Formal Methods for Reflective System Specification

J.B. de Meer
GMD-FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany;
e-mail: jdm@fokus.gmd.de

E. Abraham-Mumm
Christian-Albrechts-Universität zu Kiel, Institut für Informatik u. Praktische Mathematik
e-mail: eab@ifnformatik.uni-kiel.de

Abstract: A reflective system is able to recognize his internal state consisting of several discrete or continuous variables and, to adapt his internal state to some pre-defined target values. This can be achieved by feeding observations forward towards the sink or backwards toward the source of a service observed. At the service's sink and source discrete activities are able to be performed which remedy malfunctions or violations of a pre-defined system state. By this way of operation, reflective systems are able to provide guarantees on service qualities to end-users because their build-in remedy and adaptation capabilities.

In order to evaluate the complex behaviour of reflective systems, specifications must be formalized appropriately. However, appropriate formalization, means the representation of continuous and discrete behaviour by one unified model. In this paper several approaches, i.e. declarative, discrete, functional and hybrid methods, have been analysed with respect to their suitability to specify and to verify constraints comprising reflective system behaviour. The control procedures of the recently published multimedia standard MPEG4 have been chosen to demonstrate the capabilities of the various formal methods and to show how they could be integrated into a unified model for QoS.

Keywords: Continuous Behaviour, Reflective Systems, Quality of Service, Hybrid Models, Formal Methods.

1. Specification Requirements of Reflective Systems

A reflective system[2] is able to recognize his internal state[1] that consists of several discrete or continuous variables and that is able to adapt his internal state to some pre-defined target values. This can be achieved by feeding observations forward to the sink or backwards to the source of a service observed. At the service's sink and source discrete activities are performed which heal malfunctions or violations of a target values of the system state. The observations from which the remedy activities are derived must be done continuously or, at least so often that remedy activities will not be performed too late. A reflective system comprises both, discrete and continuous activities at the same time and can thus be designated as a hybrid system as well. Whereas by the notion of reflection the "self-healing" capability is stressed, the adjective "*hybrid"* stresses the combination of discrete and continuous behaviour existing cooperatively in one system.

From system theory, we know that the *"self-healing"* capability are technically spoken, open or closed loops carrying control information in the system. Open loops provide the principle of feed-forward controlling. A control component feeds some steering information forward to a controlled component that executes this information. Examples of open-looped systems which include this type of control are an admission control system at network edges or, the decoder buffer control system of MPEG4[8]. Closed loops enjoy the principle of feed-back controlling. Closed loops are different to open loops since they include an additional component that discriminates observations against precisely specified values. The result of discrimination which must continuously be executed, are discrete commands of remedies and are fed-back to the source of a service that effects the observed state variables. A simple example of feed-back looping is the flow control mechanism of the

1. Notice, a concrete state is described by the tuple of values of each of the variables comprising the state. Hybrid methodology uses the valuation function for assigning values to state variables.

transport protocol. At the sink site of the protocol loss of information can be observed and returned to the source which interprets the received control information as a re-transmission request of lost data.

From these considerations one can derive a set of requirements which must be achieved by formal but also by achitectural approaches which are to be applied to the specification of reflective systems. Thus, adequate formal respectively architectural methods for the specification of reflective system must be able

- to adequately describe time-delaying, i.e. of continuous, because of the streaming characteristics of reflective systems that deal with multimedia applications;
- to handle discrete, i.e. instantaneous transitions, because of adaptation mechanism that discriminate discrete targets or thresholds;
- to provide with analytical capability for the verification of dependable components arising from control loops that are the constituent elements of reflective systems;
- to provide with architectural openness, i.e. compositionality of system components, because of the necessity to observe permanently system-internal state variables;
- to allow the association of quantitative constraints to continuous or discrete events, because of the quantitative nature of quality-of-service parameters.

2. Formal Methods for the Specification of Reflective Systems

From the requirements listed above and for a formal approach which is best suited for the specification of reflective systems, one needs

- capabilities for the declaration of obligations of some service provided respectively required. The purpose of a declarative method is the marshalling of features, not their execution. Examples of declarative approaches are OMG’s Interface Definition Language (IDL)[11], TINA-C’s Object Definition Language (ODL)[12] or, the Unified Modelling Language (UML)[3].
- capabilities for the specification of discrete behaviour, i.e. message-triggered systems, which means messages trigger transitions of the system state. Examples of discrete approaches are all languages that are based on sequential transition semantics like Lotos[5,6], SDL[9], etc.
- capabilities for the specification of interconnecting functional components to a compound continuously operating system. An example of a componental specification method is the approach using “continuous functions” [4]. A similar componental decomposition schema based on mathematics is the Z-notation[13]. A Z-schema includes capabilities to describe state invariants, state transition conditions and relationships of components’ input and output.
- capabilities to integrate system parts that operate discrete with those that operate continuously. These methods are called “hybrid systems”[1] because they capture continuous and discrete behaviour in a unified model.

Since, there is no method that captures all of these requirements, several existing approach have been taken into account in order to model just one or two aspects of reflective systems. So, IDL has been found to be adequate for the specification of the QoS characteristic of streaming but also of signalling capabilities at interfaces. Sequential (transition) methods that do not provide with a built-in notion of continuity. Sequential methods can be improved by specifications modelling explicitly continuous behaviour. But much more appropriate are the hybrid specification methods which provide with a notion of real concurrency. Normally, continuous processes depend from other continuous processes in the same system, i.e. their evolution depends from their co-operation. Hence, Hybrid methods aim on the analysis of the dependability among continuously cooperating processes.

According to Alur et al, Hybrid System (HyS) are descrete programs which are embedded in an analog environment [1]. Thus, the control part of a HyS makes discrete decisions on the observation of continuous (analog) processes. In a HyS discrete decisions are modelled by descrete transitions and continuous processes or events are modelled by ‘continuous variables’. In linear HyS continuous variables obey a constant change rate. In non-linear HyS the change rate can be a higher order function. The MPEG4 example below, however, is restricted to linear HyS only. Hence, in formal terms, a HyS is modelled by ‘locations’ and discrete ‘transitions’ between locations. Locations comprise continuous activities, involving so-called ‘valuations’. An activity is time-invariant and continuous, which means its variables evolve continuouly over time. A transition is discrete and instantaneous and separates continuous state evolutions at locations. A transition is labelled with guarded sets of assignements to the variables and a location is labelled with an invariant condition. In figure 1 the HyS graph of two locations $l, l' \in Location$ connected by the transition $a \in Label$ is presented. The tran-

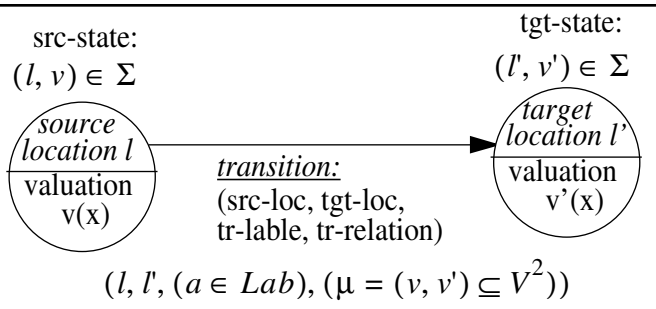

Figure 1: HyS Graph vertices and edges

sition a is enabled in state $(l, v) \in \Sigma$ if for some valution $v' \in Valuation$: $(v, v') \in \mu$ is a possible transition relation.

Whereas in HyS continuity is modelled by the valuation of continuous variables, continuous functions (*cf*)[4] model continuity by their operation on possibly infinite flows of data. Similar to HyS, in which a valuation of a location evolves as long as the invariant condition of the location holds, the cf evolves as long as the predicate of the cf holds. The predicate refers to values of variables that are collected to the so-called QoS state vector (QoS_SV). During evolvement the cf cuts the current discrete element from the input flow, generates some discrete output, may actualize some variables of the QoS state vector and will continue to evolve with the rest of the input flow. Because of functional behaviour cf can easily be connected into sequential and parallel configurations. Furthermore, for reactive systems the indispensably needed feedback loops can directly be modelled by cf-networks by feeding output links back to inputs of the same or of other cf. Figure 2 comprises both

Connectivity property:

Mutually recursive flow processing functions:

a) Sequentiality: $(h \bullet k)(z) = k(h(z))$

b) Parallelity: $(f||g)(x) = f(x_1,...,x_n),g(x_{n+1},...,x_{n+o})$

c) feedback: $(y_1,...,y_m) = h(z_1,...,z_{n-1},y_m)$

Continuity property:

A predicate on a QoS state vector q and input flow event i:

$Q(i, q) \Rightarrow Comp(q)(i \cdot flow) = o \bullet Comp(\hat{q})(flow)$, i.e.

if an input i and a certain QoS q fulfil condition Q then Comp processes i with QoS q, outputs o and possibly changes to new internal QoS $\hat{q}$ and continues with the rest of flow.

Figure 2: Properties of Continuous Functions

the specification of the connectivity and of the continuity property of cf.

Algebraic methods have no such properties. However systems behaving like continuous ones can as well be specified in terms of traces of discrete events. The behavioural model of the algebraic approach provides operators for the parallel and sequential composition of processes and operators for the alternative and sequential processing of traces of events. Those specifications providing with properties being unabandonly to reflective systems can be inherited by more complex ones. Thus, the specifications of basic properties provides a framework for reflective systems. Nevertheless a basic type of continuity is described by the transition relation of the bahavioural model. A transition $s_1 \xrightarrow{a} s_2$ from state s_1 to state s_2 is enabled if in state s_1 the action a can be observed otherwise the system will deadlock.

From the above discussed properties and features of the formal approaches we now want to derive the ingredencies from which reflective systems can be built. There are actually three elements:

1. the connectivity property among flows of data as provided by cf
2. the continuity property of activities as provided by HyS

3.the operational transition property as provided by algebraic approaches.

In figure 3 the integration of these three elements into a component is presented. It is not inended to integrate

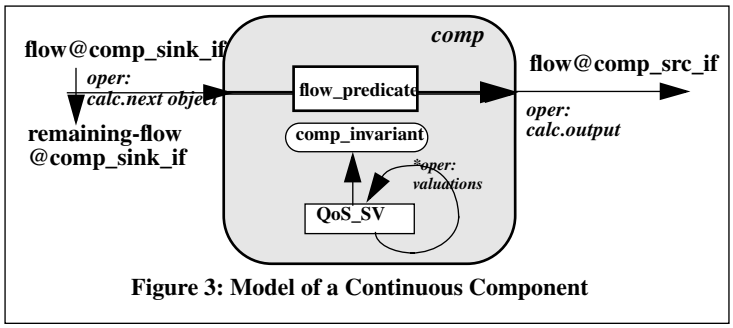

Figure 3: Model of a Continuous Component

the various methods into one unified modelleing language. It is our intention to keep the modelling languages separate but apply the notion of viewpointing[7]. Thus reflective systems are supposed to be analysed from the three viewpoints of connectivity, continuity and operationality. Accordingly, to the MPEG4 example the continuity and connectivity viewpoints have been applied and will be demonstrated.

The continuous component of figure3 has the two flow interfaces *‘comp_sink_if’* and *‘comp_src_if’*. On these flows the operations *‘calc.next_object’* and *‘calc_output’* are applied in order to separate the next discrete object to be processed by the component and, to generate an output from the calculation of the input object and the values of the QoS state vector. The QoS state vector contains continuous and discrete variables. Notice that the valuations of the continuous variables depend from the component’s invariant, but the continuous processing of the flow depend from the component’s flow predicate. They are different although they refer to the same set of variables, i.e. the QoS state vector.

4. The MPEG4 Example

The MPEG4 standard[8] exploits the open loop concept for the remote control of the decoder component. MPEG4 is thus an application that can be said to be reflective because it applies a forward loop mechanism. The quality of service of an MPEG4 system mainly depends from the synchronity of the clocks of the encoder and of the decoder. The encoding component is able to adjust the clock drift of the decoding component by issuing discrete synchronisation messages.

Additionally, the encoder component provides with the capability to control from remote the decoding and the composition time of multimedia objects by issuing time stamp messages. Since multimedia objects are parts of a more complex multimedia scene, the encoder, i.e. the source of this scene is able to control the scene presentation at the site of the sink.

The synchronisation and the scene control mechanisms are different mechanisms which do not depend from each other. But both are based on the open feed forward loop concept. Since clock synchronisation provides with continuous activities much more than the time-out activities of the scene control mechanism do, we want to restrict the examples of formal evaluation to MPEG4 clock synchronisation only.

The synchronisation mechanism is modelled by two processes which would diverge if they would not been synchronized from time to time. The two modelled processes represent the clocks of the decoder at the client’s site and the clock of the encoder at the server’s site. Both clocks obey a drift. Clock drifts can be linear or non-linear and can be positive or negative. By positive drifts clocks earn time, by negative drifts clocks loose time. When the actually measured time of both clocks differ from each other more than a certain amount, the decoder clock must be re-adjusted. In the MPEG4 standard this is achieved by sending on chance the time of the serving decoder to the encoding client. The client thus is able to adjust on receiving the serving decoder time, that is called *‘Object Time Base (OTB)’*, its own clock with the server’s clock. If the re-adjustment is done in one step loss of objects of or other information related to the scene to be presented and, thus degradation of quality of service must be taken into account. In order to avoid unnecessary degradation of QoS the discrepancy between the clocks must be resolved smoothly and without too large delays.

In order to specify a HyS continuous variables x and y for each clock at the encoder and decoder respectively is chosen. Notice that the object time base *OTB* is identical with y. There are two locations, one to model the drift of the both clocks and one location to model the compensation of the decoder clock. Both clocks are represented at both locations in order to model their distributed, i.e. independent behaviour. It is assumed that the clocks do not run with constant speed, but vary in between a minimum and maximum value $[\alpha, \beta],[\alpha, \gamma]$ respepectively. The speeds of the clocks in HyS are represented by the first derivation of the clock functions, i.e. $\dot{x}$ and $\dot{y}$ respectively. For simplicity reasons we further assume that the decoder clock follows always the encoder clock, i.e. $x \le y$. This is also the invariant condition for the continuous activity of location l_1. The second location l_2 models the smooth adaptation of x to y. Because of the distribution among a multimedia application network, there is no direct comparision between the clock's position. Thus the compensation of the deviation of the positions of y and x cannot exactly be achieved. The compensation is based on an extrapolation from the clock positions when *OTB* is received by the decoder clock x. This is the b-transition of the HyS. The positions are stored by the discrete variables x_0 and y_0. The difference of both values $y_0 - x_0$ is the adaptation path of clock x, and the adaptation time is δ. Thus the quotient $(y_0 - x_0)/\delta$ is the extrapolated correction to the speed of clock x in order to compensate the deviation only. Since clock x needs to adopt clock y's speed, the x's speed must be set to a value of y's speed. The worst case α from $[\alpha, \beta]$ is the slowest speed of y. Thus x's adopted speed sums to $\alpha + (y_0 - x_0)/\delta = \dot{x}$. The compensation activity continuously approaches the invariant condition of $x \le y_0 + \alpha \cdot \delta$. The c-transition from location l_2 to location l_1 occurs on the step before violating the invariant condition of the location, i.e. $x = y_0 + \alpha \cdot \delta$.

We defined in *PVS* the hybrid system model and an assertion-based verification method for proving safety properties of hybrid systems. With this approach one can verify invariant properties, i.e., properties that hold for each reachable state. The verification method is sound and complete.

For the MPEG4 example we have verified that, if the time delays between two consecutive clock synchronizations are always less or equal a positive real number *zmax*, and that the diference between the values of the encoder and the decoder clock is always positive but smaller or equal to *zmax* which is equal to $\beta - \gamma$. With respect to the latter verification goal the differential of the encoder clock is always between α and β and the differential of the decoder clock is always between γ and α with $\gamma < \alpha < \beta$.

Formally, for each state $\sigma = (l, v)$ of the hybrid system H:

$$\frac{\text{reachable}(H, \sigma)}{0 \le v(y) - v(x) \le \text{zmax} \cdot (\beta - \gamma)}$$

Notice, in figure 4 there is an alternative transition which is enabled if the smooth compensation cannot be done. Compensation or better adaptation is performed in one step, just by assigning the received position of clock $y = OTB$ directly to x. Because of the immediate change of the x-clock's position guarantees of loss of information, including video objects, about the MPEG4 scene cannot be given.

The specification of the continuous function is sketched in figure 5. In order to select the synchronisation, i.e.

$$(\text{sync?} \wedge \text{ldc}) \Longrightarrow$$

$$\text{Sync.Decoder}\left(\frac{dx}{dt}, inv\right)(\text{OTB.AUStream}) = \varnothing$$

$$\bullet\ \text{Sync.Decoder}((\alpha + \text{OTB-x}),(x \le \text{OTB}+\alpha\delta))(\text{AUStream})$$

Figure 5: cf of the MPEG4 Clock Compensation

compensation function from the decoder component, the flow predicate is extended with the boolean value *sync?* which becomes true on receiving the *OTB* value from the remote encoder. The *Sync.Decoder* function

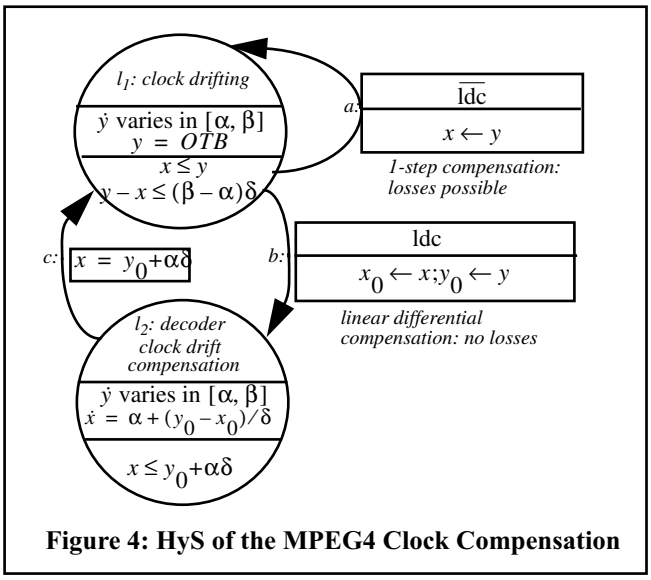

Figure 4: HyS of the MPEG4 Clock Compensation

comprises in its internal QoS state vector the continuous variable $\frac{dx}{dt}$ which models the clock x of the decoder. The invariant of x is set to the extrapolated position to be adopted by the clock. The continuous variable of the QoS state vector of the *Sync.Decoder* function behaves the same as for the HyS. It must be noted that OTB is depacked from the AU stream which is the input to the continuous component. Since clock synchronisation is an internal activity no value is output. On completion of compensation the continuous component automatically continues with the rest of AU stream.

5. Conclusions

It has been shown, that various formal methods are able to model properties of reflective systems even they are not designed explicitly to handle QoS concepts. E.g. by sequential approaches, continuity can be invented by the explicit specification of the “continuity property”. Since it is not a built-in formal concept it is up to the specifier’s taste to decide what is the correct model of continuity. But continuous functions are specialized to the input-output behaviour of components. They are thus suited for the verification of the connectivity between components configured into a system. Hybrid methods, however, integrate both, methods for the discrete and methods for the continuous behaviour of system activities. By hybrid methods we can approve that co-operating processes globally behave correctly, i.e. they do not diverge, as the MPEG4 example of the clock synchronisation has shown.

Similar to the ODP approach[7] of adopting viewpoints for the design of a distributed system, it makes sense to adopt a viewpoint approach as well for the modelling of reflective system properties. Discrete state transitions can at its best be modelled by approaches based on transition systems, interfacing and connectivity are more suited to functional approaches, continuous activities embedded in discrete systems need hybrid specification methods. It cannot be said that there exists the formal modelling method which is suited for the specification and evaluation of reflective respectively QoS control mechanisms. It might not be wishable to have one unified modelling approach, since the diverging modelling concepts to be integrated would possibly lead to a less powerful approach. For the modelling of the complex behaviour of reflective systems it is proposed to adopt various notions likely comparable to the ODP viewpoint approach. Formalisation viewpoints for reflective system specification could be: Continuous behaviour of components (1), dependability of architectural control structures (2), quantitative QoS specification (3) and Integration of continuous components into discrete control structures, i.e. hybrid system behaviour (4).

References

[1] Alur et al: The Algorithmic Analysis of Hybrid Systems, Theoretical Computer Science Vol. No. 138, pp. 3-34, 1995, Springer Verlag.

[2] Blair et al. The Impact of Aspect-oriented Programming of Formal Methods (Postion Paper), Proceedings of the ECOOP Workshop, Brussels 1998, Springer Verlag.

[3] Boch, Jacobsen, Rumbough: Unified Modelling Language UML, Rational Software Corporation.
[4] Broy et al: The Design of Distributed Systems. An Introduction to FOCUS, TUM-19202-2, SFB Nr. 342/2-2/92, 1993
[5] ISO 8807 International Standard IP OSI LOTOS - A formal description technique based on the temporal ordering of observable behaviour 1989
[6] ISO/IEC JTC1/SC21/WG7, Enhancement to LOTOS, Committee Draft 1998.
[7] ISO 10746-1...-4 International Standard IP OSI DM&ODP - Basic Reference Model for Open Distributed Processing Parts 1 to 4, 1995
[8] ISO/IEC JTC1/SC29 WG11, Coding of Moving Pictures and Audio, MPEG4 Overview, 1999.
[9] ITU-T Z.100 Recommendation: CCITT Specification and Description Language SDL, http://www-vst.fokus.gmd.de/platin/ITU/itu-t/rec/z/z100_23858.html
[10] B.C.Kuo: Automatic Control Systems, 7th edition, Prentice Hall, ISBN 0-13-304759-8
[11] OMG IDL Syntax and Semantics: The Common Object Request Broker: Architecture and Specification CORBA V2.3, Revised Edition 1995, http://www.omg.org/library/issuerpt.htm
[12] TINA-C: TINA Object Defintion Language (ODL) Manual Version 2.3., Archive Label: TR_NM.002_2.2_96, 1996
[13] Spivey: The Z Notation. A Reference Manual, C.A.R. Hoare Series Editor, Prentice Hall 1989