Composition of Reactive System Components

K. Lano, J. Bicarregui, T. Maibaum
Dept. of Computing, Imperial College, 180 Queens Gate, London SW7 2BZ
kcl@doc.ic.ac.uk

J. Fiadeiro
Dept. of Informatics, University of Lisbon, Campo Grande, 1700 Lisbon

Abstract

This paper will present the case for using a formal component-based specification technique for reactive systems, such as the Object Calculus of Fiadeiro and Maibaum. The Object Calculus provides a modular, highly declarative and abstract specification language, suitable for refinement using model-based design notations such as B or VDM.

In the Object Calculus, pre/post style specifications of the effect of actions can be given, together with temporal logic specifications of expected histories of behaviour of the system.

Keywords: Temporal logic, Reactive systems, Program specification, Object Calculus, Specification languages.

Workshop Goals: Investigate application of formal specification in component-based systems, particularly reactive systems.

1 Background

Temporal logic is an established technique for the specification of reactive systems: it has the advantage of being declarative and supporting reasoning, and it is sufficiently expressive for many practical cases. The Object Calculus adds a strong concept of encapsulation and theory composition to a basic temporal logic formalism [8], which allows reactive system components to be separately specified, instantiated and combined using category-theoretic operations, in particular, the co-limit construction:

> *"given a category of widgets, the operation of putting a system of widgets together to form some super-widget corresponds to taking the co-limit of the diagram of widgets that shows how to interconnect them" [10]*

Using this integration of category-theoretic structuring and temporal or modal logics, the development of the Object Calculus has been carried out by research groups at Imperial College and the University of Lisbon over the last 10 years. It has been taken up by other research groups and applied to systems of significant complexity, such as the steam boiler system described here.

In this paper we will use examples from a case study of an established benchmark for formal methods, the steam boiler system, to illustrate the techniques of abstract and compositional specification using the Object Calculus.

A description of the steam boiler system can be found in [2], together with different approaches to formal specification of it.

The purpose of the system is to produce a flow of steam from the boiler water tank, without letting the tank boil dry or overflow. Failures in the measuring devices involved (flow monitors on the water feed lines, steam level sensor and water level sensor) and the water pumps must be handled by an appropriate change of mode of the controller – in emergency situations this may involve a shutdown of the control system. Figure 1 shows the main components of the system.

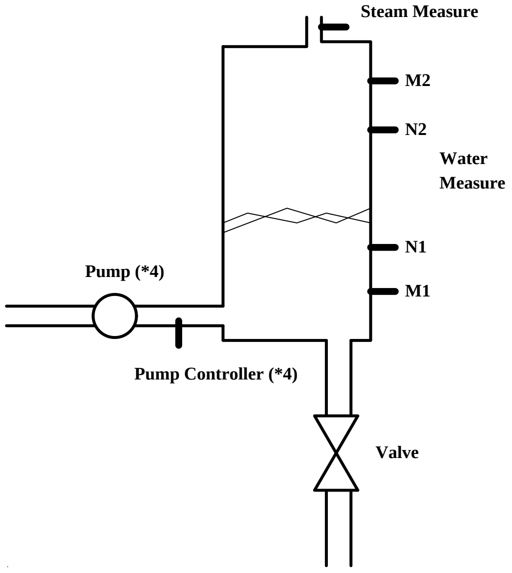

Figure 1: Steam Boiler Components

A specification in the object calculus [8] is constructed as a set of linked theories in a temporal logic. Formally, it is a diagram of objects in a category of theories and theorem-preserving morphisms. A theory consists of collections of *type* and *constant symbols*, *attribute symbols* (denoting time-varying data), *action symbols* (denoting atomic operations) and a set of axioms describing the types of the attributes and the effects, permission constraints and other dynamic properties of the actions. The axioms are specified using linear temporal logic operators: $\bigcirc$ (in the next state), $\mathcal{U}$ (weak until), $\Box$ (always in the future) and $\Diamond$ (sometime in the future). There is assumed to be a first moment. The predicate *BEG* is true exactly at this time point.

$\bigcirc$ is also an expression constructor. If e is an expression, $\bigcirc e$ denotes the value of e at the beginning of the next time interval. e itself denotes its value at the beginning of the current interval. Several actions may execute in a given interval: the formula α where α is an action, denotes that α occurs in the current interval. We express the effects of actions via axioms of the form:

$$Pre \wedge \alpha \;\Rightarrow\; Post$$

where *Pre* is a precondition, a predicate over the current state, and *Post* describes the properties of the state that results from execution of α in a state satisfying *Pre*. It may use both $\bigcirc att$ and *att* for attributes *att* of the theory.

A wide variety of properties can be expressed using such a logic. In particular it seems appropriate for the specification of the steam boiler problem as the requirements of this system are expressed in terms of reaction cycles (intervals) where a collection of events (actions) occur, including inputs to the system, its internal reactions, and outputs from the system to the physical devices. Constraints between the events in a given cycle include that multiple *level* messages in a given interval should give rise to a transmission error:

$$\neg\ \exists_1\, lev : \mathbb{Z} \cdot level(lev) \quad \Rightarrow \quad transmission\_failure$$

$\exists_1\, x$ is the "exists a unique x" quantifier.

Constraints between events in successive cycles include that three successive *stop* messages give rise to a termination (in the same cycle as the third *stop*):

$$stop \wedge \bigcirc stop \wedge \bigcirc\bigcirc\, stop \quad \Rightarrow \quad \bigcirc\bigcirc terminate$$

and the protocol for failure detection and acknowledgement:

$$\begin{array}{l} water\_measure\_failed \ \Rightarrow \\ \qquad (level\_failure\_detection\ \mathcal{U}\ level\_failure\_acknowledgement) \end{array}$$

"If the water measure fails in the current cycle, the message *level_failure_detection* is repeated until a *level_failure_acknowledgement* message is received." The use of weak until means that there is no obligation for an acknowledgement message to ever be received[1].

In order to support reasoning about the attributes which may change over a given interval, we associate to each action the set of attributes which it may change: its *write frame*. For each attribute *att* we then have a *locality axiom* of the form

$$att = \bigcirc att \ \vee \ \alpha_1 \ \vee \ \ldots \ \vee \ \alpha_n$$

where the α_i are all those actions with *att* in their write frame.

Theories are connected by means of *theory morphisms* σ which map each attribute symbol *att* of the source theory S to an attribute symbol $\sigma(att)$ of the target theory T, each action α of S to an action $\sigma(\alpha)$ of T, and so forth. Each theorem of S must become a theorem of T under this translation:

$$\vdash_S \varphi \ \text{ implies } \ \vdash_T \sigma(\varphi)$$

Preservation of the locality axioms means that no new actions (not in the image of σ) can be introduced in T which directly write attributes *att* of S. Any action with $\sigma(att)$ in its write frame must be (or must always co-execute with) the interpretation of some action α of S where *att* is in the write frame of α in S.

This form of encapsulation is close to that of B [1]: only operations declared in a given B module (machine) may directly write to variables declared in that module.

2 Position

The central problem with the specification of reactive systems is obtaining a sufficiently abstract description to avoid the high numbers of states which make verification difficult. In particular, we need to be able to describe the allowed sequencing of phases and operations without coding up details of component implementations.

We believe that the Object Calculus formalism provides a suitable framework for the description of reactive system components. Such components are often of a generic nature,

[1]Technically, this means that this property is a *safety* rather than a *liveness* property.

consisting of variations or enhancements of fundamental physical devices such as valves, tanks, pumps, etc. Particular systems are built from a combination of these components. Thus the Object Calculus seems an appropriate formalism for their specification since it allows convenient extension, adaption and composition of component specifications. Moreover, the key properties of such components concern their dynamic and reactive behaviour, for which temporal logic is ideally suited.

In the following sections we illustrate the use of the object calculus in specifying the steam boiler control system.

2.1 Abstract Specification

At the most abstract level, all actions of a system can be assumed to occur in intervals without overlap. An interval at this level of abstraction represents a cycle of the concrete system.

A theory *SData* gives definitions of the types and constants used in the system, and will be included in each of the other theories:

Types

$PState = \{running, off\}$
$PCState = \{flow, noflow\}$
$Condition = \{failed, operating\}$
$@Pump = \{p1, p2, p3, p4\}$

Constants

$M1 : \mathbb{N}$ /* *Minimum water level* */
$M2 : \mathbb{N}$ /* *Maximum water level* */
$N1 : \mathbb{N}$ /* *Minimum normal water level* */
$N2 : \mathbb{N}$ /* *Maximum normal water level* */
$W : \mathbb{N}_1$ /* *Maximum steam production* */
$C : \mathbb{N}_1$ /* *Capacity of boiler* */
$P : \mathbb{N}_1$ /* *Capacity of pump* */
$U1 : \mathbb{N}$ /* *Max rate of steam increase* */
$U2 : \mathbb{N}$ /* *Min rate of steam increase* */
$\tau : \mathbb{N}_1$ /* *Sampling interval* */

Axioms $M1 < N1 \ \wedge \ N1 < N2 \ \wedge \ N2 < M2$

Each physical component has an associated *monitor* which provides an interface between it and the controller. This monitor is responsible for managing the protocol of communications between the controller and the components, and for detecting errors in data and communications.

The axioms of the monitor theory for the water measure formalise the requirements given in [2, pages 500–509].

Attributes

$water\_measure\_condition : Condition$
$water\_quantity : \mathbb{Z}$

Actions (With write frames presented as sets of attributes:)

$level(lev : \mathbb{Z}) \quad \{water\_quantity\}$
$water\_measure\_failed \quad \{water\_measure\_condition\}$
$transmission\_failure \quad \varnothing$
$level\_failure\_detection \quad \varnothing$

level_failure_acknowledgement $\varnothing$
level_repaired $\{water\_measure\_condition\}$
level_repaired_acknowledgement $\varnothing$

Axioms Initially the water quantity is 0 and the measure is operating:

$$BEG \;\Rightarrow\; water\_quantity = 0 \;\wedge\; water\_measure\_condition = operating$$

A water measure failure event occurs if we receive a $level(lev)$ message with $lev < 0$ or $lev > C$:

$$\exists\, lev : \mathbb{Z} \cdot level(lev) \wedge (lev < 0 \vee lev > C) \;\Rightarrow\; water\_measure\_failed$$

A transmission failure occurs if we do not receive a unique *level* message in the current cycle:

$$\neg\; \exists_1\, lev : \mathbb{Z} \cdot level(lev) \;\Rightarrow\; transmission\_failure$$

Notice that this includes the case where no *level* message is received.

If a level measure failure occurs, the system must react by recording the failure:

$$water\_measure\_failed \;\Rightarrow\; \bigcirc\, water\_measure\_condition = failed$$

The signal *level_failure_detection* must then be repeated until *level_failure_acknowledgement* is received:

$$water\_measure\_failed \;\Rightarrow\; (level\_failure\_detection \;\mathcal{U}\; level\_failure\_acknowledgement)$$

A *level_repaired* signal resets the *water_measure_condition* attribute:

$$\neg\; water\_measure\_failed \wedge level\_repaired \;\Rightarrow\; \bigcirc water\_measure\_condition = operating$$

and leads to the generation of a *level_repaired_acknowledgement*:

$$level\_repaired \;\Rightarrow\; level\_repaired\_acknowledgement$$

A valid water level event sets the value of *water_quantity*:

$$\neg\; water\_measure\_failed \;\wedge\; \neg\; transmission\_failure \;\Rightarrow\; (level(lev) \;\Rightarrow\; \bigcirc\, water\_quantity = lev)$$

The theory of the steam measure monitor is identical in structure to the water measure monitor. Formally it is an isomorphic image under the morphism which maps C to W, *water_measure_condition* to *steam_measure_condition*, etc.

A similar structure can be given for the theory of the pump, pump monitor and the pump controller and its monitor. Indeed we can recognise a number of commonalities between the monitor theories (only the criteria for detecting sensor failures, and for recording the current state, are different). There are common subtheories *FailureManager* of the form

Attributes
condition : *Condition*

Actions

$component\_failed \quad \{condition\}$
$component\_repaired \quad \{condition\}$
$failure\_detection \quad \varnothing$
$failure\_acknowledgement \quad \varnothing$
$repaired\_acknowledgement \quad \varnothing$

Axioms

$$BEG \Rightarrow condition = operating$$

$$component\_failed \Rightarrow \bigcirc\, condition = failed$$

$$component\_failed \Rightarrow (failure\_detection\ \mathcal{U}\ failure\_acknowledgement)$$

$$\neg\, component\_failed \wedge component\_repaired \Rightarrow \bigcirc\, condition = operating$$

$$component\_repaired \Rightarrow repaired\_acknowledgement$$

These in their turn could be subdivided into parts dealing with the communication protocol (axioms 3 and 5) and parts dealing with the recording of failure status (axioms 1, 2 and 4).

A theory *Transmission* has the form

Actions

$component\_state(val : \mathbb{Z}) \quad \varnothing$
$transmission\_failure \quad \varnothing$

Axioms

$$\neg\, \exists_1\, lev : \mathbb{Z} \cdot component\_state(lev) \Rightarrow transmission\_failure$$

where we regard the *PState* and *PCState* types as isomorphic to $\{0, 1\}$ as in [2].

Therefore the *Water_Measure_Monitor* theory can be re-expressed in terms of *Transmission*, via the morphism *m7* of Figure 2:

$component\_state(s) \mapsto level(s)$
$transmission\_failure \mapsto transmission\_failure$

and *FailureManager*, via the morphism *m1*

$condition \mapsto water\_measure\_condition$
$component\_failed \mapsto water\_measure\_failed$
$component\_repaired \mapsto level\_repaired$
$failure\_detection \mapsto level\_failure\_detection$
$failure\_acknowledgement \mapsto level\_failure\_acknowledgement$
$repaired\_acknowledgement \mapsto level\_repaired\_acknowledgement$

The attribute *water_quantity* and axioms to initialise and set this quantity are defined locally in *Water_Measure_Monitor*, and the axiom

$$\exists\, lev : \mathbb{Z} \cdot level(lev) \wedge (lev < 0 \vee lev > C) \Rightarrow water\_measure\_failed$$

determining when a component failure occurs is also defined in this theory.

Similar constructions work for the pump and pump controller (flow monitor) components. Figure 2 shows the structure of this part of the system. Separate copies of the *FailureManager* and *Transmission* theories are included in each of the component theories, but we identify all the different *transmission_failure* actions so that a transmission failure in any component generates the same system error event. *SData* is also included into each of the theories

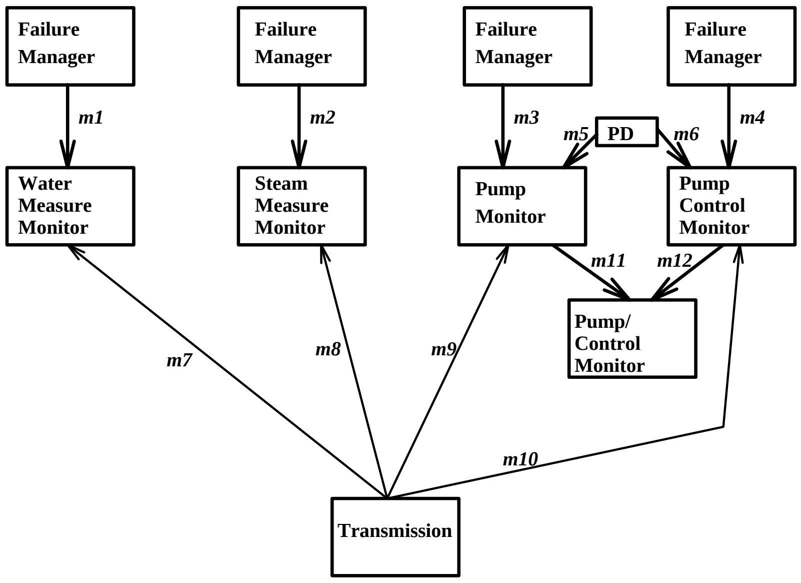

Figure 2: Construction of Component Theories

shown in this diagram.

The theory of the controller then extends the co-limit of the monitor theories with the following attributes and actions:

Types
$CState = \{initialisation, normal, degraded, rescue, emergency\_stop\}$

Constants
$hazard\_level(\mathbb{Z}) : bool$
$min\_level\_estimate(\mathbb{Z}, \mathbb{Z}) : \mathbb{Z}$
$max\_level\_estimate(\mathbb{Z}, \mathbb{Z}) : \mathbb{Z}$

Attributes
$cstate : CState$

Actions
$react \quad \{cstate\}$
$terminate \quad \{cstate\}$
$stop \quad \varnothing$
$steam\_boiler\_waiting \quad \varnothing$
$physical\_units\_ready \quad \{cstate\}$
$program\_ready \quad \varnothing$

Axioms Some example axioms of the controller are that a *terminate* event occurs if there have been three successive *stop* events, or if there has been a transmission error:

$$stop \wedge \bigcirc stop \wedge \bigcirc\bigcirc\, stop \;\Rightarrow\; \bigcirc\bigcirc terminate$$

$$transmission\_failure \;\Rightarrow\; terminate$$

Given the new mode and water level, take appropriate action:

$$\begin{aligned}
&\bigcirc cstate = normal \;\vee\; \bigcirc cstate = degraded \;\vee \\
&\qquad \bigcirc cstate = initialisation \;\Rightarrow \\
&\qquad\qquad (\bigcirc water\_quantity < N1 \;\Rightarrow\; increase\_flow) \;\wedge \\
&\qquad\qquad (\bigcirc water\_quantity > N2 \;\Rightarrow\; decrease\_flow)
\end{aligned}$$

$$\begin{aligned}
&\bigcirc cstate = rescue \;\Rightarrow \\
&(min\_level\_estimate(water\_quantity, steam\_quantity) < N1 \;\Rightarrow \\
&\qquad\qquad increase\_flow) \;\wedge \\
&(max\_level\_estimate(water\_quantity, steam\_quantity) > N2 \;\Rightarrow \\
&\qquad\qquad decrease\_flow)
\end{aligned}$$

The specification can be validated via animation [11]. The actions *increase_flow* and *decrease_flow* are general operations which will be interpreted as opening and closing certain pumps in the actual physical system: we have defined a layered architecture in which the implementation details of such abstract actions are hidden from the high-level controller.

2.2 Design and Implementation

B [1] specifications of steam boiler components were defined from the above theories. Standard techniques were used to code up temporal logic constraints. For example the temporal constraint

$$\alpha \;\Rightarrow\; (\gamma \;\mathcal{U}\; \beta)$$

can be formalised by a boolean flag *gamma_until_beta*, initialised to *FALSE*, and preconditions *gamma_until_beta* = *FALSE* on every controller action except γ and β and method definitions:

```
α  =   PRE gamma_until_beta  =  FALSE  ∧  ...
       THEN
           gamma_until_beta  :=  TRUE  ||
             ⋮
       END ;

β  =   ...
         gamma_until_beta  :=  FALSE
```

A C executable of 3098 lines of code was produced from the B implementations. A number of test scenarios were input to this executable, following the format defined in [2].

3 Comparison

We have shown that the Object Calculus can be used to provide a highly abstract and declarative specification of the behaviour of the steam boiler. We have formalised most aspects of the system. An executable controller has been produced and tested against the FZI simulator. Approximately 1 person week was used in writing the abstract specification, and 2 person weeks in developing the B design and implementation, including verification activities. B has been shown to be effective for industrial specification and to be comprehensible by 'average programmers'. We believe that the Object Calculus is also quite easy to relate to reactive system concepts and to notations such as statecharts, which it generalises. The papers [4, 5] describe how statecharts can be mapped to the Object Calculus.

Some specifications of [2] address issues which we do not consider, such as the calculation of optimal control points or the probabilistic behaviour of device failures. Our abstract specification adopts the approach of [7] in working at the *macro* step level in order to simplify the description. The B design is at the *micro* step level. The Object Calculus description is also related to the rule-based approaches used in [3] and [9], and suffers a similar problem of consistency obligations between rules. Our controller design model, like that of [6], adopts a purely reactive system approach, whereby events are assumed to happen one at a time and are reacted to in the order of their arrival.

An alternative structuring approach would be to use a time-based partitioning of modules, whereby the actions relevant to the initialisation phase of the steam boiler operation are placed in a theory separate from the actions and attributes of the running phase.

The mapping from Object Calculus to B is systematic (theories correspond to machines) but is not entirely automatic, since a design process is involved. Future work will classify different design choices for this translation, and relate B structuring formally to the Object Calculus. The Object Calculus has also been related to the UNITY approach for reactive system specification [12].

Other research directions include real-time extensions and durative actions, and deontic logic in order to handle failure states more naturally.

References

[1] J R Abrial, *The B Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[2] J R Abrial, E Borger, H Langmaack (Eds.), *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165, Springer-Verlag, 1997.

[3] C Beierle et al, *Refining Abstract Machine Specifications of the Steam Boiler Control to Well Documented Executable Code.* Pages 52–78 of [2].

[4] J.C. Bicarregui, K.C. Lano, T.S.E. Maibaum, *Objects, Associations and Subsystems: a hierarchical approach to encapsulation*, ECOOP 97, LNCS, 1997.

[5] Towards a Compositional Interpretation of Object Diagrams. J.C. Bicarregui, K.C. Lano and T.S.E. Maibaum. To appear: Proc. of IFIP TC2 Working Conference on Algorithmic Languages and Calculi, Strasbourg, February, 1997.

[6] R Büssow, M Weber, *A Steam-Boiler Control Specification with Statecharts and Z.* Pages 109–128 of [2].

[7] J Cuellar, I Wildgruber, *The Steam-Boiler Problem – A TLT Solution.* Pages 164–183 of [2].

[8] J. Fiadeiro and T. Maibaum *Describing, Structuring and Implementing Objects*, in de Bakker *et al.*, *Foundations of Object Oriented languages*, LNCS 489, Springer-Verlag, 1991.

[9] M-C Gaudel et al, *A Formal Specification of the Steam-Boiler Control Problem by Algebraic Specifications with Implicit State.* Pages 233–264 of [2].

[10] J Goguen and S Ginali, *A Categorical Approach to General Systems Theory*, in G. Klir (Ed.), *Applied General Systems Research*, Plenum 1978, pp 257–270.

[11] K Lano, *Specification of Steam Boiler Controller in RAL/VDM^{++} and B*, ROOS Project Document GR/K68783-15, 1997.

[12] K Lano, J Bicarregui, J Fiadeiro, A Lopes, *Specification of Required Non-determinism*, FME 97, to appear in LNCS, 1997.

4 Biography

Juan Bicarregui has worked on tool support for formal methods, in particular the Mural proof assistant, and on their theoretical foundations.

Jose Luiz Fiadeiro works in the area of formalisation of specification and program design paradigms using modal logics (temporal and dynamic) and of their underlying modularisation principles using category theory.

Kevin Lano is a Research Fellow at the Department of Computing at Imperial College. He has previously worked in industry in the areas of formal methods and software assessment. His current research is focussed on the use of formal techniques to provide semantics for object-oriented and reactive systems.

Tom Maibaum is Professor of Foundations of Software Engineering at the Department of Computing at Imperial College. He has worked on the development of compositional techniques for specification and design of software systems since 1974 and has published papers and books in the area.