A Framework and Patterns for the Specification of Reactive Systems

Leonor Barroca
Dept. of Computing
The Open University
Milton Keynes, U.K.

Pedro Henriques
Dep de Informática
Universidade do Minho
Braga, Portugal

Abstract

We defend the need of a dual approach for specifying reactive systems and reasoning about their timing properties; this dual approach uses a notation for the specification of behaviour, preferably a graphical one, and a temporal logic to express formally, and reason about, this behaviour. The combination of two representation schemes forms the basis of a generic framework that can be instantiated for different choices of notations. Two instantiations of the framework, illustrating its use, are discussed. To describe in detail one of the instantiations, the ArchSM method, we use the *pattern form*.

1 Introduction

This paper discusses the application of patterns to detail the development of software environments, namely object-oriented frameworks for the specification of, and reasoning about, the behaviour of reactive systems and their timing properties. *Reactive systems* are those whose behaviour is strongly determined by the response to the occurrence of external and internal events, with or without time constraints.
We have been developing notations that are formally underpinned and that can be used pragmatically to build models of systems with important timing constraints. We defend the need of a dual approach that uses a notation for the specification of behaviour, preferably a graphical one, and a temporal logic to express formally, and reason about, this behaviour. We made specific choices for the notations to be used when developing our specifications. However, we recognize that these choices are not the only ones possible. When developing environments to support specification and verification we opted for taking a more generic approach that would allow for different choices. This led us to define a generic framework that can be instantiated for each specific approach. In this paper we define the generic framework and two of its possible instantiations to illustrate its use.
One of the instantiations of the framework is described in detail using the *pattern form*. As suggested by Johnson[Joh92], the use of a pattern to give the details of the design is a good way of documenting the framework. The development of the framework led us smoothly to the identification and acceptance of the patterns used.

An *object-oriented framework* [WB88] is a set of classes that are integrated and interact in a well defined way to provide a set of services. The use of *object-oriented* technology brings in a set of facilities. The encapsulation provided by the classes supports abstraction, and the interaction through message passing facilitates the interface between components. Frameworks provide a structured way of defining the architecture of the system to be developed, by describing an abstract model to solve similar problems. They represent a strategy

for reuse in a specific domain or across a wide set of applications.
The framework discussed in this paper captures specific problem domain expertise, environments for the specification and verification of reactive systems.

According to [GHJV95], an *object-oriented pattern* is a set of interrelated objects, with well-defined interfaces, aiming at the characterization of a problem, and its context, and describing a reusable solution for it.
Three components will be defined when describing a *pattern*, namely:

- **pattern name**
- **problem description**: intent, motivation and applicability
- **proposed solution**: participants and their structure

It is often suggested that a clear identification of the **forces** and **consequences** of adopting that solution should also be present. We omit these components as our use of *patterns* is in the context of detailing *frameworks*. The concept of *pattern* does not restrict the kind of problems to be solved. It can be used to describe how to apply a methodology to accomplish a given task. In this case, the proposed solution is not concerned with implementation details; it contains the steps to be followed, or the strategy to be used.

1.1 Paper Overview

In the next section we introduce the *architectural framework* for the specification of reactive systems. We define its purpose, the sets of participating classes, and two known examples of use.
Section 3 is devoted to present two of the *patterns* designed to instantiate the proposed *framework*. We describe the approach followed to derive the *patterns* and the notation used.
Section 4 summarizes the proposal presented.

2 The Architectural Framework

The problem we want to address is the representation of, and reasoning about dynamic behaviour of reactive systems using a dual notation approach.

2.1 Purpose of the framework

The purpose of this *framework* is to describe the architecture of a specification method based on a dual approach. This description is intended to support the construction of environments for the pragmatic application of the proposed formal method.
By dual approach we mean the use of two different notations; a graphic notation to represent the behaviour of a system, and a mathematical notation to formally describe the properties of models and reason about the behaviour.

2.2 Definition of the framework

This framework consists of seven classes divided into three groups. Two *main* classes model the two notations; three application classes model application specific knowledge; two *operational* classes define the interconnections between the *application* classes.

Main Classes

- Diagram Notation—describes the diagrammatic notation used to specify the temporal behaviour of the system components and their interactions.
- Logic Notation—defines the axiomatisation of the formal language used to represent the properties and the axioms.

Application classes

- Behaviour diagrams—contains a set of drawings (in the diagrammatic notation defined above) that represent the system behaviour.
- Behaviour axioms—contains a set of sentences (in the logic defined above) that axiomatizes the system behaviour; generated by the mapping class (see below).
- Properties—contains a set of sentences (in the logic defined above) that defines the system properties to be proved; defined by the user.

Operational Classes

- Mapping—defines the translation rules deriving the axiomatisation from the diagrammatic description of the behaviour of a system.
- Theorem Prover—defines the representation schemas to describe the logic notation in the logic of the theorem prover; this class is only present in case the environment provides tool supported verification.

The *main* classes correspond to the philosophy of each specification approach; they define the concepts used in the specification.
The *operational* classes represent the strategy of transformations; they are instantiated by engines that can be implemented manually or supported by software tools.
The *application* classes are repositories of information used by the engines and built according to the main classes.

Figure 1 illustrates the proposed framework. The main classes are shown on top with a thick border; the application classes below with a thin border; the operational classes in the bottom with a dashed border. The links between the main and application classes represent a satisfaction relationship; the arrows connecting the application and operational classes establish the direction of transformation.

2.3 Uses of the Framework

In this section we present two instantiations of the framework, ArchSM and TRIO.

2.3.1 Example 1: ArchSM

A collection of techniques has been proposed under the name of *Architectural Specification Method* (ArchSM), for the formal specification of real-time systems [AB96, BFS95]. ArchSM supports modelling and reasoning about three important aspects of a reactive system: timing properties, behaviour, and structure. ArchSM uses a graphical notation based on *Timed Statecharts* (TSC) for describing the dynamic behaviour, and *Real Time Logic* (RTL) for reasoning about the temporal properties.

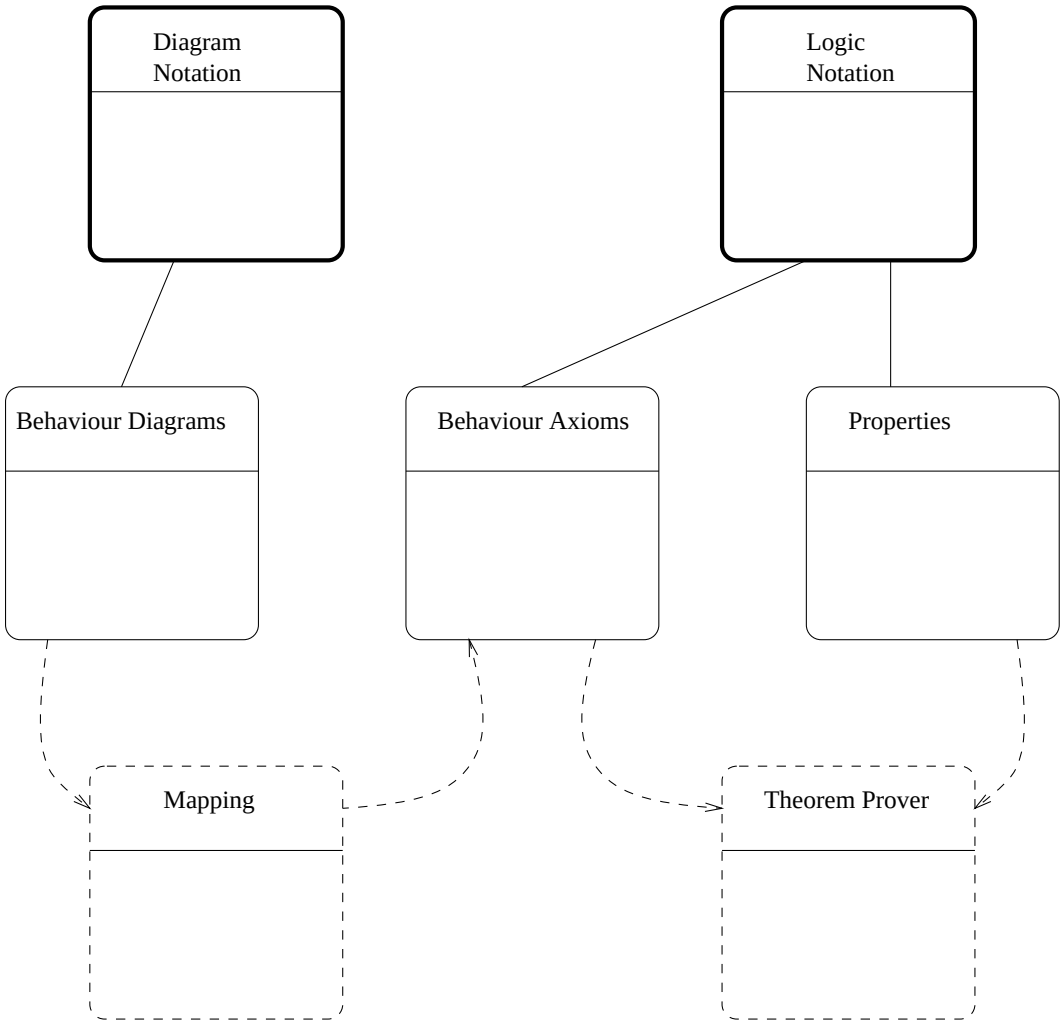

Figure 1: The Framework

This environment has been applied to the specification of several real-time systems [AB96, CBF+95, BFS95, BHV96].

Temporal Behaviour
State transition diagrams have been widely used as a design tool in industry. We chose to represent the timing behaviour of systems with a graphical notation, based on TSC. This has the advantage of being accessible to those without a great level of mathematical background, while being sufficiently expressive and formal. *Statecharts* [Har87] are an extension of state transition diagrams for the specification of *reactive* systems. *Timed Statecharts* [HMP91] extend this notion to deal with time constraints.

Reasoning about Temporal Behaviour
The production of rigorous arguments is important for the verification process. In ArchSM, RTL is used to construct the arguments about time properties of systems whose timing behaviour is expressed using TSC. *Real Time Logic* [JMS88] is a formal language for capturing the time constraints of real-time systems. It deals with time quantitatively, instead of just relative temporal order. In ArchSM, to produce rigorous statements and proofs of timing properties, statecharts are treated as theories in RTL.

ArchSM as an instantiation of the framework
ArchSM can be seen as a specialization of the generic framework defined above, according to the following instantiation.

Diagram notation	*Timed Statecharts* (TSC)
Logic notation	*Real Time Logic* (RTL)
Mapping	a set of translation rules TSC $\mapsto$RTL
Theorem Prover	HOL (alternatively Isabelle, ...)

2.3.2 Example 2: TRIO

TRIO[GMM90] is a first order temporal logic augmented with temporal operators to specify properties that change with time. A method has been proposed at the Politecnico di Milano[FMM94] to combine *Timed Petri Nets* (TPN) [MF76] with TRIO. This method is also a dual-language approach for the analysis of real-time systems. Behaviour is expressed using TPN; and the axiomatisation of the behaviour is expressed in TRIO axioms and proof rules. There are many similarities between this approach and ArchSM: TPN is used to describe the dynamic behaviour instead of TSC; TRIO is used to reason about the temporal properties instead of RTL.

TRIO as an instantiation of the framework
TRIO is also an example of the specialization of the proposed framework with the following instantiation.

Diagram notation	*Timed Petri Nets* (TPN)
Logic notation	TRIO
Mapping	axiomatisation in TRIO of TPN
Theorem Prover	The proof is carried out manually

2.4 Design of the framework

To complete the definition of the *framework* we need to specify the classes identified above. For each class we have to define its attributes, their type, and, if appropriate, its operations. The classes that describe notations (*main classes*) and those that contain application specific knowledge (*application classes*) are static and have no relevant behaviour to be described.

In the next section we specify the two *main classes*, give the ArchSM instantiation for each, and describe their design using the *pattern* form.

3 Patterns describing the Framework

In the previous section we mentioned the ArchSM realization of the proposed *framework*. In this section we describe *patterns* to give detail of that instantiation.
We considered the specification of the `DiagramNotation class` and `LogicNotation class` of the *framework*; this led us systematically to understand the information of, and the design of the *patterns*.

3.1 Notation

The notation used to define the types of attributes is based on set theory.
To describe the patterns we have adapted the template suggested by Gamma et al in [GHJV95].
The diagrams presented to define the structure of each pattern follow the OMT [RBPL91] object model notation.

3.2 Pattern 1: TScSpec

The definition of this `DiagramNotation` class is as follows.

class: `DiagramNotation`
attributes:
`atomic-elements: ℙ Token`
`compound-elements: ℙ Token`
`start-element: Token`
`structural-rules:`
`ℙ(compound-elements → ℙ (atomic-elements ∪ compound-elements))`

operations:
restrictions:
`"the start-element must be a compound-element"`

In the context of ArchSM, where the chosen notation is based on TSC, the class defined above is instantiated as follows.

object: `TSC` instance-of `DiagramNotation`
attributes:

```
atomic-elements: {state, event, action, condition, time-interval}
compound-elements: {system, eventflowdiagram, class,
                    machine, transition, label, effect,
                    trigger, source, target, initial-state}
start-element: system
structural-rules: {transition → state × state × label,
                       label → trigger × effect,
                       trigger → event ∪ time-interval,
                       effect → ℙ event × action
                       machine → initial-state × ℙ state × ℙ transition,
                       ... }
```

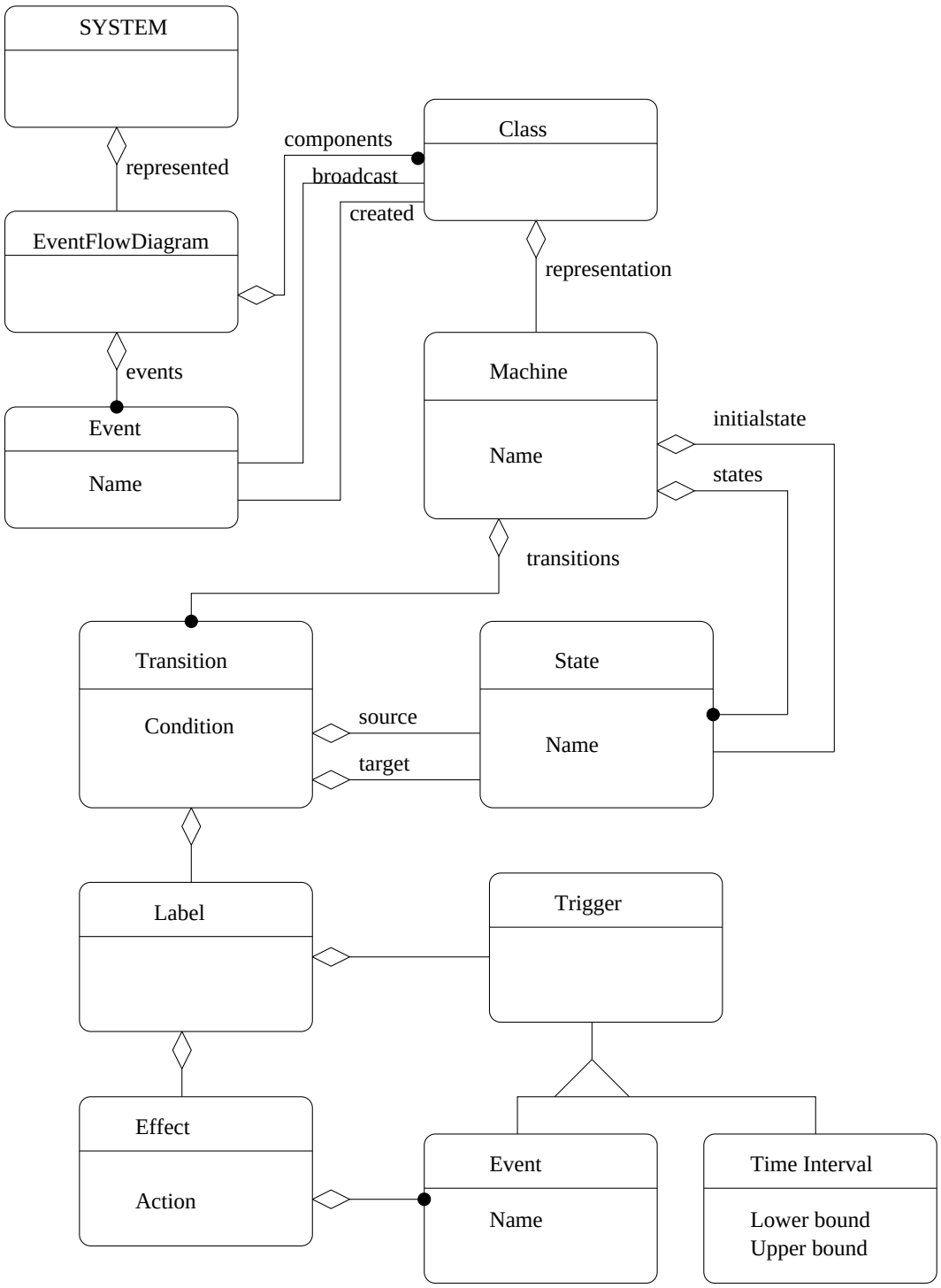

Figure 2: The pattern for *Timed Statecharts Specifications*

The instantiation above is described in detail by the following *pattern*.

Name

TScSpec (*Timed Statecharts Specification Pattern*)

Intent

Given a reactive system, describe its behaviour based on the idea of a set of machines with their own internal state, reacting to events.

Motivation

Reactive systems interact with the environment by changing state as a result of the occurrence of events or of time elapsing. As a consequence of this change of state, new events are generated and actions performed. This behaviour is usually represented by state transition machines preferably with time annotations. This *pattern* represents the concepts that are common in the representation of this type of behaviour.

Applicability

This *pattern* is applicable to all specification problems where reactive behaviour is identified. It is particularly suitable when a system can be split into interacting components exhibiting independent behaviour.

Structure

From the information about TSC given by the class definition it is easy to infer the objects, their attributes and interrelations, involved in that specification approach. The diagram shown in figure 2 illustrates the structure of the pattern.

Participants

- **System**—the entire problem to be specified.
- **Event Flow Diagram**—the partition of the system into components and the communication between them.
- **Event**—an instantaneous occurrence of and external or internal stimulus to the system.
- **Class**—a component that encapsulates an individual reactive machine as a black box.
- **Machine**—representation of the behaviour of a reactive machine.
- **Transition**—representation of a firing, its causes and consequences
- **Label**—annotation of a transition.
- **Trigger**—cause of a transition firing.
- **Effect**—consequence of a transition firing.
- **State**—a snapshot of a reactive machine that determines how the machine reacts.
- **Time Interval**—a pair of two time bounds.

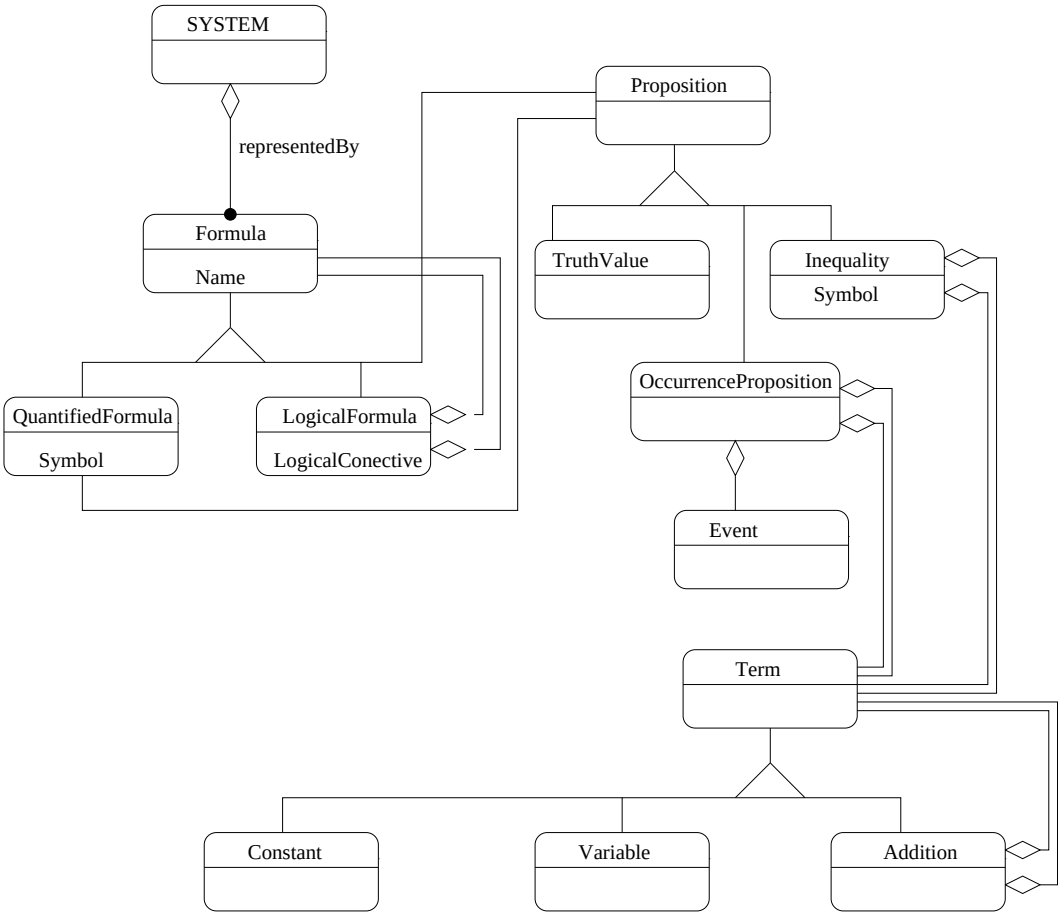

Figure 3: The pattern for *Real Time Logic Specifications*

Known Uses

The *pattern* proposed here has been used in the development of an environment for the application of ArchSM [BHV96].
This *pattern* is also used to present ArchSM every time we want to teach it.

3.3 Pattern 2: RTLSpec

The definition of the `LogicNotation class` is as follows.

class: `LogicNotation`
attributes:
`atomic-elements: ℙ Token`
`compound-elements: ℙ Token`
`structural-rules:`
`ℙ(compound-elements → ℙ (atomic-elements ∪ compound-elements))`
`operators: ℙ Token`
`axioms: ℙ Rewriting-Rule`
operations:

restrictions:

In the context of ArchSM, where the chosen notation is RTL, the `LogicNotation` class is instantiated as follows.

object: `RTL` instance-of `LogicNotation`
attributes:

```
atomic-elements: {truth-value, variable, constant, event}
compound-elements: {formula, quantified-formula, logical-formula,
                    proposition, inequality, occurrence-proposition,
                    term, addition }
structural-rules: {formula → proposition ∪ quantified-formula ∪
                        logical-formula,
                    proposition → truth-value ∪ occurrence-proposition ∪
                        inequality,
                    ... }
operators: {+, predicate-symbols (<, ≤, >, ≥, =),
            occurrence-symbol θ ,
            logical-connectives (∧, ∨, ¬, ⇒),
            quantifiers (∃, ∀ )}
axioms: {Monotonicity-Axioms, Start-Stop-Event-Axioms,
         Transition-Event-Axioms}
```

The instantiation above is described in detail by the following *pattern*.

Name

RTLSpec (*Real Time Logic Specification Pattern*)

Intent

Given a reactive system, describe its behaviour based on the idea of a set of logical formulas written in RTL.

Motivation

Reactive systems interact with the environment by changing state as a result of the occurrence of events or of time elapsing. As a consequence of this change of state, new events are generated and actions performed. This *pattern* represents an approach that is useful to prove properties of systems exhibiting reactive behaviour, mainly when absolute time constraints are imposed.

Applicability

This *pattern* is applicable to all specification problems where reactive behaviour with time constraints is identified.

Structure

The diagram shown in figure 3 illustrates the structure of the pattern.

Participants

- **System**—the entire problem to be specified.
- **RTL spec**—the temporal logic description of the behaviour of the system components.
- **Formula**—an axiom of the behaviour of the system.
- **Logical Formula**—the connection of propositions by a logical operator.
- **Quantified Formula**—an assertion universally or existentially quantified.
- **Proposition**—an assertion on the system dynamics.
- **Term**—absolute time representation.
- **Inequality**—a relational operation between terms.
- **Addition**—the addition of two terms.
- **Truth Value**—true, or false.
- **Occurrence Proposition**—assertion about the n-th occurrence of an event at a certain time.
- **Event**—an external or internal instantaneous stimulus to the system.

Known Uses

This *pattern* is used to develop specifications in RTL, and to present ArchSM every time we want to teach it.

4 Conclusion

We have been working with a group of techniques under the name of ArchSM for the formal specification of real time systems. ArchSM is intended to support modelling and reasoning about time properties and behaviour of reactive systems. When we began the development of tools for the pragmatic application of these techniques we opted to stick to a dual representation approach without necessarily adhering to a particular choice of notations.
This decision led us to model in an abstract way the method we were working with. The notion of *framework* proved to be suitable for this purpose. Although not its most common use, we found the concept of *pattern* interesting to describe detail of the *framework*; its applicability has also been demonstrated in the subsequent development of the tools referred above.
By using an *architectural framework* and *patterns* in the context of rigorous specifications, we believe that this work is a contribution to the cross-fertilization between the formal methods area and the *object-oriented patterns* research.

Acknowledgements

We thank the British Council and JNICT (Portuguese Junta Nacional de Investigação Científica e Tecnológica) for the grant under the BC/JNICT protocol, project 67/proc.423/RU, that has supported this research.

We also thank Norm Kerth who was the EuroPLoP'96 shepherd for this paper for making helpful comments that led to improvements to its first version.
We acknowledge the valuable contribution of the participants in the Writers Workshop 1: Pattern Languages at EuroPLoP'96.

References

[AB96] J. Armstrong and L. Barroca. Specification and verification of reactive system behaviour: The railroad crossing example. *Real-Time Systems*, March 1996.

[BFS95] L. Barroca, J. Fitzgerald, and L. Spencer. The architectural specification of an avionic subsystem. In *Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques*, Boca Raton, Florida, April 1995.

[BHV96] Leonor Barroca, Pedro R. Henriques, and Maria Joao Varanda. Language and Environments for the Pragmatic Applications of Formal Methods: project reports. JNICT/BC – Project Report UMDITR(9507—9601), Departamento de Informática da Universidade do Minho, February 1996.

[CBF+95] A. Coombes, L. Barroca, J. Fitzgerald, J. McDermid, L. Spencer, and A. Saeed. Formal specification of an aerospace system: The attitude monitor. In M. Hinchey and J. Bowen, editors, *Applications of Formal Methods*. Prentice-Hall, 1995.

[FMM94] M. Felder, D. Mandrioli, and A. Morzenti. Proving properties of real-time systems through logical specifications and petri net models. *IEEE Transactions on Software Engineering*, 20(2):127–141, 1994.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GMM90] C. Ghezzi, D. Mandrioli, and A. Morzenti. Trio: A logic language for executable specifications of real-time systems. *Journal of Systems and Software*, 12:107–123, 1990.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HMP91] T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Proceedings of the REX Workshop–Real-Time: Theory and Practice*, 1991.

[JMS88] F. Jahanian, A. K. Mok, and D. Stuart. Formal specification of real-time systems. Tr-88-25, Dept. of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712, 1988.

[Joh92] R. Johnson. Documenting frameworks using patterns. In *Proceedings of OOPSLA '92*, 1992.

[MF76] P. Merlin and D. Farber. Recoverability of communication protocols – implications of a theoretical study. *IEEE Transactions on Communications*, September 1976.

[RBPL91] J. Rumbaugh, M. Blaha, W. Premerlani, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[WB88] R. Wirfs-Brock. Object-oriented frameworks. *American Programmer*, 4(10), 1988.