Specifying and Implementing
Control Software Systems with Hybrid Paradigms

P. Nesi, P. Bellini, D. Rogai
DSI – University of Florence

Abstract
The realization of control software systems could be a complex task especially when we are dealing with decentralized control and when the temporal constraints that the system has to realize are not straightforward. The realization of such systems could be supported by a hybrid methodology joining the good of denotational formalisms and of operational formalisms. If the methodology supports the development from the early specification down to the implementation this facilitate the realization of a good product, reducing the application errors and software failures. The paper presents TILCO an executable temporal logic supporting the development from the specification phase to the implementation of control systems. In the implementation phase a particular focus has been given to the integration with source code (C++) for the interaction with hardware/low-level components.

1 Introduction
Generally complex systems are considered as systems with many interrelated parts, with complex interactions and also dealing with heterogeneous platforms. Such systems are difficult to specify, to model, to validate and obviously to be correctly implemented. When we consider control systems (or real-time systems in general), dealing with many temporal constraints is a fundamental problem. Since control systems are typically composed by many parts with specific time behaviour and not trivial interactions among them, suitable methodologies are needed to develop an implementation which grants global time properties to the system. Sometimes the system is not only structurally complex (many interrelated parts) the source of complexity is in the time constraints themselves. In general real-time systems cannot benefit from traditional debug phases, thus to achieve a correct behaviour a formal methodology is useful especially when it is supported by effective tools.

How to deal with temporal constraints?
Dealing with temporal constraints means:
- being able to describe the temporal constraints while specifying the system behavior;
- put in the system model the “correct” temporal behavior when designing the system;
- enforce in the system implementation the “correct” temporal behavior.

When coping with complexity the description, modeling and implementation of timing constraints is not straightforward and a validation and verification phase is often needed. The use of formal languages aids from the early development stages in the understanding and in the growth of the system.

Temporal behavior description
Temporal logics are generally considered the easiest way to describe temporal constraints (see [1] for a survey) other solutions are possible as in process algebras, where an abstract model is used to capture a specific behavior abstracting away the other details. However it seems to be more “natural” to describe properties using temporal logics rather than using an abstract model.
When considering complex time constraints the use of tools supporting the visualization, comprehension and validation of constraints is mandatory, without these tools the system designer is left alone in the complexity management.

**Temporal behavior model**
When modeling the system behavior the operational formalisms as state charts, Petri nets, timed automata, process algebras etc. are the more commonly used. But not only, in recent years also (temporal) logics can be used to build the system model thus not capturing only some possible behaviors that the system have to satisfy but specify the complete behavior of the system. Also in this phase the support of tools to verify that the modeled behavior is the correct one is of great importance particularly when we are dealing with a complex situation.

**Temporal behavior implementation**
When we come to the implementation the global behavior of the system is affected not only from the application implementation, also the operating system on which the application is running can influence the application behavior. For this reason specific real-time operating systems are used to deploy the real-time applications. Tools or specific programming languages were developed to support this phase, however passing from the model to the implementation is a critical task.

**Hybrid methodologies**
In this context, the “hybrid” attribute refers to the capability of describe and model a timed system with a uniform formalism supporting both the high-level denotational specification of system properties and the operational model of the complete system behavior. On the other side when we come to the implementation “hybrid” also refers to the possibility of integrating a formal method within a low-level development framework (e.g. C++ for peripheral communication).

In literature some examples of the presented approach are reported in [2].

### 2 The TILCO framework
TILCO [1][3] temporal logic extends FOL to express time constraints in qualitative and quantitative manner. The time is linear, implicit and discrete, the time interval is the main entity and two operator have been conceived on it: “@”, which represents the universal quantifier, and “?” which works as the existential quantifier. Other operators like since and until are provided to express ordering relationships.

TILCO specifications can be executed in real-time, in this case “execution” means that outputs can be produced on the basis on inference of system inputs and states [4]. This important feature suggest its utilization in a hybrid approach:

- to use logic to describe temporal properties of the system in concise expressions [5],
- to use logic to model the behaviour of the system with the possibility of decomposing sub-processes and formally describing the communication among them [6],
- to use a traditional programming language based on object oriented paradigm to develop other functional parts like data management, user interface etc [7].

Since such a hybrid approach can assemble different languages, a consistence management must be provided by an integrated development environment which aids the process to coordinate both modeled aspects [7].

In the TILCO framework the consistency of the global development project is granted by a global declaration of common entities between the two worlds. On the basis of this key elements both development parts are independent and the proper connection is ensured by the declared names. This solution requires different attitudes to gain most benefits and it is suitable for time-critical control systems:

- first, the time constraints must be collected and organized in a hierarchical structure of processes;
then, the common entities must be recognized and listed in declarations;
the time requirements must be specified using the identified key elements connecting the
specified temporal behavior to the traditional operational components;
the functional aspects should be programmed following the object oriented paradigm and
taking into account that the specified behavior is wrapped in objects as well;
the connection between the two styles must be actuated by using the event-driven paradigm,
in this case “events” are related with activation and deactivation of predicates in the
specification, other methods must be customized to travel information from the programmed
world to the executed specification.

Since the consistency check and common entities connection it is made automatically by the
development environment, which aids the system designer in the previously depicted steps, all the
system complexity is bounded in the denotational part: the time behaviour specification. This does
not mean that the system is not complex anymore, but that such a complexity can be faced with
formal methods tools.

Thanks to this separation the temporal aspect can be:
- exhaustively validated by proving constraint soundness and properties of safeness or
  liveness;
- easily implemented, since they are described, and a direct execution can be performed, no
coding effort (no translation from specification to programming language) has to be
  provided, thus no hard debug phase is required;
- safely linked with functional aspects by using input and output predicates in order to
  activate suitable methods to change the state in the operational side.

On the other side functional requirements can be managed as usual, with the advantage of a clear
distinction from temporal behavior, which typically requires code tricks which lower the readability
and manageability of the program. The programming language augment its capability by attaching
methods to the event-driven paradigm that are triggered from the specified behaviour and can
perform periodic tasks, mutual exclusion, synchronization constraints.

Finally a very important feature is hidden in this context independence, that is the capability of
changing one of the two sides of this approach with no dependence on the other, unless the
connection entities are respected: it is possible to change quantitative parameters without rebuild the
whole system from the beginning, and it is possible to change the functional behaviour that occurs
at a certain event without losing safeness in the time structure.

---

**Fig. 1:** (a) the development process that deals with complexity
(b) the basic architecture to integrate denotational and operational approach
References


