A. John M. Donaldson and Kenneth J. Turner. Formal specification of QoS properties. In Jan de Meer, Gregor von Bochmann, and Andreas Vogel, editors, Proc. Workshop on Distributed Multimedia Applications and QoS Verification, pages 1-14. CRIM, Montreal, Canada, June 1994.

Formal Specification of QoS Properties

A. John M. Donaldson and Kenneth J. Turner [1]

Department of Computing Science and Mathematics, University of Stirling.

Abstract. We describe the specification of communication services, with special emphasis being placed on the use of the Temporal Logic of Actions (TLA) to describe the behaviours involved. We show how, starting from Message Sequence Charts, this temporal logic may be used to describe The Joint Viewing and Tele Operating Service (JVTOS) and its associated functions; and so lead on to the specification of QoS parameters. We discuss the approach that was taken to determine the exact nature of the Quality of Service parameters, and how the method may be used to extend the specification, and probe further aspects of the services and protocols involved.

Keywords: Message Sequence Charts, User Perception, Temporal Logic, Behaviours, Actions, Liveness.

1. Introduction

The RACE project TOPIC (Toolset for Protocol and Advanced Service Verification in IBC Environments) comprises work in: verification methodologies for Integrated Broadband Communications (IBC), the use of verification tools for Quality of Service (QoS) measurement and the provision of a demonstrator for IBC verfication technology. As part of our participation in the TOPIC project we have been working on the application of a particular methodology to specify a QoS aspects of the complex multi-media tele-conferencing Joint Viewing and Tele Operating Service (JVTOS).

As will be seen, our approach to Specification (and Verification) has been to apply Lamport's *Temporal Logic of Actions* [Lam91] but it must be stressed that the content of this paper is *not about TLA in itself*, but explains steps that we have taken towards *specifying QoS parameters* in such a way that our results are both complementary to work using other more conventional methods (such as LOTOS [E.B87] and SDL [RR89]) and yet able to solve problems which may not suit those particular methods.

[1] Ths work has been conducted as part of a Research Fellowship at the University of Stirling, supported by the EC RACE R2088 Project TOPIC. The results presented in this document do not necessarily represent the opinion of the TOPIC consortium.
Correspondence and offprint requests to: John Donaldson, Department of Computing Science, University of Stirling, Stirling, FK9 4LA, Scotland. Email ajd@compsci.stirling.ac.uk (Paper presented at: The Montreal Workshop on Multimedia Applications and Quality of Service Verification, Montreal, Canada, June 1994.)

1.1. JVTOS

JVTOS is a service designed to provide audio-visual and desk-top communication in a conferencing session between two or more service users. In this service, the major participants are:

- The *Session Manager* (SM) who has the principal supervisory role and also acts as a user.
- The *Resource Allocator* (RA) is a system module for resource management.
- *User(s)* (U) who participate in JVTOS sessions.

Current development work in JVTOS [B.M92] is based on the eXtended XTP protocol (XTPX) which has additional provision for QoS information. The phases are identified are: *Call Set-up, Information Transfer* and *Call Clearing*. The session states identified are: *Idle Session, Active Session, Request to Join Active Session, Request to Leave Active Session* and *Closed Session*.

1.2. Background work

An investigation into the JVTOS service was conducted [Don93] and then a series of of Message Sequence Charts (MSCs) based on the functions such as *Start-A-Session, User-Requests-to-Join* and *Data-Transfer* identified in [GE93] were prepared.
Having determined the different functions we then considered which broad categories of QoS parameter required to be specified (e.g. delay, error rate, throughput, failure probability). We looked at the likely sequences of events taking place within the JVTOS service and prepared descriptions of different types of signal (*Poll, Reply, Control*) associated with each phase of the process (*Call, Data-Transfer, Clear*).
We initially attempted to consider Quality of Service in direct association with MSCs by some form of annotation that could be used. We even developed a BNF description of the functions, finding that this approach was accurate to a certain extent but limited in its application against our first expectations. Our interest from the outset however had been to investigate Quality of Service and to look for ways of specifying requirements and paying particular attention to :

- providing a specification that is relatively easy to comprehend.
- being able to handle aspects of the service that are not easily achieved through conventional Formal Description Techniques.

What we have been seeking is *not* an alternative to existing methods but ways that we can assist in preparing the way for these techniques and addressing problems that they cannot readily solve.

1.3. The type of problem that we are addressing

We consider a typical JVTOS example where we have two terminals communicating with each other. The user's measure of quality will be determined by (for example) the response time to receive the first screen image and the clarity with which the image is displayed. Figure 1 shows the typical picture of what is going on where we see that the higher level user perception is supported by a whole series of functions taking place in order to send an image from one terminal to another. Here we show a signal being compressed using CODEC (coder/decoder) and JPEG (Joint Picture Expert Group) which compresses entire images. To study and specify such aspects of the behaviour requires a broad approach ranging from detailed analysis of QoS/NP functioning at the transport layer and below right up to higher level considerations concerned with user perception, negotiation and renegotiation.

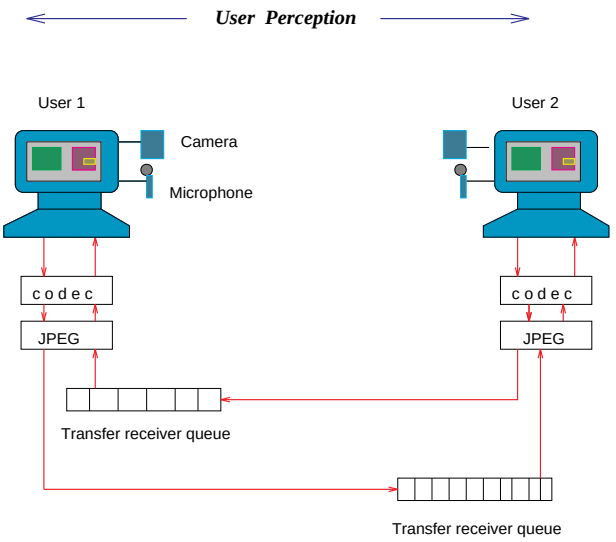

Fig. 1. Bidirectional Video Transfer

Thus we are interested is looking at ways of relating lower layer operations to performance at the application layer, and hence to (mathematically) determine user perception. To look at such a problem we have to consider the underlying functions that are taking place and then reason about the different actions that make up the behaviour that we are studying.

1.4. The choice of Temporal Logic

At this point we considered Lamport's *Temporal Logic of Actions* (TLA) [Lam91] as a way of specifying QoS aspects of the JVTOS service. This approach was appealing for a number of reasons.

- It is able to provide broad descriptions of behaviours by allowing us to reason about the actions and state transitions that take place in a concise and straightforward manner that is easy to comprehend.
- Any specification such as this invariably involves long and complicated formulas. However by using a combination of ordinary mathematics and familiar conventional programming notation, TLA provides a clean and straightforward presentation; with particular aspects being specified in individual modules.
- By using logic, relatively simple algorithms can be easily and compactly specified.
- Having been developed with concurrency issues in mind, TLA has been shown [ML92] to be well suited to handling *real time* issues.
- TLA has also been designed to cope with safety and liveness aspects.

2. JVTOS and QoS Specification

Following our review the basic functions associated with JVTOS and starting with MSCs, we have been able to describe the corresponding core functions by creation of a series of modules each of which specifies some particular aspect of the service or its related QoS requirements. What we now have is a comprehensive series of interrelated modules as shown in figure 2.

As can be seen this scheme covers the wide range of aspects that we have specified to date. We will look at some specific examples after first noting some of the salient features of TLA.

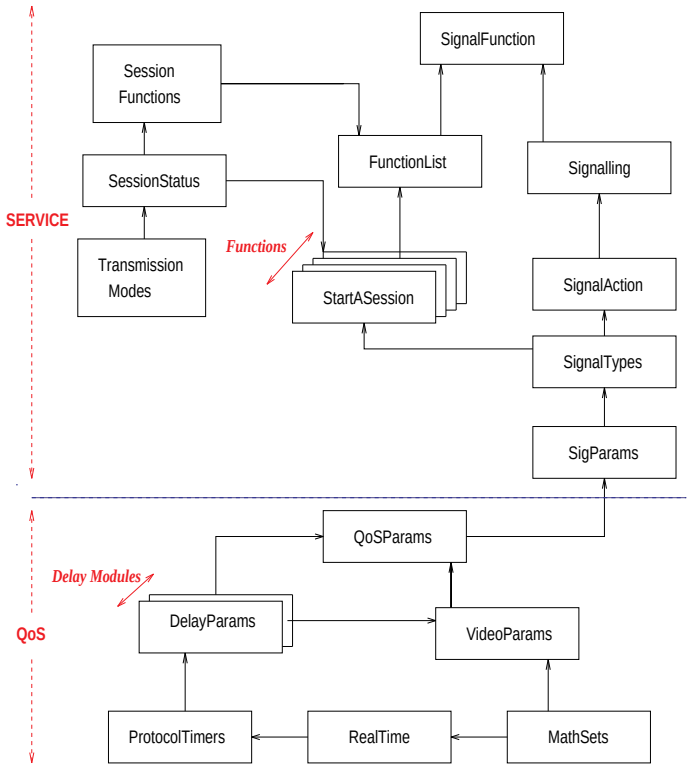

Fig. 2. The JVTOS Import Scheme

2.1. A brief overview of TLA $^+$

TLA is a logic for reasoning about systems capable of exhibiting concurrent activity and is syntactically represented by the *meta*language TLA $^+$. In TLA, we consider *states* and *state transitions* achieved through *actions* in order to describe the *behaviours* in which we are interested. *Predicates* are boolean-valued state functions used to describe actions.

- **Actions** represent the relationship between old and new states and as such map pairs of states to booleans.
  - $[\mathcal{A}]_f$ means : $\mathcal{A} \vee (f' = f)$ for any action $\mathcal{A}$ and state function f.
  - $\langle \mathcal{A} \rangle_f$ means : $\mathcal{A} \wedge (f' = f)$ for any action $\mathcal{A}$ and state function f.
  - **Unchanged** means : $(f' = f)$ for state function f.
- **Safety and Liveness** Our specification may have some action ($Next$) which deals with state transitions (v), and the temporal formula $\Box[Next]_{\langle v \rangle}$ asserts that henceforth ($\Box$) actions $Next$ are permissible. When that formula is conjoined with the predicate $Init$ (which ensures that the system starts in the correct state), then the assertion $Safe$ can be made :

$$Safe \triangleq \begin{array}{l} \wedge\, Init \\ \wedge\, \Box[Next]_v \end{array}$$

This temporal formula means that a behaviour starts in a correct initial state, and that every successive state is correct. This statement is perfectly legal, but it is also satisfied by a system that is allowed to stutter (effectively do nothing) forever. We therefore have to assert **liveness** to ensure that we are specifying a system that progresses.

The formula $Safe$ describes what may *not* happen - the behaviour may *not* start in an incorrect state, and it may *not* take an incorrect step. The assertion of a **liveness** property states that something good will eventually happen. To achieve this the $\Diamond$ operator (*eventually*) is used. ($\Diamond$ is equivalent to $\neg\Box\neg$) Liveness properties are expressed in TLA using the concept of **fairness**. We can apply two kinds of fairness to a behaviour.

- **Weak Fairness** : $WF_v(A) \triangleq \Box\Diamond\neg\, Enabled\langle A\rangle_v \vee \Box\Diamond\langle A\rangle_v$
  Weak Fairness asserts that either $\mathcal{A}$ is infinitely often disabled, or infinitely many actions $\mathcal{A}$ occur.
- **Strong Fairness** : $SF_v(A) \triangleq \Diamond\Box\neg\, Enabled(A)_v \vee \Box\Diamond(A)_v$
  Strong Fairness asserts that either $\mathcal{A}$ is becomes forever disabled, or infinitely many actions $\mathcal{A}$ occur.

WF (*Weak Fairness*) and **SF** (*Strong Fairness*) are used as operators when specifying systems in TLA $^+$.

2.2. The Modular approach

In order to keep specifications neat and simple to follow, TLA $^+$ uses modules to specify behavioural properties which may then be imported into other ones. This is essentially the same as is done with **Z** schemas [Spi89] although it should be remembered that in TLA $^+$, there is no notion of type and/or inheritance as such. Basically importing is used for clarity.

We stated above that we were interested in describing lower layer communications behaviours and relating

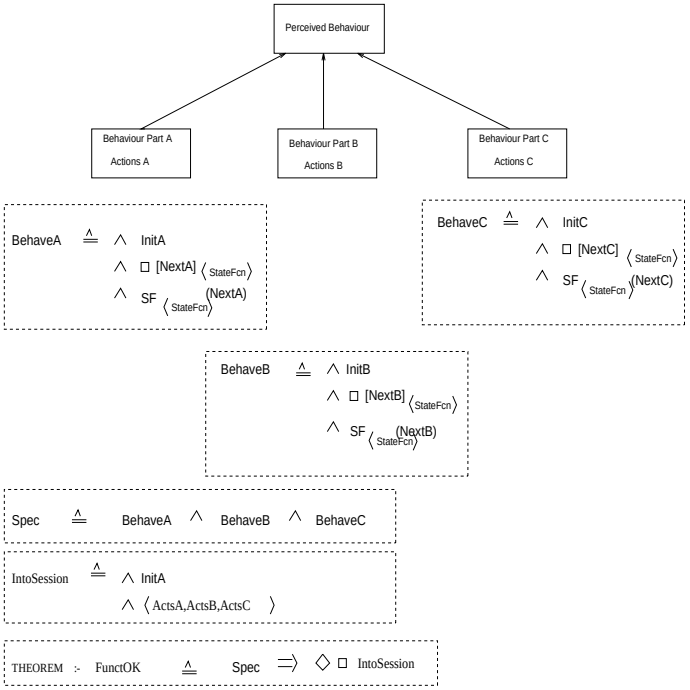

Fig. 3. Decomposition in TLA

them to higher level observations and this is achieved in the following manner.

Using TLA we can decompose our overall specification into a series of (possibly atomic) actions such as may be seen in figure 3 where our overall perceived behaviour has been decomposed into parts A B and C. These parts will probably be specified in their own individual modules each having a temporal formula of

the general form:

$$\begin{aligned} Spec \;\; &\triangleq \;\; \wedge \, Init \\ &\quad\;\; \wedge \, \Box[Next]_{\langle v \rangle} \\ &\quad\;\; \wedge \, SF_v(Next) \end{aligned}$$

- Predicate *Init* - meaning that the behaviour has started in a correct state
- Action *Next* says that the action will occur or that the system state will stutter forever
- The Temporal operator *SF* imparts liveness and says that the action *Next* will proceed or will be disabled.

We may then combine each of these parts and by use of some other derived temporal formula, assert a theorem to verify that our specification has been correct.

In a similar fashion, the module scheme (figure 2) shows a series of functions and we now look briefly at how these modules relate to each other.

2.3. Specification using TLA $^+$

Within the scheme there is a function known as `VideoTransfer` which has an MSC as shown in figure 4 The system specification (in TLA $^+$) consists of parameters, actions and temporal functions. In our video example we include the service participants, the type of signals used, system flags and variables. These definitions are extended by parameters which define sets of variables; and constants such as transmission paths, groups of QoS parameters in which we are interested and PDU data types. Predicates determine the contents, system states and ranges of values of variables. The actions specify how these states may be changed and the temporal functions are used to express the safety and liveness properties of the system. Actions may for example be described by a using predicates reflecting the signal types coming from the action's initial state and leading to the action's final state along with their appropriate parameters.
Here User 1 is communicating with User 2 in order to transfer video data, and upon receipt of the appropriate

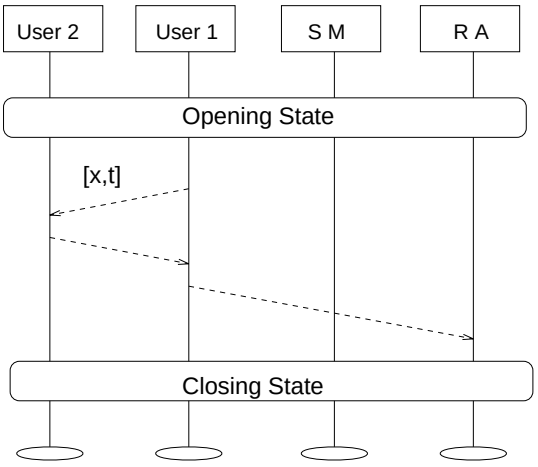

Fig. 4. JVTOS - Video Transfer

confirmation that the data has been received, the originator informs the Resource Allocator of the event.

module *VideoTransfer*

import : *SignalTypes, SessionStatus*

predicates

$User \triangleq U \in \{U1, U2, U3\}$

$S_{vt}(Vtr1) \triangleq \wedge\ tp = \langle U, U\rangle \vee \langle SM, U\rangle \vee \langle U, SM\rangle$
$\wedge\ snd \in \{U \cup \{SM\}\}$
$\wedge\ rcv \in \{U \cup \{SM\}\}$
$\wedge\ pp = [x, t]$
$\wedge\ qos = Q_b^\frown \langle Q_v\rangle$
$\wedge\ pdu =$ "call"

..... [*Other Predicates*] .....

$fn(VDatTfr) \triangleq \wedge\ TfrType =$ "PPMultiDirectionalStream"
$\wedge \langle S_{vt}(Vtr1),\ R_{vt}(Vtr2),\ I_x(Vtr3)\rangle$

The predicate $S_{vt}(Vtr1)$ defines aspects of the first video signal sent. Signal type parameters are imported from module `SignalTypes` and particular values are assigned here. The *transmission path* (`tp`) states that signals may pass between any of the users themselves and/or with the session manager. The *senders* (`snd`) and *receivers* (`rcv`) must be one of those and the *polling policy* (`pp`) allows `x` attempts each with time-out time `t`. *Quality of Service* (`qos`) is defined in this case as a tuple of the basic parameters (delay, delay jitter, error rate/burstiness and throughput) with the specifically defined video QoS parameters. The final conjunction refers to the PDU type (and hence phase) in which the signal is sent.
The other signals in this module are then defined and the final predicate states that in this case there is a Point-to-Point Multi-directional stream (defined in and imported from module `SessionStatus` and that the order of signals is *poll attempts, replies* and *control information to the Resource Allocator*.
We have seen the simple list of predicates defining the sequence of signals in a JVTOS function. In the case of the video signal S_{vt} reference is made to the parameter Q_v which are defined (elsewhere) in the module `VideoParams`, and imported through the module `SignalTypes`.

The module `VideoParams` contains a summary of the Video QoS Parameters wher m_num is the number of measures being made in Video Quality sampling and e_num is the number of measures in Time of Connection studies. S_i and R_i are the number of sent and received video TPDUs respectively and T_j represents values in a vector of times for successful point to point connection establishment. the time

1. QoVDL - The *Quality of TPDU based Video Delivery Loss* This is defined as the arithmetical mean value of the differences between the measured send and received video TPDUs along the measured time interval.
2. QoVRF - The *TPDU based Receiver Video Frequency* is the rate of displayed video TPDUs on the receiver side.
3. QoVCE - The *Quality of Video Connection Establishment* This parameter is a measure for the video connection establishment reaction time to a connection request sent by a user.
4. Q_v is a tuple of the QoS parameters QoVDL, QoVRF and QoVCE.

module *VideoParams*

import *MathSets*, *DelayParams*

parameters

$m\_num,\ e\_num$: **variables**

$R_i,\ S_i,\ T_j$: **variables**

$$QoVDL \triangleq \frac{1}{m\_num} \sum_{i=1}^{m\_num} (S_i - R_i)$$

$$QoVRF \triangleq \frac{1}{m\_num} \sum_{i=1}^{m\_num} (R_i)$$

$$QoVCE \triangleq \frac{1}{e\_num} \sum_{i=1}^{e\_num} (T_j)$$

$$Q_v = \langle QoVDL,\ QoVRF,\ QoVCE \rangle$$

3. Specification leading to Verification

We have seen wher the Video quality parameters fit into the scheme and in figure **??** we diagrammatically showed user-perception of a onference session. We may (simply) represent this in the form of a Time Sequence Diagram as shown in figure 5. In this diagram we see that in order to join the session the service has first

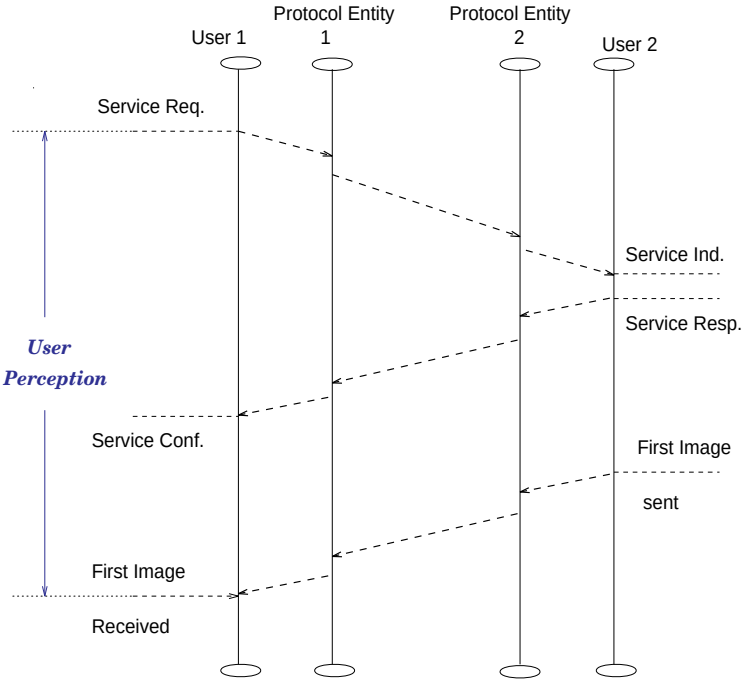

Fig. 5. Layered view of the User's perception of establishment delay

to negotiate and establish a connection and then transmit a video picture to the iniator of the request. (for clarity retries and unsuccessful attempts are not shown her but they are covered in specifications)
Suppose a user switches on a workstation and wishes to join a currently active JVTOS session. What happens

is that the signal is sent requesting the connection and the user knows that the session is functioning when the first video image is returned to the screen. Within the system itself of course we know that a number of technical functions have to be performed first.

3.1. Connection Delay

We consider the situation (figure 6) where a Connection request is made. CON.req and the appropriate CON.ind, CON.resp and CON.conf primitives occur. We take (for our example) a situation where up to `x` connection attempts are being made, each with a time-out limit of t_{out}. Furthermore, as soon as polling attempts commence, then we have stated that the system moves from a "stable" state into an "unstable" state. Here "unstable" is in the sense that the system has started with some actions, and that following that initial event, something *eventually* must happen. We also define a series of states (`conf`) which are used to define the current (and future) status of the systen after (any) confirmations have been received. We

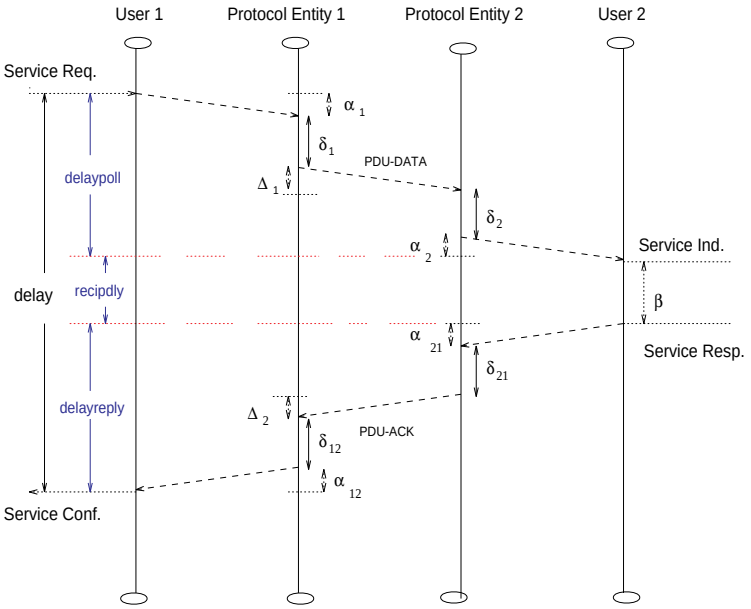

Fig. 6. A layered view of Delay

therefore define a TLA $^+$ specification of this part of the overall connection behaviour by specifying a module (`Connection` constructed in the following manner:

1. **import** : For the purposes of this example we are *assuming* that a number of qualities are being imported from other modules in the specification.
2. **variables** The variables used to record `delay` are declared
3. **predicates** :
   - Confirmation states may be : , , or (means that no confirmation has been received or is expected, and means that polling is in progress but that no response has been received.
   - The system states and signal states may be "stable" or "unstable".
   - The values for delay (figure 6) are :
     `delaypoll` - refers to polling signals (shown as a sum representing aggregate of polling attempts),

`recipdly` - refers to the recipient system response time and `delayreply` - refers to the time to send an or

- Predicate `ConInit` ensures that the process starts in a correct state.

4. **state function** : the function `vst` is defined over an n-tuple of states associated with the process.
5. **actions** : Action `Connect` states conditions for the (next) signal to happen (this includes the first attempt).
6. **temporal** : The formula `ConSpec` states our requirements that: The behaviour must start in a correct state, it will always be true that if the action `Connect` does not proceed then the associated states are invariant (and that the situation is *safe* but could stutter forever). The assertion of strong fairness `SF` implies that eventually action `Connect` will be disabled forever *or* that infinitely many `Connect` actions may occur.

module *Connection*

import : *SignalTypes*

parameters

delayval, delaypoll, delayreply, recipdly : **variables**

predicates

$conf \in \{\quad , \quad , \quad\}$

$(sigstate \in \{\text{“stable”}, \text{“unstable”}\})$

$delaypoll = \sum_1^x(\alpha_1 + \delta_1 + \Delta_1 + \delta_2 + \alpha_2)$

$recipdly = \beta$

$delayreply = \alpha_{21} + \delta_{21} + \Delta_2 + \delta_{12} + \alpha_{12}$

$$\begin{aligned} ConInit \triangleq\ & \wedge sigstate = \text{“stable”} \\ & \wedge conf = \\ & \wedge pdu = \text{“call”} \\ & \wedge delayval = 0 \end{aligned}$$

state function

$vst \triangleq \langle sigstate, conf, pdu\rangle$

actions

$$\begin{aligned} Connect \triangleq\ & \wedge ((sigstate' = \text{“unstable”}) \vee (sigstate' = \text{“stable”})) \\ & \wedge (sigstate = \text{“unstable”}) \\ & \wedge ((conf = \quad) \wedge (pdu = \text{“call”})) \\ & \wedge (conf' = \quad) \vee (conf' = \quad) \vee (conf' = \quad)) \\ & \wedge delayval = delaypoll + recipdly + delayreply \end{aligned}$$

temporal

$$\begin{aligned} \text{ConSpec} \triangleq\ & \wedge ConInit \\ & \wedge \Box[Connect]_{\langle vst\rangle} \\ & \wedge SF_{\langle vst\rangle}(Connect) \end{aligned}$$

3.2. The User's perception of Delay

When the user initiates the request the responses expected are the manifestations of a properly working service such as the appearance of the first image. As may be seen the module `SendImage` (below) is very similar to the previous module (`Connection`). The actions involved are differ only in the fact that the phase (data transfer) is different. The QoS value (delayval) is specified from the constituent delay elements of the process(`imagedelay`) and the temporal formula `ImageSpec` states the requirements for correct functioning with liveness asserted.

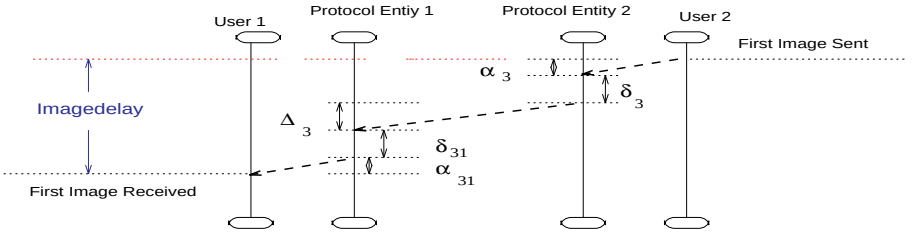

Fig. 7. Receiving the first Image

module *SendImage*

import : *SignalTypes*
imagedelay : **variables**

predicates

$conf \in \{\quad , \quad , \quad\}$
$sigstate \in \{\text{“stable”}, \text{“unstable”}\}$

$imagedelay = \sum_1^x(\alpha_3 + \delta_3 + \Delta_3 + \delta_{31} + \alpha_{31})$

$Init \triangleq \wedge\, sigstate = \text{“stable”}$
$\wedge\, ((conf = \quad) \wedge (pdu = \text{“data”}))$
$\wedge\, delayval = 0$

State Function

$vst \triangleq \langle sigstate, conf, pdu\rangle$

actions

$NextImage \triangleq \wedge\, ((sigstate' = \text{“unstable”}) \vee (sigstate' = \text{“stable”}))$
$\wedge\, (conf = \quad) \wedge (pdu = \text{“call”})$
$\wedge\, (conf' = \quad) \vee (conf' = \quad)$
$\wedge\, delayval = imagedelay$

temporal

$\text{ImageSpec} \triangleq \wedge\, ImageInit$
$\wedge\, \Box[NextImage]_{\langle vst\rangle}$
$\wedge\, SF_{\langle vst\rangle}(NextImage)$

3.3. A combined view of delay

Having specified the *Connection* and *SendImage* aspects of the user perception of delay, we know that:

- We have already accounted for the connection phase. (section 3.1).
- We have described the sending of the first image (section 3.2).
- We still have to account for the system response to the connection request. This involves a phase change "call" → "data" and we have to be careful to observe other aspects of state changes taking place.
- Taking the three "building blocks" of connection, system response and receipt of the first image, we should be able to prove that the system specification is correct by means of a theorem.

Before specifying the whole combined scenario, we therefore need to specify the link between the connection phase and the data-transfer phase. This is achieved through the module `VideoResponse`.

- **variables** - `respdly` represents delay experienced while the system responds to the connection request.
- **predicates** - `ReadyToRespond` is an inialising predicate ensuring a correct start to this stage. `respdly` is only briefly shown here, further work is being directed at specifying the exact nature of this aspect.
- **action** - `ImageResponse` represents state changes necessary for normal continuation of the connection.
- **temporal** - The formula `FirstImage` asserts that this stage of the process starts in a correct state and that the correct states for the next (data transfer) phase are satisfied.

module *VideoResponse*

import

parameter

respdly : **variable**

predicates

$$\text{ReadyToRespond} \triangleq \begin{array}{l} \wedge\ respdly = 0 \\ \wedge\ conf = \text{“call”} \\ \wedge\ ((sigstate = \text{“unstable”}) \wedge (sigstate = \text{“stable”})) \end{array}$$

$respdly = \beta_1$

action

$$ImageResponse \triangleq \begin{array}{l} \wedge\ ((conf = \quad) \wedge (conf' = \quad)) \\ \wedge\ ((pdu = \text{“call”}) \wedge (pdu' = \text{“data”})) \\ \wedge\ delayval = respdly \end{array}$$

state function

$$vst \triangleq \langle sigstate, conf, pdu \rangle$$

temporal

$$FirstImage \triangleq \begin{array}{l} \wedge\ ReadyToRespond \\ \wedge\ \Box\ [ImageResponse]_{\langle vst \rangle} \end{array}$$

3.4. The overall view

We now combine all three modules to describe the overall behaviour:

1. **include** : The modules `Connection, SendImage` and `VideoResponse` are included with the new naming `Conn, SendI` and `VidResp` respectively.

2. **parameter** : `totdelay` is a QoS parameter representing perceiver delay (it is accumulated as part of the action `Next`).
3. **predicate** : `UpAndRunning` is a predicate which is used to specify the conditions necessary for the behaviour to reach a satisfactory state.
4. **action** : Action `Next` takes the actions from the previous modules and conjoins them along with a definition of the total delay.
5. **temporal :** Formula `CheckFunction` asserts that the whole behaviour starts in a correct state and proceeds through the required stages with liveness being asserted in the form of strong fairness.
6. **theorem** : The final stage is to define a theorem (proof not shown) which states that if the conditions defined in the predicate `CheckFunction` have been met, then this this implies that eventually it will always be true that the system operates correctly (`UpAndRunning` $\Rightarrow$ "`true`").

module *PerceivedDelay*

include *Connection* **as** *Conn*
include *SendImage* **as** *SendI*
include *VideoResponse* **as** *VidResp*

parameter
totdelay : **variable**
predicate

$$UpAndRunning \triangleq \land Conn.ConSpec \land VidResp.FirstImage \land SendI.ImageSpec$$

action

$$Next \triangleq \land Conn.Connect \land VidResp.ImageResponse \land SendI.NextImage \land totdelay = Conn.delayval + VidResp.delayval + SendI.delayval$$

temporal

$$CheckFunction \triangleq \land Conn.Init \land \Box[Next]_{\langle vst \rangle} \land SF_{\langle vst \rangle}(Next)$$

theorem

$$SignalCorrect \triangleq CheckFunction \Rightarrow \Diamond\Box\, UpAndRunning$$

4. Conclusionss

Specifying JVTOS has not been our goal, but by specifying the service in the manner indicated we have been able to provide a solid base for our work, always being able to simply refer to individual modules as

necessary. Our experience has also shown that, having no *type*, we may change a single module and are not required to check through the entire specification.

We are continuing to collaborate with our colleagues in TU-Berlin in order to define and specify JVTOS QoS parameters. It is our intention to then be able to relate this work to other studies being done elsewhere where user perception is being studied. In order to achieve this we are developing a calculus to describe the atomic actions taking place, and with the aid of temporal logic we will build verifyable descriptions of those actions.

Any mathematical verification involving logic and algorithms built from a variety of disparate sources can become complicated, requiring the reader to have a deep understanding of the issues being addressed *and* the mathematics employed. Clearly any tools to assist with this task are most welcome and we are currently seeking tools to assist us to achieve our goals. We have recently acquired the TLA prover TLP 2.5 [Eng94] which is based on the Larch Prover [Gar] and have commenced its evaluation.

In conclusion we have found a relatively simple means of describing fairly complex scenarios by using TLA and applying it to those areas that interest us. We have found a method that is able to take a broad view of the JVTOS service by specifying the actions associated with its behaviour. At the same time as this we are able to reason about timing constraints and assert liveness conditions and yet still be able to give straightforward representations of measurement requirements. It is our intention now to pursue this line, and with the aid of verification environment to address many of the areas identified that still require specification and verification.

Acknowledgements

My thanks are to Professor Ken Turner (University of Stirling) for his counsel and to Dr Peter Ladkin (University of Nancy) for his invaluable thoughts on TLA. I would also like to thank my TOPIC colleagues Plamen Simeonov (TUB) and Stefan Leue (University of Berne) for there advice, and the RACE Programme for funding this work.

References

[B.M92] I.Miloucheva B.Metzler. Multimedia communication platform: Specification of the broadband transport protocol xtpx. Technical report, Technical University of Berlin / CIO RACE Project, 1992.

[Don93] A. J. M. Donaldson. Specification of quality of service measurement points in jvtos, 1993.

[E.B87] E.Brinksma. An introduction to lotos. North-Holland Publ., Amsterdam, 1987. 7th IFIP WG6.1 International Workshop on Protocol Specification Verification and Testing.

[Eng94] Urban Engberg. Tlp - the tla proof checker. Technical report, Institute of Mathematics, Aarhus University, Denmark, 1994.

[Gar] Steve Garland. A guide to lp,the larch prover. Technical report.

[GE93] M.Faloustos N.Magafossis G.Stasinopolus and E.Papachristou. Qos tests for jvtos. basic and extended methods. Technical report, Intracom / RACE II (R2088) TOPIC/WA1/ICOM, 1993.

[Lam91] Leslie Lamoport. The temporal logic of actions. Technical report, Digital Equipment Research Corporation, Systems Research Center, 1991.

[ML92] M.Abadi and L.Lamport. An old fashioned recipe for real time. Technical report, Digital Equipment Research Corporation, Systems Research, 1992.

[RR89] J.R.W.Smith R.Saracco and R.Reed. *Telecommunications Systems Engineering using SDL.* North Holland, Amsterdam, 1989.

[Spi89] J.M. Spivey. *The Z notation, A reference manual.* Prentice Hall, 1989.