

An Investigation into the Hardware Abstraction Layer of the Plural Node Architecture for IEEE 1394 Audio Devices

Nyasha Chigwamba and Richard Foss
Department of Computer Science
Rhodes University, Grahamstown, South Africa
Tel: 046 6038291 Fax: 046 6361915
Email: nchigwamba@gmail.com and r.foss@ru.ac.za

Abstract—Yamaha Corporation introduced the concept of a music Local Area Network (mLAN) which uses IEEE 1394 as its underlying network technology. Second generation mLAN is based on the Plural Node Architecture which splits connection management of audio devices over two nodes, namely a Transporter and an Enabler. A client/server implementation exists which enables mLAN connection management to be done via TCP/IP. The Open Generic Transporter (OGT) specification was introduced to provide an open interface between the Enabler and Transporter. This paper describes how the current mLAN Transporter Hardware Abstraction Layer (HAL) Application Programming Interface (API) has been modified to exploit all the benefits introduced by the OGT concept while maintaining backwards compatibility with Transporters based on the current HAL API.

Index Terms—FireWire, music Local Area Network, Open Generic Transporter, Plural Node Architecture, XML

I. INTRODUCTION

M LAN (music Local Area Network) is a digital network interface technology being promoted by Yamaha which allows professional audio equipment, PCs and electronic instruments to be interconnected using a single cable [1]. IEEE 1394, commonly known as *FireWire*, is used as the underlying networking technology due to its dual asynchronous and isochronous capabilities which enable the transmission of both control and multimedia data over the same cable [2].

The International Electrotechnical Commission defines an Audio and Music (A/M) data transmission protocol, also known as the IEC 61883-6, that governs the encapsulation and extraction of audio or MIDI by FireWire audio transmitters and receivers respectively [3]. In addition, the 1394 Trade Association has documented an architecture known as the *Plural Node Architecture* (also known as the Enabler/Transporter Architecture) which splits connection management of audio devices over two nodes, namely a Transporter and an Enabler [4]. The Transporter resides on the audio device and is responsible for the encapsulation and extraction of audio/MIDI data in accordance with the A/M protocol. The Enabler, residing on a PC workstation, is responsible for making or breaking connections

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from Telkom SA, Business Connexion, Converse, Verso Technologies, Tellabs, StorTech, THRIP and The Mandela Rhodes Foundation.

between Transporters. For each type of Transporter, a HAL plug-in is required (implemented against the current mLAN Transporter HAL API, referred to as "HAL API" for brevity) and is loaded by the Enabler when communicating with the Transporter in a device specific manner.

The mLAN project has also created a client/server architecture above the Plural Node Architecture. Here, a server application uses the Enabler to fulfil XML-based connection management requests exchanged with one or more client applications via TCP/IP.

In the context of the Plural Node Architecture, the OGT concept was introduced to ease the task of manufacturers [5]. If manufacturers design Transporters conforming to the OGT specification, there will be no need to create the manufacturers' own HAL plug-ins since they can make use of a common OGT HAL plug-in. The introduction of the OGT specification has resulted in additional capabilities that are not being realised by the current HAL API.

This paper proposes a new HAL API to remedy the deficiencies of the current HAL API. In the new HAL API, a set of functions has been defined to manipulate Transporters that adhere to either the OGT specification or a manufacturer-specific Transporter implementation for mLAN devices. The new HAL API, in addition to exposing existing Transporter capabilities, exposes enhanced capabilities resulting from the OGT specification. The impact of the proposed HAL API on the existing client/server implementation is also described.

II. BACKGROUND

A. mLAN Connection Management

In an mLAN network, audio, MIDI, synchronisation and control data are packaged into isochronous packets with each packet having a channel number within its header. Packets bearing the same channel number comprise an *isochronous stream*. A detailed description of the packet format is available in the IEC 61883-6 specification [3]. The most important concept in this specification as identified by Fujimori and Foss in "*A New Connection Management Architecture for the Next Generation of mLAN*" [6] is that of a *sequence*. An isochronous stream comprises a number of sequences where each sequence corresponds to a particular position within an isochronous

packet cluster as shown in Fig. 1. Although not shown here, up to 8 MIDI sequences may be multiplexed on one sequence.

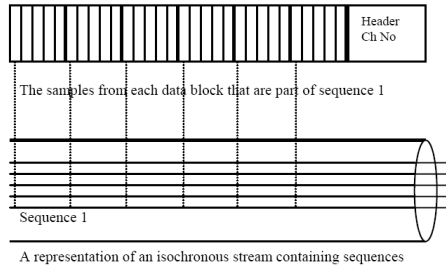


Fig. 1. An isochronous stream with sequences [6]

A number of chips (ASICs) that are responsible for the encapsulation (at the transmitter side) and subsequent extraction (at the receiver side) of audio and MIDI data according to the A/M protocol have been created. Each of the chips has a transmission FIFO that receives audio or MIDI data from its input pins, packages the data into isochronous packets with a channel number that would have been previously allocated from an Isochronous Resource Manager and sends out the packets onto the FireWire bus. In addition to allocating a channel number, a transmitter must also allocate bandwidth from the Isochronous Resource Manager [7]. For data reception, there are FIFO buffers that receive audio samples from the audio sequences and MIDI messages from the MIDI sequences. Selection of an isochronous stream and sequence within that stream is done via a pair of registers for each buffer as shown in Fig. 2.

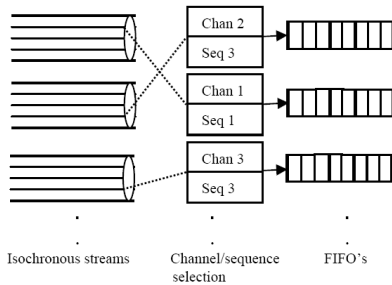


Fig. 2. mLAN Sequence Selection [6]

B. Enabler/Transporter Architecture

Section I introduced the Enabler/Transporter Architecture. More specifically, the Enabler enables connections between mLAN plugs. mLAN plugs are end points of the sequences mentioned in section II-A. On the device side, there is an "mLAN Node Controller"¹ and associated firmware that together comprise a Transporter. A Transporter is responsible for the encapsulation and extraction of audio and music data in accordance with the A/M protocol. In addition, a Transporter will rely on an Enabler to set up its A/M data parameters for transmission and reception of audio and music data sequences.

¹A collection of all chips required for the extraction and encapsulation of audio and music data

The Enabler comprises three layers, namely the mLAN Plug Abstraction, A/M Manager and Hardware Abstraction layers as shown in Fig. 3. A detailed discussion of an Enabler can be found in a PHD thesis entitled "High Speed End-To-End Connection Management in a Bridged IEEE 1394 Network of Professional Audio Devices" [8]. A brief description of each of the layers follows.

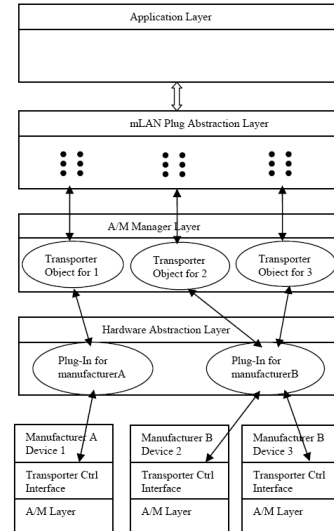


Fig. 3. The Enabler's Layers and Interfaces [6]

1) *mLAN Plug Abstraction Layer*: This is the top layer which implements a number of mLAN plugs. These plugs are terminators of the A/M data sequences that are transmitted and received by the Transporters. In particular, this layer implements input and output mLAN plug abstractions for all possible end points of all the Transporters under its control.

2) *A/M Manager Layer*: This layer is responsible for reading A/M data transmission and reception parameters from the associated Transporters and for updating these parameters in response to requests from the mLAN Plug Abstraction layer. Each of the Transporters under the control of the Enabler will have a Transporter object that keeps its state information and handles requests from both the plug abstraction layer and the actual Transporter.

3) *Hardware Abstraction Layer*: This layer is responsible for abstracting away the details of the various hardware implementations of Transporters. Non-mLAN chip manufacturers acquire mLAN compliance via this layer, in conjunction with the Transporter Control Interface. There is variation in the way in which parameters are set up for A/M data transmission and reception across manufacturers. Consequently, each manufacturer can provide a software HAL plug-in that translates A/M configuration requests from the A/M manager into proprietary requests that the Transporter Control Interface of that particular manufacturer will understand.

C. mLAN Connection Management Server (mCMS)

The mCMS is a server application that runs on the same workstation as the Enabler [9]. The mCMS accepts mLAN

device connection management requests from client applications and effects the requests via the Enabler, and reports to client applications when the status of devices on the FireWire network changes. Communication between each client and the server is in the form of XML across a TCP/IP connection. Clients may be patch bay applications running on a variety of devices ranging from PDAs and Laptops, to PC Workstations. Fig. 4 shows an mLAN Client/Server configuration.

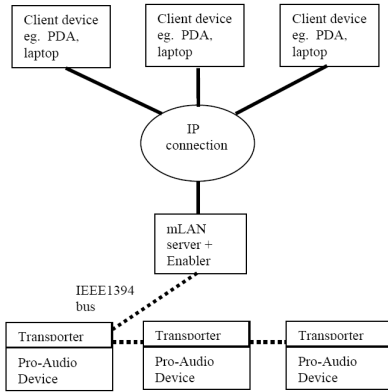


Fig. 4. mLAN Client/Server Configuration [9]

D. Open Generic Transporter Specification

Section I introduced the motivation for the OGT specification, namely vendor convenience. The specification indicates a number of functional blocks that comprise a generic Transporter and can be controlled via a set of registers [10]. This set of registers and their locations comprise the open interface.

1) *Generic Transporter Architecture*: Two sub-blocks exist within the Generic Transporter block, namely the Node Application and the Node Controller blocks. The Node Application represents the hosting device and its range of legacy audio and MIDI plugs. The Node Controller represents the IEEE 1394 node that is hosted by the device. The main components of the Node Controller are the A/M Transport block, Transport Restoration block and Transporter Control Interface. The Enabler sends control commands to the Transporter Control Interface via a number of generic registers. These commands are then forwarded to the A/M Transport block to perform actual encapsulation and extraction in terms of the underlying hardware/software supplied by the Transporter’s vendor. The Transport Restoration block allows the Transporter to resume transmissions in the absence of an Enabler.

2) *Generic Architecture of the A/M Transport Block*: Fig. 5 shows a generic architecture for the A/M Transport block. In the diagram, the Sync block handles the synchronisation functionality whilst the Plug block handles connection management functionality. A detailed description of how synchronisation is handled may be found in the IEC 61883-6 specification [3]. An important point to be noted is the introduction of the *Isochronous Stream Plugs (ISPs)* and *Node Controller Plugs (NCPs)* within the A/M Transport block. An ISP represents an input or output for a single isochronous stream to or from the IEEE 1394 bus [10]. On the other

hand, an NCP represents an input or output for a single monaural channel of audio or a single cable of MIDI to or from the Node Application. Therefore, each NCP is a sequence end point. One or more NCPs can be attached to an ISP and a sequence number can be specified for each NCP to determine the positions of audio samples or MIDI messages within transmitted or received isochronous packet clusters. The current HAL API follows an approach which assumes that it would be possible for each receiving sequence plug to be associated with a unique isochronous stream. A receiving sequence plug in this case is synonymous with an mLAN plug (sequence endpoint) described in section II-B.1, and maps directly to a receiving FIFO buffer as shown in Fig. 2. This paper proposes a new HAL API which is based on the abstraction shown by ISPs and NCPs where multiple sequence end points can be associated with an isochronous stream.

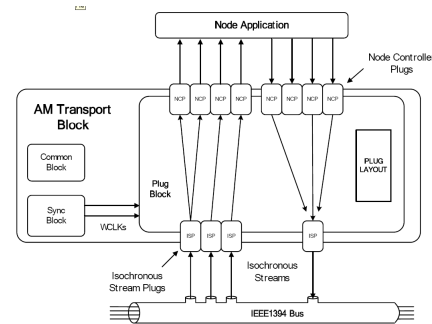


Fig. 5. Generic A/M Transport Block [10]

III. APPROACH TO THE STUDY

Two types of evaluation boards were used, namely the DICE II EVM board and the MAP4 board. The DICE II EVM board houses the DICE II chip which performs encapsulation and extraction of multimedia data, and has a firmware implementation that follows the OGT specification. The MAP4 board houses mLAN-NC1 [11] and mLAN-PH2 [12] chips which perform encapsulation of audio and MIDI data, and follows Yamaha’s manufacturer specific design on which the current HAL API was based. This combination of mLAN-NC1 and mLAN-PH2 chips is known as the NCP05 design [13].

A. Problems Identified with Existing HAL API

A critical evaluation of the current HAL API and its subsequent plug-in implementations led to the discovery of the following shortcomings when it came to the plug-in implementation for the OGT:

- 1) *Devices with Limited Receiving Channel Capabilities*
The DICE II chip, for example, has four independent 1394 audio receivers [14]. Each receiver can extract up to sixteen audio channels and eight MIDI plugs. Section II-D.2 mentioned how the current HAL API is based on the assumption that it is possible for each receiving sequence end point to be associated with a unique isochronous stream. On the contrary, the operation of an OGT is such that multiple NCPs, up to a specified

limit, may be attached to each ISP. Such attachment of multiple NCPs to each ISP leads to clustering of multiple sequence end points to an isochronous stream. For an OGT, the current HAL API does not take into account limitations in reception capabilities in cases where the number of sequence end points (NCPs) is greater than the maximum number of isochronous streams² that may be received.

The current HAL API is based on the association of FIFO buffers with a pair of registers as shown in Fig. 2, and thus exposes sequence end points (referred to by an ID called *isochID*) to the Enabler. Each of the FIFO register pairs is uniquely identifiable via the *isochID*. The OGT HAL plug-in implementation against the current HAL API only exposes NCPs (identifiable via a unique *isochID*). As an example, reference is made to the following current HAL API call which, among other calls, is called within the receiver when making a connection from a transmitter:

```
SetIsochChannel(isInput, isochID,  
isochChannel)
```

In the case of an OGT, when specifying the channel number for the isochronous stream to receive, a write is made to the channel register of the ISP associated with the NCP as opposed to the channel register of a FIFO buffer as shown in Fig. 2. To resolve this, mappings have been made within the current implementation to map ISPs to associated NCPs as well as map *NCP IDs*³ to *isochIDs*. Assuming that all ISPs are in use and no additional isochronous streams can be received, the operation fails. Evidently, the current HAL API does not deal with such a constraint in the best possible manner. This can be resolved by having a HAL API which exposes ISPs, in addition to the NCPs.

2) Dependencies in Starting/Stopping ISPs

When an Enabler makes an audio or MIDI connection between mLAN plugs, it ensures that the transmitter and receiver are sending and receiving packets respectively. In doing so, among other calls, it calls the following current HAL API call:

```
Start(isInput)
```

When *isInput* is true, the call above will ensure that all receivers on a given Transporter are set to receive isochronous packets assuming that the channel numbers for the isochronous packets to be received have already been specified. On the other hand, if *isInput* is false, all transmitters on a given device will start transmission assuming that an available channel and bandwidth have been allocated for each transmitter as mentioned in section II-A. The audio receivers (receiving ISPs) and audio transmitters (transmitting ISPs) on the DICE II chip, for example, are independent [14]. An implementation of the OGT plug-in against the current HAL API imposes a dependency on both,

audio receivers and transmitters, which leads to loss of independent operation. For transmitters, this approach results in bandwidth wastage in situations where, for example, only one of the audio transmitters is required for audio/MIDI connections but all transmitters have to be started.

3) Plug Layouts

The OGT specification “takes account of the many ‘out of the ordinary’ aspects of the various vendors’ A/M Transport implementations” [5] by introducing the concept of a Plug Layout Table which contains a number of Plug Layouts. Each Plug Layout gives information about the types and number of plugs and wordclocks⁴ a device implements. An example of one such ‘out of the ordinary’ aspect is the manner in which the number of audio channels is dependent on sampling frequency as is currently the case with the NCP05 design where, at higher sample rates, more data will be transmitted resulting in fewer sequences being transmitted or received [15]. This idea was explored further and bandwidth was identified as another possible aspect that could warrant the need for different Plug Layouts. It may be possible for a manufacturer to create an OGT-based Transporter with Plug Layouts that reduce bandwidth wastage. The current HAL API does not explicitly define this Plug Layout concept, hence the OGT plug-in implementation would only change Plug Layouts upon sample rate changes. More flexibility can be achieved by having a HAL API that allows explicit switching of the Plug Layouts.

4) Multiple Wordclock Outputs

The IEC 61183-6 specification describes a mechanism for the synchronisation of a slave device’s clock to a master device’s clock [3]. Current mLAN Transporter implementations only make use of one unit (termed *wordclock output*) on the device to handle synchronisation. However, the OGT specification allows for the possibility of more. This capability is not supported by the current HAL API, hence the OGT plug-in implementation does not make use of the additional capability.

5) Wordclock Source Selection

Synchronisation occurs between a master device’s clock and a slave device’s clock. The current HAL API is based on the assumption that a Transporter will have two clocks, namely an internal clock and an SYT clock. The internal clock is used when a Transporter is acting as a master, while the SYT clock is used when a Transporter acts as a slave within a synchronisation relationship. The OGT specification also indicates the possibility of having more than one internal type of clock, in addition to a possible SYT clock. Again this is not exposed to the Enabler by the current HAL API.

²Equivalent to the maximum number of receiving ISPs.

³A unique index for each NCP within the OGT register set.

⁴Related to the synchronisation process that should be done before making audio connections.

B. Design and Implementation

In view of the problems raised in section III-A a new HAL API was created that sought to resolve the shortcomings of the current HAL API when implemented for an OGT-based Transporter, while retaining backwards compatibility with manufacturer specific Transporters such as the NCP05 design. The ISP/NCP concept of the OGT, introduced in section II-D.2, was viewed to be generic enough to encapsulate the functions of a design such as the NCP05 and hence forms the basis of the new HAL API. Consequently, the resulting HAL API is biased towards the OGT, although in a manner that allows plug-in implementations for manufacturer-specific designs such as the NCP05. Key aspects of the new HAL API include:

- Based on ISP/NCP concept
- Inclusion of Plug Layout switching capabilities via a new HAL API call:
SetCurrentPlugLayout(plugLayoutID)
- Support for multiple wordclock outputs via a new HAL API call:
*GetNumWordclockOutputs(pluglayoutID, * numWCLKs)*
- Support for wordclock source selection via a new HAL API call:
SetCurrentClockSource(pluglayoutID, wclkID, clockID)

Two proof-of-concept implementations are described below.

1) *OGT Plug-in Implementation*: A number of important aspects have been modified within this implementation. The mappings mentioned in part 1 of section III-A have been removed. ISPs and NCPs are identified by ISP IDs and NCP IDs which are unique indices relative to the OGT register set.

It is now possible to have individual control over ISPs. For example, when more than one ISP exists, it is possible to start only one of the ISPs via a new HAL API call such as:

Start(ispID)

The flexibility resulting from this is evident when compared with a similar call in the current HAL API mentioned in part 2 of section III-A.

Audio channel and MIDI plug extraction and encapsulation restrictions such as those mentioned in part 1 of section III-A are being represented more clearly to the Enabler with the use of the new HAL API call:

*GetMaxAttachableNCPs(ispID, ncpType, *numNCPs)*

2) *NCP05 Transporter Plug-in Implementation*: To demonstrate backwards compatibility of the new HAL API some rules were created to guide this plug-in implementation. In particular, only two Plug Layouts are exposed by this architecture. One to cater for all the audio channels that are available for sample rates lower than or equal to 48kHz and another for sample rates greater than 48kHz. Unlike the OGT plug-in implementation where a change in the Plug Layout configuration in use results in a write to a plug layout register, for the NCP05 this results in a change in a sample rate register. In addition, there shall only be one wordclock output and two possible wordclock sources, internal and SYT, as discussed in part 5 of section III-A.

For transmission, each of the chips (mLAN-NC1 and mLAN-PH2) now represents a uniquely identifiable transmitting ISP for which isochronous resources (channel and bandwidth) should be allocated prior to starting transmission. Transmitting NCPs are *virtual in-memory objects* that do not map to any registers but determine the number of sequences that the associated ISP will transmit. Note that the number of sequences being transmitted is directly proportional to the amount of bandwidth in use [16] hence this was a critical component in the implementation.

Part 2 of section III-A describes how the current HAL API only took into account collective starting/stopping of transmitters or receivers. The NCP05 design requires such a dependency for its transmitters (one transmitter for the mLAN-NC1 and another for mLAN-PH2) [15]. Given the fact that the new HAL API allows for independent control of ISPs, unless specified, a solution had to be provided. Fortunately, the OGT specification allows for constraints to be defined for certain attributes of ISPs or NCPs [10]. In resolving this problem, constraints were therefore put in place for both transmitting ISPs that correspond to the mLAN-NC1 and mLAN-PH2 chips. These constraints are required to notify the Enabler that both ISPs need to be started/stopped at the same time. The Enabler will then be able to allocate/deallocate resources, via the Isochronous Resource Manager, for each of the chips.

For reception, receiving ISPs are *virtual in-memory objects* that do not map to any registers, but rather keep track of the unique isochronous streams being received by the Transporter. Each of the FIFO buffers shown in Fig. 2 now represents a uniquely identifiable receiving NCP. In the diagram, there is a pair of registers to represent the channel and sequence number. In order to go about this, we followed an approach where the channel register is only written to when a receiving NCP is associated with a receiving ISP. Fig. 6 below is a modified version of Fig. 2 that shows how the ISP/NCP model has been incorporated in this example implementation.

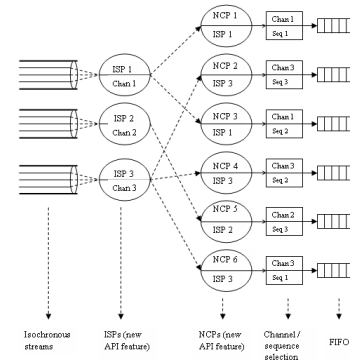


Fig. 6. Modified mLAN Sequence Selection with NCPs and ISPs.

C. Results

A client application was modified to show the new HAL API in operation. Connections could be made between mLAN plugs on a DICE II EVM board and mLAN plugs on a MAP4 board showing backwards compatibility of the new

HAL API. Fig. 7 shows a section of the patch bay display for the client application. There are lists of source and destination mLAN plugs. In addition, an 'AdditionalInfoForm' window, superimposed over the destination mLAN plug list, shows some of the aspects introduced by the new HAL API, namely 'Possible Connections', 'Number of Plug Layouts' and the 'Current Plug Layout ID'⁵. 'Possible Connections' informs a user of the number of additional isochronous streams a device can still receive and thus alleviates the problem mentioned in part 1 of section III-A without "surprising" a user.

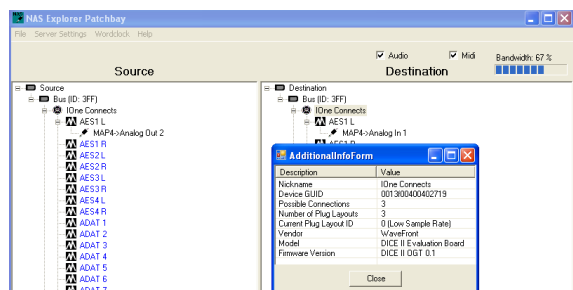


Fig. 7. Patch bay display showing mLAN plugs and device information

Fig. 8 shows how the concepts of multiple wordclock outputs and wordclock source selection have been incorporated in the client application. As shown on the top left section of the diagram, the current Transporter implementation for the OGT has only one wordclock output although the client can handle more. The bottom half of the diagram shows how the wordclock source can be specified for a given master wordclock output.

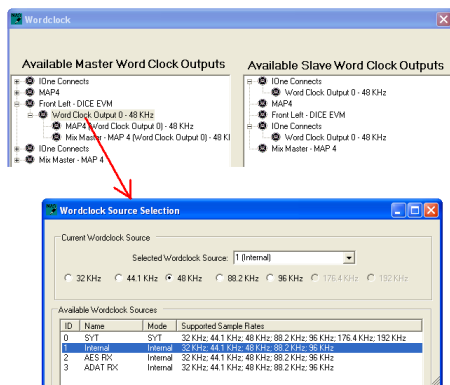


Fig. 8. Multiple wordclock outputs and wordclock source selection

The new HAL API has resulted in more powerful end-user client application capability that alleviates problems identified in section III-A. With reference to Fig. 4, it is also clear that any communications protocol other than TCP/IP can be used for client/server communication. This is facilitated by the platform-independent nature of XML.

IV. CONCLUSION

We have shown how the introduction of the OGT specification created the need to investigate the HAL of IEEE

1394 audio devices by highlighting some problems resulting from an OGT plug-in implementation against the current HAL API. By defining a new HAL API, we have attempted to alleviate the problems resulting from a HAL API that had been defined without the OGT concept in mind. An example of a working implementation has been provided to demonstrate the new HAL API at work and interoperability between OGT-compliant and manufacturer-specific Transporters. We have also shown how the new HAL API has resulted in more powerful end-user client applications in a client/server environment. In particular, capabilities that were previously not available to end-user client applications have been made possible by defining a more comprehensive HAL API. Such capabilities include, explicit switching of Plug Layouts, the ability to display multiple wordclock outputs and allowing wordclock source selection for each of the wordclock outputs.

REFERENCES

- [1] Yamaha Corporation, "Otari, Yamaha to Promote mLAN Digital Network Interface Technology." [online]. Available at: <http://www.yamaha.co.jp/english/news/00092102.html>. [Accessed 30 April 2006].
- [2] IEEE Std. 1394-1995, "Standard for a High Performance Serial Bus".
- [3] IEC 61883-6 - Ed. 2.0, "Consumer Audio/Video equipment - Digital Interface - Part 6: Audio and Music Data Transmission Protocol".
- [4] IEEE 1394 Trade Association Working Draft, "Plural Node Implementation of a Professional A/M Device", September 2004.
- [5] K. Kounosu, J. Fujimori, R. Laubscher, and R. Foss, "Convention Paper: An Open Generic Transporter Specification for the Plural Node Architecture of Professional Audio Devices," in *Convention Paper, 118th AES Convention*, (Barcelona, Spain), Audio Engineering Society, May 2005.
- [6] J. Fujimori and R. Foss, "A New Connection Management Architecture for the Next Generation of mLAN," in *Convention Paper, 114th AES Convention*, (Amsterdam, The Netherlands), Audio Engineering Society, March 2003.
- [7] Don Anderson, "FireWire System Architecture". New Jersey: Mindshare Inc., 2nd ed., 1999.
- [8] H. Okai-Tetty, "High Speed End-To-End Connection Management in a Bridged IEEE 1394 Network of Professional Audio Devices". PhD thesis, Rhodes University, Grahamstown, 2005.
- [9] J. Fujimori, R. Foss, B. Klinkradt, and S. Bangay, "An mLAN Connection Management Server for Web-Based, Multi-User, Audio Device Patching," in *Convention Paper, 115th AES Convention*, (New York, USA), Audio Engineering Society, October 2003.
- [10] Audio Engineering Society - Standards Committee, *Draft AES Standard for Audio over IEEE 1394 - Specification of Open Generic Transporter*. Audio Engineering Society, 2005.
- [11] Yamaha Corporation, "mLAN-NC1 PH1 Block Specification", 2001. Confidential.
- [12] Yamaha Corporation, "mLAN-PH2 (YTS440-F) Specification", 2003. Confidential.
- [13] Yamaha Corporation, "MAP4 User's Manual", 2003.
- [14] TC Applied Technologies, "Specification: DICE (Digital Interface Communications Engine) - User Guide".
- [15] Yamaha Corporation, "NCP04/05 Transporter Specifications", 2002. Confidential.
- [16] Yamaha Corporation, "Guide for Bandwidth Efficiency", 2002.

Nyasha Chigwamba holds a Bachelor of Science (Software Development) degree from Rhodes University. He is currently reading for a Masters degree in Computer Science at the same institution. His research interest is in the area of IEEE 1394 audio networking.

⁵The current Plug Layout ID is modified via a new HAL API call.