Real-Time Temporal Control of Musical Processes

Raphaël Marczak
INSERM, U642, Rennes,
F-35000, France
Université de Rennes 1,
LTSI, Rennes, F-35000, France
raphael.marczak@gmail.com

Myriam Desainte-Catherine
Université de Bordeaux
IPB, UMR5800,
SCRIME, ENSEIRB-Matmeca
Talence, France
myriam.desainte-catherine@labri.fr

Antoine Allombert
LIPN
Université Paris 13
Villetaneuse, France
antoine.allombert@lipn.univ-paris13.fr

***Abstract*—The temporal control of the execution of multimedia processes is a crucial point for a number of application fields. We propose a formalism for authoring multimedia scripts that involves dynamic triggerings. In addition, we propose an abstract machine able to execute these scripts, by adapting the temporal organization of the processes according to the dynamic triggerings. This machine must implement some temporal features such as fast forward or GOTO functionalities. We present some algorithms performing these functionalities. A last, we mention some evaluation by users and possible future works.**

***Keywords*-Musical Processes; Petri network; Real-Time; Flexible time; Iscore**

I. Introduction

Controlling the temporal unfolding of multimedia processes is a useful but challenging task for several kinds of applications. Software systems like *Flash* are quite useful for creating multimedia contents for animation but they fail at precisely specifying synchronization between several processes. From live-performance to composition of interactive music or even oral presentations, the conception of a complex script implies controlling the temporal sequence of several processes as well as their coordination.

In most applications from the musical domain, time control is often performed by introducing continuous control giving the tempo of a score. In such systems, durations of musical events are defined according to a time unit that is stretched in order to fit real-time purposes. For example, a score-follower [1] or a real-time accompaniment system [2] is listening to a performer playing a score, which is known. By analyzing the sound played by the musician, the system is able to extract the notes and to follow the fluctuations of the tempo played by the musician. In consequence, the system can use those fluctuations of tempo to control the time unit of the score that he must play to accompany the musician. But such systems often fail at synchronizing precisely processes beginning or ending.

In the VIRAGE project [3], we addressed the problem of executing a script of multimedia processes for controlling light, sound or video, for live-performance while adapting the launching of the processes according to the play of actors on the stage. We introduced and implemented the *Iscore* system that uses discrete controls to precisely synchronize multimedia processes according to a script [4].

In this paper, we present added features to the *Iscore* system that permit real-time temporal control. The innovation of this system lies in the melding of discrete and continuous controls over temporal unfolding of the processes. With this system, the user can firstly place processes on a time-line and specify temporal constraints between starting or ending dates of processes. Secondly, he can specify interactive points that will come during execution flow of the system. The link between those two paradigms, that is time-line and time-flow, holds thanks to temporal relations that are verified by the system both at writing and execution time. At last, during execution, the user can modify the speed of the processes execution in a continuous way and he is also able to use a GOTO feature enabling him to skip or repeat some parts of the script.

In Section II, we briefly introduce the *Iscore* system and the implementation of the management of discrete events thanks to an abstract machine constituted of a Petri Net. In Section III, we present the new features, that is modification of the execution speed and GOTO, and how they are implemented in the context of the abstract machine.

II. Iscore system

The *Iscore* system has been fully described in [5], let us remember the bases of its conception and implementation.

The main question addressed by the *Iscore* system is the authoring and interpretation of musical scores of electro-acoustic music. This problem has been enlarged to a more general model for interactive multimedia scripts. The *Iscore* has typically two sides : the authoring side allows an author to design a multimedia script that can be modified during its execution, while the performance side executes the multimedia scripts and allows a performer to take benefit from the interaction possibilities. We studied the case in which the interaction possibilities consists in modifying the date of some steps of the multimedia processes involved in a script, as well as the speed of execution of these processes.

 ISBN: 978-1-61208-129-8

In order to prevent excessive modifications of a script during its execution, an author can define some boundaries that a performer must always respect. Since we only consider temporal modifications as interaction possibilities, the limits upon the interaction possibilities consist in temporal relations that must be respected during each performance.

A. Authoring side

One can find an example of a script on Figure 1. In this representation, the time-line is horizontal and left-to-right oriented. The author can temporally organized some *temporal objects*, which are represented as boxes. A *temporal object* can be *simple*, i.e., it represents the execution of a multimedia process such as the objects *sound* and *red* on the example ; or it can be *complex*, i.e. it represents the execution of a group of *temporal objects* such as the *lights* object. Each complex temporal objects holds its own time-line with a specific time speed. Therefore, a script can be performed with heterogeneous time speeds in its complex *temporal objects*.

A *temporal object* presents some *control points*, represented by circles on the top and bottom borders of the objects. *Control points* represent some particular moments of the execution of the *temporal object*. The beginning and the end of an object are naturally considered as particular moments. Other intermediate moments can also be considered. The author can temporally organize a script by adding some temporal relations between the control points of the objects involved in this script.

Since these relations can be defined between points, they can be of two types : *precedence* and *posteriority*. Formally, a temporal relation tr can be defined by a 6-uple :

$$tr =< t, p_1, p_2, \Delta_{min}, \Delta_{max} >$$

- t is a type (*Pre* or *Post*)
- p_1 and p_2 are control points
- Δ_{min} and Δ_{max} are real values in $[0, \infty]$

If tr is a precedence relation, then it imposes the inequality :

$$\Delta_{min} \leq d(p_2) - d(p_1) \leq \Delta_{max}$$

where $d(p_i)$ is the date of p_i.

The inequalities imposed by the temporal relations must be respected during each execution.

The possibilities of interaction are expressed through *interaction points*, represented by red circles. An *interaction point* turns a control point into a dynamic one. A dynamic control point must be explicitly triggered by the performer during the execution. On the contrary, the other control points (the static ones) are triggered by the system.

In order to execute interactive scripts, we had to design an abstract machine that can trigger the static control points, accept the triggering of the dynamic control points and respect the temporal relations.

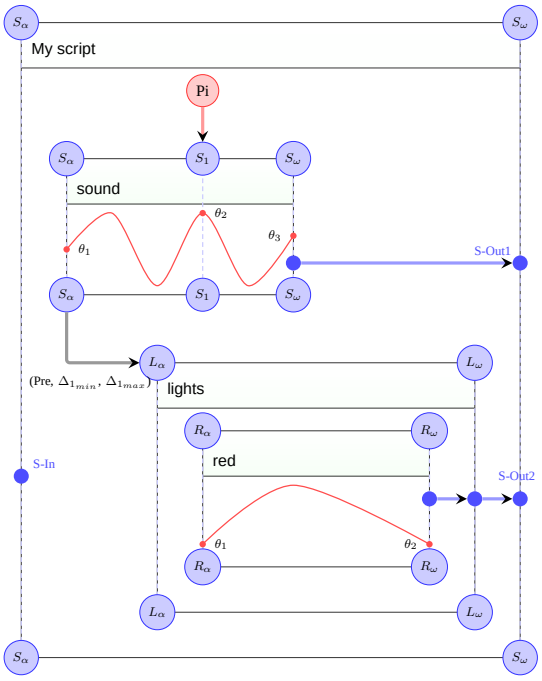


Figure 1. An example of an interactive script

B. ECO machine

We call the abstract machine designed for executing the interactive scripts the *ECOMachine*, for *Environment*, *Controls* and *Outputs*. One can find a representation of this machine on Figure 2. The term *Environment* must be understood as *temporal environment*. It carries all temporal information specified in the script. This information is represented in a time-stream Petri net [6]. One can see an example of such a net on Figure 3. To generate the environment associated to a script, we transform the script into a Petri net according to the following method. Each control point is turned into a transition. If a temporal constraint imposes the simultaneity of different control points, their transitions are merged. If a precedence relation is specified between a control point p_1 and a control point p_2, a sequence arc/place/arc is added between the transition of p_1 and the transition of p_2. The type of Petri net that we use accepts a time range on each arc. This time range allows us to represent the possible values for a time interval between control points. In addition, the crossing of a transition that represents a dynamic control event is conditioned by receiving an external control message.

Then, the term *Controls* represents the flow of control messages that trigger the dynamic control points. When a control point of a temporal object is triggered, the associated step of the process represented by the object is triggered. The part of the machine, which runs the multimedia process, is called the *processor*. It receives messages from the Petri net and sends the data produced by processes through the

 ISBN: 978-1-61208-129-8

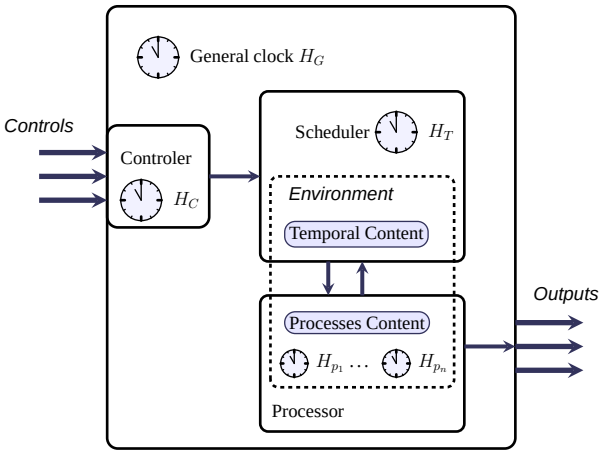


Figure 2. The ECOmachine

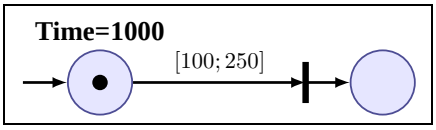


Figure 3. Here is a part of a Petri net produced by the transformation of an interactive score. During the execution, a token is produced in the left place at absolute time 1000. This leads to the creation of a START action labeled with the date 1100, as well as an END action labeled with date 1250. These two actions are put in the priority FIFO.

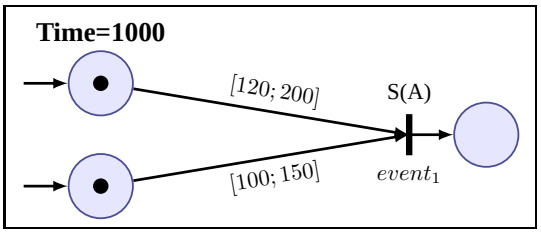


Figure 4. An example of a dynamic control point

Outputs flow.

The structure of this machine allows us to temporally control the triggering of the control points, as well as the speed of execution.

III. Real-time temporal control

We present the *Iscore* system without speed modification, we provide some algorithms to manage this system in an efficient way, and then we introduce what modifications are needed to allow speed modification as well as the GOTO feature.

A. Discrete Temporal Control System

For the discrete temporal control, we apply the petri network crossing rules.

1) Static And Dynamic Transitions: A script without trigger points, i.e., without interaction, only contains static transitions. These transitions are triggered by the system. When an interaction is needed, the compositor can decide to add a trigger point on a processes start or end, the transition becomes dynamic [3]. He can also decide a time range in which this event can be triggered.

In the Petri network, this mechanism is implemented by a transition, which is waiting for an event. Each of its ingoing arcs has minimum and maximum values that match the time range decided by the compositor. The minimum value corresponds to the minimum time after which we can cross the transition. The maximum value corresponds to the time when the transition will be crossed, whether the event was received or not.

2) Priority FIFO Algorithms: During the editing process, the time taken by each user action is not really important. However during real time execution, we cannot allow a procedure to be time consuming. The most many-time called procedure is the one, which decides if a transition should be crossed or not. It is impossible to check all the transitions all time because it would be a long process. So we decided to use a priority FIFO that can be filled with priorityActions.

PriorityAction

A PriorityAction is defined by a transition, a date, a type (START or END) and a boolean stating if the action is still enabled. The type is START when a transition could be crossed (for example when we just wait for the event to come), and END when a transition must be crossed whether the event was received or not.

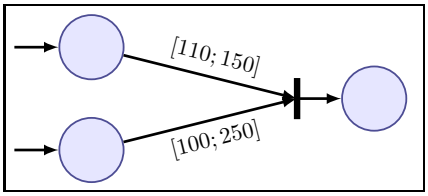
Filling and updating the FIFO

These actions will be the elements of a chronologically priority FIFO.

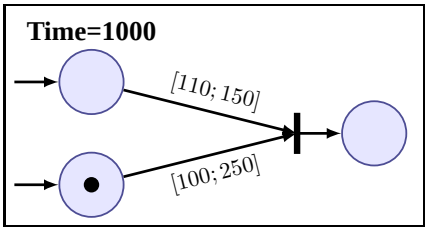
Filling the FIFO is not an easy procedure. If fact, START and END actions should be computed in real time, but they depend on how the tokens arrived in the places before the transition. The Figure 4 shows a simple example. In this example, the considered transition represents a dynamic control point. This means that during the execution, the system will wait for receiving a external trigger message called $event_1$. When this message is received, the system crosses the transition, which leads to start a process called *A*. The system must respect the time ranges introduced by the user. Therefore, if the two tokens are produced simultaneously at absolute time 1000, the transition can be crossed between time 1120 and time 1150. Then the system will ignore a message $event_1$ that would arrive before time 1120 and it will automatically crossed the transition if no message has been received at time 1150.

One can find a more complex example on Figure 5.

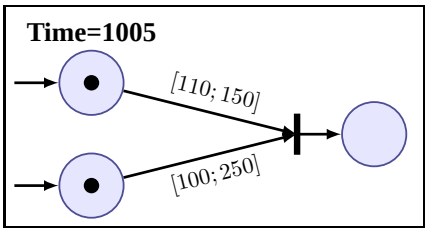
A transition knows at every time, which ones of its ingoing arcs are active, i.e., when a token is present in the



(a) The considered transition is preceded by two arcs with different time ranges.



(b) At the absolute time 1000, a token is produced is one of the preceding places. This will create a END action at 1250.



(c) At absolute time 1005, a token is produced in the second place. This will create a START action labeled with date 1115 (the maximum of 1000 + 100 and 1005 + 110) and a update the END action labeled with date 1155 (minimum of 1005 + 150 and 1000 + 250)

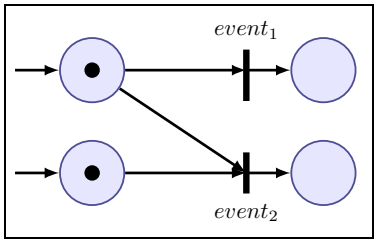
Figure 5. An example in which, two tokens arrive at different times. This implies that the system needs to update the date of an action

place linked by this arc. When the incoming arcs are not all active, we can only calculate and update the END action. When all arcs are actives, we can add the START action. When we update an action, we just disable the previous one and add a new one, for efficiency purpose.

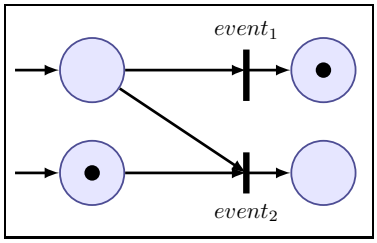
When a token arrives in an empty place, all the outgoing arcs are stated as active. At this moment, the END and START actions are computed. (See the Algorithm 1).

Computing the FIFO elements

At each ECO Machine cycle, the makeOneStep procedure (Algorithm 2) is called. It handles all the actions, which dates are lower or equal to the current date. If an action is disabled, it simply removes it. If an action is END, it forces the transition to be crossed (if it is impossible, an exception should be thrown). If an action is START, it fills a list of sensitized transition, i.e., transition waiting for an event. After that, it handles all the sensitized transitions. A transition that can finally not be triggered will be removed



(a) The two transitions can be crossed



(b) The message $event_1$ is received, the corresponding transition is crossed. The second transition cannot be crossed anymore.

Figure 6. In this example, the crossing of a transition leads to the impossibility to cross another transition.

from the list (for example if a place is involved in two or more transitions, and another transition was previously crossed, consuming the token).

Updating transition state

The last algorithm (Algorithm 3) handles a transition crossing. It consumes and produces token in the corresponding places, executes the external actions (i.e., the start or end of a processes), and resets the transition state. In fact, when a token is consumed, it could destabilize the system (for example when a token is involved in two or more transitions, as seen in the Figure 6).

B. Continuous And Discrete Temporal Control System

It is very useful to have the possibility to accelerate or decelerate the script. For example if we need to accelerate the fade out for music and light. But these features could not be possible without the concept of numbered tokens.

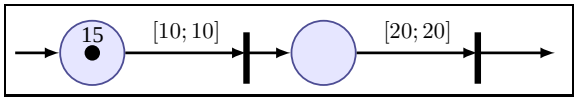
1) Deceleration: Decelerating is not a complex part of the speed modification. In fact, if all the time values are set in a millisecond precision, the precision of the Petri network time is in microsecond. So for decelerating, a simple multiplication of the next computed delta time by a factor between 0 and 1 is sufficient.

2) Acceleration: Accelerating is much more challenging. A first idea could be to call makeOneStep more often, but this would too much computation time, and it would be unsafe in real-time processing. Our solution is to use stamped tokens, but it is not straight forward solution. Each number on a token represents the remaining time to be handled.

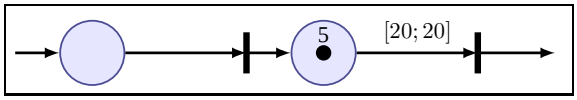
 ISBN: 978-1-61208-129-8

Algorithm 1 setArcAsActive

Require: $arc\ transition\ petriNet$
1: $transition.labelInGoingArcAsActive(arc)$
2: $currentDate \leftarrow petriNet.getCurrentDate()$
3: $startDate \leftarrow arc.getMinDate() + currentDate$
4: $endDate \leftarrow arc.getMaxDate() + currentDate$
5: **if** $startDate > transition.getStartDate()$ **then**
6: $transition.setStartDate(startDate)$
7: **end if**
8: **if** $endDate < transition.getEndDate()$ **then**
9: $transition.setEndDate(EndDate)$
10: $petriNet.addPriorityTransitionAction(transition, END, endDate)$
11: **end if**{/* if an END action already exist for this transition, it will be disabled */}
12: **if** $transition.allInGoingArcAreActive()$ **then**
13: $petriNet.addPriorityTransitionAction(transition, START, startDate)$
14: **end if**{/* if an START action already exist for this transition, it will be disabled */}



(a) The token is labeled with a value 15



(b) After the crossing of the first transition, the token is label-led with the value 5

Figure 7. An example of the spreading of a label-led token

- The makeOneStep algorithm will precise a token value when a transition is crossed, matching the remaining time after the crossing (currentTime - actionTime).
- When an arc is stated as active, the token value must be subtracted from the minimum and maximum arc values. See Figure 7.
- If this subtraction makes the transition crossable (for example if the value on the second arc on the Figure 7 is $[15; 20]$), this transition should be added to a transition list, handled at the end of the makeOneStep algorithm. And this should be repeated as often as the numbered token are disseminated in the Petri network.

3) Processes: The new speed must also be given to all currently running processes for them to adapt their computation, and must be provided to each processes launched next.

4) GOTO: The GOTO currently implemented can be seen as a very fast acceleration, where all dynamic transitions are turned into static transitions. It was an acceptable solution for a first version of the VIRAGE sequencer, but it has several limits.The artists using the VIRAGE sequencer made a lot of feedbacks about it.

Algorithm 2 makeOneStep

Require: $petriNet$
1: $transitionActionFIFO \leftarrow petriNet.getTransitionActionFIFO()$
2: $currentDate \leftarrow petriNet.getCurrentDate()$
3: **while** $(transitionActionFIFO.size() \neq 0) \wedge (transitionActionFIFO.top().date() \leq currentDate)$ **do**
4: $topAction \leftarrow transitionActionFIFO.top()$
5: $topTransition \leftarrow topAction.getTransition()$
6: $transitionActionFIFO.pop()$
7: **if** $topAction.isEnable()$ **then**
8: **if** $topAction.getType() = START$ **then**
9: $topTransition.declareAsSensitized()$
10: **else if** $topAction.getType() = END$ **then**
11: **if** $topTransition.allInGoingArcsAreActive()$ **then**
12: $topTransition.crossTransition()$
13: **else**
14: $throw\ incoherentStateException$
15: **end if**
16: **end if**
17: **end if**
18: **end while**
19: $sensitizedTransitionList \leftarrow petriNet.getSensitizedTransitionList()$
20: **for** each $currentSensitizedTransition$ in $sensitizedTransitionList$ **do**
21: **if** $!currentSensitizedTransition.allInGoingArcsAreActive()$ **then**
22: $sensitizedTransitionList.remove(currentSensitizedTransition)$
23: **else if** $(petriNet.hasReceivedEvent(sentizedTransition.getEvent())) \vee (sensitizedTranstion.isStatic())$ **then**
24: $sentizedTranstion.crossTransition()$
25: $sensitizedTransitionList.remove(currentSensitizedTransition)$
26: **end if**
27: **end for**
28: petriNet.resetEvents()

The first one is that if all processes are played rapidly, they are also played integrally. When a GOTO is performed, the artists usually do not want all the intermediate values, but only the last state of each processes. For example, if a processes computes a sound fade-in in a normal execution

Algorithm 3 crossTransition

Require: $transition\ petriNet$
1: $inGoingArcs \leftarrow transition.getInGoingArcs()$
2: **for** each $inArc$ in $inGoingArcs$ **do**
3: $inArc.consumeToken()$
4: **if** inArc.nbToken = 0 **then**
5: $transitionList = inArc.getPlace().getSuccessorsTransition()$
6: **for** each $transitionToReset$ in $transitionList$ **do**
7: $transitionToRest.resetArcState()$
8: **end for**{/* Resetting a transition arc state means looking for all predecessors places (after disabling the END and START action), and activate corresponding arc (with the previous values) if there is still a token in the place */}
9: **end if**
10: **end for**
11: **for** each $externAction$ in $transition.getExternAction()$ **do**
12: $externAction.execute()$
13: **end for**{/* an action could be a process start or end */}
14: $outGoingArcs \leftarrow transition.getOutGoingArcs()$
15: **for** each $outArc$ in $outGoingArcs$ **do**
16: $outArc.produceToken()$
17: **end for**
18: $transition.resetArcState()$

; in a GOTO situation, only the last value is useful. A good solution can be to execute all processes without sending the results, and only broadcast the last results of each processes. Another solution can be to regularly save the ECOMachine state, and its processes, and perform the GOTO from the closest saved state.

The second one is that some processes can not be accelerated, for example a light, which need 5 seconds to be correctly initialized. A solution can be to precise some processes as GOTO-rigid, and execute them completely even in a GOTO situation.

Finally, when some processes are in the correct state, for example the light is correctly initialized by a previous execution, artists do not want to have a complete reinitialization. A solution can be to have an interaction with this processes by asking is current state, and skip the GOTO-rigid part if the initialization is already performed.

C. Validation

These features were tested and validated by the artists involved in the VIRAGE project. An Agile method (SCRUM) was set up to improve communication between developers and artists. A bug tracker allowed fast corrections and performances were made to test and present these features during frequent meetings [3].

IV. Conclusion

In this paper we presented a novel system for controlling in real-time the temporal unfolding of multimedia processes. For that purpose, we mix several temporal paradigms. Firstly, we use a time-line model to place processes start and end dates as well as temporal relations between them. Secondly, the execution uses a time-flow model in which the processes are executed while holding the temporal relations stated on the time-line between dates. In this last model, temporal synchronization is performed thanks to discrete controls associated to processes dates, while continuous controls can be performed to control the speed of the processes. The implementation of such continuous controls in a discrete model as a Petri net was not straight-forward. We proposed solutions that have to be enhanced. In particular, the GOTO feature should not execute all processes but only those, which have a persistent effect on the future. As a matter of fact, a fade-in, which is followed by a fade-out of the light can be completely skipped. However, the move of a camera should be performed before the recording process execution. Logical relations have to take place between processes in order to skip processes properly. In the future, we want to open the scripts in order to give choices to the user, which can depend on what is going on in real-time. Such a system should be useful also in the context of museography and improvisation.

References

[1] A. Cont, “Antescofo : Anticipatory synchronization and control of interactive parameters in computer music,” in *Proc. of the International Computer Music Conference (ICMC08), Belfast, North Irland*, 2008.

[2] R. Dannenberg, “A language for interactive audio applications,” in *Proc. of the International Computer Music Conference (ICMC02), San Francisco, USA*, 2002.

[3] A. Allombert, R. Marczak, M. Desainte-Catherine, P. Baltazar, and L. Garnier, “Virage : Designing an interactive intermedia sequencer from users requirements and the background,” in *Proc. of the International Computer Music Conference (ICMC10), New-York, USA*, 2010.

[4] A. Allombert, M. Desainte-Catherine, J. Larralde, and G. Assayag, “A system of interactive scores based on qualitative and quantitative temporal constraints,” in *Pr. of the 4rd International Conference on Digital Arts (ARTECH 2008), Porto, Portugal*, November 2008.

[5] A. Allombert, “Aspects temporels d’un système de partitions musicales interactives pour la composition et l’interprétation,” 2009.

[6] M. Diaz, *Petri Nets: Fundamental Models, Verification and Applications*. Wiley-Blackwell, 2008.

 ISBN: 978-1-61208-129-8