Sinnott, R.O. (1999) *Specifying multimedia configurations in Z.* In: Verma, B. (ed.) Proceedings: Third International Conference on Computational Intelligence and Multimedia Applications: ICCIMA '99, September 23-26, 1999, New Delhi, India. IEEE Computer Society, Los Alamitos, USA, pp. 331-335. ISBN 9780769503004

http://eprints.gla.ac.uk/7239/

Deposited on: 18 September 2009

Enlighten – Research publications by members of the University of Glasgow
http://eprints.gla.ac.uk

Specifying Multimedia Configurations in Z

Dr Richard O. Sinnott
GMD-Fokus,
Kaiserin-Augusta-Allee 31,
Berlin, Germany
sinnott@fokus.gmd.de

Abstract

In this paper we illustrate how the formal specification language Z can be used to reason about the temporal and throughput constraints associated with multimedia flows of information. In particular we show how it is possible to specify issues related to maximum delays, throughputs and jitter of information flows and how control of these flows can be achieved. What makes our work particularly interesting is that we deal with temporal aspects of systems without the use of a temporal logic. Rather, we highlight the versatility of the Z language in modelling systems with real time constraints.

1. Introduction

The formal modelling of systems that deal with flows of information place a great many constraints on the specification languages chosen for that purpose. Many of these constraints arise from the timing aspects associated with the information flow. Most approaches to modelling multimedia systems, e.g. [1] typically use a temporal logic [2] to reason about the timing issues of the flows. Whilst temporal logics are well suited to reasoning about timing aspects of systems, they are limited in their modelling of the behavioural aspects of systems. In this paper we show the power of the Z language in modelling both of these aspects. To make our approach less abstract we specify a producer and consumer flow. For simplicity, we avoid the issues involved in determining whether information sent from producers is that required by consumers. Instead we focus on the timing aspects of sending and receiving of informations and how control over these timing aspects can be achieved.

2. Background Concepts for Modelling Information Flows

We shall consider a generic idea of information flow represented by a sequence of *frames* (*seq Frame*) where a frame may be regarded as a particular item in the flow of information. Each frame can be considered as a unit consisting of data (this may be compressed) which we represent by *Data* and a time stamp used for modelling the time at which this particular frame was sent or received. It is often the case in multimedia flows that particular frames are required for synchronisation, e.g. synchronisation of audio and video. Therefore we associate a particular *Label* with each frame which can be used for selecting frames from the flow as required. We thus model a frame as: $Frame \triangleq [\ d: Data; ts: \mathbf{R}; l: Label\ ]$

Information flows have inherent characteristics given by the nature of their flows, e.g. they can be isochronous where frames are sent/received in equal time segments:

$$\forall f_1,f_2: Frame \mid < f_1,f_2 > \in frames \bullet f_2.ts - f_1.ts = constant$$

Here all frames in the flow are separated by equal time intervals (the rate) which is a constant. Alternatively, flows can be bursty which implies that time intervals between successive frames are not necessarily equal:

$$\exists f_1, f_2, f_3: Frame \mid < f_1,f_2> \in frames \wedge < f_2,f_3 > \in frames \bullet f_2.ts - f_1.ts \neq f_3.ts - f_2.ts$$

Producer and consumers information flows may be either flowing or stopped which we represent by the free type: $Status ::= Stopped \mid FlowIn \mid FlowOut$

Authorized licensed use limited to: UNIVERSITY OF GLASGOW. Downloaded on September 18, 2009 at 06:23 from IEEE Xplore. Restrictions apply.

3. Formalising the State of Producers and Consumers

Interfaces that produce or consume flows are required to access variables modelling time. It is possible to represent time in several ways in Z. Time might be represented by a variable global throughout the specification and is accessed and modified each time events (modelled as schemas) that consume time occur. Alternatively, it is possible to model time locally to schemas. We consider here a local model of time where the producers and consumers of information flows have local variables modelling the current local time. The local time is incremented when operations occur, i.e. information items are sent or received. The amount that the time is incremented by depends on the rate of production or consumption of flows. As we shall see, having a local time model also allows rates of production or consumption of frames to be modified. We may model a producer state by the following Z fragment:

PState

$ps: seq\ Frame$
$ptnow, prate: R$
$pstat: Status$
$pCons: ProducerCons$

$\forall f_1,f_2: Frame \mid < f_1,f_2 > \text{in } ps \bullet f_2.ts > f_1.ts \land pstat \in \{ Stopped, FlowOut \} \land ...$

Here we state that the producer state is given by a sequence of frames (*ps*), the current time (*ptnow*), the rate of production of frames (*prate*), i.e. the time it takes for a producer to produce a frame, the current status (either stopped or flowing out from the interface), and a set of constraints associated with the producer information flow. We also state that all frames in the sequence of frames to be sent should have increasing time stamps. Here the dots are dependent upon the constraints associated with the multimedia flow and how they influence the flow itself. To allow these predicates to be given, it is necessary to be more prescriptive regarding the constraints of interest. We consider the maximum delay, maximum throughput and (bounded) jitter associated with the information flow which may be represented as: $ProducerCons \triangleq [\ pmt, pmd: \mathbf{R}; pjb: \mathbf{R} \times \mathbf{R}\ ]$

We note here that it is normally the case that Z specifications are written in a definition before use manner. Semantically the ordering is unimportant provided the Z text can be ordered in such a way that all Z fragments are defined before they are used. In the schema *ProducerCons*, *pmt* represents the producer's maximum throughput, *pmd* the maximum delay and *pjb* the upper and lower limits on jitter. The maximum delay for isochronous flows is simple the rate. The maximum delay for bursty flows is given by:

$maxDelay: \text{seq } Frames \rightarrowtail\!\!\!\!\!\!+\ \mathbf{R}$

$\forall sf: \text{seq } Frames \bullet maxDelay(sf) = max\{ f_1,f_2: Frame \mid < f_1,f_2 > \in sf \bullet f_2.ts - f_1.ts \}$

The maximum delay of a multimedia flow may be regarded as the upper limit on the time window at which a frame is expected, e.g. a consumer may be able to wait for a certain time for the next frame to arrive. For bursty flows, the maximum delay is the maximum time difference between two successive frames in the sequence.

The maximum throughput may be regarded as the number of frames a producer of a flow wishes to produce, or that a consumer wishes to consume. Isochronous flows should have a consistent throughput (given by the reciprocal of the rate), whereas bursty flows may have situations where more frames are output (or input) than at other times. In bursty flows it is especially useful to put an upper limit on the maximum throughput of data. The maximum throughput of a bursty flow is the maximum flow subsequence with a timestamp difference of less than or equal to one second from its first and last elements (assuming the throughput was measured over one second). Thus throughput may be represented as:

$maxThru: \text{seq } Frames \rightarrowtail\!\!\!\!\!\!+\ \mathbf{R}$

$\forall sf: \text{seq } Frames \bullet maxThru(sf) =$
$\quad max\{ s: \text{seq } Frames; f_1,f_2: Frame \mid s \text{ in } sf \land f_1 = head\ sf \land f_2 = last\ sf \land 1 \geq f_2.ts - f_1.ts \bullet \# s \}$

We shall see that the values for maximum delays and throughputs can be assigned non-deterministically. With this, a more complete model of the producer's state is given by:

Authorized licensed use limited to: UNIVERSITY OF GLASGOW. Downloaded on September 18, 2009 at 06:23 from IEEE Xplore. Restrictions apply.

$$\forall f_1,f_2: Frame \mid < f_1,f_2> \text{ in } ps \bullet f_2.ts - f_1.ts < pCons.pmd \land 1 \text{ div } prate < pCons.pmt$$

Here we state that all consecutive frames in the sequence should have time differences less than the maximum allowed delay. We also state that the rate (*prate*) of production of frames must be less than some upper throughput limit. For simplicity we assume that throughput is measured over a single time unit. We note here that whilst these constraints can be asserted directly and used to influence the production of frames, it is not possible to write explicit predicates on the jitter. The jitter depends on the actual sending and receiving of frames. To model jitter we assume a similar model of state for a consumer as that for a producer, i.e.with a sequence of frames to be expected (*cs*), a current time (*ctnow*), a current status (*cstat*) which is either consuming or stopped, a rate of consumption (*crate*) and a set of constraints which we assume are on delays (*cmd*), throughputs (*cmt*) and jitter (*cjb*).

We may represent the constraints that might be associated with a producer or consumer generally by a parameterised free type definition. This can be represented as:

$$Constraints ::= prodBindCons \langle\langle ProducerCons \rangle\rangle \mid consBindCons\langle\langle ConsumerCons \rangle\rangle$$

We represent the satisfaction of constraints associated with producers and consumers as:

$$Satisfies: Constraints \leftrightarrow Constraints$$

For brevity we omit the associated predicates. Satisfaction of throughput and delay constraints require that the maximum delay of a producer is less than the maximum delay a consumer can tolerate and that the consumer can accept a higher throughput than the producer can produce. Satisfaction of jitter constraints requires the producer allowed jitter is within the consumer jitter range.

Producer and consumer initialisation may be represented as:

PÇInit

$$Pstate'$$
$$CState'$$

$$\exists pc: ProducerCons; cc: ConsumerCons; ss: seq Frames \mid ss \neq <> \bullet$$
$$(consBindCons(cc),prodBindCons(pc)) \in Satisfies \land (prodBindCons(pc),prodBindCons(pCons)) \in Satisfies \land (consBindCons(cc),consBindCons(cCons)) \in Satisfies \land$$
$$pCons' = pc \land cCons' = cc \land ps' = ss \land cs' = <> \land ptnow' = ctnow' \land$$
$$pstat' = cstat' = Stopped \land prate' = 1 \text{ div } pc.pmt \land crate' = 1 \text{ div } cc.cmt$$

This states that there exist producer and consumer constraints such that: the values the producer and consumer constraints are set at, are satisfactory to the consumer and producer respectively; the values the producer's constraints are set at are satisfactory to the producer and the values the consumer constraints are set at, are satisfactory to the consumer.

These constraints (if they exist) are assigned to the producer and consumer constraints. The producer is also instantiated with a non-empty sequence of frames to send and the consumer initially has received no frames. The producer and consumer local times are synchronised and both have the current status *Stopped*. Finally we state that the producer and consumer rates are set to values of the reciprocal of the maximum throughputs.

4 Formalising Interfaces for Controlling Information Flows

As well as having interfaces for producing and consuming information flows, producers and consumers of information flows are likely to have operations for manipulating these information flows. We term these *control* interfaces. It is likely that these interfaces will follow an RPC-like mode of interaction, where messages are sent and responses are given to these messages. For simplicity here, we assume a basic set of messages that are passed between producers and consumers. These may be represented by the free type definition:

$$Message ::= ExceedsMax \mid AlreadyStopped \mid AlreadyStarted \mid NotStarted \mid ArrivedEarly \mid ArrivedLate \mid Ok$$

The most elementary operations are starting and stopping flows. A flow being started can be represented by: $ProdStartFlow = ProdStartFlowOk \setminus ProdStartNotOk$ where:

ProdStartFlowOk

$$\Delta PState$$
$$res!: Message$$

$$pstat = Stopped \land ps' = ps \land ptnow' = ptnow \land$$
$$pstat' = ptnow \land prate' = prate \land pCons' = pCons \land res! = Ok$$

ProdStartFlowNotOk

$$\Xi PState$$
$$res!: Message$$

$$pstat = FlowOut \land$$
$$res! = Already Started$$

Authorized licensed use limited to: UNIVERSITY OF GLASGOW. Downloaded on September 18, 2009 at 06:23 from IEEE Xplore. Restrictions apply.

The preconditions for *ProdStartFlowOk* require that the flow is currently stopped. The postconditions state that the status of the producer is changed to *FlowOut*. All other variables associated with the producer state are unchanged. It might well be the case that the occurrence of this operation schema might take a non-negligible time. In this case, the local variable modelling time should be incremented by some amount. We assume that the operation takes zero time and that time is only incremented on the sending of frames. ProdStartFlowNOk shows the failure case when the producer is already producing the flow.

For brevity we do not provide the operations for stopping a flow which are similar to the starting of a flow with the obvious changes to the predicates. For similar reasons we also do not provide the operations for starting or stopping a consumer flow which are similar to the operations given for starting and stopping producer flows.

The rate of production of information flows may be increased or decreased depending on consumer demands. A successful operation to increase the producer flow rate is:

ProdFasterOk

$\Delta PState$
nr?: **R**
res!: Message

$pstat = FlowOut = pstat' \land nr? < prate \land pCons.pmt >= 1 \text{ div } nr? \land$
$ps' = ps \land ptnow' = ptnow \land prate' = nr? \land pCons' = pCons \land res! = Ok$

This operation requires that the producer is currently outputting information and the new rate, i.e. the time to produce frames, is less than the existing one but within the throughput limits. The postconditions state that the status, current time, sequence of frames and constraints are unchanged. An output message is sent stating the operation was successful.

We consider two cases in which this operation might fail: if the requested rate would result in the maximum throughput constraints being exceeded or if a request to produce faster occurs when the producer has not yet been started. These may be represented as:

ProdFasterExceedsMax

$\Xi PState$
nr?: **R**
res!: Message

$pCons.pmt \geq 1 \text{ div } nr? \land res! = ExceedsMax$

ProdFasterNotStarted

$\Xi PState$
nr?: **R**
res!: Message

$pstat = Stopped \land res! = NotStarted$

The total operation showing how the producer flow can have its rate increased is:

$$ProdFaster \triangleq ProdFasterOk \lor ProdFasterExceedsMax \lor ProdFasterNotStarted$$

We omit the operations showing how the information flow rates can be reduced. These are similar to those given for the increase in production but with the obvious predicate changes, e.g. checking that the rate does not become zero or negative. For reasons of brevity we also do not provide the operations for increasing or decreasing the consumer consumption rate. These are similar to the operations given for changing the producer flow rate but with the predicates modified accordingly.

5. Formalising the Production and Consumption of Flows

The production of frames requires that the frames are time stamped with the local time when they are to be sent. The sending of a frame from a producer may be represented as:

PSendFrame

$\Delta PState$
f!: Frame

$ps' \neq <> \land pstat = FlowOut = pstat' \land ps' = tail\ ps \land prate' = prate \land pCons' = pCons \land$
$(\exists t: \mathbf{R} \mid (ptnow + prate + second(pCons.pjb) \geq t \geq (ptnow + prate - first(pCons.pjb)) \bullet prate' = t \land$
$(\text{let } nf == (\mu\ Frame \mid d = (head\ ps).d \land ts = t \land l = (head\ ps).l) \bullet f! = nf))$

Several things should be pointed out here. Firstly, we require that the producer has frames to send. The producer should currently have the producing frames status. Sending a frame removes that frame from the sequence of frames to be sent. The current rate and constraints associated with the flow are unchanged. The actual time at which the frame was sent is given by the current time plus the current rate and a value within the jitter bounds.

Authorized licensed use limited to: UNIVERSITY OF GLASGOW. Downloaded on September 18, 2009 at 06:23 from IEEE Xplore. Restrictions apply.

The actual frame sent is the head frame in the sequence of frames to be sent. This is time stamped with the value for the time calculated previously. We note that the use of the definite description requires that a proof obligation is fulfilled to ensure the frame sent is unique. This is satisfied through modelling all frames in the sequence with increasing (i.e. non-equal) time stamps. We also require that the sequence of frames is non-empty.

A consumer may receive a frame successfully provided the constraints are satisfied.

CGetFrameOk

$\Delta CState$
$f?: Frame$

$cstat' = FlowIn = cstat \land crate' = crate \land cCons' = cCons \land ctnow' = ctnow + crate \land$
$ctnow + second(cCons.cjb) \geq f?.ts \geq ctnow - first(cCons.cjb) \land cs' = \langle f? \rangle^\frown cs$

The preconditions for this operation are that the consumer can currently receive frames, i.e. its status is *FlowIn*. The operation itself does not affect the rate of consumption, the current status or consumption constraints. The frame is only accepted provided the time stamp is within the allowed jitter range associated with the consumer constraints. If the time constraints are satisfied then the frame is accepted and added to the sequence of frames received and the current time is incremented by whatever value the rate is currently set at.

We consider two situations in which a frame may be rejected by a consumer. When the frame arrives outside the time constraints, e.g. too early or too late, or when the consumer is not yet ready to start consuming frames. These can be represented as:

CGetFrameTimeNOk

$\Xi CState$
$f?: Frame$
$res!: Message$

$(f?.ts \geq (ctnow + second(cCons.cjb)) \land res! = ArrivedLate) \lor$
$((ctnow - first(cCons.cjb) \geq f?.ts) \land res! = ArrivedEarly)$

CGetFrameNotStarted

$\Xi CState$
$f?: Frame$
$res!: Message$

$cstat = Stopped \land res! = NotStarted$

The actual sending of a frame from a producer to a consumer may be represented as:

$ProdSendToCon \triangleq PSendFrame \land (CGetFrameOk[f!/f?] \lor$
$CGetFrameTimeNOk[f!/f?] \lor CGetFrameNotStarted[f!/f?])$

6. Conclusions

This paper has shown the flexibility of the Z language for modelling multimedia flows of information and how these flows can be controlled. We have focused here on showing how delays, throughputs and jitter can be specified and controlled directly. Extensions to the work are also possible, e.g. modelling the information (*Data*) in more detail and abstract representations of the required compression/decompression techniques required to access the data. The work could also be extended to model other forms of jitter, e.g. bounded jitter. Similarly extensions such as the synchronisation of clocks that have drifted apart, modelling unreliable media and latency related issues can be modelled directly [3].

7. References

[1] G.Blair, L. Blair, H. Bowman and A. Chetwynd. Formal Support for the Specification and Construction of Distributed Multimedia Systems (the TEMPO project), TR-MPG-93-23, University of Lancaster, 1993.
[2] Z. Manna, A. Pneuli. The Temporal Logic of Reactive and Concurrent Systems, Springer-Verlag, 1992.
[3] R.O. Sinnott. An Architecture Based Approach to Specifying Distributed Systems in LOTOS and Z. PhD Thesis, University of Stirling 1997.

Authorized licensed use limited to: UNIVERSITY OF GLASGOW. Downloaded on September 18, 2009 at 06:23 from IEEE Xplore. Restrictions apply.