IMPROVING THE EFFICIENCY OF OPEN SOUND CONTROL WITH COMPRESSED ADDRESS STRINGS

Jari Kleimola
Department of Signal Processing and Acoustics
Aalto University School of Electrical Engineering
Espoo, Finland
jari.kleimola@aalto.fi

Patrick J. McGlynn
Sound and Digital Music Technology Group
National University of Ireland, Maynooth
Co. Kildare, Ireland
patrick.j.mcglynn@nuim.ie

ABSTRACT

This paper introduces a technique that improves the efficiency of the Open Sound Control (OSC) communication protocol. The improvement is achieved by decoupling the user interface and the transmission layers of the protocol, thereby reducing the size of the transmitted data while simultaneously simplifying the receiving end parsing algorithm. The proposed method is fully compatible with the current OSC v1.1 specification. Three widely used OSC toolkits are modified so that existing applications are able to benefit from the improvement with minimal reimplementation efforts, and the practical applicability of the method is demonstrated using a multitouch-controlled audiovisual application. It was found that the required adjustments for the existing OSC toolkits and applications are minor, and that the intuitiveness of the OSC user interface layer is retained while communicating in a more efficient manner.

1. INTRODUCTION

Open Sound Control (OSC) [1], [2], [3] is a widely used content format for communicating between media-related applications. Its popularity can be attributed to the intuitive and extensible addressing scheme, which, together with the ability of describing typed parameter spaces, enables setups that may easily adapt to a wide variety of application scenarios. Furthermore, the unidirectional communication protocol simplifies the connection setup between OSC compliant end points.

In OSC, the transmitted information stream flowing between the end points, from an OSC producer/controller to the OSC receiver, is quantized into time-tagged frames. Each frame contains one or more OSC messages, which are described by a human-readable URL-style address part and a typed data vector (see Table 1). The data vector is further divided into type tags and the actual arguments carrying the instantaneous data values. The address part and the type tags are transmitted as strings, whereas the arguments are kept in binary format. The fields of the message are aligned on 4-byte boundaries.

Address part	**Data vector**	
address	type tags	arguments
"/LPF/1/cutoff"	"f,"	0.50

Table 1. The structure of an OSC message.

Figure 1 shows an example communication sequence for a standard OSC v1.1 compliant parameter update procedure. The controller addresses a synthesizer parameter located at "/LPF/1/cutoff", sets its value to 0.50, and then immediately updates the value to 0.51. As can be seen, the controller simply needs to push the updated parameter values to the synthesizer, which parses the address string, and updates its data structures accordingly with the binary argument values. The communication language (i.e., the namespace, data types and value ranges of the arguments) is defined by the receiving end, while the controller is configured to speak in the same language. The benefits of OSC are clear: the communication language is intuitive, extensible, and the stateless communication paradigm results in simplified application models for both end point implementations.

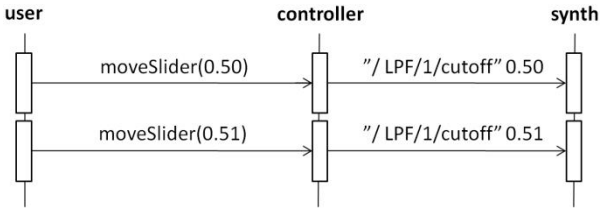

Figure 1. Standard OSC communication sequence.

The previous example also illustrates the inefficiency of the standard OSC implementations: although the human-readable addressing scheme is intuitive, long URL-style address strings waste communication bandwidth because the address part of the message is transmitted repeatedly with each parameter update. Furthermore, the receiving end needs to perform time-consuming string parsing operations for each received message. These points were criticized in [4], which also suggested that the inefficiencies might be eliminated if certain established IP-based techniques were adapted into the core of the OSC specification. However, the previous work did not elaborate how these techniques might be adapted in practice.

This paper explores the actual adaptation process of DNS-inspired [5] address mapping methods, and introduces an OSC v1.1 compliant address compression tech-

nique to improve the efficiency of OSC-based communication. The proposed method is especially useful for resource-constrained devices, where the CPU and power consumption efficiency is of great importance. It is also beneficial for systems where the physical connection between the end points is over a wireless medium, because increased network traffic has a tendency of raising the amount of lost packets.

The remainder of this paper is organized as follows. Section 2 introduces the OSC address compression technique, while Section 3 describes its implementation in three popular development environments and toolkits. Section 4 demonstrates the practical applicability of the approach using a multitouch-based interaction device in an audiovisual application setup. Section 5 provides discussion and suggestions that might be useful when speculating upon the next major OSC specification update. Finally, Section 6 concludes the paper.

2. OSC ADDRESS COMPRESSION

In order to improve the efficiency of the standard OSC communication mechanism, it seems beneficial to a) reduce the amount of transmitted data, and b) to simplify the parsing algorithm on the receiving end. This can be accomplished by compressing the potentially lengthy address string of the OSC message into an integer token, which is then transmitted in binary form inside the variable length data vector of the message, and used subsequently as the input for the receiving end parsing algorithm. The actual address part of the message is transmitted as a single character "/", denoting a compressed message, and at the same time ensuring backwards compatibility with the current OSC specification.

The internal server implementation of SuperCollider [6] utilizes a related method which is, however, incompatible with OSC and only supported by a few toolkit implementations. The technique proposed in this paper is fully OSC compliant, and therefore usable from any environment already supporting OSC. For example, a Pure Data [7] (Pd) patch running on the unmodified legacy OSC externals is able to take advantage of the compressed transmission stream, and base its computational logic on parsing the passed integer token. Another benefit of backwards compatibility is that the proposed method may be used concurrently with the standard OSC messaging mechanism. For instance, languages such as TUIO [8] may transmit their well-defined address strings as integer tokens, while still retaining user-extensibility: the receiving end parsing algorithm can easily differentiate between the compressed messages – which are transmitted using "/" as the address string – from the standard uncompressed OSC messages transmitted with longer address strings.

Figure 2 shows the communication sequence of the previous example in the proposed streamlined form. The user is interacting with a universal OSC controller application, which has a slider widget in its graphical user interface. During the initial setup phase, the user links the slider with the synthesizer parameter ("/LPF/1/cutoff"). In response, the controller application requests string-to-integer mapping information from the receiving end, and associates the slider with the supplied token value. The transformation from the URL-style address strings into the integer form tokens is thus carried out discreetly behind the scenes – the user of the system still views the address space of the receiving end in the intuitive string-form notation. Moreover, this procedure needs to be performed only once per session, as all slider movements are thereafter transmitted using the integer-based parameter addressing method.

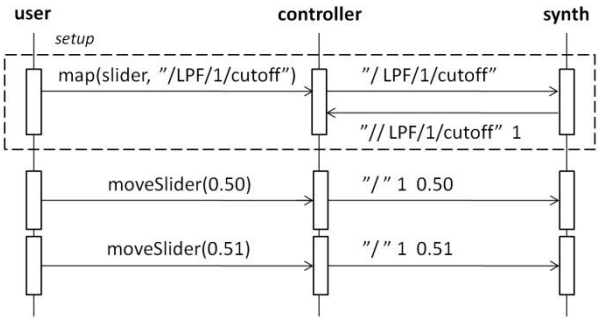

Figure 2. Proposed OSC communication sequence, showing one of the alternative mapping mechanisms within the dashed rectangle.

The reduction amount of the transmitted data depends on the complexity of the receiver defined namespace[1]. The reduction ratio R between the proposed technique and the standard OSC implementation is given by expression $R = 1/\mathrm{ceil}[0.25(L+1)]$, where L is the length of the address string of the standard OSC implementation. The reduction in the processing complexity of the parser implementation depends on the parsing algorithm, e.g., on the complexity of the hashing function. For a straightforward string comparison approach, the proposed method is substantially more efficient, as it has up to L times less comparisons than the standard OSC implementation, not counting the time wasted by non-matching string comparisons.

OSC also supports a wildcard-based parameter addressing mode, which enables control of multiple parameter values using a single message. The proposed method supports this by treating the integer token as a bitmask. To interpret the semantics of the integer parameter at the receiving end, the address string of the bitmask-based message is transmitted as "/?" instead of "/" used in the normal one-to-one messages.

As shown in Figures 1 and 2, the cost of the improved efficiency is the more complicated setup phase. This is required because the controller needs to know the mapping between the string and the integer form representations. We considered two alternative mapping mechanisms: shared dictionaries [3] and request-reply mapping, as discussed in the following section.

[1] It is not uncommon to have address strings that are even longer than 20 characters.

2.1 Mapping Mechanisms

In the shared dictionary mapping mechanism, the receiver publishes its entire address space dictionary as a shared resource. This can be done by using a text-formatted file, where each line defines a single address string – integer pair, separated by a comma. Resource-constrained controller devices with fixed functionality may also opt to hard-code the transmitted integer tokens and leave the file-based mapping interpretation to the receiving end.

The setup phase of the shared dictionary mapping mechanism consists of searching the user-supplied address string from the first column of the file, and once it is found, reading the corresponding integer token value from the second column. The URL of the mapping file may be defined as a TXT record entry of a Zeroconf-based service discovery process [9]. If such technology is unavailable, the URL may be entered by the user during the setup phase, or be simply hard-coded in the controller code.

In the request-reply mapping mechanism (see Figure 2) the controller sends a request to the receiver, which replies with the integer token of the controller-supplied address string. The address string of the reply consists of the requested string prefixed with a slash. This allows non-blocking implementation at the controller end, because the controller is then able to associate the reply in an asynchronous manner.

Although this mechanism requires more complicated implementation than the shared dictionary-based alternative, it has the additional benefit of supporting the wildcard enhanced address string – bitmask mapping mode: the receiver is responsible of resolving the wildcard formatted address string into the returned bitmask value.

Following the best practices described in [3], the request-reply mechanism calls for a TCP-based connection between the controller and the receiver. Because the receiver may already use a TCP port for normal parameter updates, it needs to recognize a GET request from the normal OSC parameter updates, i.e., SET messages. For this, we propose that GET messages are sent with an empty data vector, so that the receiver end can simply test the number of arguments in the received message, and if it is zero, interpret the address string as a GET request. A similar approach has also been utilized in [10].

TCP connection enables also request-reply –based shared dictionary access: the controller may send the reserved pattern "`//*`" as the address string, which the receiver recognizes and replies by dumping the contents of the mapping file back to the controller.

2.2 Topological Considerations

The proposed method works in both distributed peer-to-peer and centralized network configurations. In the former approach the controller and the receiver are in direct connection with each other, as shown in Figure 2.

The centralized hub-based topology requires a dedicated service in the network, into which the receivers register, and into which the controllers send their parameter updates. The hub is responsible for redirecting the received update messages to the registered receivers. Since the IP address:port of the controller-induced SET message is already pointing to the hub itself, the address string of the SET message should be prefixed with the name of the destination. This name should be the one that the receiver supplied when registering with the central OSC service.

The benefit of the centralized solution is that the address string – integer token mapping requests may be handled in a single location, simplifying the receiver end implementations. Another benefit is that the central hub may act as a router, which can map controller-centric parameter spaces (e.g., /finger/1) into the receiver end parameters, even providing simple transformations for the data values as is demonstrated in [11]. The hub may also be used to define OSC processing chains, where the outputs of distributed OSC modules are connected into the inputs of other OSC modules. For example, a gesture recognition module might receive raw data from a multi-touch controller, turn it into higher level gestural tokens, which are successively routed into another OSC processing module. However, the implementation of the centralized service is outside of the scope of this paper and left for future work. Instead, we will next look at three widely used OSC libraries, and describe how they can adapt the proposed address compression technique.

3. IMPLEMENTATIONS

Existing OSC applications are often built using third-party OSC toolkits. Accordingly it makes sense to look at how these commonly-used libraries need to be adjusted in order to benefit from the proposed technique. The existing applications can then adapt to the new method with minimal reimplementation efforts. This section looks at three open source implementations, and describes the required steps for their modification. The source code for these modifications is available at the accompanying website of this paper [12].

3.1 Implementation of the Proposed Technique

The proposed technique is implemented as a singleton *Streamliner* class, which holds two hashing tables for the string – integer mappings. The tables have identical content, but are indexed either with string or integer values. This enables fast conversion into the compressed representation (string-to-integer), or back to the standard OSC form (integer-to-string). The Streamliner class has two public methods for making these mapping conversions, and one public method for reading a file into the hashing tables.

Additionally, the message class implementation of the OSC toolkit needs to be subclassed by overriding the constructor in order to map the supplied address string into the integer token (using the Streamliner class). The constructor also inserts the mapped token into the argu-

ment vector of the OSC message, and sets the address string of the address-compressed message to "/".

The receiver code does not usually require additional modifications at the library level, because the toolkits are natively capable of extracting the first integer argument from the argument list of the received OSC message.

3.2 Processing and oscP5

Processing [13] is a Java-based environment for building interactive audiovisual applications. Its OSC implementation is based on the object oriented oscP5 library [14], which is straightforward to modify. The OscMessage class was subclassed with OscMessage2, and a singleton OscStreamliner class was implemented in Java. To take the advantage of the proposed method, existing applications simply need to define their OSC message objects as instances of OscMessage2 instead of OscMessage, as shown in Listing 1.

```
osc2 = new OscStreamliner("map.txt");
…
OscMessage2 msg = new OscMessage2("/lpf/cutoff");
msg.add( 0.50 );
oscP5.send(msg, destination);
```

Listing 1. Sending compressed messages in oscP5.

The oscP5 library routes all received OSC messages into the oscEvent(…) handler, which can then perform the parsing based on integer arithmetic, as shown in Listing 2. The received integer token can be converted back into the string form presentation, if so desired, using the OscStreamliner method osc2.address(iaddr). The OscPlugin mechanism of oscP5 was not addressed in this work.

```
void oscEvent(OscMessage msg) {
  int iaddr = msg.get(0).intValue();
  switch (iaddr) {
    case 1: …
```

Listing 2. Parsing compressed messages in oscP5.

3.3 oscpack

The oscpack library is a "set of C++ classes for packing and unpacking OSC packets" [15]. The required modifications included augmenting the library with the osc::Streamliner class, and modifying the code of a) BeginMessage class with a new constructor, taking an integer token argument, and b) streaming this token into the argument list of the OSC message at the end of the OutboundPacketStream << (BeginMessage) handler. Listing 3 shows an example of the streamlined oscpack sending procedure.

Although it is possible to hide the string-to-integer transformation of the address pattern entirely inside the BeginMessage class, as was done in the oscP5 adaptation, the form described below is more efficient in terms of CPU load.

```
osc::Streamliner osc2("map.txt");
…
osc::int32 token = osc2.iaddress("/lpf/1/cutoff");
p << osc::BeginBundleImmediate
  << osc::BeginMessage( token ) << (float)0.50
  …
```

Listing 3. Sending compressed messages in oscpack.

The oscpack ReceivedMessage class supports the non-standard SuperCollider address patterns. In order to add additional support for the proposed method, the class was augmented with a method to extract the integer token from the received message. The parsing is simplified into the form shown in Listing 4. As in oscP5, the address pattern may be regenerated from the received integer token using osc2.address(iaddr) invocation, if needed.

```
try {
  osc::int32 token = m.iaddress();
  switch (token) {
    case 1: …
```

Listing 4. Parsing compressed messages in oscpack.

3.4 Pd and OSCx externals

The OSC implementation (OSCx) of Pd consists of sendOSC, dumpOSC and OSCroute externals, and of a static code library containing the actual OSC implementation. To retrofit the proposed method, the sendOSC external was augmented with two functions (i.e., one for generating the mapping dictionary, and another for the actual sending routine). In addition, the *setmap* message was added to the interface to initiate the setup phase mapping dictionary construction. This message has to be invoked from the Pd patch before the actual control session is started (see Figure 3).

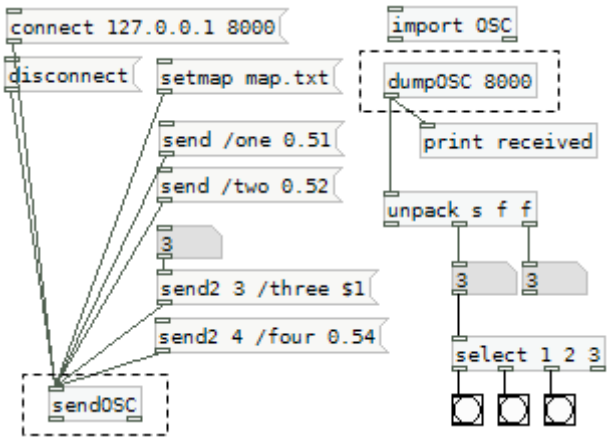

Figure 3. Streamlined Pd implementation. The OSC terminals are highlighted with dashed rectangles.

In the initial retrofit, the setmap message was the only modification visible to the user, as he still sends the OSC parameter updates using the original OSCx *send* message objects. However, because the external has to parse

the address string parameter of the send message for each invocation, a *send2* message was added to the interface as well. The parameters of send2 are identical to those of send, except that the parameter list is prefixed with an additional integer, which is unique to all send2 message boxes in the particular Pd patch (it does not need to match with the actually transmitted token value). The sendOSC external uses the prefixing integer to associate the message box instance with the actually transmitted integer token. This relieves the external from repeated string-form parsing actions, resorting to a simple integer-to-integer lookup for each sent message.

The parsing of the received OSC messages can be accomplished using the standard Pd *select* object. Thus, the more inefficient OSCroute becomes obsolete in this context.

4. EXAMPLE APPLICATION

The practical applicability of the proposed method is demonstrated by the following transparent example – wherein a touch screen is used to toggle playback of a selection of loops and apply some simple filtering. This demo uses the Pd implementation as described in Section 3.4 to both transform and distribute performance cues sent via a multi-touch application running on the Apple iPad.

In addition to the enhanced CPU efficiency, this approach reduces network traffic and hence the possibility of lost packets – which can especially cause concern in a live performance context that features wireless interfaces. In situations where the wireless transmission takes place on a mobile device, power consumption can also be reduced.

The example application can be conceptually split into three stages – the input stage, transformation stage, and the output stage, as shown in Figure 4.

Input stage: The iPad is running TUIOpad [16] – an open-source application which sends multi-touch data formatted using the TUIO protocol. This data is transmitted via a wireless connection to a laptop running a simple Processing [13] sketch, which extracts useful high-level information such as the number of fingers in contact and recognizes gestural cues. Relevant data is passed forward using typical hierarchical OSC namespaces (e.g., /touch/1).

Transformation stage: The resulting high-level messages are received by Pd, which is running a patch similar to the one illustrated in Figure 3. The augmented *sendOSC* function associates the incoming messages with a unique integer token according to the data in the mapping file. The data is now compressed – the new mapping procedure eliminates the need to repeatedly send address patterns.

Output stage: The compressed OSC messages control several looping buffers and band-pass filters in Pd. In addition to the audio control, a second Processing sketch generates a visual representation of the data. Processing is relieved of the burden of performing a string comparison every time an OSC message is received – a simple integer-recognition function will suffice. The visual accompaniment demonstrates the relatively concise data being parsed, in comparison with the lengthy address patterns that would otherwise be necessary.

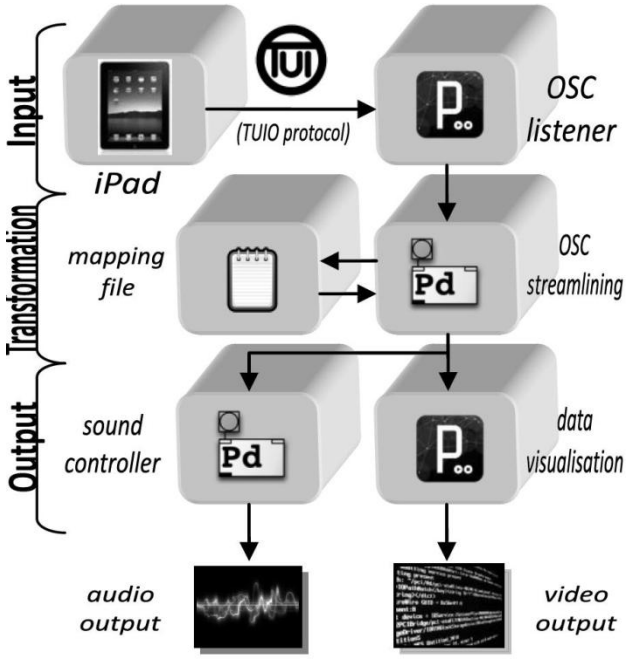

Figure 4. The data flow in the example application.

In this example, the proposed method has been demonstrated entirely within Pd in order to provide a lucid explanation. While performance issues are not necessarily of great concern in this kind of system, the benefits of implementing OSC compression at the 'input stage' – e.g., upon a mobile device sending compressed OSC with hardcoded integer tokens – is worthy of further investigation.

5. DISCUSSION

The proposed technique reduces the amount of transmitted data, simplifies parsing, and is straightforward to implement in existing OSC libraries and OSC compliant environments, as was demonstrated above. However, looking at the examples given in this paper, one might initially think that the technique flattens the intuitive tree-form address space of OSC into a continuous range of integers. This is not the case, because the receiving end – which defines how the integer tokens are mapped into the address strings – may construct the integer address space in a way similar to the IP addresses (e.g., 192.168.1.1). The 32-bit integer token can thus be interpreted hierarchically in a tree-form structure, and the widths of the bit fields are completely application-configurable. The hierarchical interpretation allows also wildcard-based mapping of the addresses, albeit consequently, it defines also a limit to the expansion of the address space (which is of no concern in the string-based representation). Since the

integer token is transmitted inside the data vector, and is therefore typed, it is possible to overcome this limitation by using 64-bit integers as address tokens instead of the 32-bit entity described earlier.

The downside of the proposed approach is that the message interpretation requires mapping information, and therefore the stateless presentation of OSC is not fully preserved. However, content formats with fixed address spaces may describe their a priori mapping definitions already at the specification level, which makes the messages self-contained, albeit still not truly stateless.

Section 2.1 discussed briefly the means of interpreting the received OSC message either as a SET or a GET request. Naturally, the interpretation should not be based on the amount of arguments in the message, but rather, the command verb should be included in the header part of the OSC message. Because RESTful paradigm has proved to be successful in many internet-based applications, the next major OSC specification update might consider redesigning the header part of the message accordingly. Based on our observations, a sufficient minimum set of verbs would consist of SET, GET, ADD and DELETE, possibly augmented with PUBLISH and SUBSCRIBE.

On the other hand, Zeroconf appears to be a viable solution for the service discovery and name resolution processes. In addition to working in a peer-to-peer fashion, it offers means of acquiring IP addresses without dedicated DHCP servers, which might be advantageous in distributed OSC setups. The TXT record field of the multicast DNS can also be used to describe the schema of the service, which, as discussed in Section 2.1, can be utilized to describe the string-to-integer mapping of the proposed technique.

XML-based schema language would offer advantages because it is extensible, standardized, and supported by a wide variety of libraries and related techniques (such as XPath, for example). However, since XML-based document parsing is a resource intensive process, a lighter format, such as JSON or OSC itself might prove to be more attractive for resource-constrained devices.

6. CONCLUSION

This paper proposed a technique that improves the efficiency of OSC-based communication. It decouples the user interface and the transmission layers of the protocol by mapping the string-form address representation into a corresponding integer form, which is then used in the compressed transmission stage. Three commonly used open source libraries were patched to take advantage of the proposed technique, and its practical applicability was demonstrated with an interactive audiovisual application. Finally, the paper concluded by providing suggestions for adapting certain established IP-based techniques within the OSC specification.

The OSC community is encouraged to explore the proposed technique in custom applications and OSC toolkits. We believe that the technique improves the efficiency of OSC communication protocol in a transparent manner, but given the diversity of OSC-based application scenarios, its full potential is revealed after it is supported by a wider installation base. The source code for the discussed implementations is available at [12].

Acknowledgments

This work has been supported by the Academy of Finland (project no. 122815) and the National University of Ireland Maynooth (John and Pat Hume Scholarship).

7. REFERENCES

[1] M. Wright and A. Freed, "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers," in *Proc. Int. Computer Music Conf.* (ICMC'97), Thessaloniki, Greece, Sept. 25-30, 1997.

[2] *The Open Sound Control 1.0 Specification*, 2002. http://opensoundcontrol.org/spec-1_0 (accessed 19.5.2011)

[3] A. Freed and A. Schmeder, "Features and Future for Open Sound Control version 1.1 for NIME," in *Proc. 9th Int. Conf. New Interfaces for Musical Expression* (NIME 2009), Pittsburgh, PA, USA, June 4-6, 2009.

[4] A. Fraietta, "Open Sound Control: Constraints and Limitations," in *Proc. 8th Int. Conf. New Interfaces for Musical Expression* (NIME 2008), Genova, Italy, June 5-7, 2008.

[5] C. Liu and P. Albitz, "*DNS and BIND*," 5th ed., O'Reilly Media, 2006.

[6] J. McCartney, "SuperCollider: a New Real Time Synthesis Language," in *Proc. Int. Computer Music Conf.* (ICMC 1996), Hong Kong, China, Aug. 19-24, 1996.

[7] M. Puckette, "*The Theory and Technique of Electronic Music*," World Scientific Press, River Edge, NJ, USA, 2007.

[8] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, "TUIO - A Protocol for Table-Top Tangible User Interfaces," in *Proc. 6th Int. Workshop on Gesture in Human-Computer Interaction and Simulation* (GW 2005), Vannes, France, 2005.

[9] D. Steinberg and S. Cheshire, "*Zero Configuration Networking: The Definitive Guide*," O'Reilly Media, 2005.

[10] *oscit Homepage*, http://lubyk.org/en/software/oscit (accessed 19.5.2011).

[11] J. Malloch, S. Sinclair, and M. Wanderley, "A Network-Based Framework for Collaborative Development and Performance of Digital Musical Instruments," *Lecture Notes in Computer Science*, Springer, 2008.

[12] *Accompanying webpage of this paper* http://www.acoustics.hut.fi/go/smc2011-osc

[13] *Processing Homepage*, http://processing.org/ (accessed 19.5.2011).

[14] *oscP5 Homepage*, http://www.sojamo.de/libraries/oscP5/ (accessed 19.5.2011).

[15] *oscpack Homepage*, http://code.google.com/p/oscpack/ (accessed 19.5.2011).

[16] *TUIOpad Homepage*, http://code.google.com/p/tuiopad/ (accessed 19.5.2011).