Converting NCL Documents to Smix and Fixing Their Semantics and Interpretation in the Process

Guilherme F. Lima
glima@inf.puc-rio.br
PUC-Rio, Rio de Janeiro, Brazil

Roberto Gerson de Albuquerque Azevedo
razevedo@inf.puc-rio.br
PUC-Rio, Rio de Janeiro, Brazil

Sérgio Colcher
colcher@inf.puc-rio.br
PUC-Rio, Rio de Janeiro, Brazil

Edward Hermann Haeusler
hermann@inf.puc-rio.br
PUC-Rio, Rio de Janeiro, Brazil

ABSTRACT

In this paper, we present the conversion of NCL to Smix and discuss its main implications. NCL is a declarative language for the specification of interactive multimedia presentations which was adopted by the ITU-T H.761 recommendation for interoperable IPTV services. Smix is a synchronous domain-specific language with a similar purpose, but with a simpler and more precise semantics. By implementing NCL over Smix, we bring to the former the notions of reaction and execution instants, and with them some benefits. From a practical perspective, we fix the semantics of the converted documents, get a leaner NCL player (the Smix interpreter), and simplify further conversions. From a systems-design perspective, the structured conversion of NCL to Smix helps us tame the complexity of mapping the user-oriented constructs of NCL into the machine-oriented primitives that realize them as a multimedia presentation. In the paper, we present NCL and Smix, discuss related work on document conversion, and detail the conversion process and a prototype implementation.

CCS CONCEPTS

• **Applied computing → Format and notation**; • **Information systems** → *Multimedia content creation*;

GENERAL TERMS

Languages; design; theory

KEYWORDS

Document conversion; multimedia; synchronous language; NCL; Smix; DietNCL

ACM Reference format:
Guilherme F. Lima, Roberto Gerson de Albuquerque Azevedo, Sérgio Colcher, and Edward Hermann Haeusler. 2017. Converting NCL Documents to Smix and Fixing Their Semantics and Interpretation in the Process. In *Proceedings of WebMedia'17, October 17–20, 2017, Gramado, RS, Brazil.*, , 8 pages.
DOI: https://doi.org/10.1145/3126858.3126876

1 INTRODUCTION

In this paper, we describe the conversion of NCL documents to Smix programs and discuss how this process, at the same time, fixes the semantics of the converted documents and serves as a basis for their structured interpretation. NCL (Nested Context Language) is a declarative language for the description of interactive multimedia presentations. NCL version 3.0 is both the standard language for interactive applications in the Brazilian digital terrestrial TV system [1] and an ITU-T recommendation [13] for IPTV systems. Smix (Synchronous mixer) is a synchronous domain-specific language for the construction of interactive multimedia applications [18]. By calling Smix a synchronous language, we mean that its programs operate under the synchronous hypothesis [2, 3], i.e., that they behave as event-driven systems whose reactions to external events occur in discrete, instantaneous computational steps. There are practical and methodological advantages in implementing NCL over Smix.

From a practical perspective, the NCL-Smix conversion is advantageous for at least three reasons. First, the output programs inherit the properties of the Smix semantics: its synchronous operation, determinism, and guaranteed reaction-termination [18]. The intuitive event propagation model of NCL (depending on how the NCL specification is interpreted) is subject to dysynchrony, non-determinism, and infinite feedback-loops. The conversion rules proposed here attach meaning (derived from Smix) to the NCL constructs they are converting from, making their behavior unambiguous and fixing the intuitive event propagation model of NCL. Second, we get a novel and leaner NCL player, viz., the Smix interpreter, which realizes the intended semantics. And third, the conversion to Smix simplifies further conversions and enables direct interpretations. For instance, Smix programs can be directly interpreted as Lua coroutines [19], whose syntax and model of operation bear no resemblance to NCL.

From a methodological, systems-design perspective, the conversion from NCL to Smix is advantageous because it provides a structured approach to tackle the problem of transforming the high-level description of media objects and their temporal relationships, which comprises the NCL document, into the low-level digital signal-processing primitives and operations that produces the corresponding multimedia presentation. Such a structured, hierarchical decomposition of abstractions (in this case, languages) is a recurrent theme in computing-systems design in general, but not so common in the context of multimedia languages, whose specifications and implementations tend to be monolithic. (We discuss exceptions in Section 2.)

WebMedia'17, October 17–20, 2017, Gramado, RS, Brazil.

To implement NCL over Smix, we adopted the following series of conversion steps:

$$\text{NCL} \xrightarrow{1} \text{Raw NCL} \xrightarrow{2} \mu\text{NCL} \xrightarrow{3} \text{Plain Smix} \xrightarrow{4} \text{Smix} \xrightarrow{5} \text{DSP Dataflow}.$$

In step (1), we convert the input NCL document to a restricted but compatible format, called Raw NCL. In step (2), we reduce the Raw NCL document even further, converting it to a minimalist format, called Micro NCL (μNCL). In step (3), we convert the μNCL document to Plain Smix, which is a syntactically richer dialect of Smix. And in step (4), we macro-expand the Plain Smix program into the final Smix program. Each step in the series transforms a representation that is closer to humans into one that is closer to the machine. The last step in the series, step (5), stands for the actual interpretation of the output Smix program, which is translated at run-time by the Smix interpreter into a multimedia digital-signal processing dataflow (viz., a GStreamer pipeline [9]).

In this paper, we are mainly concerned with the step (3) of the above series and with the implications of the NCL-Smix conversion process as a whole. In Section 2, we discuss some related work on multimedia document conversion. In Section 3, we present the NCL and Smix languages, and the dialects μNCL and Plain Smix. In Section 4, we detail the conversion from μNCL to Plain Smix. In Section 5, we discuss the implications of the conversion from a semantical point of view. In Section 6, we detail the implementation of a prototype converter. And in Section 7, we draw our conclusions and point out future work.

2 RELATED WORK

The idea of a multi-layered multimedia framework with higher level languages targeting document authors and lower level languages targeting player implementors is not new. The architecture of the Extensible MPEG-4 Textual Format (XMT) [16, 22] framework, for instance, consists of two such layers (or languages), viz., XMT-A and XMT-O. The XMT-A language is an XML representation of the MPEG-4 Binary Format for Scenes (BIFS) [12], which is a low-level multimedia scene description format. The XMT-O language is based on SMIL [4] and offers a higher level representation of MPEG-4 features. To be presented, XMT-O documents are first translated to equivalent XMT-A documents, which are then translated to MPEG-4 BIFS or directly interpreted by the document formatter. There are proposals for the direct conversion of subsets of SMIL to subsets of MPEG-4 BIFS [24] and vice versa [14].

Recently, with the popularization of HTML5 [10], there has been a surge in proposals that use it as a target language. Many of these implement novel document formats or re-implement old formats by converting them statically or dynamically to HTML5, which ends up playing the role of a lower level representation format. The justification of such proposals is usually grounded on interoperability with the Web ecosystem. They are less concerned with an efficient simulation of the input documents, or with the conscious use of HTML5 as an intermediate step in their structured interpretation (which are both goals of our NCL-Smix conversion proposal).

As examples of works that propose the dynamic conversion of parts of SMIL and NCL to HTML5, we can cite TimeSheets.js [5] and WebNCL [21]. And as examples of works that do it statically, we can cite Kim et al. [15], for the conversion of MPEG-4 XMT to HTML5, and NCL4WEB [25], for the conversion of NCL to HTML5.

There are also works on the conversion of NCL and SMIL documents to formal models with the purpose of validating them. As examples of these we can cite Picinin et al. [23] and dos Santos et al. [7] for NCL, and Chung et al. [6] and Gaggi et al. [8] for SMIL. For obvious reasons, these works devote special attention to document behavior. Although we are also interested in formalizing document behavior, we do so by converting the input document to a model that is closer to it, and to one that fixes known problems of the intuitive semantics of NCL, as we discuss in Section 5. Finally, our proposal is especially concerned with real-time performance—the previous works on NCL validation use complex simulation models, whose real-time evaluation is impractical.

3 NCL AND SMIX

In this section, we present the NCL and Smix languages, and detail the dialects μNCL ($\subset$ NCL) and Plain Smix ($\supset$ Smix) which are, in effect, the source and target languages of the conversion process we describe in Section 4. The conversion process itself is defined inductively on the syntax of the input documents, which means that it transforms, inductively, XML strings ($\supset$ μNCL documents) into Lua tables ($\supset$ Plain Smix programs). As it would be cumbersome to use XML and Lua in the description of the conversion process, we introduce abstract versions of the concrete syntaxes of μNCL and Plain Smix. We use these abstract versions with the tacit assumption that given the appropriate parameters every abstract document or program can be easily instantiated in the concrete syntax.

3.1 (Micro) NCL

A Micro NCL (μNCL) document is an NCL document containing only contexts, media objects, anchors, properties, and links, and in which links are in a restricted, basic format. More precisely, a μNCL document is an NCL 3.0 Raw Profile [26] document whose link-connector pairs are in the first normal form (NF1) [20].

The NCL 3.0 Raw Profile (or Raw NCL) is a trimmed-down version of the full NCL [1] (EDTV Profile) which preserves the expressiveness of the full version and is at the same time compatible with it. As a result, every Raw NCL document is by definition a valid (full) NCL document, and for each such NCL document there is a Raw NCL document that produces the same presentation.

In NCL, either Raw or full, link specification consists of two parts: the connector, which is a template for the link, and the link itself, which is an instantiation of the connector. The first normal form (NF1) theorem for link-connector pairs in NCL [20] is an analytical result that establishes that for any NCL document D there is an equivalent document D' such that (i) each connector element in D' is referenced by exactly one link element, and (ii) all link-connector pairs of D' contain exactly one condition and at most two (sequential) actions.

By adopting the Raw profile plus NF1 in the definition of μNCL, we reduce considerably the complexity of the conversion to Smix, and we do it in a structured way: the preprocessing step now consists in converting the input NCL document to a Raw NCL document and, subsequently, in putting this Raw NCL document in NF1. The resulting μNCL document is then fed to the next step.

μNCL syntax and intuitive semantics. We now turn to the definition of μNCL. Its abstract syntax is given by the following grammar:

$$\begin{aligned}
U &::= \ \textbf{context}\, x\, SML\, \textbf{end} \\
S &::= \varepsilon \mid \textbf{port}\, x\ S_1 \\
M &::= \varepsilon \mid \textbf{media}\, x\ M_1 \mid U\, M_1 \\
L &::= \varepsilon \mid C, P\ \textbf{do}\ A\ L_1 \mid C, P\ \textbf{do}\ A_1, A_2\ L_1 \\
C &::= \textbf{onBegin}\, x \mid \textbf{onPause}\, x \mid \textbf{onResume}\, x \mid \textbf{onEnd}\, x \\
&\quad \mid \textbf{onAbort}\, x \mid \textbf{onSelect}\, x \mid \textbf{onSet}\, x.u \\
A &::= \textbf{start}\, x \mid \textbf{pause}\, x \mid \textbf{resume}\, x \mid \textbf{stop}\, x \mid \textbf{abort}\, x \\
&\quad \mid \textbf{select}\, x \mid \textbf{set}\, x.u \coloneqq e
\end{aligned}$$

A μNCL document is a string of the form **context** x SML **end**, where x is an identifier, S are zero or more ports, M are zero or more components (i.e., media objects or nested contexts), and L are zero or more links. Each **port** x establishes that component x shall be started when the port's parent context is started. And each link of the form C, P **do** A or C, P **do** A_1, A_2 establishes that whenever the event waited by condition C occurs and, simultaneously, predicate P evaluates to true, the events denoted by the actions $A_1, \ldots, A_n$ are to be generated one after another.

A μNCL predicate P (assessment statement in NCL terminology) is a propositional logic formula involving the state or property values of media objects. As μNCL predicates are almost identical to Smix predicates, we will neither detail their structure nor their mapping to Smix. For the same reason, we will omit the internal structure of media object declarations (**media** x) and the mapping of NCL properties into Smix properties. It suffices to say that media objects denote presentation atoms (texts, images, audio clips, video clips, etc.) and that their properties define the audiovisual characteristics of their presentation (e.g., property "transparency" determines the transparency applied to the visual samples of an object with visual representation). Besides properties, media objects may have anchors, which represent segments of object's content. We will defer the description of media-object anchors and their translation to Smix to Section 4.2.

In μNCL, as in full NCL, every media object has a presentation event (or interval) which may be in one of three possible states: "occurring", "paused", or "stopped" (the initial state). If the object's presentation event is in state occurring, then its content is being presented; if it is in state paused, its content is paused; and if it is in state stopped, the content is not being presented and the object's properties assume their initial values. The transitions between these presentation-event states are commanded by actions and trigger corresponding conditions. Table 1 shows the mappings between actions, presentation-event transitions, and triggered conditions.

Table 1: NCL presentation-event transitions.

Action	State transition	Condition triggered
start x	stopped→occurring	**onBegin** x
pause x	occurring→paused	**onPause** x
resume x	paused→occurring	**onResume** x
stop x	occurring/paused→stopped	**onEnd** x
abort x	occurring/paused→stopped	**onAbort** x

In addition to the presentation event, each media object defines a selection event and, for each of its properties, an attribution event. The former represents the selection of the object by the user (via key presses or pointer clicks) and the latter represents the attribution of a value to an object property. In full NCL, the selection and attribution events are analogous to the presentation event: both define three states and five associated action-condition pairs. For simplicity, and without loss of generality, in the abstract syntax of μNCL we will assume a single action-condition pair for each of these events, which behave as follows: action **select** x triggers condition **onSelect** x, and action **set** $x.u \coloneqq e$ sets the property u of x to the value of expression e and triggers condition **onSet** $x.u$.

Some μNCL actions can be thought as being generated implicitly by the environment (the NCL player). There are three cases in which such implicit actions are generated:

(1) When the document starts, an action **start** x is generated to its utmost context x. And when a context x starts it generates an action **start** y to each of its immediate child components y such that there is a port of the form **port** y in its port list. Thus, when a context starts, all components mapped by its ports are started.
(2) When media object x is selected by the user, an action **select** x is generated. (Contexts cannot be selected.)
(3) After the content of media object x is exhausted, an action **stop** x is generated. And when the last child component of context x is stopped, an action **stop** x is generated.

The document presentation terminates when the presentation event of its utmost context transitions to stopped, i.e., when all media objects and contexts that comprise the document are stopped.

Example. Consider the following μNCL document.

$$\begin{aligned}
&\textbf{context}\ x \\
&\quad \textbf{port}\ x_1 \quad \textbf{media}\ x_1 \quad \textbf{media}\ x_2 \quad \textbf{media}\ x_3 \\
&\quad \textbf{onBegin}\ x_1, \top\ \textbf{do}\ \textbf{start}\ x_2, \textbf{start}\ x_3 \\
&\quad \textbf{onEnd}\ x_2, \top\ \textbf{do}\ \textbf{pause}\ x_1 \\
&\quad \textbf{onSelect}\ x_3, \top\ \textbf{do}\ \textbf{set}\ x_3.\text{transparency} \coloneqq .5, \textbf{resume}\ x_1 \\
&\textbf{end}
\end{aligned}$$

The above document consists of a context x containing a port, three media objects, and three links. The port establishes that when the document (context x) starts, media object x_1 shall be started. In the links, symbol $\top$ denotes a predicate that is always true. Thus the first link establishes that whenever media object x_1 starts, objects x_2 and x_3 shall be started. The second link establishes that whenever media object x_2 stops, object x_1 shall be paused. Finally, the third link establishes that whenever media object x_3 is selected by the user, object x_3 shall have its transparency set to 50% and object x_1 shall be resumed.

Assuming that x_1, x_2, and x_3 are video objects, when the above document is started the three videos will be started: x_1 due to the port, and x_2 and x_3 due to the first link. The three videos will be reproduced until x_2 ends, i.e., its presentation event transitions from state occurring to state stopped. At this moment, the second link is activated and x_1 is paused (if it is not already stopped). Finally, during the whole presentation, if the user selects x_3 while x_3 is being presented, the third link is activated, x_3's transparency is set to the half of its natural value, and x_1 is resumed (if it was paused).

3.2 (Plain) Smix

A Smix program consists of a set of media object declarations together with a sequence of links. A media object is a presentation atom; it has a content, a state, a (playback) time, and a property table. The content is a sequence of audiovisual samples. The state is either "occurring", "paused", or "stopped". (In Smix, the states are associated with the media object as a whole, and not with its events as in NCL.) The object time is the number of clock ticks to which the object was exposed while in state occurring. And its property table is an associative array that maintains the value of its properties, which are analogous to NCL properties.

Smix media objects are manipulated by actions which are triples of the form

$$(\text{predicate} \mathrel{?} \text{target} : \text{argument}),$$

where the predicate is a propositional logic formula involving the state, time, or property values of media objects, the target specifies the operation—start ($\triangleright$), pause ($\shortparallel$), stop ($\square$), seek ($\triangleright\!\!\triangleright$), or set ($\circ$)—and main operand (media object or property) of the action, and the argument is an extra operand required by seek and set actions.

The execution of actions is conditioned by the validity of their predicates. Thus, to evaluate an action, the Smix interpreter first evaluates its predicate. If the predicate is false, the action is discarded; if it is true, the interpreter proceeds to execute the action: it evaluates the extra argument (if any) and tries to execute the specified operation with the given operands. When writing actions, we often omit the predicate, question mark, and parentheses when the predicate is tautological (always true). Table 2 shows some example Smix actions and their intended readings.

Table 2: Smix actions and their intended readings.

$(\top \mathrel{?} \triangleright x)$	$\approx$ *start x unconditionally (abbreviated as $\triangleright x$)*
$(\bot \mathrel{?} \triangleright x)$	$\approx$ *skip (do nothing, as $\bot$ is always false)*
$(\text{state}(y) = \triangleright \mathrel{?} \shortparallel x)$	$\approx$ *pause x if y occurring*
$(\text{time}(x) \geq 1 \wedge \text{time}(x) \leq 5 \mathrel{?} \square x)$	$\approx$ *stop x if its time is between 1 and 5 ticks*
$(\text{prop}(x, u) = 0 \vee \neg(\text{time}(x) > 1) \mathrel{?} \triangleright\!\!\triangleright x{:}\, 10)$	$\approx$ *seek x by 10 ticks if x.u is 0 or if x's time $\leq$ 1 tick*
$(\text{time}(x) = \text{time}(y) \mathrel{?} \circ x.u{:}\, \text{time}(x) \div 2)$	$\approx$ *set x.u to the half of x's time if this is equal y's time*

A Smix link is a synchrony relation of the form

$$a \longrightarrow a_1 a_2 \ldots a_n,$$

which establishes that whenever some action with target a is executed, actions a_1, a_2, …, a_n shall also be executed, in this order. Notice that on the left-hand side of the symbol $\longrightarrow$ we have an action target a (not a complete action), while on its right-hand side we have a nonempty sequence of complete actions (each consisting of predicate plus target plus argument).

Example. Consider the following Smix program.

$$\begin{aligned} \triangleright\lambda &\longrightarrow \triangleright x \\ \triangleright x &\longrightarrow \triangleright y \,\square z \\ \triangleright y &\longrightarrow \triangleright z \\ \square x &\longrightarrow \square \lambda \end{aligned}$$

This program is written in the abstract syntax of Smix, in which a program is represented by sequence of links and media object declarations are omitted. The above program has four links that operate on four media objects: the ordinary objects x, y, and z, and the implicit object lambda (λ) which stands for the program itself. The first link establishes that when the program starts, media object x shall be started. The second link establishes that whenever x starts, object y shall be started and object z shall be stopped. The third link establishes that whenever y starts, object z shall be started. And the fourth link establishes that when x stops the whole program shall be stopped.

To start the above program the interpreter generates an action $\triangleright\lambda$ (the bootstrap action), which activates the first link. This causes the interpreter to execute action $\triangleright x$, which starts media object x and activates the second link. This, in turn, causes the interpreter to execute two actions: (i) $\triangleright y$, which starts y and triggers the third link (which starts z), and (ii) $\square z$, which stops object z which has just been started. Thus, after the bootstrap action $\triangleright\lambda$, the sequence of actions $\triangleright x \triangleright y \triangleright z \square z$ is executed by the Smix interpreter and this leaves the program in a state where media objects λ, x, and y are occurring and object z is stopped. We call the whole process of propagating an action through the program links (and reaching a final state) a *reaction.*

Smix reactions have some interesting properties. First, they operate under the synchronous hypothesis, i.e., logical time does not pass while actions are being executed and propagated through links. (Clock ticks correspond to "seek by 1" actions generated periodically by the environment.) Second, links are evaluated in a depth-first fashion. If we look at the previous program as the adjacency list of a graph, the propagation of an action through its links becomes a depth-first traversal in the graph. And third, due to the way in which the Smix semantics is defined, such traversal is deterministic (there is no choice involved) and always terminates.

The Plain Smix macro set. Plain Smix is a set of macros defined on top of the pure Smix language. We use two of these macros in Section 4. The first one is the conditional-link macro, which behaves as a link whose activation depends on a predicate. (Note that pure Smix predicates are associated to actions, not links as in NCL.) In Plain Smix, a conditional link is written as

$$(a, p) \longrightarrow a_1 a_2 \ldots a_n$$

and it expands to the pair of pure Smix links

$$\begin{aligned} a &\longrightarrow (p \mathrel{?} \circ\lambda.u{:}\, \eta) \\ \circ\lambda.u &\longrightarrow a_1 a_2 \ldots a_n, \end{aligned}$$

where u is a novel dummy (innocuous) property of media object λ that does not occur in any of the other links of the program, and η stands for the null value. Using the above definition we can write links that resemble those of μNCL, whose activation is conditioned by some predicate p.

The other Plain Smix macro we use in Section 4 is the if-else action, which provides a restricted form of branching in action sequences. To define if-else actions, we first need to define pinned actions and limited iteration, which are pure Smix (not Plain Smix) constructs with no counterpart in NCL.

Pinned actions are actions that do not trigger links. We write $\mathring{\triangleright}$, $\mathring{\text{▯▯}}$, $\mathring{\square}$, $\mathring{\triangleright\!\triangleright}$, $\mathring{\circ}$ (with the pin above the action symbol) for the pinned version of ordinary Smix actions. Limited iteration is a construct used to execute a sequence of actions repeatedly within a reaction. We write

$$\{e * a_1 a_2 \ldots a_n\}$$

for exactly m repetitions of the sequence $a_1 a_2 \ldots a_n$, where m is the number to which expression e evaluates at the time the iteration construct is considered. If $m \leq 0$, then the iteration construct together with its content is discarded by the interpreter.

Plain Smix uses pinned actions and limited iteration to define an if-else action of the form

$$(p \mathbin{?} \mathring{a}_1 \mid \mathring{a}_2),$$

which establishes that if predicate p holds, then pinned action $\mathring{a}_1$ is executed; otherwise pinned action $\mathring{a}_2$ is executed. This macro expands to the following sequence of pure Smix actions

$$(\top \mathbin{?} \circ\lambda.u\colon -1)(p \mathbin{?} \circ\lambda.u\colon 1)\{\mathrm{prop}(\lambda, u) * \mathring{a}_1\}\{-1 * \mathrm{prop}(\lambda, u) * \mathring{a}_2\},$$

where u is a novel dummy property of media object λ. The above gymnastics is necessary to ensure that predicate p is evaluated only once prior to the execution of $\mathring{a}_1$ or $\mathring{a}_2$. (The naive solution "$(p \mathbin{?} \mathring{a}_1)(\neg p \mathbin{?} \mathring{a}_2)$" does not work as the execution of $\mathring{a}_1$ may cause $\neg p$ to evaluate to true.)

4 FROM μNCL TO PLAIN SMIX

We are now in a position to define the conversion of μNCL to Plain Smix; we will do so incrementally. In Section 4.1, we deal only with the conversion of μNCL documents consisting of a single context into Plain Smix. In Section 4.2, we discuss the conversion of NCL temporal anchors. And, in Section 4.3, we discuss how the single-context conversion can be generalized to arbitrary μNCL documents.

4.1 Single-context conversion

The mapping of a single-context μNCL document into a corresponding Plain Smix program is given by function h below, where ε denotes the empty string, and the definitions of $h(M)$, $h(P)$, $h(u)$, and $h(u, e)$, which denote, respectively, the translation of media-object declarations, predicates, property names, and expressions, are omitted. (As we said earlier, their translation is direct.)

Contexts, ports, and links

$$\begin{aligned}
h(\varepsilon) &= \varepsilon \\
h(\textbf{context}\, x\, SML\, \textbf{end}) &= h(S)\, h(M)\, h(L)\, \varphi \\
h(\textbf{port}\, x\, S) &= \triangleright\lambda \longrightarrow (\top \mathbin{?} \triangleright x)\, h(S) \\
h(C, P\ \textbf{do}\ A\, L) &= (h(C), h(P)) \longrightarrow h(A)\, h(L) \\
h(C, P\ \textbf{do}\ A_1, A_2\, L) &= (h(C), h(P)) \longrightarrow h(A_1)\, h(A_2)\, h(L)
\end{aligned}$$

Conditions

$$\begin{aligned}
h(\textbf{onBegin}\, x) &= \triangleright x \\
h(\textbf{onPause}\, x) &= \text{▯▯} x \\
h(\textbf{onResume}\, x) &= \circ x.r \\
h(\textbf{onEnd}\, x) &= \square x \\
h(\textbf{onAbort}\, x) &= \circ x.a \\
h(\textbf{onSelect}\, x) &= \circ x.\mathrm{input} \\
h(\textbf{onSet}\, x.u) &= \circ x.h(u)
\end{aligned}$$

Actions

$$\begin{aligned}
h(\textbf{start}\, x) &= (\top \mathbin{?} \triangleright x) \\
h(\textbf{pause}\, x) &= (\top \mathbin{?} \text{▯▯} x) \\
h(\textbf{resume}\, x) &= (\mathrm{state}(x) = \text{▯▯} \mathbin{?} \mathring{\circ} x.r_f\colon 1 \mid \mathring{\circ} x.r_f\colon 0) \\
&\ (\mathrm{prop}(x, r_f) = 1 \mathbin{?} \mathring{\triangleright} x) \\
&\ (\mathrm{prop}(x, r_f) = 1 \mathbin{?} \circ x.r\colon \eta) \\
h(\textbf{stop}\, x) &= (\top \mathbin{?} \square x) \\
h(\textbf{abort}\, x) &= (\mathrm{state}(x) \neq \square \mathbin{?} \mathring{\circ} x.a_f\colon 1 \mid \mathring{\circ} x.a_f\colon 0) \\
&\ (\mathrm{prop}(x, a_f) = 1 \mathbin{?} \mathring{\square} x) \\
&\ (\mathrm{prop}(x, a_f) = 1 \mathbin{?} \circ x.a\colon \eta) \\
h(\textbf{select}\, x) &= (\top \mathbin{?} \circ x.\mathrm{input}\colon \eta) \\
h(\textbf{set}\, x.u \coloneqq e) &= (\top \mathbin{?} \circ x.h(u)\colon h(u, e))
\end{aligned}$$

The conversion procedure implemented by function h above uses media object λ to represent the state of the whole single-context μNCL document. Each port in the input μNCL document becomes a link that starts the object mapped by the port when λ starts. And each link in the input μNCL document becomes a conditional Plain Smix link in the output program. The condition-action pairs **onBegin**/**start**, **onPause**/**pause**, **onEnd**/**stop**, and **onSet**/**set** of μNCL are translated directly to the corresponding Plain Smix targets and actions (viz., $\triangleright$, ▯▯, $\square$, $\circ$). The remaining conditions and actions are simulated via media-object property attributions.

The pair **onSelection**/**select** is simulated via the attribution of Smix property "input", which contains the last input data directed to the object. And the pairs **onResume**/**resume** and **onAbort**/**abort** are simulated via the attribution of the dummy properties r and a of the target object x. Thus, an attribution to $x.r$ means that object x was resumed, while an attribution to $x.a$ means that x was aborted. The translation of the **resume** and **abort** actions use the extra dummy properties $x.r_f$ and $x.a_f$ which are flags that make sure that the simulated actions behave as expected. (E.g., an object cannot be aborted twice in a row: if the first abort action is successfully executed, then the second abort must be discarded.)

In the definition of function h, symbol φ stands for the sequence of Smix links

$$\begin{aligned}
\square x_1 &\longrightarrow (\mathrm{state}(x_1) = \square \wedge \cdots \wedge \mathrm{state}(x_n) = \square \mathbin{?} \square\lambda) \\
\circ x_1.a &\longrightarrow (\mathrm{state}(x_1) = \square \wedge \cdots \wedge \mathrm{state}(x_n) = \square \mathbin{?} \square\lambda) \\
&\ \vdots \\
\square x_n &\longrightarrow (\mathrm{state}(x_1) = \square \wedge \cdots \wedge \mathrm{state}(x_n) = \square \mathbin{?} \square\lambda) \\
\circ x_n.a &\longrightarrow (\mathrm{state}(x_1) = \square \wedge \cdots \wedge \mathrm{state}(x_n) = \square \mathbin{?} \square\lambda),
\end{aligned}$$

where $x_1, \ldots, x_n$ are the identifiers of all media objects in the input single-context µNCL document. This sequence of links simulates the terminating condition of µNCL: whenever an object x_i is stopped or aborted, if all objects of the document are stopped, then its presentation terminates.

As a final remark, note that the number of links in the output Plain Smix program is $O(n)$, where n is the number of media objects, ports and links in the original document.

Example. If we apply function h to the example µNCL document of Section 3.1, we get the following Plain Smix program, where $\varphi(x_1, x_2, x_3)$ denotes the sequence of links that implement the termination condition of the original document.

$$\begin{aligned}
\triangleright\lambda &\longrightarrow (\top\,?\,\triangleright x_1)\\
(\triangleright x_1, \top) &\longrightarrow (\top\,?\,\triangleright x_2)(\top\,?\,\triangleright x_3)\\
(\square x_2, \top) &\longrightarrow (\top\,?\,\text{▯▯} x_1)\\
(\circ x_3.\text{input}, \top) &\longrightarrow (\top\,?\,\circ x_3.\text{transparency}\colon .5)\\
&\qquad (\text{state}(x_1) = \text{▯▯}\,?\,\mathring{\circ} x_1.r_f\colon 1 \mid \mathring{\circ} x_1.r_f\colon 0)\\
&\qquad (\text{prop}(x_1, r_f) = 1\,?\,\mathring{\triangleright} x_1)\\
&\qquad (\text{prop}(x_1, r_f) = 1\,?\,\circ x_1.r\colon \eta)\\
\varphi(x_1, x_2, x_3)&
\end{aligned}$$

In the concrete syntax of Smix, this program becomes a Lua table [11] such as the following, where the above tautological-conditional links are represented as ordinary links and Plain Smix actions are expanded.

```
{{x1={···}, x2={···}, x3={···}},  -- media descriptions
 {{'start', lambda},              -- link #1
   {true, 'start', 'x1'}},
 {{'start', 'x1'},                -- link #2
   {true, 'start', 'x2'},
   {true, 'start', 'x3'}},
 {{'stop', 'x2'},                 -- link #3
   {true, 'pause', 'x1'}},
 {{'set', 'x3', 'input'},         -- link #4
   -- action #4.1
   {true, 'set', 'x3', 'transparency', .5},
   -- action #4.2
   {true, 'set', lambda, 'u', -1},
   {function (m) return m.x1.state == 'paused' end,
          'set', lambda, 'u', 1},
   {'iter', function(m) return m[lambda].u end,
     {true, 'set', 'x1', 'r_f', 1, 'pinned'}},
   {'iter', function(m) return -1 * m[lambda].u end,
     {true, 'set', 'x1', 'r_f', 0, 'pinned'}},
   -- action #4.3
   {function(m) return m.x1.r_f == 1 end,
          'start', 'x1', nil, nil, 'pinned'},
   -- action #4.4
   {function(m) return m.x1.r_f == 1 end,
          'set', 'x1', 'r', nil}},
 ⋮
                                  -- φ(x1, x2, x3)
}
```

4.2 Simulating NCL temporal anchors

In NCL, a temporal anchor denotes a temporal segment of a media object presentation. Each temporal anchor defines a partially independent presentation event, whose state transitions can be manipulated by ordinary conditions and actions. More specifically, in µNCL-like notation, an action **start** $x.w$, **pause** $x.w$, **resume** $x.w$, **stop** $x.w$, or **abort** $x.w$, respectively, starts, pauses, resumes, stops, and aborts the presentation event of temporal anchor w of media object x, and, consequently, triggers condition **onBegin** $x.w$, **onPause** $x.w$, **onResume** $x.w$, **onEnd** $x.w$, or **onAbort** $x.w$. Each anchor w defines a begin-time w_b and an end-time w_e that when reached trigger the implicit generation of corresponding **start** $x.w$ and **stop** $x.w$ actions by the environment (NCL player).

There are two uses for temporal anchors in NCL. The first use is to schedule a sequence of actions to execute when the object's presentation reaches a particular time, which can be either the begin time or end time of the anchor. In µNCL, this is accomplished by links of the form

$$\begin{gathered}
\textbf{onBegin}\, x.w, P \;\textbf{do}\; A_1, \ldots, A_n, \quad \text{or}\\
\textbf{onEnd}\, x.w, P \;\textbf{do}\; A_1, \ldots, A_n,
\end{gathered}$$

which can be translated to Plain Smix as, respectively,

$$\begin{aligned}
&(\triangleright\!\!\triangleright x, \text{time}(x) = w_b) \longrightarrow h(A_1), \ldots, h(A_n), \quad \text{and}\\
&(\triangleright\!\!\triangleright x, \text{time}(x) = w_e) \longrightarrow h(A_1), \ldots, h(A_n),
\end{aligned}$$

where w_b and w_e denote the begin and end time of anchor w of x, and h is the conversion function defined in Section 4.1.

The second use for temporal anchors in NCL is to request the presentation of a segment of the object's content, i.e., to start the object's presentation from the anchor begin-time and stop it when the anchor end-time is reached. In µNCL, this is done via a link of the form

$$C, P \;\textbf{do}\; A_1, \ldots, A_i, \textbf{start}\, x.w, A_{i+2}, \ldots, A_n,$$

which can be translated to the Plain Smix links

$$\begin{aligned}
(h(C), h(P)) &\longrightarrow h(A_1) \ldots h(A_i)(\text{state}(x) = \square\,?\,\circ x.w'\colon \eta)\\
&\qquad h(A_{i+2}) \ldots h(A_n)\\
\circ x.w' &\longrightarrow (\top\,?\,\mathring{\triangleright} x)(\top\,?\,\triangleright\!\!\triangleright x\colon w_b)\\
(\triangleright\!\!\triangleright x, \text{time}(x) = w_e) &\longrightarrow \square x,
\end{aligned}$$

where w_b and w_e denote the begin-time and end-time of anchor w of x, w' is a dummy property of target object x, and h is the conversion function of Section 4.1.

Although when considered in isolation the above translations are correct, in general, to avoid undesired interactions with other program links, it may be necessary to use dummy Smix timer objects (i.e., media objects without content) to represent the temporal anchors of an object. In this case, instead of operating directly on the object, the previous translations would operate on the timer that represents the anchor. Timers objects are also used to simulate in Smix the attributes *delay* and *explicitDur* of NCL. The former specifies that an action should take effect only after a certain amount of time, and the latter establishes an explicit duration to the object's presentation.

4.3 Simulating NCL contexts

An NCL context combines into a group a sequence of ports, a set of properties, a set of components (media objects or other contexts), and a sequence of links. The context itself is a self-contained module that interacts with the environment, i.e., external components or the NCL player (in the case of the utmost context), only through its ports. Each port exposes an internal component to the environment, allowing it to be manipulated by the environment. Thus once an internal component is mapped by a context port, the environment can submit actions (**start**, **stop**, **pause**, etc.) to it or listen for its conditions (**onBegin**, **onEnd**, **onPause**, etc.).

Besides interfacing with the context ports, the environment can also address the context as a unit. For instance, if x is a context and if the environment submits an action **start** x to it, then this action is propagated to all components x_i such that there is a port of the form **port** x_i in context x. The exact behavior of external actions over the context unit depends on the action type and on the state of the context presentation event. Every context maintains a presentation event and, for each of its properties, an attribution event—these events are analogous to those of media objects. (Contexts do not have a selection event, as they cannot be selected.)

The translation procedure of μNCL contexts to Smix is a generalization of that defined by function h of Section 4.1. In the generalized version, instead of using the Smix media object λ to represent the whole program, we represent each context—which from the point of view of its components *is* the whole program—by a dummy timer object that acts as a proxy for the context events. Similarly, we simulate context properties and its ports via dummy properties of the proxy timer object.

5 THE INDUCED SEMANTICS

The semantics induced by the NCL-Smix conversion fixes three problems of the intuitive (informal) semantics of NCL. The first one has to do with the representation of logical time within documents. The intuitive semantics of NCL treats logical time as something whose representation and manipulation can be influenced by physical phenomena, such as processing or communication delays, which are unpredictable and implementation dependent. Because of this, it is hard (if not impossible) to predict how the language constructs affect time or are affected by it.

To make matters concrete, consider an NCL document D containing n media objects, each of which with a temporal anchor whose begin time is 2s, and such that as soon as D starts, all n media objects are started. In principle, all n objects should be started simultaneously at instant 0s, their clocks should advance at the same rate, and all should have their anchors activated simultaneously at instant 2s. In practice, it may be even conceivable that that can happen for a small n. As n gets larger, however, each object starts to perceive a slightly different global time and becomes dyssynchronized in relation to the other objects. This happens because the actual code that process each object takes a small but significant time to execute—and since in the intuitive semantics of NCL the notion of logical time is tied to the actual running time, logical time passes while objects are being processed, so delays accumulate and instances that are executed later experience a greater time skew.

Smix avoids this problem by adopting the synchronous hypothesis: its program reactions are conceptually instantaneous, which means that logical time does not pass while the reaction is being computed. So, if we repeat the previous experiment with Smix, after the first reaction, which by definition terminates instantly, all n objects will be started at instant 0s. And after two times the number of ticks ("seek by 1" actions) corresponding to a second, say t, the playback time of all n objects will be exactly the same: $2t$. The trick here is that a Smix program can only perform in real-time if it reacts fast enough, i.e., if the physical time consumed by each of its reactions does not exceed the period of its logical ticks.

The second problem with the intuitive semantics of NCL that is fixed by the NCL-Smix conversion is nondeterminism. In NCL, links (and "par" actions) are evaluated in an arbitrary order. This arbitrary choice together with the imprecise notion of logical time discussed previously can lead to nondeterministic behavior. Such behavior is undesirable as deterministic programs decompose better and are much easier to specify and analyze than nondeterministic ones [3]. In Smix, reactions are guaranteed to be deterministic: the same input action in the same state will always lead to the same final state—no choice is involved. Similarly, the same history of input events will always lead to the same presentation history.

The third and last problem of intuitive NCL that is fixed by the NCL-Smix conversion is the possibility of infinite feedback-loops in event propagation. For instance, a naive evaluation of the following μNCL link leads to an infinite loop:

$$\textbf{onBegin}\, x, \top \;\; \textbf{do} \;\; \textbf{stop}\, x, \textbf{start}\, x.$$

When the link is activated, object x is stopped and then started, which causes the same link to be activated. (Less trivial cycles are also possible.) Smix avoids such feedback-loops by pre-compiling the possible propagation paths in the program graph into imperative programs that are guaranteed to terminate and which implement all possible reactions in the original program [19]. A side benefit of this pre-compilation phase is that it simplifies the definition of procedures for program optimization.

6 IMPLEMENTATION

We implemented the conversion procedure of Section 4 in the DietNCL tool [17], which is an NCL processor program and library written in Lua [11]. The DietNCL tool transforms an input NCL document by pushing it through sequence of filters. Each filter is a Lua module that performs an orthogonal transformation. For instance, the first filter parses the input XML file (or string) into a Lua table that represents the document; subsequent filters operate over this table, and the last filter eventually serializes the table back to a resulting XML file (or string).

Smaller filters can be combined into larger ones. The complete conversion of a full NCL document to Plain Smix consists of three such larger filters connected in series: *to-raw*, *to-micro*, and *to-smix*. Figure 1 depicts the structure of this series.

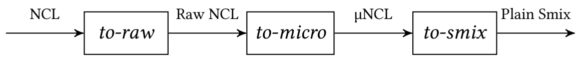

Figure 1: Filter series for converting NCL to Plain Smix.

The first filter, *to-raw*, takes a (full) NCL document and converts it to a Raw NCL document. Internally, this filter applies a series of simpler filters that remove redundant elements of the input document by rewriting them into equivalent combinations of more basic elements. The result of the *to-raw* filter is an equivalent Raw profile document containing only these basic elements.

The next filter in the series, *to-micro*, is concerned with the normalization of the input Raw NCL document. Internally, it applies in a series the five filters that correspond to the pre-normalization steps described in [20], and then applies a filter that manipulates the links and connectors in a way that the resulting link-connector pairs comply with the NF1, i.e., each consists of a simple condition plus predicate together with either a simple action or a sequence of exactly two simple actions.

The last filter in the series, *to-smix*, is a prototype implementation of the recursive translation procedure defined by function h of Section 4.1. The *to-smix* filter takes a μNCL document and converts it to a Lua table which is the corresponding Plain Smix program. This table can be either written to a file or string or can be used directly by the caller, in case DietNCL is being used as a library.

This last approach, using DietNCL as library, allows us to integrate the conversion directly into the Smix interpreter. By doing so we avoid one extra parsing step from the Smix program text to the corresponding the Lua table. Another possibility is the direct interpretation of this table. For instance, in [19] we describe the interpretation of Smix programs via Lua coroutines. Using this interpretation, we can write an *eval* function that takes as input an NCL document, uses the DietNCL library to obtain a Smix program, and then builds and returns a Lua coroutine that implements the corresponding presentation. To start this presentation, all one would have to do is start the coroutine.

7 CONCLUSION

In this paper, we described the structured conversion of NCL documents into Smix programs. We argued that this conversion is justified because it allows us to fix—correct and establish—the semantics of the input NCL documents, while producing a novel and leaner NCL player (the Smix interpreter) and enabling further conversions and interpretations.

The conversion *per se* was only possible (in theory) because Smix is sufficiently expressive to simulate any NCL construct. And it was only viable (in practice) because Smix is similar enough to NCL to avoid a complete simulation, and because we could define it over μNCL. By using the preexisting reductions—from NCL to Raw NCL and from this to μNCL (NF1)—we greatly reduced the number of combinations we had to deal with, and consequently a significant part of the complexity of the process.

Currently, we are improving the prototype implementation of the *to-smix* filter in the DietNCL tool and also working on the formalization and implementation of the coroutine interpretation of Smix (which can lead to an even simpler NCL player). On the specification side, a possible future work is to "bubble up" the semantics induced by Smix to full μNCL itself, i.e., to use the Smix conversion to formalize and give a synchronous interpretation to the semantics of μNCL, and consequently to full NCL.

REFERENCES

[1] ABNT 15606-2. *Digital Terrestrial TV — Data Coding and Transmission Specification for Digital Broadcasting — Part 2: Ginga-NCL for Fixed and Mobile Receivers: XML Application Language for Application Coding.* ABNT, São Paulo, 2007.

[2] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.

[3] G. Berry, P. Couronne, and G. Gonthier. Synchronous programming of reactive systems: An introduction to ESTEREL. In K. Fuchi and M. Nivat, editors, *Proceedings of the First Franco-Japanese Symposium on Programming of Future Generation Computers, Tokyo, Japan, 6–8 October, 1986*, pages 35–55, Amsterdam, 1988. North-Holland Publishing Company.

[4] D. Bulterman, J. Jansen, P. Cesar, S. Mullender, E. Hyche, M. DeMeglio, J. Quint, H. Kawamura, D. Weck, X. G. Pañeda, D. Melendi, S. Cruz-Lara, M. Hanclik, D. F. Zucker, and T. Michel. Synchronized multimedia integration language (SMIL 3.0). Recommendation, W3C, December 2008.

[5] F. Cazenave, V. Quint, and C. Roisin. Timesheets.Js: When SMIL meets HTML5 and CSS3. In *Proceedings of the 2011 ACM Symposium on Document Engineering*, DocEng '11, pages 43–52, New York, 2011. ACM.

[6] S. M. Chung and A. L. Pereira. Timed Petri net representation of SMIL. *IEEE Multimedia*, 12(1):64–72, 2005.

[7] J. dos Santos, C. Braga, and D. C. Muchaluat-Saade. A rewriting logic semantics for NCL. *Science of Computer Programming*, 107(C):64–92, September 2015.

[8] O. Gaggi and A. Bossi. Analysis and verification of SMIL documents. *Multimedia Systems*, 17(6):487–506, 2011.

[9] GStreamer. GStreamer: Open source multimedia framework. http://gstreamer.freedesktop.org. Accessed June 6, 2018.

[10] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. O'Connor, and S. Pfeiffer. HTML5: A vocabulary and associated APIs for HTML and XHTML. Recommendation, W3C, October 2014.

[11] R. Ierusalimschy. *Programming in Lua.* Lua.org, 4th edition, 2016.

[12] ISO/IEC 14496-11:2005. *Information Technology — Coding of Audio-Visual Objects — Part 11: Scene Description and Application Engine.* ISO, Geneva, 2005.

[13] ITU-T Recommendation H.761. *Nested Context Language (NCL) and Ginga-NCL.* ITU-T, Geneva, November 2014.

[14] H.-S. Kim. Conversion mechanism of XMT into SMIL in MPEG-4 system. In Y.-S. H and H.-J. Kim, editors, *Advances in Multimedia Information Processing — PCM 2005: 6th Pacific Rim Conference on Multimedia, Jeju Island, Korea, November 13–16, 2005, Proceedings, Part II*, pages 912–922. Springer, Heidelberg, 2005.

[15] H.-S. Kim and C. Dae-Jea. Conversion mechanism for MPEG-4 contents services on Web environment. In T.-J. Cham, J. Cai, C. Dorai, D. Rajan, T.-S. Chua, and L.-T. Chia, editors, *Advances in Multimedia Modeling: 13th International Multimedia Modeling Conference, MMM 2007, Singapore, January 9–12, 2007. Proceedings, Part II*, pages 627–634. Springer, Heidelberg, 2006.

[16] M. Kim, S. Wood, and L.-T. Cheok. Extensible MPEG-4 textual format (XMT). In *Proceedings of the 2000 ACM Workshops on Multimedia*, MULTIMEDIA '00, pages 71–74, New York, 2000. ACM.

[17] Lab. TeleMídia. DietNCL: A tool to remove the syntactic sugar from NCL documents. http://github.com/TeleMidia/DietNCL. Accessed June 6, 2018.

[18] G. F. Lima. *A synchronous virtual machine for multimedia presentations.* PhD thesis, Department of Informatics, PUC-Rio, Rio de Janeiro, 2015.

[19] G. F. Lima, C. Braga, and E. H. Haeusler. The Smix synchronous multimedia language: Operational semantics and coroutine implementation. In *Anais da 1a Escola de Informática Teórica e Métodos Formais (ETMF 2016), Natal, RN, Brazil, 22–23 November, 2016*, pages 145–154, Porto Alegre, 2016. SBC.

[20] G. F. Lima and L. F. G. Soares. Two normal forms for link-connector pairs in NCL 3.0. In *Proceedings of the 19th ACM Brazilian Symposium on Multimedia and the Web*, WebMedia '13, pages 201–204, New York, 2013. ACM.

[21] E. L. Melo, C. C. Viel, C. A. C. Teixeira, A. C. Rondon, D. de Paula Silva, D. G. Rodrigues, and E. C. Silva. WebNCL: A web-based presentation machine for multimedia documents. In *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web*, WebMedia '12, pages 403–410, New York, 2012. ACM.

[22] F. C. Pereira and T. Ebrahimi. *The MPEG-4 Book.* Prentice Hall PTR, Upper Saddle River, NJ, 2002.

[23] D. Picinin, J.-M. Farines, and C. Koliver. An approach to verify live NCL applications. In *Proceedings of the 18th ACM Brazilian Symposium on Multimedia and the Web*, WebMedia '12, pages 223–232, New York, 2012. ACM.

[24] B. Shao, L. M. Velazquez, N. Scaringella, N. Singh, and M. Mattavelli. SMIL to MPEG-4 BIFS conversion. In *Second International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution*, AXMEDIS 06, pages 77–84, December 2006.

[25] E. C. O. Silva, J. A. F. dos Santos, and D. C. Muchaluat-Saade. NCL4WEB: Translating NCL applications to HTML5 Web pages. In *Proceedings of the 2013 ACM Symposium on Document Engineering*, DocEng '13, pages 253–262, New York, 2013. ACM.

[26] L. F. G. Soares and G. F. Lima. The NCL handbook. Monographs in computer science, Informatics Department, PUC-Rio, Rio de Janeiro, 2013.