactive software components (modified of [3])



3. OBJECT-ORIENTED PETRI NETS IN INTERACTIVE SOFTWARE COMPONENT

In order to present our approach to the design of interactive application, we will relate it to widely acknowledged Seeheim model. This conceptual architecture model for interactive application describes the user interfaces as structured in three modules: *the presentation, the dialogue and the core function* (see Figure 2).

1. The presentation module handles the lexical aspects of the interaction such in input as well in output.
2. The dialogue control module handles the syntactic aspects of the interaction and is responsible for the dynamic of the system.
3. The core function provides a semantic interpretation of the information received for the dialogue component.

3.1 Integrating Object-Oriented Petri Nets

A software component could be used effectively to code the software abstractions of an interactive application and specifying graphical user interface using object-oriented Petri nets. Petri nets come into play very naturally for the design of the Dialogue component of the Seeheim model (see figure 2). They allow for an easy description of complex, concurrent control structures, they offer several structuring construct, and, for the high-level models, they cleanly integrate the data structure aspects by allowing tokens to hold structured data.

In our approach, we will consider that (as it is often the case with current development methods) the presentation component is handled by specialized tools of the UIMS (User Interface Management System) category. Moreover, we will consider that the non interactive application kernel is designed in an object oriented approach. If this is not the case (for example the application kernel could be a relational database) the applications interfaces component will provide the necessary object-oriented layer

The behavior of a interactive software component specifies how it reacts to external stimuli according to its inner state. This behavior is described by a high-level Petri net of the component. A Petri net is a directed bipartite graph whose nodes are either *places* (depicted as circles) or *transitions* (depicted as rectangles). Places and transitions are connected by arcs. Each place may contain any number of *tokens*. In high-level Petri nets, tokens may carry values. In our formalism, tokens may hold conventional values (integer, string, etc.) or references to other objects in the application. A transition may feature a *precondition* that is a Boolean expression that may involve the variables labeling the input arcs of the transition. A transition *is fireable* (may occur) if and only if:

- Each of its input places carries at least one token
- If the transition features a precondition; it exists tokens in the input.

When a transition is fired, it removes one token from each of its input places, and sets one token in each of its output places. A transition features an *action* part, which may request services from the tokens involved in the occurrence of the transition, or perform arbitrary algorithms manipulating the values of tokens.

In this way an iteractive software componente offers a set of services that define the interface (in the programming language meaning) offered by the component to its environment. In the case of user-driven application, this environment may be either the user or other objects of the application. Each service is related to at least one transition ob the Petri net, and a service is only available when at least one of its related transitions is fireable.

3.2 Architectural Consideration

In the architectural Seeheim model a Petri net plays the role of the *Dialogue* component (see figure 2). The *core function is* modeled by the classes of the tokens flowing in the net. The *Presentation* component is made of a set of interactors (widgets) that may display and edit data (for example text entry fields or radio buttons), or trigger events of interest to the application (for example menu items or buttons).

The communication between the *Dialogue* component and the *Core function* is thus described both by the flow of tokens in the net and by the call of tokens methods in the transitions' actions.

The communication between the *Dialogue* component and the *Presentation* component is more complex to describe, since several aspects are to be taken into consideration:

- The *Presentation* component influences the dialogue through the occurrence of events. This occurrence is modelled in the petri net by special places called event places. The *Presentation* component is able to deposit tokens in those event places after the occurrence of an event. A transition in the petri net may have at most one input event place. A transition with an input event place is called an event transition. The very notion of interface place is made necessary by the fact that a given incoming event may trigger different actions in the system, according to the system's inner state. This is modelled by two or more event transitions in the petri net sharing a common event place. Those transitions are therefore in structural conflict, and this indeterminism has to be relieved by the structure of the petri net
- Conversely, the state of the *Dialogue* component (i.e. the marking of the petri net) influences the *Presentation* component: according to this state, several events may be disabled, and their associated interactor greyed out. This is described by associating event transitions to one or several interactors in the presentation: when a transition is not fireable, all of its associated interactors are greyed out or disabled.

Lastly, the state of the petri net must be displayed by the presentation. This is done by associating a rendering action to each place of the petri net Such actions may call methods of the tokens held in the place in order to display whatever information is appropriate.

4. CASE STUDY

We propose a case study wich features a interesting behaviour in order to exemplify the integration of object oriented petri nets in the interactive software components. The example chosen to illustrate our

Figure 3. A snapshot of thermostat at start time

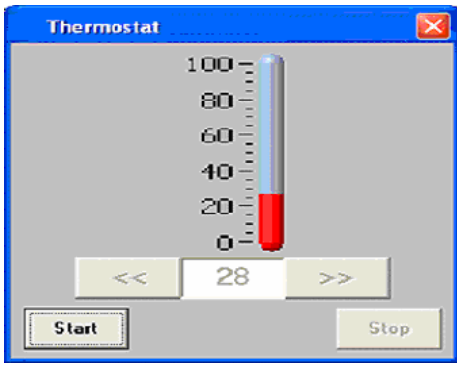
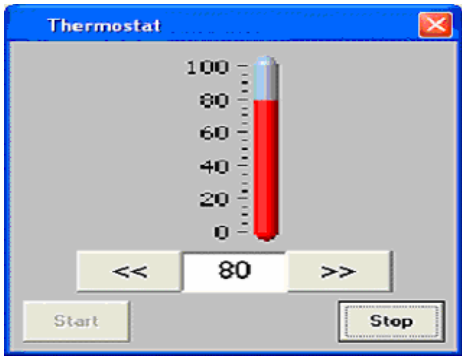


Figure 4. A snapshot of thermostat at runtime



approach is a fairly common one: an interactive thermostat taken and modified of Englander's book [6].

4.1. Description of the Case Study

The interactive thermostat is a simple application allows users to enter a temperature percentage using several input elements and to have an immediate feedback of the percentage on different output elements. The figure 3 and 4 shows the presentation part of this application.

The input elements are:

- A button ">>" that increments the value of themperature by one point. This button is not available to the user if the value of temperature equals 100.
- A button "<<" that decrements the value of themperature by one point. This button is not available to the user if the value of temperature equals 0.

The input/output elements are:

- A text widget where the user can type out a new value of temperature
- A thermometerchart widget that can be directly manipulated by the user. Using a mouse, the themperature can be modified, first by pressing the left button of the mouse then by dragging the mouse to the desired value.

An important feature of the application is that the abstraction (corresponding to the value of the variable themperature) is always consistent with its various graphical representations. Thus, the value in the text zone and the graphical value of the thermometerchart always correspond to the same value and each user action on the widgets modifies both the graphical representations and the abstraction. A static representation of the application with consistent presentations is shown on figure 3.

Figure 5. Presentation specification of thermostat application

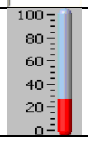
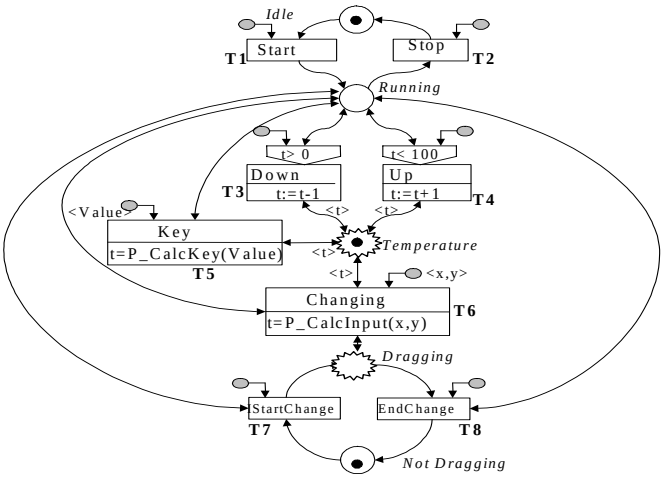
Services	Widgets	Graphical representation	Events
Start	ButtonStart	Start	Click
Stop	ButtonStop	Stop	Click
Up	ButtonUp	>>	Click
Down	ButtonDown	<<	Click
Key	TextZone	80	Key-Press
StartChange	ThermometerChart		Mouse-Move
Changing			Mouse-Down
EndChange			Mouse-Up

Figure 6. Dialogue specification of thermostat application.



4.2. Design of Thermostat Interactive Application

One of the goals is modeling in the software component interface of the interactive thermostat the user actions in order to make formal and non ambiguous such natural language informal requirements.

The presentation module of Seeheim model is specified mapping the user services with the interactive objects of graphical user interface as the figure 5 shown.

The dialogue module is represented by the object oriented Petri net of figure 6, this Petri net specifies that at the beginning, the application is stopped and the user can only start it by pressing the Start button. Then all the interactions described above are available until the user presses the Stop button that stops the application and puts it back into the initial state. This initial state is represented in figure 3.

In this case study the *Core function* module of our thermostat application is modeled by a class named T*emperature*. The declaration of this class feature a constructor, used to generate a new instances.

The code for this constructor should query the various widgets in the edit zone to gather the values for the new Customer's attributes. This code is not shown here for it is highly dependent on the graphical toolkit providing the user interfaces. The constructor should also take care of inserting the new instance in the repository of software component. Conversely the destructor called on object deletion, should take care of removing the instances from the persistence storage offered by the repository. Lastly, the class features a method called *Render*, whose purpose is to display the values of the instance's attributes in the window.

Figure 7. OO class Temperature as specification of core function of thermostat application

```
Class Temperature {
Public:
     Temperature ()
     ~ Temperature()
     SetTemperature(Real)
     Real GetTemperature()
     void Render() const
private:
     aTemperature: Real
}
```

5. FORMAL VERIFICATION

Formal verification aims at checking properties over a formal specification of a system. Some generic properties are of great interest for system specification and Petri nets theory offers predefined techniques for checking them. Such properties are liveness, boundedness and reinitialisability.

- *A system is live*, if for every state of the system, there exists a sequence of action that can trigger any given action of the system. That is to say that there is no dead branch in the system.
- *A system is bounded* if there is no production or consumption of resources during the activity of the system. This is an important property as it guaranties the fact that the system does not wear, i.e. its actual use does not jeopardise its future use .
- *A system is reinitialisable* if for any state the system can be in, it is always possible to find a sequence of actions that will set the system back in its initial state. This property relates to the previous although they are orthogonal [7], i.e. a system can be reinitialisable and not bounded and reciprocally.

5.1 Marking Graph

The marking graph of a Petri net is an automaton and can easily be automatically generated from a Petri net modelling a system with a finite number of state.

Most of the properties that one might be interested to check over a Petri net model, can be checked on the marking graph of this Petri net. However, when large-scale the marking graph can be rather difficult. This is not the case for the thermostat application and the marking graph corresponding to the Petri net of figure 6 is represented on figure 7.

As the marking graph is finite, the related Petri net model is bounded. It can be easily seen that the Petri net is live as every action can be reached from any state of the marking graph. The same reasoning can be applied in order to prove that the Petri net is reinitialisable.

Figure 8. Marking graph corresponding to the Petri net of figure 6

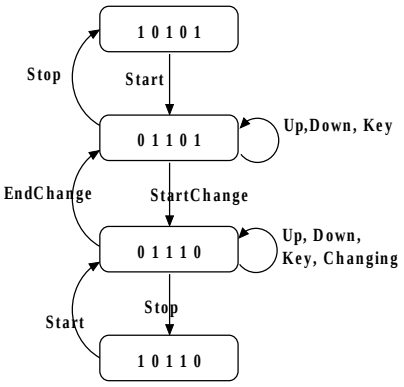


Table 1. Interactive aplicación specification technique

	Behavioral aspects	Structural aspects
ICO[1]	*High level Petri nets*	*Objects*
AOCE [8]	*Aspects*	*Objects*
Tadeus [5]	*High level Petri nets*	*Interaction Table*
ExecutionModel [9]	*StateTransition diagram*	*Object-oriented component*
DiaMODL[4]	*StateChart*	*Objects*
ISC	*High level Petri nets*	*Object-oriented software components*

6. RELATED WORK

This section presents a comparison of our approach with other works wich has purposed specfication techniques to design interactive applications.

All the works register in the previous table rely on well on mathematical well found approaches; they attempt to take into account both the dynamic and the structure aspects of an interactive application. Note that our approach called "Interactive Software Component" (ISC) is characterized by the integration of Petri nets into software components in order to design interactive applications.

7. CONCLUSIONS

This paper proposes integrate the object-oriented Petri the software component for interactive application in order to specify independently of any language programming the graphical user interface. In addition, we show how Petri nets could be used only for the specification phase, allowing to state in a concise manner complete and non ambiguous requirements for the control structure of this type of applications. Some of the interaction aspects in terms of usability factors have been exemplified on a meaningful case study. The designer can make an easy maintenance and reuse of software component model, improving in this way the communication with the people involved with the software development.

Finally, one expectation of the present work is to specify learning multimedia software component taking into account the combination of text, icons, sound and the visual animation feedback.

8. REFERENCES

[1] Bastide, Rémi and Palanque, Philippe. Conformance and Compatibility between Models as Conceptual Tools for a Consistent Design of Interactive Systems. CHI 99 workshop on Tool Support for Task-Based User Interface Design. 99.

[2] S. Clemens, Component Software – Beyond Object-Oriented Programming, Addison-Wesley and ACM Press, 2002

[3] Dix, Alan J. , Russell, Beale, and Wood, Andy. Architectures to make Simple Visualisations using Simple Systems. Advanced Visual Interfaces - AVI2000. 51-60. 2000. Italy, ACM Press.

[4] Hallvard, Traetteberg. *Dialog modelling with interactors and UML Statecharts A hybrid approach*. Design, Specification and Verification of Interactive Systems 2003. 2003. Springer-Verlag.

[5] Schlungbaum, Egbert. *Support of Task-based User Interface Design in TADEUS*. CHI'98 Workshop . 1998.

[6] R. Englander, *Developing Java Beans Anonymous* O'reilly, 1999

[7] Murata T. *Petri nets: properties, analysis and applications*. Proceeding of the IEEE 1989;77 (4).

[8] Grundy, J.C. *Multi-perspective specification, design and implementation of software components using aspects*, in International Journal of Software Engineering and Knowledge Engineering, Vol. 20, No. 6, December 2000.

[9] P. H. Frohlich and M. Franz. *Stand-alone messages: A step towards component-oriented programming languages*. In J.Gutknecht and W. Weck, editors, volume 1897 of Lecture Notes in Computer Science, 2000. Springer-Verlag.