Specifications from Multiple Viewpoints: The Temporal-Causal Way

Jaelson Freire Brelaz de Castro *

Departamento de Informática – UFPE

Caixa Postal 7851– 50732 Recife– PE

Brazil

jbc@di.ufpe.br

Abstract

In this paper we argue that a single representation scheme is often not enough to capture the various features of system behaviour. Instead, multiple viewpoints should be used to partition the domain of information.

We present a technique that supports the specification of systems from three points of view: informal, causal and temporal-causal. The *Informal View* describes in natural language and graphically the requirements of the component. It is based on the *Conic Environment* which provides a language-based approach to the building of distributed system. The *Causal View* relies on enhanced Petri Nets to describe internal structure, distributed control and safety requirements. The third view makes use of temporal-causal logic to describe temporal features of the components such as liveness requirements. Previous causal properties are preserved. Temporal-causal logic is the result of the integration of two well-known formalisms, namely Petri Nets and Temporal Logic.

Keywords

Distributed Systems, Formal Methods, Specification.

*I gratefully acknowledge CNPq, under grant 301052/91-3(NV) for its financial support. I also acknowledge the usefull discussions with Jeff Kramer at Imperial College.

1 Introduction

Our primary goal is to develop a framework to support specification, i.e. modeling and analysis, of concurrent and distributed systems. A *system* will be composed of separate, interacting *components*, possibly highly independent of each other. Components will have well-defined interfaces and their local state that changes over time. A system will be described and managed in terms of their configuration [5].

It is well known that despite the considerable research work on automated software construction and formal methods, precise specifications for complex systems are still proving to be extremely difficult to develop. This lack of impact on the *real world* may be attributed to the inadequacy of existing techniques. Often, they are suitable for representing some limited aspects of the systems, but are unable to characterize many other features. Therefore, a single representation scheme is often not enough to capture all the features of the behaviour of a system.

For example, the standard Petri Net technique provides a graphical notation that has direct and immediate appeal for small examples. It is very suitable for the initial phases of the description, when one is still working out the details and is mainly interested in capturing the flow of control. The possibility of using several levels of abstraction is clearly an advantage. But, for larger systems it usually leads to large and unstructured graphs that are difficult to understand. Besides, temporal properties are not directly stated within this framework.

On the other hand, temporal logics have proved to be good at modeling the reactive aspect of systems and expressing temporal properties. Yet, large specifications written directly in temporal logics are unstructured collections of formulae from which it is difficult to produce or even extract the meaning.

We argue that a single representation scheme is often not enough to capture the various features of system behaviour. Instead, multiple viewpoints should be used to partition the domain of information. Its success rests upon the selection of appropriate representation schemes, the careful definition of the relations between them and the process by which such specifications are built within those representation schemes.

Indeed, the major feature of research in software engineering and formal methods in recent years has been the trend towards combinations and integrations of different approaches [8, 9, 3].

In this paper we show how multiple representation schemes, such as temporal logics and Petri Nets, can be used to describe the behaviour of systems. This temporal-causal framework will contribute towards the provision of a more effective basis for software development, enabling system specification from multiple points of view. The main advantages are the ability to express explicitly, clearly and compactly the causal and temporal relationship between the events of the system. Formal reasoning is also supported, i.e local and global properties are derived [2].

We propose the specification of systems from three points of view: informal, causal and temporal-causal. The *Informal View* describes in natural language and graphically the requirements of the component. It is based on the *Conic Environment* that provides a language-based approach to the building of distributed system. The *Causal View* relies on enhanced Petri Nets to describe internal structure, distributed control and safety requirements. The third view makes use of temporal-causal logic to describe temporal features of the components such as liveness requirements. Previous causal properties are preserved.

In [4] we show how Petri Nets and Temporal Logic can be integrated. In essence, the basic Petri Net model is enhanced (to include types, guarded transitions and the annotation of places with a logical scope) and given a logical proof-theoretic characterization. This logic is then merged with standard temporal logic. The resulting logic is called *Temporal-Causal Logic* [1].

A methodology for the temporal-causal specification of systems is briefly discussed in **Section 2**. It shows how one can start from an informal description and end up with a temporal-causal specification. **Section 3** demonstrates the viability of the specification technique. The components of the patient monitoring system are described in detail. **Section 4** summarizes the discussions and concludes the paper.

2 Methodology

Given a certain design (possibly obtained with a constructive design method [6]), which identifies a collection of components as well as their structure, our job is to specify the individual components of the systems as well as the structural configuration of the system. The first task consists of:

- Identification of the basic components that will the basis for the system;

Having done that, one should now concentrate on the specification of the individual components of the system. As discussed above, a component specification consists of three complementary views. The informal one is the starting point of the process. The objective is to recognize the interface ports of the components, as well as messages that are going to flow through. A rectangle with ports should be drawn to represent the component. If possible, message types should be associated with the ports. A description in natural language (e.g. English) should be attached to the graphical description. Since we are going to concentrate on events and their effects, it should list all the events associated with the component. Observe that each "port" has a communication event associated with it. The remaining events are "internal" ones, i.e. related to the control flow of the activities. For each event there should be a rule describing what are the enabling conditions and what is the effect of its occurrence.

Having produced the informal description of the component, the next step is to provide the *causal view.* As explained above, it will correspond to an enhanced net. At a macro level one could portray a component as one transition whose input places are associated with the entryports of the informal view, and output places corresponds to the exitports. These places are typed according to the messages that they suppose to carry.

At a more detailed level, the initial transition should be refined into a rectangle with has the same input/output typed places. The interior of the rectangle may contain several other transitions (one for each rule identified in the informal view) and places. The idea is to be able to represent graphically the flow of control of activities that the component may perform. Transition and places will correspond to circle and bars respectively. Conditional transitions (those which occurrences depend upon the holding of a guard) are depict as hatched bars. A flow relation captures the relationship between the conditions and transition. Scopes also may be associated with transitions and inhibitor arcs are also allowed. Recall that all the entities present in the enhanced net diagram are typed. We shall adopt the convention of underlining condition predicates and representing inhibitor arcs as dashed lines which end in black dots (not arrow). Besides, an outbound arrow associated with a transition, may have underneath it a "scope", i.e. a formula of the Enhanced net language. It is expected that several versions of the enhanced net may be required until the specifier is "convinced" that it indeed captures the required structural behaviour of the component. Only then, the enhanced net description should be coded-up into causal (the effect of transition occurrence) and safety (when a transition is enabled to fire) axioms.

The last stage is the characterization of the further temporal properties of the component, i.e. liveness. For instance, it will be possible to capture when a transition must fire. These requirements are expressed as temporal-causal logic axioms. Any other property that the specifier feels that may be required to be expressed, also should be written as additional logical axioms. The result is a set of axioms, which represent causality aspects as well as temporal properties.

Having defined the set of logical axioms that characterize the behaviour of the component, one should then make use of the logical machinery available to prove properties of the specification. It is through these logical manipulations that confidence in the specification is gained.

Systems are constructed as a composition of the various individual components. In the remaining of this paper we shall concentrate on the component specification, leaving system specification for [4].

3 An example: The Patient Monitoring System

The example to be used to illustrate the temporal-causal technique is the patient monitoring system (PMS) [7]. The intensive care ward in a hospital consists of a number of factors, such as pulse, temperature and blood pressure. For each patient the current readings can be displayed both at the bedside and at the nurse unit. If any of the factor readings of a patient is outside the preset limits, then an alarm is sent to the central nurse station. The ward system will be composed of two sorts of components: patient and nurse. Bellow we specify the **Patient Component**.

3.1 Informal View

Three interface ports are required: alarm, bed req and bed reply. Exitport alarm conveys messages which carry alarm information (i.e the status of each sensor attached to the patient). Entryport bed req receives signal messages requesting the latest bed readings. Exitport bed reply relays messages with the most updated patient readings.

The next task is the identification of the basic events of the component. Recall that for each port there should be one event. Therefore, let us denote them as "**send** bed.alarms **to** alarm", "**receive** signal **from** bed req" and "**send** bed **to** bed reply". Three internal events are also necessary, "**new reading**(data)", "**Alert nurse**" and "**check reading**". The first one models the readings of the sensors attached to the

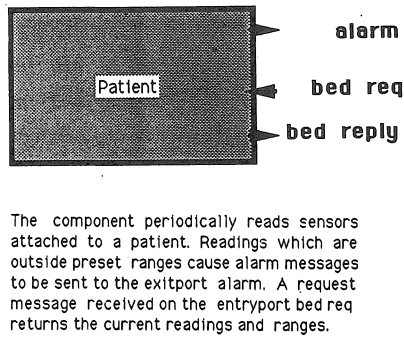

Figure 1: Patient Component: Informal View

patient, the second models the decision to alert the nurse (conditional on problems being detected with the latest patient readings) and the last event indicates that a new reading of the sensors should be performed (conditional on the last readings being normal). Last, the behaviour of the patient component could be informally described as a set of rules, one for each event.

- **R1 Receiving a request for patient information;** a patient is enabled to receive a signal requesting the latest patient readings only if he is in "waiting for new request" condition. The reception of the signal causes the patient to prepare a message (with the latest readings) to be sent.
- **R2 Sending the latest patient readings;** a patient is enabled to send a message with the latest readings only if the appropriate message has been prepared. The sending of the message causes the patient to return to the condition of "waiting for a new request".
- **R3 Performing a new reading of the sensors;** a patient component is enabled to perform a new reading of the sensors attached to a patient only if it is in "read sensors" condition. The reading of the sensors causes a condition where the new "data" acquired must be tested. A variable (bed) is updated with the most recent patient readings and sensors are checked to detect any abnormality. If a problem is detected, then a predicate (inwarning) must be set
- **R4 Alerting the nurse;** a patient is enabled to alert the nurse only if the latest readings have been tested and provided that an abnormality had been detected (predicate inwarning holds). Alerting a nurse causes an alarm message to be prepared to be sent.
- **R5 Sending an alarm message;** a patient is enabled to send an alarm message only if the appropriate alarm message has been prepared. Sending it causes the patient to return

to the condition of "read sensors".

- **R6 Check again the sensors** a patient is enabled to check again the sensors only if the latest readings have been tested and provided that there were no abnormalities (predicate inwarning does not hold). This event causes the condition "read sensors" again to hold.

3.2 Causal View

Based on the informal view given above, the following enhanced Petri Net of Figure 2 can be obtained. Observe that there are two concurrent cycles of activities. The top one deals with the generation of alarm messages. The bottom one addresses the request and reply of patient readings.

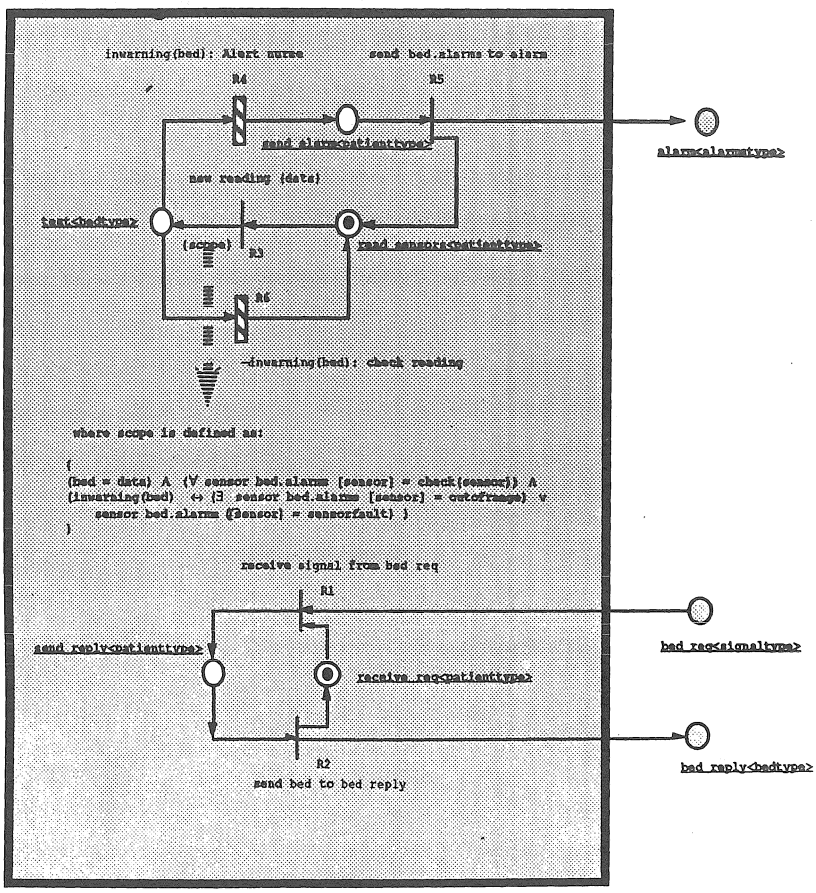

Figure 2: Patient Component: Enhanced Net Description

3.3 Patient Temporal-Causal View

A component specification will consist of a component-id, data-types definition, port-declaration, local-definitions and behaviour. In this example the component-id will be "Patient" and data-types are imported from the definition module "*monmsg*". Among the local-definitions there are variables, condition predicates (for flow control purpose), (normal) predicates and events. The **Behavior** is given in terms of two views: Causal (see Figure 2) and Temporal-Causal presented below. The Temporal-Causal View has four segments: **Causal Properties**, **Safety Properties**, **Liveness Properties** and **Initialization**. The first two are based on the enhanced net description. Each transition in the causal view will correspond to an event in the temporal-causal view, which has a causal and safety property associated with it. One could envisage a scenario in which these two segments are obtained automatically from the causal view with the help of a "translator tool". The **Initialization** segment could possibly be obtained from the enhanced net. It will correspond to the initial "*case*" as well as the setting of the initial values of other logical entities such as predicates.

Liveness properties are unique to the temporal-causal view. They are meant to express when an event must or has to occur. For instance, if a component receives a request for its latest readings (condition predicate bed req(signal) it is committed eventually to reply (occurrence of event "**send** bed **to** bed reply"). Also, if some patient readings are out of range (predicate inwarning(p) holds), then an alarm message will be sent out (occurrence of event "**send** bed.alarms **to** alarm").

Patient Specification

Component Patient;
import monmsg: bedtype, signaltype, alarmstype, patienttype;
exitport alarm: alarmstype;
entryport bed req: signaltype;
exitport bed reply: bedtype;
Variables
bed, data: ⟨bedtype⟩;
p: ⟨patienttype⟩;
signal: ⟨signaltype⟩;
Condition Predicates
bed reply, test: ⟨bedtype⟩;
alarm: ⟨alarmstype⟩;
bed req: ⟨signaltype⟩;
send reply, receive req, read sensors, send alarm: ⟨ patienttype ⟩;
Predicate

inwarning: ⟨bedtype ⟩;

Events

receive *signal* **from** *bed req*: ⟨ signaltype, signaltype ⟩;	(to be used with **R1**)
send *bed* **to** *bed reply*: ⟨ bedtype, bedtype⟩;	(to be used with **R2**)
new reading(data) : ⟨bedtype⟩;	(to be used with **R3**)
Alert nurse: ⟨patientype⟩;	(to be used with **R4**)
send *bed.alarms* **to alarm**: ⟨alarmstype, alarmstype⟩;	(to be used with **R5**)
check reading: ⟨patienttype⟩;	(to be used with **R6**)

Behaviour

Causal View (see Figure 2);

Temporal-Causal View

Causal Properties

P 1 $([\ receive\ \ signal\ \ from\ \ bed\ \ req\ \rangle \overbrace{\underline{send\ \ reply(p)}}^{post-condition} \wedge \overbrace{\underline{\neg bed\ \ req(signal) \wedge \neg receive\ \ req(p)}}^{resetting})$; *(from* **R1***)*

P 2 $([\ send\ \ bed\ \ to\ \ bed\ \ reply\ \rangle \overbrace{\underline{bed\ \ reply(bed) \wedge receive\ \ req(p)}}^{post-condition} \wedge \overbrace{\underline{\neg send\ \ reply(p)}}^{resetting})$; *(from* **R2***)*

P 3 $([\ new\ \ reading(data)\ \rangle\ \overbrace{\underline{test(data)}}^{post-condition}\ \wedge\{\overbrace{(bed = data)}^{scope.1} \wedge \overbrace{(\forall sensor\ bed.alarms[sensor] = check(sensor))}^{scope.2} \wedge$
$\overbrace{(inwarning(bed) \leftrightarrow (\exists\ sensor\ bed.alarms[sensor] = outofrange) \vee (\exists\ sensor\ bed.alarms[sensor] = sensorfault))}^{scope.3}\}) \wedge$
$\overbrace{\underline{\neg read\ \ sensors(p)}}^{resetting}$; *(from* **R3***)*

P 4 $([\ inwarning(bed) : Alert\ \ nurse\ \rangle \overbrace{\underline{send\ \ alarms(p)}}^{post-condition} \wedge \overbrace{\underline{\neg test\ \ (data)}}^{resetting})$; *(from* **R4***)*

P 5 $([\ send\ \ bed.alarms\ \ to\ \ alarm\ \rangle \overbrace{\underline{alarm(bed.alarms \wedge read\ \ sensors(p))}}^{post-condition} \wedge \overbrace{\underline{\neg send\ \ alarm(p)}}^{resetting})$; *(from* **R5***)*

P 6 $([\ \neg inwarning(bed) :\ \ Read\ \ again\ \rangle \overbrace{\underline{read\ \ sensors(p)}}^{post-condition} \wedge \overbrace{\underline{\neg test(data)}}^{resetting})$; *(from* **R6***)*

Safety Properties

P 7 $(\overbrace{\underline{bed\ \ req(signal) \wedge receive\ \ req(p)}}^{precondition} \rightarrow \bigcirc Occ(receive\ \ signal\ \ from\ \ bed\ \ req))$; *(from* **R1***)*

P 8 $(\overbrace{\underline{send\ \ reply(p)}}^{precondition} \rightarrow (\bigcirc Occ(send\ \ bed\ \ to\ \ bed\ \ reply))$; *(from* **R2***)*

P 9 $(\overbrace{\underline{read\ \ sensors(p)}}^{precondition} \rightarrow (\bigcirc Occ(new\ \ reading(data))$; *(from* **R3***)*

P 10 $(\overbrace{\underline{send\ \ alarm(p)}}^{precondition} \rightarrow (\bigcirc Occ(send\ \ bed.alarms\ \ to\ \ alarm))$; *(from* **R5***)*

P 11 ($\overbrace{\underline{test(data)}}^{precondition} \wedge inwarning(bed)) \rightarrow (\bigcirc Occ(Alert\ \ nurse))$; *(from* **R4***)*

P 12 ($\overbrace{\underline{test(data)}}^{precondition} \wedge \neg inwarning(bed)) \rightarrow \bigcirc(Occ(check\ \ reading))$; *(from* **R6***)*

Liveness Properties

P 13 $\underline{bed\ \ req(signal)} \rightarrow \Diamond Occ(send\ \ bed\ \ to\ \ bed\ \ reply)$

P 14 $inwarning(bed) \rightarrow \Diamond\ Occ(send\ \ bed.alarms\ \ to\ \ alarm)$

Initialization

P 15 $\underline{read\ \ sensors(p)} = true \wedge \underline{receive\ \ req(p)} = true \wedge inwarning(p) = false$

end.

The above example demonstrates how the technique can be used to specify individual components of the system. The next step would be the specification of the system as an interconnected set of component instances described by a configuration specification (see [4]). System requirements will be satisfied by the composition of the component specifications.

4 Conclusion and Remarks

We argued that a single representation scheme is often not enough to capture the various features of system behaviour. Instead, multiple viewpoints should be used to partition the domain of information. Its success rests upon the selection of appropriate representation schemes, the careful definition of the relations between them and the process by which such specifications are built within those representation schemes.

We presented a technique that supports the specification of systems from three points of view: informal, causal and temporal-causal. The *Informal View* describes in natural language and graphically the requirements of the component. It is based on the *Conic Environment* which provides a language-based approach to the building of distributed system. The *Causal View* relies on enhanced Petri Nets to describe internal structure, distributed control and safety requirements. The third view makes use of temporal-causal logic to describe temporal features of the components such as liveness requirements. Previous causal properties are preserved. Temporal-causal logic is the result of the integration of two well-known formalisms, namely Petri Nets and Temporal Logic.

Because we concentrated on causal and temporal aspects of system specification, the natural representation schemes were Petri Net and Temporal Logic. The temporal-causal logic was used to define the relations between the techniques.

We then proceeded to give some guidance on how to obtain temporal-causal specification of systems and described the patient component of the patient monitoring system.

5 BIBLIOGRAPHY

References

[1] J. Castro. Temporal-Causal Logic: A marriage of convenience. Technical Report 91/4, Imperial College of Science and Technology, Department of Computing, March 1991.

[2] J. Castro. Reasoning about distributed system specification: The temporal-causal way. In *Proceedings of X Brazilian Symposium on Computer Networks (X SBRC)*, April 1992. Recife, Brazil.

[3] J. Castro and J. Kramer. Temporal-Causal System Specifications. In *Proceedings of IEEE International Conference on Computer Systems and Software Engineering (CompEuro90)*, pages 210–217, May 1990. Tel-Aviv, Israel.

[4] Jaelson F. B. Castro. *Distributed System Specification Using A Temporal-Causal Framework.* PhD thesis, University of London, October 1990. Imperial College of Science, Technology and Medicine, Department of Computing.

[5] J. Kramer. Configuration Programming - A Framework for the Development of Distributable Systems. In *Proceedings of IEEE International Conference on Computer systems and software Engineering (CompEuro90)*, pages 374–384. IEEE Computer Society Press, May 1990.

[6] J. Kramer, J. Magee, and A. Finkelstein. A Constructive Approach to the Design of Distributed Systems. In *Proceedings 10th International Conference on Distributed Computing Systems (ICDCS)*, pages 580–587, Paris, France, June 1990.

[7] W. Myers, G. Myers, and L. Constantine. Structured design. *IBM System Journal*, 13-(12):115–139, 1974.

[8] Wolfgang Reisig. Towards a temporal logic for causality and choice in distributed systems. In J.W Bakker, W. P. de Roever, and G. Rozemberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 603–627. Springer-Verlag, 1989. LNCS 354.

[9] Ichiro Suzuki and HarnGdar Lu. Temporal petri nets and their application to modeling and analysis of a handshake daisy chain arbiter. *IEEE Transactions on Computers*, 38(5), May 1989.