Instrumentation of Synchronous Reactive Models for Performance Engineering*

Alberto Valderruten Vidal, Manuel Vilares Ferro and Jorge Graña Gil

Laboratorio de Fundamentos de la Computación e Inteligencia Artificial
Departamento de Computación, Universidad de La Coruña
Campus de Elviña, 15071 La Coruña, España.
{valderruten,vilares,grana}@dc.fi.udc.es

Abstract. Synchronous Reactive Modelling provides an optimal framework for the modular decomposition of programs that engage in complex patterns of deterministic interaction, such as many real-time and communication entities. This paper presents an approach which includes performance modelling techniques in the Synchronous Reactive Modelling method supported by ESTEREL. It defines a methodology based on timing and probabilistic quantitative constructs which complete the functional models. A monitoring mechanism provides performance results during the simulation. This methodology is applied to a protocol modelling case study. Performance metrics are computed and compared with known reference results.

Key Words: Performance Engineering, Synchronous Reactive Models, Real-Time & Embedded Systems, Model Development, Instrumentation, Simulation and Monitoring.

1 Introduction

Developing systems with the use of formal modelling methods is steadily growing. Their use allows an unambiguous and precise description of a system and leads to easier formal verification and validation. Synchronous Reactive Models are used to describe computer-based systems which must react *instantaneously* to external events, such as the operative commands sent to an embedded system in a satellite or the alarms produced by sensors.

However, this modelling method was not intended to support quantitative analysis and the prediction of the system behaviour with respect to non-functional requirements. Performance evaluation techniques can help the developer, advising him about the estimated performance of a design option in order to justify the choices. Performance is defined by the IEEE [6] as the degree to which a system acomplishes its functions within given constraints such as speed, accuracy or resource utilisation.

The increasing complexity of cooperative systems in general, and that of real-time and embedded systems, leads us to focus on the integration of performance

* Work partially supported by the project XUGA10501A93 of the *Xunta de Galicia*.

evaluation and formal description techniques. This will allow specification, implementation and analysis of such systems taking into account both functional and non-functional requirements [12]. When formal models include inputs with regard to performance evaluation, we can obtain performance models addressing the service quality (speed, reliability...), very early in the system development life-cycle. The major goal of this approach is to define and validate rules allowing an easier definition of performance models and then to compute performance metrics.

In this paper we attempt to obtain performance results from Synchronous Reactive Models. These models describe tightly coupled and deterministic processes, communication being achieved by instantaneous broadcasting; they completely ignore the timing and probabilistic aspects of the system, and so give no information with regard to performance. In order to fill this gap, the idea is to consider the performance requirements of a design as dedicated constructs for checking performance constraints by simulation. The semantics of these constructs must differ from those of the Synchronous Reactive System, and then unambiguities must be avoided.

We have chosen different known approaches as a reference for the validation of our models. All of them are based on the use of Formal Description Techniques such as LOTOS [10, 13] and PETRI NETS [9, 7]. By obtaining a performance model from a Synchronous Reactive Model, and solving it by simulation, we would have to find the same results as those analytically found as reference. In order to support our modelling methodology, we use the ESTEREL [5] language for Synchronous Reactive Modelling, and AGEL [1] which is a user-friendly tool that supports the related modelling methodology.

This paper presents the first results of our work. The application example is the *Stop & Wait* protocol.

2 Synchronous Reactive Modelling Using ESTEREL

The ESTEREL language was designed to develop *Synchronous Reactive Models* [3]. In these models, a module reacts instantly to each input event by modifying its internal state and creating output events. The reactions are then *synchronous* with the inputs: the processing of an event is uninterruptible, and the system is sufficiently fast to process its input events. Such a system has an underlying finite state machine. Real-time embedded controllers and communication protocol entities are good examples of reactive systems.

ESTEREL is an imperative concurrent language, with high level control and event handling constructs. Modules are the basic entities which can be interfaced with the environment or with other modules. Each module has an event interface used to communicate with its environment. Events are the main objects handled by the modules: they may be composed of several signals, which can be either present or absent during the reaction to an input. The `present` construct is used to test the presence of a signal in the current event:

```
present <signal> [then <instruction-1>]
                [else <instruction-2>]
end
```

In order to be present in a reaction, an output signal can be emitted by a module using the `emit <signal>` instruction. This emission is *broadcast* to all modules having the signal defined as input. Signals are hence used for synchronization and communication description.
ESTEREL instructions are of two kinds:

- *Instantaneous instructions*, in the basis of the perfect synchrony hypothesis, which are completed in the same reaction it was activated: signal emission (`emit`), signal presence test (`present`) and sequencing (`<instruction-1> ; <instruction-2>`) are examples of such instructions.
- *Waiting instructions*, the simplest of which is the `await <signal>` statement. Their execution is blocked until a specified event occurs.

A parallel operator `||` allows the control flow to be transmitted to several instructions simultaneously. It terminates when its two branches are terminated, and again, it takes no time by itself.

The `watching` statement deals with behaviour preemption, one of the most important features of ESTEREL. A complex `watching` construct is defined with the syntax:

```
do <instruction-1>
watching <signal>
timeout <instruction-2>
end
```

Here `instruction-1` will be interrupted and `instruction-2` executed if an occurrence of `signal` happens before `instruction-1` ends. This instruction terminates when `instruction-1` does, or when `instruction-2` does if a timeout occurs.

In general, more user-friendly statements are derived from a set of primitive or kernel statements. A detailed description of its semantics is presented in [2]. The complete list of kernel statements is:

```
nothing
halt
emit <signal>
<instruction-1> ; <instruction-2>
loop <instruction> end
present <signal> then <instruction-1> else <instruction-2> end
do <instruction> watching <signal>
<instruction-1> || <instruction-2>
trap <trap> in <instruction> end
exit <trap>
signal <signal> in <instruction> end
```

Finally, even if the ESTEREL compiler can work as a front-end to several languages, we only use the language C interface in order to instrument the Synchronous Reactive Models for Performance Engineering purposes.

3 Case Study: the *Stop & Wait* Protocol Specification

Let us consider the design of the datalink layer of the OSI reference model [8]. The problem is to find an algorithm to achieve a reliable, efficient communication between two sites physically connected by a communication channel. Data is transmitted in one direction, from Host A to Host B. The sender sends a frame, the receiver may only send an acknowledgement if the data is correctly received. Since the communication channel is noisy, when a damaged frame reaches the receiver, it has to be discarded; after a while, the sender is faced with a time-out and sends the frame again. This process stops when the frame finally arrives intact.

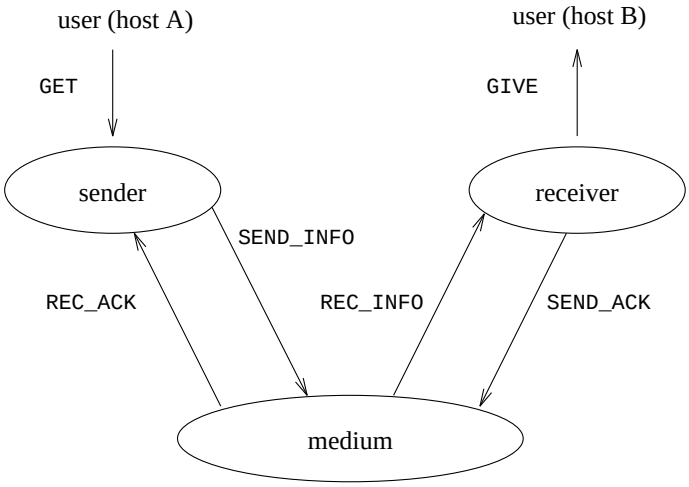

Fig. 1. Communication Abstract Model

The receiver acknowledges the data receipt. But the acknowledgement frame may be completely lost and the sender has no way of knowing about this fact. When the sender times out, having not received an acknowledgement, it incorrectly assumes that the data frame has been lost or damaged and sends it again. The duplicate frame arrives and is unwittingly passed to Host B. To avoid this, the receiver needs a way of distinguishing a retransmission from a frame which is being seen for the first time. The obvious way to achieve this is to let the sender put a sequence number in the header of each frame it sends. Then the receiver can check the sequence number of each incoming frame to find out whether it is

a new frame or a duplicated one to be discarded. A one bit sequence number is sufficient, the protocol is known as the alternating bit protocol [11].

When using a resource oriented specification approach, this protocol is modelled as three processes: a transmitter, a receiver and a communication channel (medium). The components are then composed in parallel as is shown below:

```
module protocol:
input TIMEOUT, CHANNEL_ERROR, TRANSMISSION_TIME;
output GET, GIVE;
signal SEND_INFO, SEND_ACK, REC_INFO, REC_ACK
in
        run sender
    ||
        run medium
    ||
        run receiver
end signal
end module
```

Sender and receiver reactive modules are respectively as follows:

```
module sender:
input REC_ACK, TIMEOUT;
output GET, SEND_INFO;
loop
    emit GET;
    emit SEND_INFO;
    do
        loop
            await TIMEOUT;
            emit SEND_INFO;
        end loop;
    watching immediate REC_ACK;
    end;
end loop
end module
```

```
module receiver:
input REC_INFO;
output GIVE, SEND_ACK;
loop
    await REC_INFO;
    emit GIVE;
    emit SEND_ACK;
end loop
end module
```

In this study, we focus on the communication channel: since the medium is unreliable in both directions, we must take into account the possible loss of frames. With regard to the transmitter and the receiver, an abstraction is made over the reliability operations, e.g. frame sequence numbering and verification, duplicate frames discarding...

```
module medium:
input SEND_INFO, SEND_ACK, ERROR, TRANSMISSION_TIME;
output REC_INFO, REC_ACK;
loop
    await
        case immediate SEND_INFO do
            do
                await ERROR
            watching TRANSMISSION_TIME
            timeout
                emit REC_INFO
            end
        case immediate SEND_ACK do
            do
                await ERROR
            watching TRANSMISSION_TIME
            timeout
                emit REC_ACK
            end
    end await
end loop
end module
```

Note that in this model, events related to time (`TIMEOUT, TRANSMISSION_TIME`) and the loss probability (`ERROR`) are declared as external inputs. It corresponds to a purely functional viewpoint, which does not include the performance behaviour in the model itself. We search for a way to add this behaviour in order to compute the interesting performance measures by using the simulation facility of AGEL.

4 Considering Performance

Taking into account performance issues during the system life-cycle is done by means of a performance modelling process, involving a set of system designers and modelling expert activities [4]. In this case, the aims of modelling include performance requirements, which may include throughput, timing and utilization rate constraints.

We present an approach which in a first step completes the functional specification with all performance (non functional) information, allowing the computation of performance results by simulation.

In order to compare our performance analysis of the *Stop & Wait* protocol specification with the approaches using Queueing Networks derived from

LOTOS [13], Timed Petri Nets [9], Stochastic Petri Nets [7], and Stochastic LOTOS [10], we make similar assumptions. The expected result is the throughput computation under the following constraints:

- The link baud rate is 9600.
- The information and acknowledgement frames contain 1024 bits [2].
- The next information frames are available as soon as previous information frames were successfully sent: only a small overhead delay of 1 ms is added.
- The medium can loose both information and acknowledgement frames with a 5% probability.
- The timeout is 1 s.
- The time for processing the information and acknowledgement frames is 13.5 ms.

In order to add this kind of performance information to the ESTEREL functional specification, the necessary quantitative information below must be introduced:

1. The estimated processing time relevant to each *timed action*. A timed action is an action which consumes time, e.g. the time for processing a frame, the transmission time... are relevant in performance modelling, but have no sense in a Synchronous Reactive Model. Our modelling choices must avoid any semantical incongruence with the perfect synchrony hypothesis.
2. The probability associated with each possible *alternative behaviour*. When a probability defines the system behaviour, a degree of non-determinism must be introduced in the Synchronous Reactive Model, which is deterministic by nature. Once again, the modelling choices must treat this point carefully.

In order to express this information, we complete the Synchronous Reactive Model with two types of performance constructs:

1. The timed action construct assumes a specific global signal which represents the time, `TS`. The adopted time unit is stated by the model abstraction, i.e. it is a modelling choice. Note that this clock signal is an input event from the reactive module viewpoint.
   In this construct, `await n TS` determines the time `n` associated to the current timed action. This action is defined by using two events: the initial event and the final one. For instance, the information frame construction in the sending process, modelled as `emit SEND_INFO` in the sender reactive module, may be described as follows:

```
emit SEND_INFO_I;
await 10 TS;
emit SEND_INFO_F;
```

[2] transmission time is then 106.7 ms.

where each occurrence of the TS event corresponds to 0.1 ms; the overhead of 1 ms for this action is hence taken into account. The protocol timeout and the transmission time[3] are modelled with the same clock convention, i.e. as `await 10000 TS` and `await 1067 TS` respectively.

2. The alternative behaviour construct uses an externally implemented function, `probability (value)`, which returns `true` with the probability given as a parameter. Its use is shown in the example of the module medium; we state that the `REC_INFO` emission is done with a 95% probability:

```
module medium:
function probability(integer):boolean;
...
        if probability(95)
        then [
                emit REC_INFO_I;
                await 135 TS;
                emit REC_INFO_F; ]
        end
...
```

5 Model Instrumentation

The complementary work that must be done in order to obtain performance measures from our models is to instrument them with a suitable monitoring mechanism. We propose to add a new module, the *monitor*, which is in charge of observing a set of signals defined by the designer. Is the responsibility of the designer to decide which events are necessary in order to compute the wanted performance metrics.

The monitor calls an auxiliary function, `save_entry (<time>,<string>)`, at each occurrence of a signal we want to observe. This function writes the date of the occurrence and the name of the signal on an output file.

```
module monitor:
procedure save_entry () (integer,string);
input TS, GET, GIVE;
var TIME := 0 : integer in
loop
        await TS;
        TIME := TIME + 1;
        present GET then call save_entry () (TIME,"GET"); end;
        present GIVE then call save_entry () (TIME, "GIVE"); end;
end loop
end var
end module
```

[3] `TIMEOUT` and `TRANSMISSION_TIME` signals in the model presented in section 3.

The proposed monitoring mechanism only offers the basic observation functionality that is needed for performance measuring. A complete statistical tool must include at least the computation of the confidence intervals that are needed to control the simulation time.

6 Feasibility Study

In the first step we analyse the feasibility of the instrumentation of ESTEREL models with the proposed performance constructs and measurement mechanisms for their use as a performance evaluation technique.

Our approach of feasibility study relies on the modelling of the protocol written in Stochastic LOTOS. LOTOS gives rules to compute the dynamic model from a given specification; this model can be represented by a finite state machine. Using Stochastic LOTOS, Rico and Bochmann [10] have modified some of these rules in order to introduce probability and action duration into the specification. The modified rules allows the performance labels of the finite state machine to be computed. Figure 2 is the result of the application of these new rules to the *Stop & Wait* protocol specification.

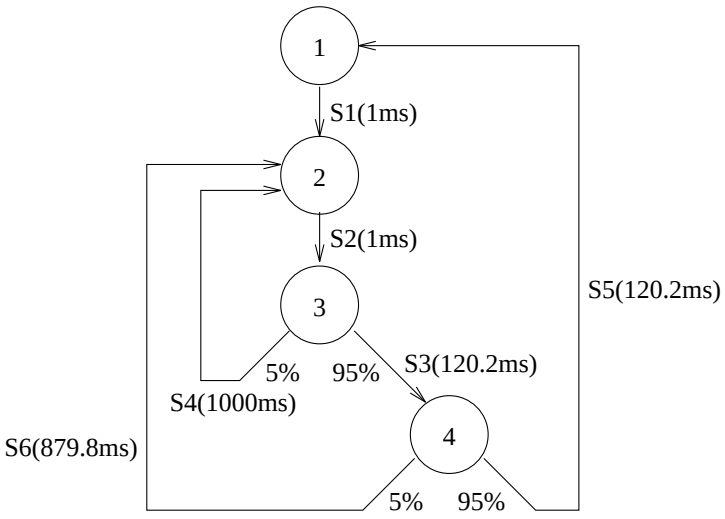

Fig. 2. Finite state machine

A simple ESTEREL model is obtained from this finite state machine, defining a reactive module for each state and a signal for each transition. The model includes the performance information by using the proposed performance constructs. For instance, the module associated to the third state is:

```
module reactive_3:
function probability(integer):boolean;
input S2, TS;
output S3, S4;
loop
        await immediate S2;
        if probability(5)
                then [  await 10000 TS;
                        emit S4;  ]
                else [  await 1202 TS;
                        emit S3;  ]
        end if
end loop
end module
```

In order to allow computation of throughput from the simulation, we only need to observe the occurrences of signal `S5`, corresponding to a `GIVE` event. A monitor is then placed in parallel. The resulting code for the main module is as follows:

```
module all_reactives:
input TS;
signal INIT, S1, S2, S3, S4, S5, S6 in
            emit INIT;
        ||
            run reactive_1
        ||
            run reactive_2
        ||
            run reactive_3
        ||
            run reactive_4
        ||
            run reactive_monitor
end signal
end module
```

Note that an initial signal (`INIT`) must be emitted to start the simulation, since the `reactive_1` module is waiting for it.

The simulation was accomplished by using the corresponding AGEL functionality. This tool supports a timer which can be associated to any input signal, the `TS` input in this case. Simulation can be stopped at any moment, hence one can stop it, compute the performance measures (throughput) and continue the simulation if stability is not yet reached. The obtained throughput is 2.819 frames per second, which is close to the value computed when using Markov chains on Stochastic LOTOS [10](2.852) or analytical methods on Derived Queueing Networks [13](2.853). It also comes close to the 2.75 frames per second obtained with Stochastic Petri Nets and exponential distributions [7].

Therefore this example allows the relevance of the proposed instrumentation methodology to be validated in order to model protocol performance by using adapted ESTEREL models.

7 Performance Model

A complete performance model was designed for the *Stop & Wait* protocol specification presented in section 3. A user-level viewpoint was added in order to represent the communications between users and protocol entities. The main module becomes the `system` reactive:

```
module system:
input TS;
output GET, GIVE;
signal GET_I, GET_F, GIVE_
in
        run hostA
    ||
        run protocol
    ||
        run monitor
    ||
        run hostB
end signal
end module
```

The module `hostA` (resp. `hostB`) is a single loop waiting `GET_I` and `GET_F` signals from the sender entity (resp. `GIVE_` from the receiver), then emitting a `GET` (resp. `GIVE`) signal to the environment.

The `protocol` reactive only differs from that presented in section 3 in the interface declaration: `TS` is the only input, the outputs are `GET_I`, `GET_F` and `GIVE_`, and the set of local signals is `SEND_ACK_`, `SEND_INFO_I`, `SEND_INFO_F`, `REC_ACK_I`, `REC_ACK_F`, `REC_INFO_I` and `REC_INFO_F`.

Receiver and sender reactive modules now include several timed action constructs, becoming respectively:

```
module receiver:
input REC_INFO_I, REC_INFO_F, TS;
output SEND_ACK_, GIVE_;
loop
   await REC_INFO_I;
   await 135 TS;                      % Processing a frame: 13.5 ms
   await immediate REC_INFO_F;
   emit GIVE_;
   emit SEND_ACK_;                    % without delays
end loop
end module
```

```
module sender:
input REC_ACK_I, REC_ACK_F, TS;
output GET_I, GET_F, SEND_INFO_I, SEND_INFO_F;
loop
   emit GET_I;
   await 10 TS;                        % 1 ms delay
   emit GET_F;
   emit SEND_INFO_I;
   await 10 TS;                        % 1 ms delay
   emit SEND_INFO_F;
   do loop
         await 10000 TS;               % Timeout: 1 s
         emit SEND_INFO_I;
         await 10 TS;                  % 1 ms delay
         emit SEND_INFO_F;
      end loop;
   watching immediate REC_ACK_I
   timeout
      await 135 TS;                    % Processing a frame: 13.5 ms
      await immediate REC_ACK_F;
   end;
end loop
end module
```

Note the presence of the alternative behaviour construct in the module medium, which is now defined as follows:

```
module medium:
function probability(integer):boolean;
input SEND_INFO_I, SEND_INFO_F, SEND_ACK_, TS;
output REC_INFO_I, REC_INFO_F, REC_ACK_I, REC_ACK_F;
loop
   await
      case immediate SEND_INFO_I do
         await 10 TS;                  % 1 ms delay
         await immediate SEND_INFO_F;
         await 1067 TS;                % Transmission time: 106.7 ms
         if probability(95)            % Probability of loose: 5%
         then [ emit REC_INFO_I;
                await 135 TS;          % Processing a frame: 13.5 ms
                emit REC_INFO_F; ] end
      case immediate SEND_ACK_ do
         await 1067 TS;                % Transmission time
         if probability(95)            % Probability of loose: 5%
         then [ emit REC_ACK_I;
                await 135 TS;          % Processing a frame: 13.5 ms
                emit REC_ACK_F; ] end
   end await
end loop
end module
```

Finally, the module monitor is the same as the one presented in section 5. `GET` and `GIVE` signals are observed in this way, allowing throughput computation during the simulation.

After simulation, we have obtained very similars values to those which we had for reference. The throughput is 2.858 frames per second, that can be compared to our first result (2.819), and to the simulation result obtained when deriving Queueing Networks from LOTOS [13](2.859).

8 Conclusion

This study shows the possibility of obtaining performance results from instrumented Synchronous Reactive Models. An instrumentation methodology that involves two steps is proposed:

- The use of performance constructs in order to take into account the quantitative information needed for performance evaluation. Two constructs were defined: a timed action construct based on a global time referential, and an alternative behaviour construct implementing probabilities.
- The monitoring of the model using a monitor reactive that tests at any event the presence of predefined signals. Choosing the convenient signals, the generated simulation trace allows computation of performance measures.

We have applied this methodology in the *Stop & Wait* protocol case study, validating the models obtained with ESTEREL as modelling language and AGEL as a development tool supporting simulations.

Our present work concentrates on:

- The generalization of a set of instrumentation rules in order to ensure the coverage of the specific semantic of ESTEREL.
- The improvement of the simulation framework, enhancing the simulation control and the computation of performance measures.

Performance evaluation models can then be directly obtained from formal specifications. Designers can be able to choose between several design options according to the provided performance estimations, without a costly redesign of the system, as Performance Engineering states.

References

1. AGEL, Workshop Manual, Version 3.0. ILOG S.A., 2 avenue Galliéni, 94253 Gentilly, France
2. Berry, G.: The Semantics of Pure Esterel. In Program Design Calculi, Broy, M. (ed.), NATO ASI Series, Computer and System Sciences **118** (1993) 361–409
3. Berry, G., Gonthier, G.: The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. Science Of Computer Programming **19**, 2 (1992) 87–152
4. Conquet, E., Valderruten, A., Trémoulet, R., Raynaud, Y., Ayache, S.: Un modèle du processus de l'activité d'évaluation des performances. Génie Logiciel et Systèmes Experts **27** (1992) 27–31
5. ESTEREL V3, Language Reference Manual. CISI Ingénierie, Les Cardoulines B1, 06560 Valbonne, France
6. Standard Glossary of Software Engineering Terminology. IEEE 610.12-90 (1990)
7. Molloy, M.K.: Performance Analysis using Stochastic Petri Nets. Transactions on Computers **31**, 9 (1982) 913–917
8. ISO: The Reference Model. DIS 7498 Part 1/4 (1987)
9. Razouk, R., Phelps, C.: Performance Analysis using Timed Petri Nets. In Protocol Specification, Testing and Verification IV, IFIP (1984) 561–576
10. Rico, N., Bochmann, G.v.: Performance Description and Analysis for Distributed Systems using a variant of LOTOS. In Protocol Specification, Testing and Validation X, IFIP (1990)
11. Schwartz, R.L., Melliar-Smith, P.M., Vogt, F.H.: Interval Logic: A Higher-Level Temporal Logic for Protocol Specification. In Protocol Specification, Testing and Verification III, Rudin, H., West, C.H. (eds.), IFIP (1983)
12. Valderruten, A.: Modélisation des Performances et Développement de Systèmes Informatiques: une étude d'intégration. Thèse d'Informatique, Université Paul Sabatier, Toulouse (1993)
13. Valderruten, A., Hjiej, O., Benzekri, A., Gazal, D.: Deriving Queueing Networks Performance Models from Annotated LOTOS Specifications. In Computer Performance Evaluation'92: Modelling Techniques and Tools, Pooley, R., Hillston, J. (eds.), Edinburgh University Press (1993) 120–130

This article was processed using the LaTeX macro package with LLNCS style