Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 232 (2009) 55–73
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Performance Analysis of the ARIA Adaptive Media Processing Workflows using Colored Petri Nets

Maurizio Garelli[a,1] Marco Gribaudo[a,2,3]

[a] *Dipartimento di Informatica*
Università di Torino
Torino, Italy

Abstract

Multimedia systems are one of the most complex and interesting applications that are nowadays proposed to the users. Their complexity derives mainly from the fact that multimedia systems have to process huge amounts of data, while respecting real-time deadlines. For this reason performance evaluation of the underlaying workflow is a key issue in the design process of a new Multimedia system.
In this paper we consider the ARchitecture for Interactive Arts (ARIA), an adaptive media processing workflow, developed at the Arizona State University, and outline a semi-automatic procedure to translate its specification into Colored Petri Nets. We then provide guidelines on how to compute the parameters for the performance models, and apply the proposed methodology to a realistic example of a face recognition application.

Keywords: Multimedia, Adaptive Media Processing, Performance Evaluation, Colored Petri Nets, Statistical Analysis.

1 Introduction

During the last years the complexity of systems has grown considerably, together with the need of fast and performing architectures. In this paper we will focus on multimedia systems, where there is the need to process huge amounts of data, while respecting real-time deadlines. Several approaches have been proposed in the literature, for example: Telegraph [22], STREAM [2] and Aurora [6]. In this paper we will focus on the ARIA [17] system. ARIA is a middleware for describing and executing media processing work-flow capable of processing, filtering, fusing sensory inputs and actuating corresponding reactions. More precisely it extracts

[1] Email: garelli@di.unito.it
[2] Email: marcog@di.unito.it
[3] This work was supported by the Italian project PRIN No. 2007J4SKYP_003

doi:10.1016/j.entcs.2009.02.050

various features from streamed data, then fuses and maps media streams onto output devices, obeying to QoS requirements. ARIA has been used in several projects, like the Intelligent Stage built at Art, Media, and Engineering (AME) Center at Arizona State University [1] where it allowed performers to have real-time control of the stage. The system was able to detect and classify the movements of the performers, and to create changes in the environmental elements (creating spontaneous lighting and sound effects) in response.

Although the ARIA system provides very efficient real-time combinatorial optimization algorithms that can reduce the quality of the considered stream to respect the deadlines, it lacks the ability to test the system before actually building it. The goal of this work, is to define a methodology capable of translating a general ARIA workflow into a more performance evaluation oriented formalism. In particular, we propose a semi-automatic technique to convert an ARIA system into an equivalent Colored Petri Net [11]. We can then use some of the various tools created to analyze CPN, to study the system and evaluate its performance. We also propose a methodology to analyze the various multimedia components in isolation, that allows the definition of meaningful parameters that characterize the corresponding CPN models.

In order to prove the validity of the proposed technique, we apply it to a realistic example: an automatic employee barrier, that implements a face recognition algorithm to grant or deny the access.

The general problem of the face recognition can be summarized as follows: given an image or video, the system must identify the person using the information stored in a database. To solve this problem a series of steps are necessary: face segmentation (face detection) from cluttered scene, features extraction from the face regions and finally a recognition or verification.

In an identification problem, the system has to determine the identity of an unknown face from a database of known individuals. In a verification problem instead, the system has to confirm or reject the claimed identity of the input face.

The face recognition is part of the biometric research and it is basically divided into two branches: 2D and 3D face data recognition. There is also a third branch that is the fusion of the two previous and is still an open research area [20]. On 2D face recognition the standard de facto was the eigenfaces [9,12,23] but many other techniques have been implemented. The most famous are the PCA [21], 2D Discrete Cosine Transform (2D-DCT) and its extension explored in [14] where the 2D-DCT features are modeled using Gaussian Mixture Models (GMMs), and morphable models [3]. The open problems of this area mainly concern the illumination, the pose variation and (in part) the problem associated with expression variation.

In the 3D face recognition area the more frequently used techniques are the PCA [7], the point signatures [24], and the isometric transformations [4]. Unlike the 2D, illumination and pose variation do not affect 3D face recognition. In order to simplify the face recognition problem, in this paper we have simply used a matching operator belonging to the ImageMagick Library [13] which does a simple pixel by pixel matching between two images. However, this does not reduce the generality

of the approach, since the focus of this paper is not on face recognition, but on performance analysis of ARIA workflows.

The paper is organized as follows. The ARIA system is briefly described in Section 2, and procedure to transform an ARIA workflow in an equivalent Colored Petri Net is outlined in Section 3. Techniques to obtain parameters for the equivalent Colored Petri Net models are presented in Section 4. The proposed methodology is then applied to the face recognition example in Section 5. Finally, the paper concludes in Section 6.

2 The ARIA system

ARIA [19], ARchitecture for Interactive Arts, is a middleware for describing and executing media processing workflows [17]. Its main task is to process, filter, fuse sensory inputs and actuate responses in real-time on an intelligent stage. The intelligent stage is equipped with a matrix of floor sensors for object localization, a microphone array for sound localization, plus a beam forming and a motion capture system.

Using this system we can specify mappings from sensory inputs to audio-visual responses. Based on the specifications, the sensory inputs are streamed, filtered and fused, and finally they actuate controllable projection, surround sound and lighting systems. The key feature of the ARIA system is that responses take place in real-time and satisfies QoS requirements, allowing live performances.

The ARIA system has been developed by the Arizona State University at the Computer Science and Engineering Department. It offers a new medium which allows artists to integrate novel sensing, interaction, content, and response mechanisms in their stage performances. This enables digital media and art researchers to expand the expressive and interactive possibilities in performative, multimedia environments.

An ARIA media processing workflow [15] describes how the data, sensed through media, will be processed and what audio-visual responses will be actuated. Figure 1 shows an example of an ARIA model of an intelligent stage where there are three types of sensors (3D Tracker, Pressure Sensor and Microphone), three types of filters (3D Label Differentiator, Pressure Label Differentiator and Pitch Extractor), five operators (Shape Detector, Trace Detector, Image DB Access, Audio DB Access and Distance Calculator) and two types of actuators (Projector and Speaker).

Workflows [18] are modeled as directed graphs where nodes represent sensors (data acquisition), filters and fusion operators (data manipulation) and actuators (data output). Edges are used instead to define connections that stream objects between components [5]. Each path in the media workflow graph, describes how objects are streamed between ARIA processing components.

The basic information unit is a data object, which can be as simple as a numeric value or as complex as an entire image. Each object has two types of information: *object payload* and *object header*. The *object payload* can be a numeric value, a string or an image. The *object header* consists of two fields:

- *object property descriptor* which describes the object state, and contains information such as object size and object precision
- *object history descriptor* that containing a set of *timestamps* acquired by object's predecessors as they go through various operators. These informations are automatically generated by the library, and store the time at which a particular object was accessed. This data gives information about the overall age of an object and the corresponding resource consumption.

Sensors are the ARIA object sources, which can make objects available at different frequencies, sizes, and precisions.

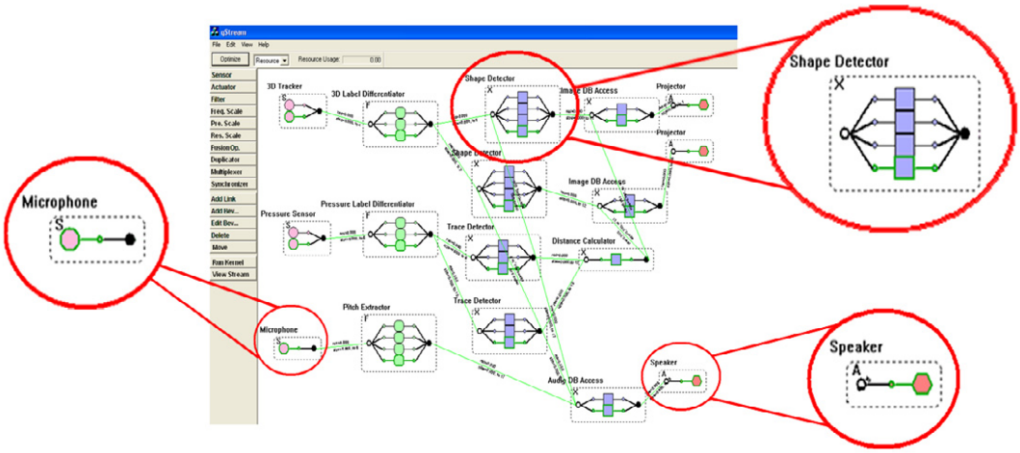


Fig. 1. Example of a general ARIA GUI media workflow.

Operators are programmable components whose delay and quality characteristics can be controlled via input parameters. They have input and output queues and a set of behaviors. During its computation, an operator picks a set of objects from each of its input queue and applies one of its behaviors. It then builds an object result and puts it into the output queues.

Filter and fusion operators are adaptable components that can provide multiple precisions. They perform analysis, aggregation and filtering operations such as information clustering and data cleaning. The main difference between filter and fusion operators is that filters take a single object stream as input, while fusion operators take multiple input object streams.

Each operator can have n input queues and m output queues. Each queue has a bound on the number of objects and a bound on the memory occupancy (in bytes). Each queue is monitored by a manager that in the case of overflow, can remove some of the objects according to a drop policy. The drop policy can be based on some of the parameter of the object, like its age, size, precision, history, amount of resources used so far, etc.

Every queue has a *window size* (which denotes the number of objects that must be present in the queue for the operation to take place), an *object priority criteria*

(that determines the order in which objects in the queue are considered) and an *object ignore criteria* (that determines which objects in the queue should be ignored). Each behavior is essentially a different implementation, with different processing, delay and quality characteristic. A particular behavior is chosen depending on an *execution condition*. When an operator has multiple behaviors ready for triggering (i.e. there is more than one behavior whose condition is satisfied) the ARIA kernel picks the most appropriate one, based on the quality, delay, or resource constraints [16].

Looking at the example shown in Figure 1, we can notice that most of the nodes have multiple/alternative behaviors: this is a key feature that allows the system to adapt to the particular dynamic runtime characteristics, allowing it to respect its QoS specifications. These specifications may include constraints on sensor-to-actuator delay, precision, resource usage, and throughput. In general, due to the various transformations available during the processing of a media object, the system tries to find the best tradeoffs between desired characteristics, choosing for example the set of behaviors that minimize the delay, without increasing too much the perceptual distortion.

3 Colored Petri Net analysis of ARIA workflows

Colored Petri Nets [11] (*CPN*) are a modeling formalism used in performance evaluation and in many other fields. In particular, they are an extension of ordinary Petri Nets, where tokens are augmented with attributes. Attributes are called *colors*, and belong to specific classes called *types*. Each token has associated a set of types that defines its attributes. When a transition fires, it removes some of the tokens from its input places, and collects their attributes into *variables*. At the same time, the firing of a transition inserts tokens into its output places. The attributes of the generated token are computed as functions of the variables collected from the input places. For a tutorial on CPN, the reader can refer to [10].

In this paper we will transform each ARIA component in a block of a CPN. In particular we will model multimedia objects with *tokens*, operators, actuators and sensors with *transitions*, and queues with *places*.

3.1 ARIA objects

ARIA objects are modeled by CPN tokens. We do not include the *object payload*, and we focus only on the *object headers*. We also neglect the *object history*, and we focus only on the *object properties*.

Object properties are modeled using colors and types. Each property of a model is stored in a type. In order to obtain more tractable models, the set of possible values that a property can assume is partitioned into subsets. A different color is used to represent properties whose value falls in that particular subset.

Consider for example an object that represents a JPEG compressed image. The object has two properties: the compression quality (an integer between 1% and 100%), and the image resolution (a product of two integers). We represent the

(a)

Quality	Color
1% - 10%	q_1
11% - 40%	q_2
41% - 90%	q_3
91% - 100%	q_4

(b)

Resolution	Color
$0Mp$ - $0.1Mp$	r_1
$0.1Mp$ - $0.4Mp$	r_2
$0.4Mp$ - $1Mp$	r_2
$1Mp$ - $2Mp$	r_2
$> 2Mp$	r_5

Table 1
JPEG image object attribute classes

object with a token with two attributes $q \in \mathbf{q}$ and $r \in \mathbf{r}$. Depending on the original object quality, attribute q will be chosen according to Table 1(a). In a similar way, depending on the resolution of the original object, attribute r will be set according to Table 1(b). For example the color attributes of a 800×600 ($0.48Mp$) image compressed at 80%, will be (q_3, r_2).

3.2 Sensors and Actuators

Sensors and *actuators* are modeled using CPN transitions. A *sensor* is implemented by a transition that has only one output arc (see Figure 2a). In a similar way, an *actuator* is modeled by a transition with one input arc (see Figure 3a). However, an *actuator* is a bit more complex, since it must store the data it receives in a queue. This queue is modeled by a CPN place.

The firing of a transition representing a *sensor* models the acquisition of a multimedia object. The attributes of the multimedia object are injected into the color of the produced token, according to the procedure described earlier. In a similar way, the firing of a transition representing an *actuator* models the consumption of a multimedia objects. In this case the attributes of the token might be used to determine particular aspect of the specific action performed by the actuator. For example the attribute representing the resolution of a token that corresponds to an image, can be used to determine the firing time of a transition that models the transfer of the picture to a storing device.

Both *sensors* and *actuators* can have an additional *state* that can be used to better characterize its behavior. For example, a switch alternately generates events of "switch on" and "switch off". This might be included in the model by adding a place that represents the local state, as shown in Figure 2b and Figure 3b.

3.3 Operators

ARIA Operators process data acquired by *sensors*, and generates outputs that can then be routed to the *actuators*. Each operator may require several multimedia objects to perform its task. Each Operator may also generate several multimedia objects as a result of its operation. Operators are thus implemented as CPN transitions (see Figure 4a), with one input arc for each multimedia object required,

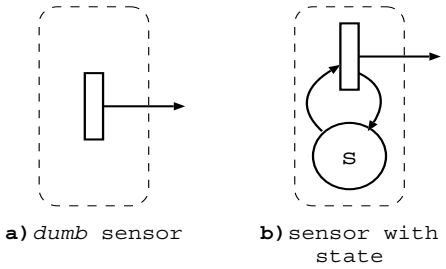


Fig. 2. CPN submodel of a *sensor*: a) dumb, and b) with internal state.

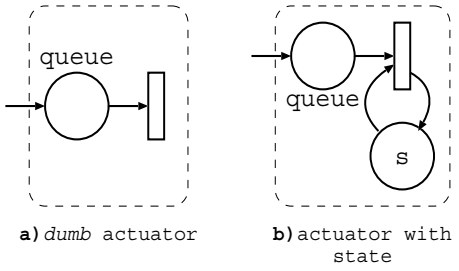


Fig. 3. CPN submodel of an *actuator*: a) Dumb, b) with internal state.

and one output for each multimedia object produced. Each input object should be enqueued, and queues are modeled by places interposed between the input and the transition. Operators may also have a local state as *sensors* and *actuators*. In this case memory will be implemented using a specific CPN place (not shown in Figure 4a for brevity) as it was done for both *sensor* and *actuators*.

One of the key features of ARIA operators is the ability to have different implementations, each with its own characteristics and parameters. In the CPN submodel of an operator, each implementation is modeled by a separate transition, connected to same set of input and output places (see Figure 4b). Each transition is then connected to place that enable or disable it. The behavior currently in use is characterized by the presence of the token in the corresponding enabling place. In this case, a domain specific sub-model must be included to dynamically select the correct behavior depending on the condition of the CPN.

3.4 Synchronization, Reset and Queue management

ARIA components can be very complex objects that might implement very sophisticated algorithms, which require synchronizations, interactions and specific queuing policies. The proposed translation procedure is perfectly capable to model basic operators. However, in order to produce accurate models, the advanced features mentioned above should be taken into account. Figure 5 shows a way in which these features can be included in the model. In particular, as a programmer is

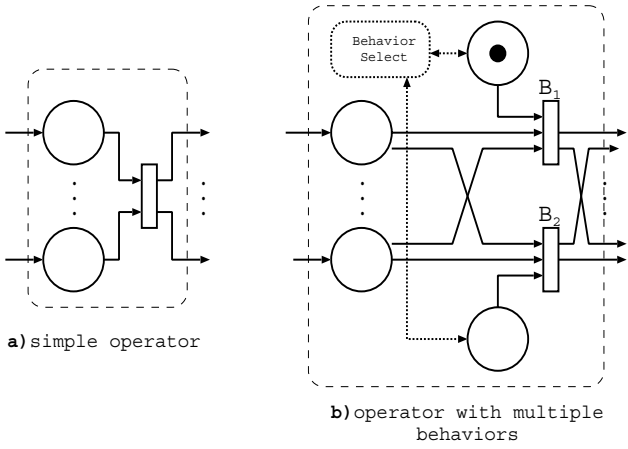


Fig. 4. CPN submodel of an Operator: a) Simple operator, B) Operator with different behaviors.

required to write specific fragments of codes to implement advanced synchronization and system management policies, here a modeler is required to write specific CPN sub-model to obtain the same result. However, some interaction might be standardized, as shown in Figure 5. In particular:

Synchronization, that is the ability of an event to occur only when specific conditions hold, can be implemented by adding a specific sub-model. The purpose of this sub-model is to model the synchronization protocol inter-connecting the transitions of the *sensor*, *actuator* or *operator* involved.

Reset, that is the possibility to restore a specific state of a state-based component, can be implemented by connecting the transitions that represent the events that reset the *sensor*, *actuator* or *operator*, to the place that models their state with both input and output arcs. The purpose of the input arc is to empty the place, and the one of the output arc is to fill it with the required state.

Queue management defines policies that a specific *actuator* or *operator* uses to manage its input objects. In this case it might be implemented by adding a sub-model that intercepts the firing of the transition representing the component, and operates the required actions on the place representing the corresponding queue.

4 System Measures

In order to define the performance parameters (such as the transition firing-time distribution) of the CPN composed using the blocks described in section 3, we have modified the ARIA system to measure the single operators in isolation.

4.1 How extracts the statistics

The arrival and departure time can be obtained from the timestamps of the *object history descriptor* introduced in Section 2. This set of timestamps contains informa-

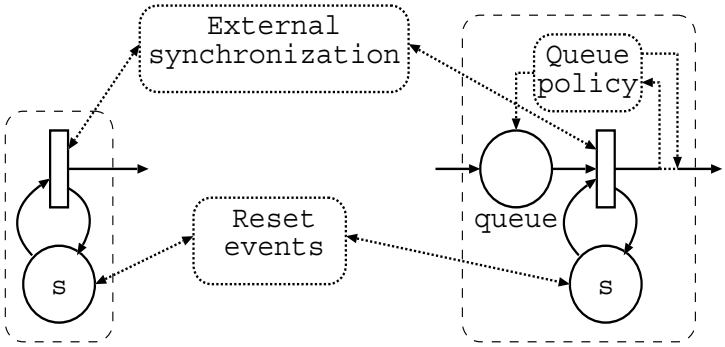


Fig. 5. Example of *synchronization*, *external reset* and *queue management* in CPN model of ARIA workflow.

tion about the time in which each object went through sensing, filtering, and fusion operations. This set of data may also be used to calculate the total time delay incurred by the object. From these values we can extract the arrival and departure time of all the objects arrived or departed from the input and output queues.

For example, to compute the processing time distribution of the matching operator described in the next session, we used a timer that counts the seconds elapsed from the start to the end of the operation. In this case, the operator was implemented using the the ImageMagick Library [13], and times were considered using the timer structure implemented in the library. This allowed us to optimize the counting operation and to reduce the errors due to the use of external data structures.

Instead, to measure the end-to-end delay of the whole system, we used measurement obtained using the standard C function *time*().

ARIA objects are very complex entities, which are characterized by several parameters. Most of the operators changes or defines the values of these parameters, and they greatly influence the behavior of the downstream nodes. It is thus very important to characterize how values changes between the operators.

We have chosen to model the evolution of the system in a probabilistic way. Let us call i the set of input parameters, of all the input queues to an operator. We define the output parameters o, according to a conditional probability distribution $P(o|i)$ that corresponds to the probability of generate a value o in output, given that the value of the parameter in input is i.

We can compute these probabilities by analyzing the data obtained by the measures performed on the system. In the case of the study of the matching operator, for example, the distribution of the output value is the measured on a spreadsheet of data collected from various experiments. The model, however, requires only a binary variable that tells whether a match has been found. We can exploit the output value probability distribution, by using a spreadsheet where we define a threshold and we apply it to the output value to determine the state of the match: we will have a positive match only if the value of the output is below this threshold. In par-

ticular we used 3 different threshold values, such that: a) unsure match are accepted and the system grant the access also to persons whose identity is not secure (*loose* threshold); b) the majority of unsure match are rejected, and the access is granted only if the system if sure enough of the employee identity (*medium* threshold); and c) all the unsure match are rejected, and the access is granted only if the system is absolutely sure of the identity of the employee (*tight* threshold).

This design allows to easily change the threshold value during the analysis of the data to study the system behavior, and indirectly add the possibility to use different threshold in the CPN model. The statistics were taken separately for different input characteristic: in this way we have been able to determine the conditioned probabilities.

5 System Implementation

In order to test the proposed methodology, we created a model of a simple ARIA application that describes an automatic office entrance door, with employees face recognition. The architecture of the application is composed by four elements: two sensors, an operator and an actuator. The logic model is presented in Figure 6. The first sensor is a camera that acquires the image from the real world, the second sensor instead reads continuously a database containing an image library. The operator takes the two images provided by the sensors and applies the match operation, after this takes the result and sends it to the actuator (turnslide). The actuator takes the result from its queue and according to its values performs the appropriate action, by either granting or denying the access.

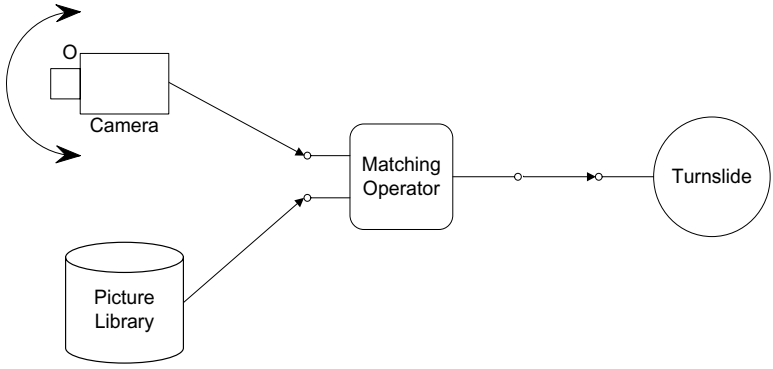


Fig. 6. The implemented ARIA workflow.

5.1 Face recognition example

In this specific example the first sensor is the camera before the office door that acquires the employee images, while the second sensor reads the databases containing the registered employee pictures. The operator does the matching and sends the numeric result to the actuator that analyzes it. If the result is under the threshold the match is positive and so the actuator opens the door giving the access to the

employee, otherwise it doesn't open the door. The corresponding model of this example implemented in ARIA is show in Figure 7.

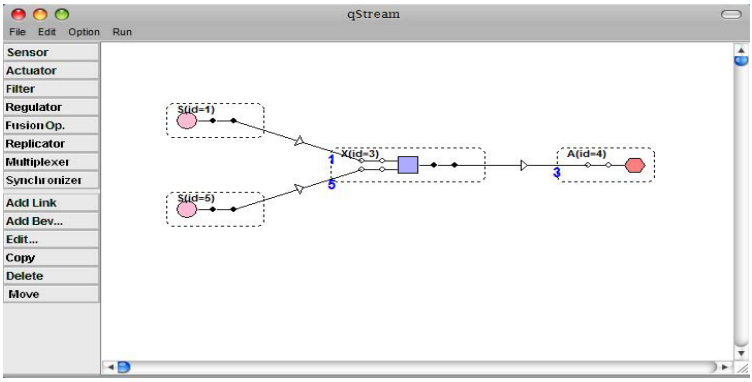


Fig. 7. The face recognition example implemented in the ARIA system.

5.2 Data Set

For the execution of the example we have used the Computational Vision Frontal Face dataset [8]. This is composed by 450 face images, with 27 unique people with different lighting, expressions and backgrounds. We have chosen a subset of this database and modified it by placing each of the characters over the same background. This has been done to allow the use of a simplified matching algorithm.

We built a database of 20 employees with a single face expression, see Figure 8. For the input we selected 9 people and for each of them we have chosen an image with a different expression, see an example on Figure 9. Of these 9 people 6 have access permission, and they have a match in the database, while the other 3 have no corresponding images. The goal of the system is to prevent the access to these three people who are trying to sneak in.

For the simulation we divide the dataset in various groups depending on resolution and quality. We used three different resolution: 896×592, 512×348, 320×240; and three different quality values: 99%, 50%, 8%.

We remind you that these are all JPG images so the attribute quality corresponds to the compression level.

5.3 Description of the statistics extracted from the operators

To perform the statistical analysis introduced in Section 4, we have extracted from the ARIA system a set of key information regarding the image characteristics, the operators processing times and the queue waiting times. For every image processed by the system we have collected several pieces of informations, including: the *file name*, *resolution* (expressed on pixels), *quality*, *file size* (expressed on bytes), *number of colors*, *metric used* and *returned difference*.

The *quality* parameter refers to the compression level used by JPEG, PNG, and MIFF images. The *metric* is an option of the matching that express how this

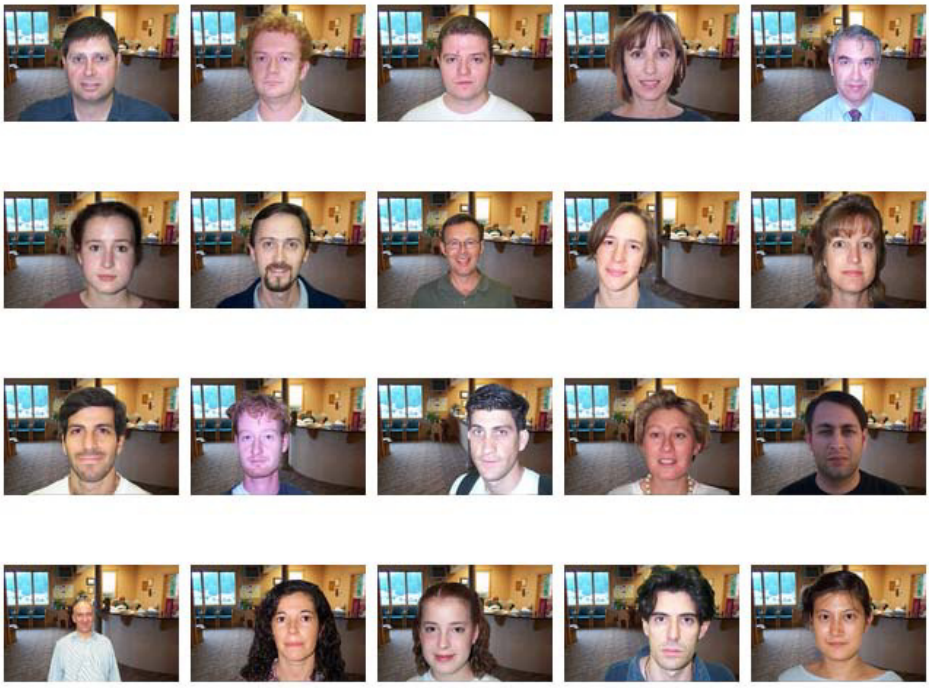


Fig. 8. Database employees' images.

Fig. 9. Example of input images. Three different expressions of the same person.

operator performs the match of two images. The ImageMagick Library used in this example allowed the use of several algorithms, like for example the *absolute number of different pixels* or the *mean absolute error*. The $difference$ is the output returned by the matching operator that express the difference between the two images as a numerical value.

To analyze the service time distribution of the matching component, we have extracted the time processing of the matching operator. This was done by computing the difference of the time when the image was put to the output queue minus the time in which the same image arrived to the input queue.

To characterize the arrival and departure object distributions we have extracted the time value when the image arrived to the input queue of the sensor and the time value when the image left the output queue of the actuator.

The last parameter computed is a truth value that shows if the input image

	Input images		Database images	
Run	Resolution	Quality	Resolution	Quality
1	896×592	99%	896×592	99%
2	512×348	99%	512×348	99%
3	512×348	99%	512×348	50%
4	512×348	99%	512×348	8%
5	320×240	99%	320×240	99%

Table 2
The five run executed during the simulation experiments

match with the image on the database or not.

5.4 *Organization of the simulation experiments*

To collect the data, we have performed five run of simulation experiments, each with different resolution and image quality. The five experiments are listed in Table 2.

The matching operator can compare only images with the same resolution, so we compared separately the three resolutions.

For the medium resolution we tested also different qualities on the database. We changed only that because we assume that the quality of the image captured by the camera cannot be changed. This parameter however has a great impact on the database, since an increase in quality, produces also an increase in the dimension of the images file size.

5.5 *Matching operator*

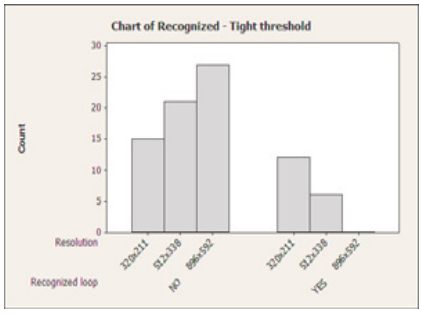
Fig. 10. Application of the matching operator at two example images.

The matching operator used in this application compares one picture with another, by subtracting the "brightness value" of corresponding pixels from the two images and taking the absolute value. This produces an image that has bright values where the images are different and dark values where the two pictures are nearly the same. An example of image comparison is reported in Figure 10. Note that this operator can be used for face recognition only because we have created a specific database of images. In real application, more sophisticated algorithm have to be used.

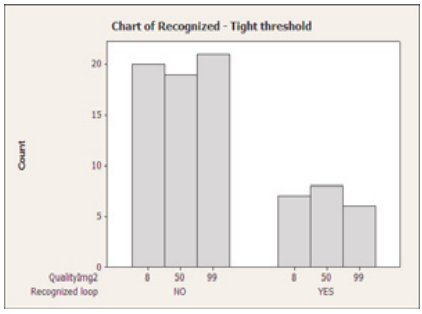
5.6 Analysis of the Matching component in isolation

We start by analyzing the changes of the results, as function of both the resolution and the threshold value. Observing Figure 11 we can see that with the loose and the medium threshold value the results are almost independent of the resolution. Instead with the tight threshold the only resolution that produces good results is 896×592. So if we want a perfect recognition we must use the biggest resolution.

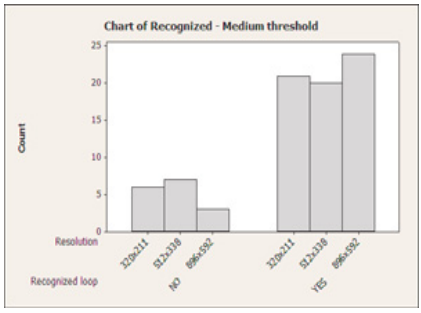
Now we analyze the behavior of the system with images of the same resolution, 512×348, but compressed at three different qualities: 99%, 50% and 8%. From the Figure 11 we can see that there isn't a quality value that performs much better than the others, but the biggest value, 99%, seem to be the best choice.



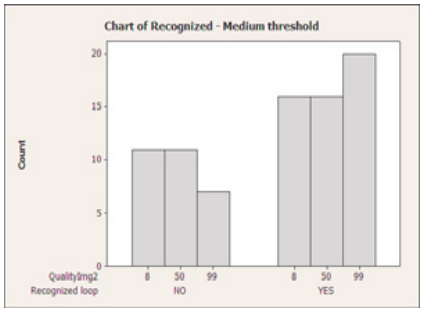
(a) Tight threshold: Resolution



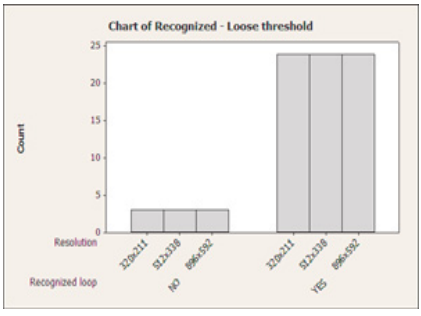
(b) Tight threshold: Quality



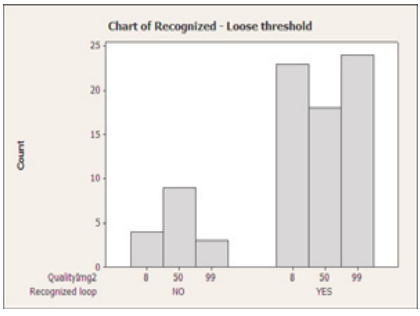
(c) Medium threshold: Resolution



(d) Medium threshold: Quality



(e) Loose threshold: Resolution



(f) Loose threshold: Quality

Fig. 11. Bar chart of the results of recognition divided by quality and resolution, and with different threshold value.

After this analysis we can conclude that the resolution can be an important parameter for the recognition, and that larger resolutions provide better results. Quality instead, has not a big impact, but also in this case the higher is the better.

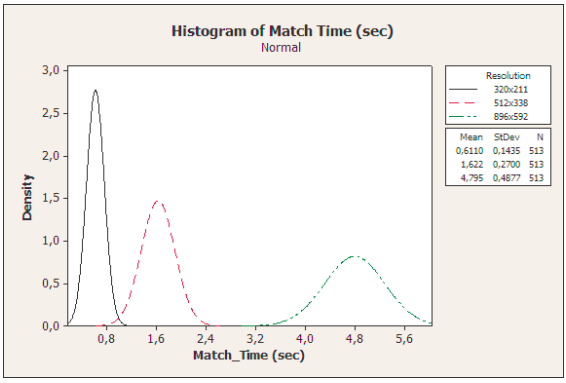


Fig. 12. Histogram of the Matching Time grouped by resolution.

We then analyze the Matching Time, i.e. the time necessary at the operator to make the matching operation between two images. If we watch Figure 12 where the data are divided by resolution, we clearly see that the data are very different. This means that there is a correlation between the resolution and the Matching Time.

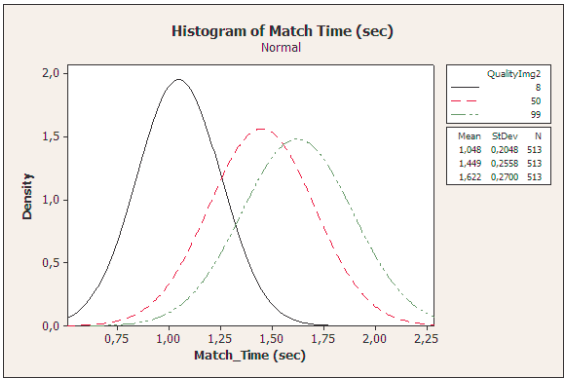


Fig. 13. Histogram of the Matching Time grouped by quality.

We then analyze time as function of the quality variable, Figure 13. The data are overlapped, especially those relating to the values of lower quality. Therefore we cannot conclude that there are any relationship between this variable and the Matching Time. We can then conclude that the Matching Time increases only when we increase the resolution.

These two results are in accordance with what one may expect, since the match operator performs a pixel by pixel matching. This means that if there are more pixels to analyze, the process will take longer. But not only, in fact an image of larger resolution will also have a larger size, and the load operator will also require more time to load the image.

All these statistics are important because they can allow us to simplify the Colored Petri Net model that will be built in the next section.

5.7 Colored Petri Net Model

Figure 14 presents the CPN model of the face recognition example described in Section 5.1. The model has been produced using the procedure described in Section 3.

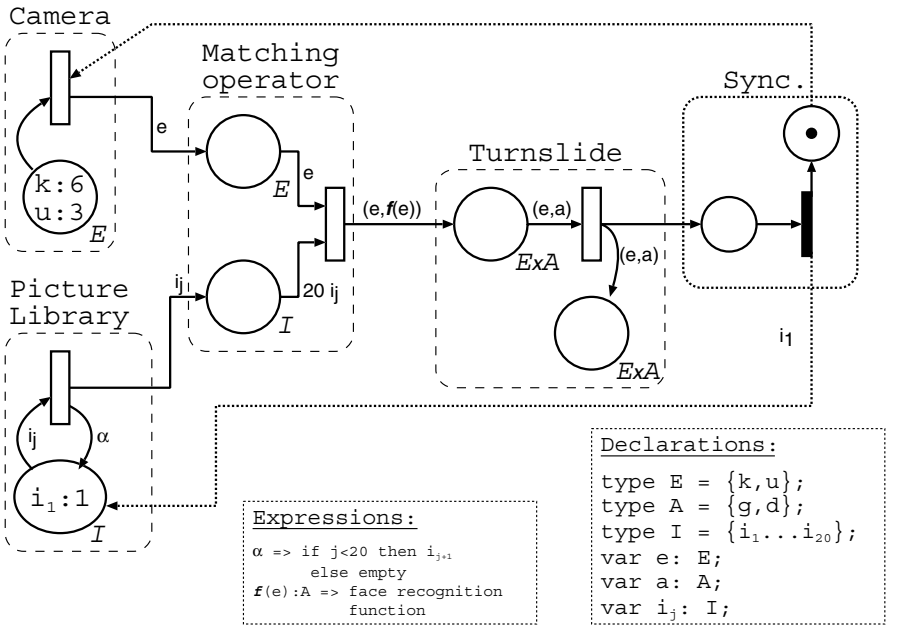


Fig. 14. The CPN corresponding to the face recognition workflow presented in Figure 6.

The *Camera* sub-model models the sensor that acquires an image of the incoming employee. Its firing corresponds to the entrance of a new employee. In this example, we imagine that 9 employees will arrive in front of the camera: 6 belonging to the DB, and 3 not included in the images library. Each token representing an arrival has an attribute e that belongs to the set $E = \{k, u\}$. When $e = k$, the employee has his picture stored in the DB, while $e = u$ represents an unknown visitor. The sensor has a state that is used to count the number of visitors that still have to enter the face-recognition system. The corresponding place is initialized with 3 tokens of color u and 6 tokens of color k.

The *Picture Library* component represents the employee DB. It is implemented as a sensor that loads the images of the DB in the matching operator. Each picture is modeled as a token with an attribute i_j belonging to the set $I = \{i_1, \ldots, i_{20}\}$. The state of the sensor keeps track of the next image to be considered. Every time an image is loaded (firing of the corresponding transition), the DB starts loading the next one, until all the pictures have been considered. When the last image has been loaded, the place that represents the state is emptied. This prevents new objects from being generated until the operator is reset.

The *Matching Operator* performs the actual match between the image of the arriving employee e with all the 20 the pictures in the DB. For this reason the

Size	320 × 240			512 × 348		
Data	Measure	Simulation interval		Measure	Simulation interval	
Mean end-to-end delay	403.67s	394.8s	418.76s	503.33s	491.47s	519.43s
Correct	0.48148	0.429263	0.472959	0.44444	0.448914	0.502197
False positive	0.22222	0.333333	0.333333	0.11111	0.257911	0.286533
False negative	0.29630	0.193707	0.237404	0.44444	0.228075	0.276369

Table 3
Result of the simulation divided by different sizes

corresponding transition can fire only when all the 20 images have been loaded into its input place. The matching is performed by function $f : E \to A$, with $A = \{g, d\}$, that returns g if there is an image among the 20 that matches the one in input, and d if it is not included in the DB. The result of the matching $a = f(e)$, together with the attribute e that represents whether the employee being considered is known or not, are stored in the attributes (e, a) of the token generated by the transition that represents the operation. This token is immediately passed to the sub-model that represents the Turnslide.

The *Turnslide* models the actuator that opens the door or denies the access to the visitor. It has a state, represented by a token whose color belongs to the type $E \times A$, that keeps track of the considered employees, and sorts them between correctly identified (k, g) and (u, d), erroneous rejection (k, d), and erroneous recognitions (u, g).

The *Sync* sub-model models the synchronization among the various components. In particular, it allows only one employee to enter at a time, by blocking the transition that represents the sensor used to model the entrance of a new employee. It also reset the DB, by restoring the state of the corresponding actuator to the first image.

5.8 Result comparison

In the Table 3 we show the comparison between the result obtained through measurements of the execution of the ARIA system and those obtained by simulating the corresponding Colored Petri Net model. Transition firing time distributions, and the matching function $f(e)$ were computed using the techniques presented in Section 4. Colored Petri Nets where solved using an ad-hoc discrete event simulator.

As we can see, we have a good match for what concerns the mean end-to-end delay, and the probability that system performs a correct identification of the incoming employ. Results concerning wrong identification classified either as False positive or False negative, are much less accurate. These measure however where taken on a very small population (just 9) individuals, and this explains some of the fluctuations in the results.

The proposed methodology allows for example to study different matching policies (for example loading the images only when required, and stop as soon as a match is found) or queuing strategies (for example start the matching of the new employee, while the turnslide is opening to let the previous employee in) by applying only simple changes to the CPN, and see whether they might improve the

performance significantly.

6 Conclusions

In this paper we have proposed a methodology to transform an ARIA model into a CPN with a semi-automatic procedure. This could be the basis for a tool that might allow the design of an ARIA system before actually building it. We have also presented a methodology to study the single components in isolation, and then use the results to set the parameters of the generated CPN. Moreover, the methodology proposed here could be exploited to study similar component based systems, such as the one used in other similar application domain like flexible-manufacturing.

Acknowledgement

We would like to thank Prof. Maria Luisa Sapino (University of Torino - Italy) and Prof. K. Selçuk Candan (University of Tempe - Arizona) for their precious contribution in completing this work.

References

[1] *Aria project* (cited July 2008). URL http://aria.asu.edu/

[2] Babu, S. and J. Widom, *Continuous queries over data streams*, SIGMOD Record **30** (2001), pp. 109–120.

[3] Blanz, V. and T. Vetter, *Face recognition based on fitting a 3d morphable model*, IEEE Trans. Pattern Anal. Mach. Intell. **25** (2003), pp. 1063–1074.

[4] Bronstein, A. M., M. M. Bronstein and R. Kimmel, *Expression-invariant 3D face recognition*, in: *Proc. International Conference on Audio- and Video-based Biometric Person Authentication*, Lecture Notes in Computer Science **2688**, Guildford, UK, 2003, pp. 62–70.

[5] Candan, K. S., G. Kwon, L. Peng and M. L. Sapino, *Modeling adaptive media processing workflows*, ICME (2006), pp. 573–576.

[6] Carney, D., U. Çetintemel, A. Rasin, S. B. Zdonik, M. Cherniack and M. Stonebraker, *Operator scheduling in a data stream manager*, in: *VLDB*, 2003, pp. 838–849.

[7] Chang, K. I., K. W. Bowyer and P. J. Flynn, *Face recognition using 2D and 3D facial data*, in: *ACM Workshop on Multimodal User Authentication*, Santa Barbara, California, 2003, pp. 25–32.

[8] *Computational vision at caltech* (cited July 2008), 2008 - California Institute of Technology. URL http://www.vision.caltech.edu/

[9] Gómez, C. and B. Pesquet-Popescu, *A simple and efficient eigenfaces method*, in: *ACIVS*, 2007, pp. 364–372.

[10] Jensen, K., *A brief introduction to coloured petri nets*, in: *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems workshop. - TACAS97*, Enschede, 1997, pp. 203–208.

[11] Jensen, K., "Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use (2nd edition)," Springer, 1997.

[12] Kim, H.-C., D. Kim, S. Y. Bang and S.-Y. Lee, *Face recognition using the second-order mixture-of-eigenfaces method*, Pattern Recognition **37** (2004), pp. 337–349.

[13] *Llc imagemagick studio* (cited July 2008), 1999-2008. URL http://www.imagemagick.org/

[14] Lucey, S., *The symbiotic relationship of parts and monolithic face representations in verification*, cvprw **05** (2004), p. 89.

[15] Peng, L. and K. S. Candan, *Adaptive multi-sensor, multi-actuator media workflow system for interactive arts*, in: *ICDE Workshops*, 2005, p. 1171.

[16] Peng, L., K. S. Candan, C. Mayer, K. S. Chatha and K. D. Ryu, *Optimization of media processing workflows with adaptive operator behaviors*, Kluwer Academic Publishers **33** (2007), pp. 245 – 272.

[17] Peng, L., K. S. Candan, K. D. Ryu, K. S. Chatha and H. Sundaram, *Aria: an adaptive and programmable media-flow architecture for interactive arts*, in: *ACM Multimedia*, 2004, pp. 532–535.

[18] Peng, L., G. Kwon, K. S. Candan, K. D. Ryu, K. S. Chatha, H. Sundaram and Y. Chen, *Media processing workflow design and execution with aria*, in: *ACM Multimedia*, 2005, pp. 800–801.

[19] Peng, L., G. Kwon, Y. Chen, K. S. Candan, H. Sundaram, K. Chatha and M. L. Sapino, *Modular design of media retrieval workflows using aria*, International Conference on Image and Video Retrieval (2006), pp. 491–494.

[20] Phillips, P. J., P. J. Flynn, W. T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min and W. J. Worek, *Overview of the face recognition grand challenge*, in: *CVPR (1)*, 2005, pp. 947–954.

[21] Sclaroff, S. and A. P. Pentland, *Modal matching for correspondence and recognition*, IEEE Trans. Pattern Anal. Mach. Intell. **17** (1995), pp. 545–561.

[22] Shah, M. A. and S. Chandrasekaran, *Fault-tolerant, load-balancing queries in telegraph*, in: *SIGMOD Conference*, 2001, p. 611.

[23] Tsalakanidou, F., D. Tzovaras and M. G. Strintzis, *Use of depth and colour eigenfaces for face recognition*, Pattern Recognition Letters **24** (2003), pp. 1427–1435.

[24] Wang, Y., C.-S. Chua and Y.-K. Ho, *Facial feature detection and face recognition from 2D and 3D images*, Pattern Recognition Letters **23** (2002), pp. 1191– 1202.