9

Actors and Virtual Time: an Experience using Time Warp, Timed Petri Nets and Cellular Networks

R. Beraldi, L. Nigro, F. Pupo
Dipartimento di Elettronica, Informatica e Sistemistica
Università della Calabria, I-87036 Rende(CS) - Italy
Voice: +39-984-494748, Fax: +39-984-494713
email: {r.beraldi,l.nigro,f.pupo}@unical.it

Abstract

This paper describes an actor based framework which allows the development of distributed time dependent applications. The approach is centred on light-weight actors and on reflection techniques which provide a time-sensitive message scheduling structure. As a significant application, the paper reports an experience using an implementation of a Time Warp mechanism upon which an effective simulation model of generalised timed Petri nets (TPN) is built. The TPN structure is then used to analyse a formal model for large cellular networks. Some performance measures are finally provided.

Keywords

Actors, reflection, modularity, virtual time, Time Warp, timed Petri nets, cellular networks

1 INTRODUCTION

The Actor model (Agha, 1986) is a well established computational framework suitable for general, time independent, distributed applications. The model is based on the concept of actors, i.e., autonomous agents which communicate to one another by asynchronous and buffered messages. An actor hosts an internal thread on behalf of which a message at a time, received into the actor mail queue, is fetched and processed. Message execution is atomic. At the end of a message execution the actor is ready to process the next message and so forth.

In the last years, Agha et al. (Ren, 1995) (Ren, 1996) (Saito, 1995) have proposed extensions to the actor model in order for it to be applicable to real-time systems. The extensions rely on capturing message *interaction patterns* among actors through a *RTsynchroniser* construct (Ren, 1995). RTsynchronisers are declarative in character. They involve a group of actors and specify constraints on the execution of relevant messages. Constraints reflect upon timing issues of messages (i.e., its invocation time) and the visible state of observed actors. RTsynchronisers provide an elegant and formal tool to control message exchanges. Their concrete application depends on the possibility of ensuring a global time notion in a distributed system. Moreover a selected scheduling structure is required in order to guarantee the timing constraints of messages are ultimately met. A major benefit of the use of RTsynchronisers is modularity, which stems from a separation of functional from timing requirements in a development. Actors are firstly defined according to functional issues only. Then timing aspects are separately specified through RTsynchronisers which affect scheduling.

The work described in this paper aims at experimenting with the concept of actor programming for distributed time sensitive applications. An architecture - DART (Nigro, 1995) (Nigro, 1996a) (Kirk, 1997) Distributed Architecture for Real Time- has been designed to be exploitable in popular object-oriented languages (e.g., C++, Java, ...). Key factors of DART are a light-weight notion of actors which are orchestrated by a customisable reflective control machine which regulates scheduling on a per processor basis. The control machine can reason upon virtual or real time. The control machines of a distributed application can co-operate according to an interaction policy in order to fulfil system-wide timing constraints. A control machine depends on a scheduler actor whose responsibility is to filter messages and to reason upon timing constraints of groups of related application actors. A scheduler can implement a set of RTsynchronisers in the sense of (Ren, 1995).

This paper focuses on virtual time. Section 2 gives an overview to DART concepts using the sequential simulation paradigm. In section 3 the basic principles of distributed simulation are briefly reviewed and the features of an implemented actor-based Time Warp mechanism described. Section 4 presents a

timed Petri nets formalism together with an effective execution model which fits naturally in the context of the developed Time Warp kernel. As a significant application, section 5 shows the use of TPNs in the modelling and analysis of the time behaviour of large cellular systems. The speedup of the distributed simulation of a modelled cellular system on a heterogeneous computing environment is shown. Finally, the conclusions and some directions of future work are presented.

2 AN OVERVIEW OF DART

DART (Nigro, 1995) (Nigro, 1996a) (Kirk, 1997) is a specialisation of the Actor model designed for the development of distributed and time dependent systems. At the programming in-the-large level a system consists of a collection of subsystems, one per processor, interconnected through a (possibly deterministic) communications system (e.g., CAN (Kirk, 1995)). Each subsystem hosts a control machine, a scheduler actor and a set of application actors. Discipline of programming in-the-large can suggest having only one *administrator* actor being referenced at the system level by a unique identifier. It receives message requests and possibly delegates its processing to inner subsystem actors. The control machine provides the necessary support for scheduling and message select and dispatch.

Actors are the basic building blocks in-the-small. An actor is modelled as a finite state machine which evolves through a lifecycle (Shlaer, 1992), i.e., a succession of states (behaviours). It is a reactive object which responds to incoming messages on the basis of current state and message contents. Message reception is implicit. An actor is at rest until a message arrives. Message processing triggers a state transition and the execution of an action. Action execution is atomic and cannot be pre-empted nor suspended. Messages can be unexpected in the current state. Their processing can be postponed by remembering them on local data or in states of the lifecycle. Basic operations on actors are: (a) *new*, for creating a new actor, (b) *become*, for changing the actor state, (c) (non blocking) *send* for transmitting messages to acquaintances actors (including itself). Differences from the Actor model are:

- an absence of an internal thread in actors. DART actors are instance of a (passive) class which extends the Actor base class. Life cycle is provided by a *message handler* (Reiser, 1992) which is invoked by the control machine at message dispatch time
- an absence of the mail queue per actor. Rather, a single message queue is introduced by the control machine upon which sent messages are scheduled, selected and dispatched according to a proper control structure

- source of concurrency among actors in a same subsystem is *action interleaving* ensured by the control machine dispatching scheme. Concurrency among actors allocated upon different processors is true parallelism.

Besides actors, DART allows for the existence of passive objects too which are lacking of a life cycle. Their methods can be invoked by the usual procedure-call synchronous semantics. Since the atomic character of action execution in actors, passive objects behave naturally as monitors and there is no need to introduce conventional mutual exclusion primitives.

2.1 Reflection and control machine

The dynamic behaviour of actors is influenced by a control discipline (Nigro, 1994) adopted at the control machine level. Here the control structure can be pure event-driven as in the Actor model, useful for concurrent applications, pure time-driven suited for hard real time systems (Nigro, 1996b), or a combination of the two. The control machine relies on the concept of reflection (Maes, 1987). It can reason on a time dimension and is causally connected with application actors. Basic components of a control machine (see also Figure 1) are:

- a *clock*
- a *message queue* (i.e., a calendar, plan or event list)
- a *controller*
- a (programmer defined) *scheduler* actor.

The control machine is fed of a particular scheduler at its initialisation. The time notion can be virtual or "real" time. In a simple case, messages are coupled with a *timestamp* indicating the occurrence time of the event captured by the message. More in general, for real-time applications, a message can be accompanied by a validity time window $[t_{min}, t_{max}]$ which specifies possible delivery times (Nigro, 1996b) (Kirk, 1997). The controller method of a simulation control engine repeats a basic message loop (sequential simulation engine). At each iteration

- a message is selected from the message queue and its timestamp used to adjust the simulated clock
- the message is delivered to its target actor by invoking the relevant message handler which in turn causes a state switch and triggers an action into execution
- at the action termination, the control loop is resumed and the scheduler is given a chance to schedule the "just sent messages".

The loop is repeated until the virtual time exceeds the simulation period or the simulation deadlocks.

The controllers of a set of distributed control machines can co-operate in order to fulfil system-wide synchronisation constraints. To this purpose, they can rely on a limited set of *control messages* which are hidden to application actors.

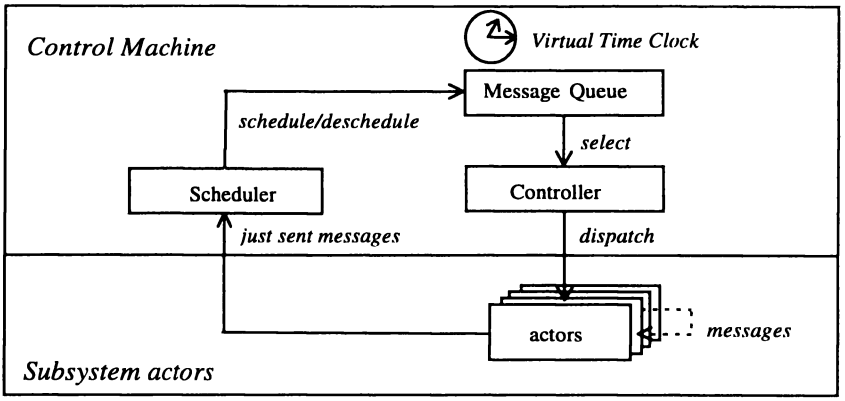

Figure 1 The organisation of a sequential simulation Control Machine

The time behaviour of a subsystem actors is specified separately from actor functional behaviour in a scheduler actor which knows application time requirements. The following interface is offered for programming a scheduler (Ren 1995) (Kirk, 1997):

- msg.*cause*() which returns the identity of the message whose processing generated msg
- msg.*iTime*() which returns the dispatch time of message msg
- *schedule*(msg, timestamp) which schedules message msg by attaching timestamp to it
- *deschedule*(msg, timestamp) which removes a previously scheduled message msg
- *now*() which returns the current time.

A scheduler hosts a set of *time clauses*. To exemplify, a cyclic actor of a class Source can have a message class Generate an instance of which is continually sent to itself for sustaining a generation process. The timestamp of the message can be defined by an exponential random variate. The following is a possible time clause for scheduling such a message (using a Java-like syntax):

```
if( msg instanceof Generate && msg.cause()==msg )
  schedule( msg, now()+Random.exponential( lambda ) );
```

where lambda is the generation mean of the physical process modelled by Source class.

A key factor of DART is a smooth transition from the event-driven to the time-driven paradigm, with actors which are not aware of the specific control discipline enforced by the control machine. This also favours modularity since actors can be reused according to different application requirements.

The control machine and scheduler actor can effectively be programmed using basic mechanisms (dynamic binding, polymorphism and runtime type identification) of an object-oriented language like Oberon-2, C++, Java ... The resultant actor programming style is safe, clean and sound (Nigro, 1995) (Kirk, 1997).

3 DISTRIBUTED SIMULATION AND TIME WARP

The simulation of complex, asynchronous and discrete event systems, e.g. formal models of large personal communications systems, can be very expensive in terms of the required computational resources (memory space and cpu time) on a single processor machine. Such simulations can potentially be accelerated by using parallel or distributed architectures. Different parts -*logical processes* LPs- of the modelled system are allocated on different processors and simulated locally by a sequential simulation engine. However, concurrent execution of messages related to different points in the simulation, on different LPs/processors, while is the source of possible speedup it also introduces a need of *synchronisation protocols* which are at the heart of the parallel or distributed simulation problem, and which can shorten the actual achieved speedup. Such protocols must ensure the ordering of events with respect to virtual time as in the sequential simulation, thus preserving the causality of events. Synchronisation mechanisms broadly fall into two categories: *conservative* and *optimistic*. Conservative protocols (Misra, 1986) (Fujimoto, 1990) prevents causality errors ever occurring (by blocking an LP would there be a chance to process an "unsafe" event, i.e., one for which causal dependencies are still pending). Optimistic protocols (Jefferson, 1987) (Fujimoto, 1990), using a *detection and recovery* approach, provide for an LP to redo the simulation of an event should it detect that premature processing of local events is inconsistent with causality conditions generated by other LPs.

Time Warp (TW) (Jefferson, 1987) (Fujimoto, 1990) is an optimistic strategy supporting distributed discrete event simulation. Every LP has a local clock (Local Virtual Time, LVT) and a separate message queue which drives a local simulation algorithm. LPs are allowed to proceed asynchronously, thus experimenting different values of LVTs at a same real time. However, if an LP receives an external message (*straggler*) whose timestamp is lower than LP's LVT, the basic synchronisation problem of TW arises. To avoid *causality errors* (the future can't influence its past!), the state of the LP must be *rolled back* at a virtual time less than or equal to straggler's timestamp, by undoing the effects of incorrectly performed forward computation. The undo process can propagate to partner LPs.

A rolled back LP proceeds again (*coasting forward phase*) into its future towards the straggler's timestamp. During this phase external messages are not re-sent. The straggler is then processed. After that forward computation is restarted according to one of two basic techniques: *aggressive cancellation* (AC) or *lazy cancellation* (LC) of previously sent (possibly erroneous) messages. *Anti-messages*

are maintained for any externally sent message. AC sends immediately all the anti-messages as a prompt undo request. An anti-message gives rise to a roll back in an LP if the corresponding positive message was already processed. LC refrains from doing so. Only in the case the original LP, during its new forward computation, doesn't re-generate an external message, the corresponding anti-message is sent. AC and LC have shown their relative performance into different application domains (Fujimoto, 1990).

TW is characterised by its necessary frequent saving of LP's state (*checkpointing*). The Global Virtual Time (GVT) is the so far committed simulation time of the whole simulation model. No LP can be rolled back at a time prior to GVT. As a consequence, each LP can free the memory space of no more useful state versions (*fossil collection*).

A critical issue of TW is how frequent GVT should be updated, since a high frequency conserves memory but wastes real time, a low frequency can result in an out-of-memory exception.

3.1 A DART based Time Warp Mechanism

Time Warp can be viewed as a particular interaction policy among the control machines of a DART simulation model. The following summarises key points of an achieved TW mechanism based on C++ and PVM (Geist, 1994). An LP includes a control machine and a collection of application actors (subsystem). A single LP is allocated onto one physical processor. A message is characterised by the tuple (*sender-LP*, *timestamp*). The timestamp consists of the pair (*ts*, *tr*) where *ts* is the "send time", i.e., the LVT of the sender LP at the moment of transmission; $tr \geq ts$ is the "receive time", i.e., the virtual time at which the receiver LP should process the message.

A distinguishing factor of the implemented TW is the adoption of a *modified aggressive cancellation* (MAC) (Beraldi, 1996) technique which operates as follows. An LP affected by roll back at time τ must undo all the erroneous computation prematurely carried out at times $\tau' > \tau$. This first requires, depending on the availability of state versions, a saved state to be withdrawn with a time $\tau'' < \tau$ and re-installed. After the coasting phase from τ'' to τ, the incorrect external computation is cancelled. In order to minimise the communication overhead, a single *undo* message is transmitted to each partner LP to which messages have been sent with $ts \geq \tau$. It requires the cancellation of all the messages sent by the rolled back LP, whose $tr \geq undo.tr$. The cancellation process annihilates positive messages still pending in the receiver LP or it raises a further roll back. MAC is capable of annihilating previously sent messages very quickly, a required feature in *self-driving* simulations (Carothers, 1994).

Performance of TW is critically affected by the values of State Save Rate (SSR) and Maximum Number of State Versions (MNSV) parameters (Beraldi, 1996). A value *s* of SSR indicates that the LP status will be copied just before the *s*-th LVT

change from the last saved version. A value *m* of MNSV specifies that the GVT updating procedure will be invoked after *m* state versions are stored on SVL and there is the need to checkpoint the LP again. A preliminary step for an effective use of TW consists of *parameter tuning*, which in turn is tied to load balancing conditions (see later in this paper).

4 DISTRIBUTED SIMULATION OF TIMED PETRI NETS

Petri nets (Murata, 1989) are widely used as a modelling tool for studying asynchronous concurrent systems. Quantitative properties of modelled systems can be evaluated by exploiting the notion of time explicitly added to the classical definition of Petri nets. In the following the class of timed Petri nets (TPNs) where timing information is associated to transitions (Ghezzi, 1991) (Chiola, 1993a) is considered.

A TPN is a tuple (P, T, F, W, τ, M) where

- P = $\{p_1, p_2, ..., p_{nP}\}$ is a finite set of P -elements (places),
- T = $\{t_1, t_2, ..., t_{nT}\}$ is a finite set of T-elements (transitions) with $P \cap T = \varnothing$ and a non empty set of nodes ($P \cup T \neq \varnothing$).
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs between P-elements and T-elements.
- $W: F \rightarrow \mathrm{N}$ assigns weights $w(f)$ to elements of $f \in F$ denoting the multiplicity of unary arcs between the connected nodes.
- $\Pi: T \rightarrow N$ assigns priorities π_i to T-elements $t_i \in T$.
- $\tau: T \rightarrow \Re$ assigns firing delays τ_i to T-elements $t_i \in T$.
- $M: P \rightarrow N^{+0}$ is the marking $\mu_i^{(0)} = \mu^{(0)}(p_i)$ of P-elements $p_i \in P$.

There are three important cases of firing delays, namely the case when $\tau_i=0$ (*immediate transitions*), the case when $\tau_i \in \Re$ is a deterministic time value (*deterministic timed transitions*) (Ghezzi, 1991), and the case when τ_i is an instance of a random variable (*stochastic timed transitions*). If T contains only stochastic timed transitions where the firing delay random variable is exponentially distributed, T belongs to the class of *Stochastic Petri Nets* (SPNs). *Generalized* SPNs *(GSPNs)* (Chiola, 1993a) allow a combination of non timed (immediate) and stochastically timed transitions.

4.1 Timed enabling and firing semantics

- Let $I(t)$ and $O(t)$ denote the set of input and respectively output places of $t \in T$. A transition $t \in T$ is *enabled* in some marking μ at time τ, iff $\forall p \in I(t), \mu(p) \geq w(p(t))$ in μ.

- If $E_\tau(\mu)$, i.e. the set of all transitions enabled in μ at time τ, contains immediate and timed transitions, then immediate transitions are always selected prior to timed transitions for firing.
- If $E_\tau(\mu)$ contains only timed transitions, and RET(t_i), $t_i \in E_\tau(\mu)$ is the remaining enabling time of t_i, then the transition t that *must* fire next is the one with $t \mid min_i\ RET(t_i)$.
- If $\forall p \in I(t), \mu(p) \geq cw(p(t))$, and $c > 1$, then t is said to be multiply enabled at degree c. If t can only fire one enabling at a time, it adopts the *Single Server* (SS) semantics. *Infinite Server* (IS) semantics occurs when *any* amount of enablings can be fired at a same time, expressing a notion of parallelism among them.

There is a variety of different firing rules defined upon TPNs. In SPN, for example, transition firing is atomic. A random time elapses between the enabling and the firing of a transition t_i, during which the enabling token(s) reside(s) in the input place(s). Transition t_i must be continuously enabled during the time τ_i, and fires at that time (*race or preemptive policy*). Another execution rule for TPNs occurs when an enabled transition fires in three phases: in a "start firing" phase tokens are removed from the input places, remaining invisible for a "firing in progress" phase until they are released into output places in the "end firing" phase (*preselection or non preemptive policy*).

4.2 A TPN execution model based on DART/Time Warp

TPNs easily allow modelling and analysis of dynamic discrete event systems. Basically the causality of events is directly expressed by the net structure, whereas the dynamic behaviour is modelled by associating firing delays (i.e., events) to transitions.

The execution of a TPN model can be instrumented on a single machine (sequential simulation). However, for the animation of complex systems parallel or distributed simulation should be used (Chiola, 1993b) (Ferscha, 1994). The basic idea is to separate topological TPN parts to be simulated by logical processes (LPs). For performance reasons (Chiola, 1993c) the TPN model can be partitioned into a number of *regions* in such a way that conflicting transitions together with all their input places reside in the same LP. In the following the attention will be on TPNs where, for simplicity, $w(f)=1\ \forall f \in F$, and the enabling and firing semantics are centred on preselection and infinite server policy. Moreover, the enablings can use inhibitor arcs.

An obvious mapping of TPNs on DART/TW would be that of associating distinct actors to transitions and places of a TPN, with an initial configuration which creates all the actors of a region and initialises them according to the region topology in terms of acquaintances relations. Transition and place actors would interact by suitable messages during conflict resolution of enabled

transitions and firings. However, a different architecture was actually chosen with the aims of

- allowing for the configuration of coarse grained regions adequate for a distributed system of workstations
- minimising the number of exchanged messages, which is a critical factor for the simulation of complex systems
- facilitating the saving/restoring operations of the region (LP) status required by Time Warp.

A single actor of a basic Region class is introduced per LP which is initialised by a (passive) data structure describing the region topology. The Region class implements the functional TPN execution model. The internal status of a region actor includes a vector *Marking* having an entry for each place in the region, and a vector *Firings* which registers the history of transition firings for statistical data collection. A region understands two messages: *transition_fire* and *token_arrival*. A message *transition_fire* is sent by a Region actor to itself for firing a given transition *t*. The corresponding action carries the following execution steps

1. vector *Firings* is modified by annotating the firing of transition *t*
2. vector *Marking* is changed by depositing a token into each place $p \in O(t)$
3. a *list of enabled transitions*, determined by step 2, is built and for each enabled transition a new *transition_fire* message is generated and sent. The parameter of *transition_fire* consists of a transition identifier or a list of conflicting transitions. Resolution of conflicts is out of the scope of a Region actor.

A token generated into a region *r'* but destined to a place located into a different region *r* is carried by a *token_arrival* (external) message. The corresponding action accomplishes the steps 2 and 3 of a *transition_fire* action.

A MetaRegion actor is introduced which is used as a scheduler and arbiter and reflects on the timing behaviour of a corresponding Region actor. A meta region is fed of all the timing attributes of the corresponding region net. In particular, for each timed transition its exponential random variate mean λ, and possibly priority and firing probability of immediate transitions, are given. A meta region captures *transition_fire* messages and resolves conflicting immediate transitions by applying associated priorities and then by taking into account the probability attributes. For a list of conflicting timed transitions, the meta region finds a winner t_w by evaluating a minimum timestamp among the transitions, and actually schedules the *transition_fire* with *only* t_w as a parameter. The proposed timestamp of a conflicting timed transition t_c is defined by *now()*+t_c'next sample random variate. It is worthy of note that all the *transition_fire* messages corresponding to the list of enabled transitions determined at step 2, are *instantaneous*. In other words, they are "just sent messages" which will be evaluated by the meta region starting from the same time horizon given by current value of LVT. This is important to ensure that the infinite server firing semantics is correctly applied.

The separation of concerns between the actors region and meta region favours region reusability in the presence of different timed enabling and firing semantics. The meta region could adopt, transparently, infinite or single server semantics (by dropping multiple enabled conflicting transitions carried in a same set of "just sent *transition_fire* messages"), non preemptive or preemptive policy (by descheduling a previously scheduled *transition_fire* message related to a conflicting transition whose timestamp is greater than that of a newly scheduled *transition_fire* message).

5 A MODEL FOR LARGE CELLULAR NETWORKS

A TPN model is presented for a *personal communication services* network (PCS) (Carothers, 1994), i.e., a cellular communication system providing voice services to *Mobile Subscribers* (MS). The service area has a wrap-around Manhattan-like topology partitioned into regular sub-areas called *cells*. The service within a cell is supplied by a *Base Station* (BS) which is identified by its coordinates (x,y).

The behaviour of a mobile user, modelled as a stateless token, is formalised in Figure 2 where a TPN model of a generic cell is portrayed.

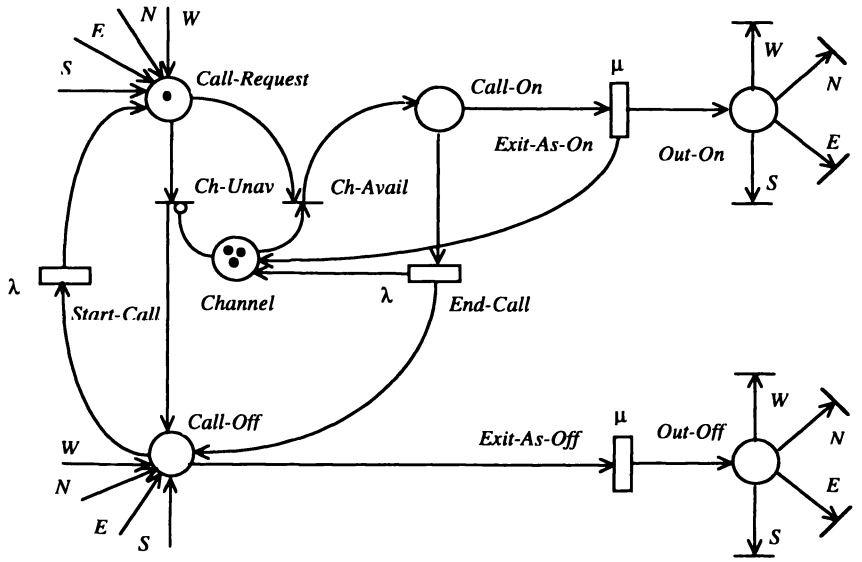

Figure 2 A TPN model of a cell.

Each cell is assigned a fixed number of channels. Movement and call issues are orthogonal one to another. A user can enter a cell with or without a call in progress. If a call is on, the old channel is released to the exiting cell and a new channel requested to the entering cell (*handover* procedure). If no channel is available, the call is blocked. Otherwise the call is continued with a newly

assigned channel. Quality of service can require designing the system (e.g., number of channels per cell) so as to keep below a given value the *blocking probability* (e.g., 10^{-4}).

In Figure 2 timed transitions are portrayed by squares. A user with a call in progress is received into the Call-Request place where the handover procedure is handled. A user without a call in progress is accepted into the Call-Off place. From Call-Off a new call may initiate (transition Start-Call). Would a channel be not available (transition Ch-Unav), a user-token moves from Call-Request into the Call-Off place (a call is rejected or a handover fails). In the Call-On state the user can abandon this cell (transition Exit-As-On) or he/she can terminate the call (transition End-Call) moving to the Call-Off place. In both cases, the utilised channel is first released to the Channel pool. Finally, a user can exit the cell without a call in progress (transition Exit-As-Off). Out-On and Out-Off are exiting places respectively of users with or without a call in progress. From an output place a user-token can non deterministically move to one of the four neighbouring cells, respectively located at its North, East, South and West. The exit transitions from Out-On (resp. Out-Off) are linked to incoming arcs to Call-Request (resp. Call-Off) places of near cells.

Statistical timed behaviour is achieved by timed transitions and related exponential random variate means. Transitions Exit-As-On and Exit-As-Off have mean $1/\mu$ (*dwell time*), transitions Start-Call and End-Call have mean $1/\lambda$ (*call interarrival time*). Non determinism among input/output W, N, E, S immediate transitions is achieved by attaching to them the probability 0.25.

The cell TPN model can easily be used to collect statistical data of the physical system. To exemplify, one can estimate the blocking probability by counting the total number of firings of the transition Ch-Unav and dividing it by the total number of firings of transitions Ch-Unav and Ch-Avail.

5.1 System partitioning and load balancing

The cell model of Figure 2 has been used to build large (e.g., 30x30 cells) cellular systems by spatially replicating identical cells, and to analyse them by simulation on a heterogeneous distributed system composed of a Sun Sparc1, Sun Sparc4, Sun Sparc5 and a HP9000 Apollo connected through a standard and shared with other users Ethernet at 10 Mb/s. A system is partitioned into four rectangular regions $<R_0,R_1,R_2,R_3>$. It is intended that the cells in a region Ri can directly communicate with cells in the neighbouring region. A configuration is specified by a tuple (x_0,x_1,x_2). A region is captured in a different LP/processor with a particular pair (Region, MetaRegion) of actors. The meta region is initialised with the timing attributes of a single cell: the means of timed transitions λ and μ and a single random generator on the basis of which the exponential variates are achieved. The saving/restoring of the meta region saves/restores the state of the

random generator. *transition_fire* messages identify enabled transitions through their name and unique identifier derived from the belonging cell coordinates.

Before simulating a cellular system, a configuration (x_0,x_1,x_2) capable of ensuring good performance through parameter tuning (see section 3.1) of the Time Warp engine was determined. As a measure of good static load balancing was assumed a homogeneous number of roll backs on the various processors. More precisely, a certain number of preliminary runs have been conducted to establish suitable values of checkpointing parameters SSR and MNSV. After that, some experiments have been carried out for performance evaluation of the TPN based PCS distributed simulation versus a corresponding sequential one. Some results are summarised in the next section.

5.2 Experimental results

A system with 30x30 cells was investigated under 10 channels per cell, a varying number of users and an assigned dwell time $1/\mu$ and interarrival time $1/\lambda$. A good partition was found with the configuration (11,18,25) to which corresponds the regions R_0=[0,10]x[0,29] allocated on Sparc4, R_1=[11,17]x[0,29] allocated on HP9000 Apollo, R_2=[18,24]x[0,29] allocated on Sparc5 and R_3=[25,29]x[0,29] allocated on Sparc1. The values SSR=0 and MNSV=3 improve Time Warp performance for the chosen system.

Speedup is shown in Figure 3. It was calculated as a ratio between the completion time of the sequential simulation, carried on the fastest machine (Sun Sparc4) and the completion time of the distributed simulation. Each point was determined by taking the mean of 3 runs. Good speedup is obtained for a number of users above 2700. Under such load conditions, processors do not experiment an excessive difference in their LVTs which in turn diminishes the number of observed roll backs.

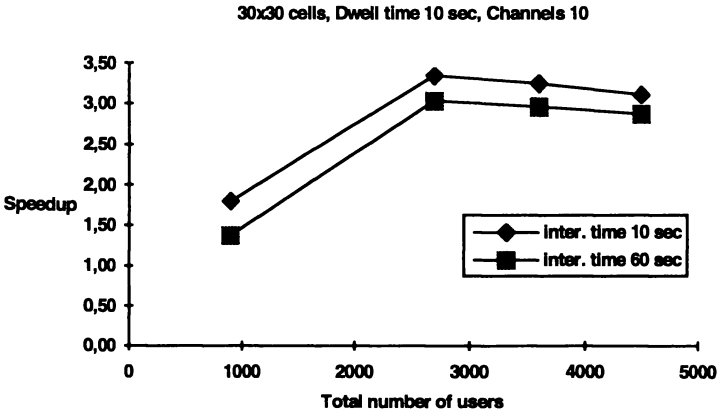

Figure 3 Speedup (4 processors)

As one can see from Figure 3, when the number of users exceeds 3000 the speedup starts decreasing. This behaviour relates to the augmented number of messages (and size of Message Queue) to be handled by each TW control machine. Although the roll back number remains equilibrated among the different processors, the state saving (checkpointing) and roll back operations (cancellation of no longer useful state versions, coasting forward, undo message handling) become more costly. Additional computational resources (processors) should be used. Table 1 depicts some measured values of the blocking probability.

Table 1 Measures of blocking probability

number of users	*blocking probability*
3600	1.63×10^{-4}
4500	3.75×10^{-4}

6 CONCLUSIONS

This paper describes an actor based architecture, DART, which is suited to the development of distributed and time dependent applications. DART is centred on the concepts of light-weight actors and control machine which hides a particular control engine orchestrating the evolution of a cluster of tightly coupled actors (subsystem). The control machine is capable of reasoning on local timing constraints. A programmer-defined scheduler object can host a set of timing clauses which directly affect message scheduling. Timing issues are confined and dealt with in the scheduler and control machine. This in turn improves modularity and reusability of actors which are not aware of timing.

Control machines of a distributed system can co-operate according to a system-wide interaction policy in order to fulfil system-level timing requirements. The paper summarises the achievement in DART of a Time Warp mechanism which is simple yet effective. It has been used to concurrently execute timed Petri nets, i.e., a powerful formalism for modelling and analysing discrete and asynchronous systems. As a significant application, the modelling and simulation of complex cellular networks has been described, along with some experimental results.
The on-going research aims at

- extending the DART based execution model of timed Petri nets by implementing different policies of timed enabling and firing semantics, so as to widen their field of applicability
- applying the separation of concerns approach of DART to distributed real-time systems modelled, visualised and analysed both from the point of view of functional and temporal behaviour through timed Petri nets (Nigro, 1996a)

- porting DART in Java in order to improve object design, memory management and platform neutral realisations.

Acknowledgements

The authors are grateful to Francesco Tisato and Brian Kirk for the helpful discussions during the preparation of the paper.

REFERENCES

Agha G. (1986) Actors: A model for concurrent computation in distributed systems. MIT Press.

Beraldi R., Marano S., Nigro L. (1996) Distributed simulation of PCS networks using a Time Warp mechanism, in *Eurosim'96, HPCN*, Dekker L., Smit W. and Zuidervaart (eds), North-Holland, 307-314.

Carothers C.D., Fujimoto R.M., Lin Y.-B., England P. (1994) Distributed simulation of large-scale PCS networks. *Mascots'94*.

Chiola G., Ajmone Marsan M., Balbo G., Conte G. (1993a) Generalized stochastic Petri nets: a definition at net level and its implications. *IEEE Transactions on Software Engineering* **19**(2), 89-107.

Chiola G., Ferscha A. (1993b) Distributed simulation of Petri nets. *IEEE Parallel and Distributed Technology* **1**(3), 33-50.

Chiola G., Ferscha A. (1993c): Distributed simulation of timed Petri nets: exploiting the net structure to obtain efficiency. *Proc. of the 14th International Conference on Application and Theory of Petri Nets*, Chicago, Illinois. Springer Verlag.

Ferscha A. (1994) Concurrent execution of timed Petri nets. *Proc. of the 1994 Winter Simulation Conference*, Lake Buena Vista, Florida, USA, 229-236.

Fujimoto R.M. (1990) Parallel discrete event simulation. *Communications of the ACM* **33**(10), 30-53.

Geist A. et al. (1994) PVM: Parallel Virtual Machine - A users guide and tutorial for networked parallel computing. The MIT Press.

Ghezzi C., Mandrioli D., Morasca S., Pezzè M. (1991) A unified high-level Petri net formalism for time-critical systems. *IEEE Transactions On Software Engineering*, **17**(2), 160-172.

Jefferson D., et al. (1987) Distributed simulation and the Time Warp Operating System. *ACM Symposium on Operating Systems Principles*, 77-93.

Kirk B. (1995) Real time protocol design for control area networks. *Proc. of Real Time'95 Conf.*, Ostrava (Cz Rep.), 251-268.

Kirk B., Nigro L., Pupo F. (1997) Using real time constraints for modularisation. *Proc. of Joint Modular Languages Conference'97*, 19-21 March, Linz (Austria), Springer-Verlag, LNCS 1204, 236-251.

Kumar D., Harous S. (1994) Distributed simulation of timed Petri nets: basic problems and their resolution. *IEEE Transaction on Systems, Man, and Cybernetics* **24**(10), 1498-1510.

Maes P. (1987) Concepts and experiments in computational reflection. *Proc. of OOPSLA'87, ACM SIGPLAN Notices*, **22**(12), 147-144.

Misra J. (1986) Distributed discrete event simulation. *ACM Computing Surveys* **18**(1), 39-65.

Murata T. (1989) Petri nets: properties, analysis and applications. *Proc. of IEEE*, **77**(4), 541-580.

Nigro L. (1994) Control extensions in C++. *J. of Object Oriented Programming*, **6**(9):37-47.

Nigro L. (1995) A real time architecture based on Shlaer-Mellor object lifecycles. *J. of Object Oriented Programming*, **8**(1):20-31.

Nigro L., Pupo F. (1996a) Modelling and analysing DART systems through high level Petri nets. Springer-Verlag, LNCS 1091, 420-439.

Nigro L., Tisato F. (1996b) Timing as a programming in-the-large issue. *Microsystems and Microprocessors J.*, **20**(4), 211-223.

Reiser M., Wirth N. (1992) Programming in Oberon - steps beyond Pascal and Modula. Addison-Wesley.

Ren S., Agha G. (1995) RTsynchroniser: language support for real-time specification in distributed systems. *ACM SIGPLAN Notices*, **30**(11), 50-59.

Ren S., Agha G., Saito M. (1996) A modular approach for programming distributed real-time systems. *J. of Parallel and Distributed Computing*, Special issue on Object-Oriented Real-Time Systems.

Saito M., Agha G. (1995) A modular approach to real-time synchronisation, in Object-Oriented Real-Time Systems Workshop, 13-22, *OOPS Messenger, ACM SIGPLAN.*

Shlaer S., Mellor S. (1992) Object lifecycles - Modelling the world in states. Yourdon Press.

BIOGRAPHY

Roberto Beraldi has received in 1996 a PhD degree in Computer Science from University of Calabria. His research interests include: distributed systems, parallel simulation, Time Warp, performance evaluation of wireless systems.

Libero Nigro is an Associate Professor of "Fondamenti di Informatica" at the University of Calabria. He is also responsible of "Sistemi di Elaborazione". His research interests include: object oriented software engineering, actor systems, scheduling control, distributed real time systems, parallel simulation, timed Petri nets.

Francesco Pupo is a PhD student at University of Calabria. He is interested in Petri nets, actor systems, real time, discrete event simulation.