Nsync - A Constraint Based Toolkit for Multimedia

Brian Bailey and Joseph A. Konstan
Department of Computer Science
University of Minnesota

Nsync - A Constraint Based Toolkit for Multimedia*

Brian Bailey
Joseph A. Konstan
Department of Computer Science
University of Minnesota
{bailey, konstan}@cs.umn.edu
http://www.cs.umn.edu/Research/GIMME

Abstract

Nsync (pronounced ‘in-sync’) is a declarative multimedia synchronization toolkit, implemented entirely in Tcl, designed to ease the complexity of designing innovative, interactive multimedia applications. Nsync does not represent a new multimedia synchronization model; rather, it provides a set of building blocks in the form of temporal and non-temporal constraints useful for specifying both the synchronization and interaction properties of an application. Nsync depends only on the logical time system provided by the Berkeley Continuous Media Toolkit, thus allowing any application using a similar notion of time to also benefit from the Nsync constraint mechanism. Nsync presents a new and powerful environment for rapid development of highly interactive multimedia applications.

1 Introduction

A multimedia application can be partitioned along three axes:

- *Content*. The text, graphics, images, audio, or video used within the application.
- *Synchronization*. The temporal ordering of the content.
- *Interaction*. The control the user exercises over both content and synchronization.

As Figure 1 shows, the complexity in building a multimedia application increases as the desire for content, synchronization, and interaction increases. Most multimedia toolkits, such as the Berkeley Continuous Media Toolkit [10], have primarily focused on providing content as opposed to synchronization or interaction. Previous efforts to provide synchronization support for applications have resulted in the development of numerous synchronization models [6, 9, 12]. These models can typically be categorized as:

- *Timeline.* Defines actions such as the starting or stopping of a media object to occur at a specific time.
- *Hierarchic*. Provides two operators, “parallel” and “serial”, which can be applied to the endpoints of different media objects.
- *Reference point*. Synchronization points are defined within media objects which may inhibit or cause other media objects to playout.
- *Event based*. Applications express interest in system events, such as the starting or stopping of a media object, and are notified when those events occur.

As expected, each of these models has its own strengths and weaknesses, and none provide a complete set of synchronization abstractions (see [8] for a good overview and comparison of these models). Modeling the interaction properties of an application has received attention from a few systems such as [4], but in general has not been addressed.

2 Background

Because our work primarily focuses on providing support for the synchronization and interaction aspects of multimedia applications, we decided to use an existing media toolkit to provide the necessary content. The media toolkit chosen was the Continuous Media Toolkit (CMT) developed at the University of California Berkeley by the Plateau project [10]. CMT is a flexible low-

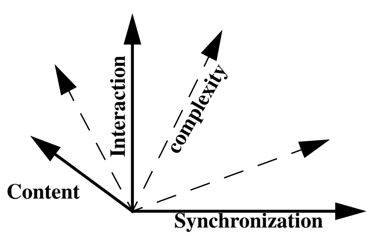

Figure 1: The Content, Interaction, and Synchronization axes of a multimedia application. Complexity is increased by traversing any of the axes outward.

* Supported by NSF IRI-94-10470 and a grant from DMRC.

level toolkit for building distributed continuous media (CM) applications. CMT provides support for a variety of video and audio formats, transport protocols, and hardware playout devices. Each of the CMT media objects can be dynamically loaded as a Tcl extension. These objects then provide an associated Tcl interface for creating, configuring and removing the object.

CMT also provides a distributed clock object called the Logical Time System (LTS) for media stream control. An LTS is shared by all the CM processes that map real time to "logical" time. Logical time has no start or end and can be thought of as an infinite timeline in both directions. An LTS object maintains three attributes: *value*, *speed* and *offset*. *Value* represents the current value of the clock, which we commonly refer to as *media time*, and which can be set to a new value in order to support random access within a media stream. *Speed* represents the ratio of the *media time* speed to real time. For example, a speed of 2.0 indicates that the media is being played at twice its natural rate and a speed of -1.0 indicates that it is being played in reverse at normal speed. *Offset* is a scalar that is used to complete the mapping between *media time* and real time:

$$\textit{Media-Time} = \textit{Speed} * \textit{Real-Time} + \textit{Offset}.$$

An LTS can be set by setting any two of {*media-time*, *speed*, *offset*}, and the third will be computed. See Figure 2 for LTS code examples.

CMT media objects are attached to an LTS which is used to provide the timing basis for the playout of individual media frames. An LTS, which from now on will be referred to as a clock object, has the following properties:

- Multiple media objects or streams can be attached to a single clock to achieve fine-grained synchronization; i.e., lip-sync quality.
- Multiple clocks can be created and attached to different media streams to support unsynchronized (or differently synchronized) playback, or to support other application objects that also need a notion of time; e.g., animation.
- *Speed* and *offset* do not change during normal playout without intervention by either the user or other system events.

```
(a) set clock [lts ""]
(b) $clock config -speed 1 -value 0
(c) $clock destroy
```

Figure 2: Tcl code examples for **(a)** creating, **(b)** configuring, and **(c)** destroying an LTS.

- Clocks can be perfectly synchronized by setting the speeds and offsets to identical values. Since *media time* is computed from real time, two clocks with the same *speed* and *offset* will always have the same *media time*.
- Clocks always progress in a piecewise linear fashion (see Figure 3 below).

Because of its piecewise linear property, the amount of time it will take a clock with its current *value*, *speed*, and *offset*, to reach a future time instant can be predicted. However, if any of the three attributes of a clock change, then this predicted value will no longer hold. Utilizing the predictable nature of a clock will be shown later, but for now it suffices to realize that a method for clock attribute change notification is needed. Together, the OAT [11] and TclProp [5] systems provide this functionality. OAT, or object attribute trace, provides a protocol for adding traces on new types of Tcl objects; e.g. clocks and their attributes. TclProp further builds on OAT to provide triggers and one-way constraints.

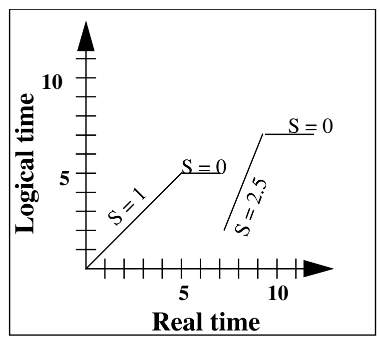

Figure 3: A graph of the CMT clock object. From real time 0 thru 5, logical time progresses at the same speed as real time, signified by speed (S) = 1. From real time 5 thru 7, S = 0, and logical time does not change. At real time 7, random access has been performed by setting the clock's value attribute back to logical time 2, and the speed is also set to 2.5. Logical time now progresses from logical time 2 at 2.5 times the rate of real time. At real time 9, S = 0, and logical time continues unchanged.

3 The Nsync Toolkit

With CMT providing the basic media stream support for applications, we focused our efforts on providing support for both the synchronization and interaction properties. The goal was to develop a simple methodology for defining a broad base of useful relationships involving clock objects (thus controlling the temporal layout of the media) and user interface events; e.g., button presses. This methodology has been embodied in the Nsync (pronounced 'in-sync') synchronization toolkit which has been implemented entirely in Tcl. At the core of Nsync is a declarative constraint language which supports each of the following:

- *Temporal constraint*. A Tcl expression consisting of constants, scalar variables, arithmetic operators, equality or inequality relationships, boolean connectives, and at least one clock object. Temporal constraints are useful for specifying the synchronization aspects of a multimedia application (see Figure 4a).
- *Non-temporal constraint*. A Tcl expression consisting of constants, scalar variables, arithmetic operators, equality or inequality relationships, and boolean connectives. Non-temporal constraints are useful in user interface development and therefore address the interaction component of multimedia applications (see Figure 4b).
- *Combination*s of temporal and non-temporal constraints. Both disjunction (||) and conjunction (&&) can be applied to any combination of temporal or non-temporal constraints. These combinations allow the synchronization and interaction issues to be simultaneously addressed (see Figure 4c).
- *Enforcement action*. A user-defined Tcl command invoked when the constraint expression becomes true (See Figure 4).

Examples of constraint specification in Nsync are given in Figure 4. Each of the constraints are specified using the following syntax:

`When` *expression action*

where `When` is the Tcl procedure name, *expression* is the constraint to be maintained, and *action* is the enforcement action. Semantically, the constraint in Figure 4a states that *whenever* `clock1` *becomes* greater than `clock2` plus the value in the `skew` variable OR *whenever* `clock2` *becomes* greater than `clock1` plus the value in the `skew` variable, set the value of `clock1` equal to the value of `clock2`. Because the attributes of either clock may be changed at any time; e.g., through temporal access controls, the constraint may need to be enforced (by calling the user-defined enforcement action) many times.

Several properties about Nsync constraints are notable:

- *Declarative*. Only the what is specified to the Nsync system without specifying any of the how. In other words, the constraints are described to the system and the system determines when to invoke the corresponding enforcement action.
- *Dynamic.* Nsync constraints can be created, deleted, or modified at run-time. For example, the `skew` variable from Figure 4a can be modified at run-time which will affect when the enforcement action is invoked.
- *Expressive*. Nsync constraints can be used to specify a wide variety of different constraint relationships.
- *Understandable*. Nsync constraints are very intuitive as they can simply be read from left to right. The English equivalent of a constraint can be stated by using the following grammatical template: "whenever *expression* becomes true, perform this *action*."

With the ability to explicitly combine temporal and non-temporal constraints, Nsync directly supports both the synchronization and interaction aspects of multimedia applications.

```
set action [list $clock1 config -value \[$clock2 cget -value \]]
When "($clock1 > $clock2 + \$skew) || ($clock2 > $clock1 + \$skew)" $action
```

(a)

```
When "(\$skew_desired == 1)" {set skew $current_skew_value}
```

(b)

```
set action [list $clock1 config -value \[$clock2 cget -value \]]
When "(\$skew_desired == 1) && (($clock1 > $clock2 + \$skew) || ($clock2 > $clock1 + \$skew))"
        $action
```

(c)

Figure 4: Tcl code examples for **(a)** Temporal constraint defining skew control **(b)** Non-temporal constraint to retrieve the skew value when desired, signaled by clicking a checkbox, and **(c)** a combination of a temporal and non-temporal constraint to enforce the skew relationship of **(a)**, to be within the limit obtained from **(b)**, but only when desired by the user. Skew control requires the logical time of two clocks to be within a specified value from one another.

4 Implementing Nsync with Tcl

The entire Nsync toolkit has been implemented using the Tcl language. Tcl was chosen as our implementation language for the following reasons:

- *Constraint specification is not time critical*. Most constraints will be specified long before they are actually enforced. Thus, the specification of the constraints is not time critical, however we do recognize that the enforcement of the constraints may be time critical.
- *Dynamic code evaluation*. By using the `subst` and `eval` commands, the Nsync constraint mechanism is very flexible. For instance, any valid Tcl command, whether it is a procedure call or the setting of a variable, can be used as the enforcement action of a specified constraint.
- *Existence of needed tools*. Nsync leverages existing tools such as the Berkeley Continuous Media Toolkit for basic stream support, OAT for object attribute trace support, and TclProp for both object change notification and non-temporal constraint specification.
- *Timed event queues*. Tcl already provides excellent support for timed event queues through the `after` command.
- *User interface components*. Nsync also provides an integrated set of Tk based mega-widgets such as vcr controls, jog shuttle controls, and random access bars.

The Nsync toolkit architecture is shown in Figure 5.

5 Nsync Implementation

Looking at the skew constraint shown in Figure 4a, it may not be obvious why neither the Tcl `eval` mechanism nor TclProp can be used to evaluate the constraint expression. The key reason is the inclusion of *media time*. *Media time* contains several properties that make either approach impractical:

- *Media time* is continuously changing, except when the clock speed is 0. In TclProp, formulas and triggers would need to be re-evaluated continuously which is impossible and impractical.
- *Media time* often should be compared using inequalities (such as whether a particular clock's *media time* is >= another clock's *media time*) which are not supported by TclProp.
- The truth value of the constraint expression is no longer just TRUE or FALSE. An additional value of WILL BECOME TRUE (WBT) is necessary and will be introduced in section 5.2. Obviously, neither the Tcl `eval` mechanism nor TclProp can produce this temporal truth value.

The Nsync implementation consists of four components:

- *Compiler*. Parses the constraint expression and converts it into a postfix expression stack for fast runtime evaluation.
- *Evaluator*. Determines the truth value of the constraint expression. If the expression evaluates to TRUE or WBT, then the Evaluator calls upon the Scheduler to invoke the enforcement action immediately (TRUE) or at the predicted time (WBT).
- *Change Monitor*. Watches the relevant scalar variables and clock attributes used in the constraint expression. If any of these change, the Scheduler is notified.
- *Scheduler*. Schedules the enforcement action to be invoked at the requested time.

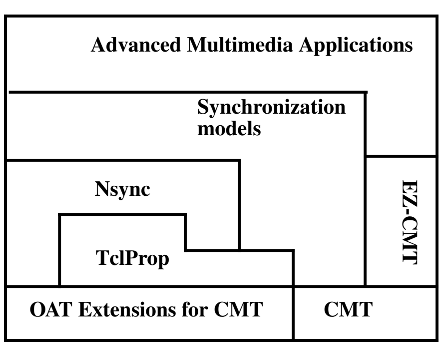

Figure 5: The Nsync toolkit architecture. Nsync leverages existing components such as CMT, OAT, and TclProp to provide synchronization and interaction support.

5.1 Compiler

The Compiler is invoked by calling the `When` command with two parameters. The first parameter represents the constraint expression and is treated as a Tcl string. The second parameter represents the enforcement action and is simply stored for later use by the Scheduler. Within the constraint expression, any attribute of a clock may need to be referenced and some method for clearly identifying the attribute needed to be developed. To address this issue, Nsync defines the notion of a *typed reference*. A typed reference is a string conforming to the following format:

{TYPE object attribute}

When the user requires an LTS attribute in the constraint expression; e.g., *speed*, the string *{LTS ltsname speed}*

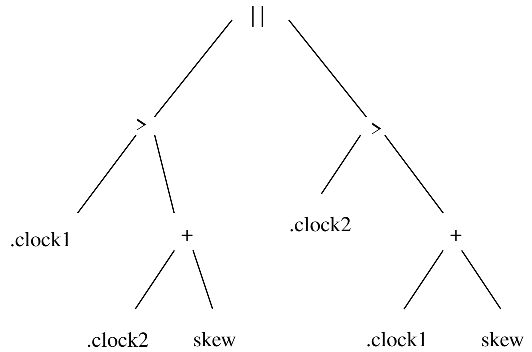

Figure 6a: The parse tree representation of the skew constraint example given in Figure 4a.

||
>
+
skew
.clock1
.clock2
>
+
skew
.clock2
.clock1

Figure 6b: The postfix expression stack for the parse tree shown in Figure 6a.

is specified[*]. Alternatively, the user may simply invoke an Nsync API which takes a clock variable and attribute, and returns the proper format.

To reduce the need for special quoting, the `When` command assumes all variable substitutions have been made prior to procedure invocation. However, if the user wants to pass in a scalar variable, and not its current value, thus deferring substitution until the constraint is evaluated, then a backslash should be placed before the dollar sign[†].

Once the `When` command is invoked, a lexical analyzer tokenizes the input string (constraint expression) and passes each token to the parser upon demand. The parser uses the tokens to build a parse tree according to the BNF language description[‡]. The parse tree is built using a combination of a Tcl array and several Tcl lists. Each parse tree node contains a value (representing the lexical token) and a list of elements, each of which is an index to another parse tree node. Once the constraint expression is successfully parsed, an equivalent postfix expression stack, used for efficient run-time evaluation, is created. Because of Tcl's built in support for list manipulation, converting the parse tree to an expression stack was extremely simple. See Figure 6a and 6b for the parse tree and stack representation of the skew constraint example in Figure 4a.

[*]The clock variables used in Figure 4 are actually typed references with the attribute being *value*, but are not shown for clarity.

[†]See the examples in Figure 4.

[‡]The BNF is available as part of the Nsync distribution available at http://www.cs.umn.edu/Research/GIMME

5.2 Evaluator

The Evaluator is responsible for determining the truth value of the constraint expression. With the addition of temporal variables, constraint expressions now have *three* possible values:

- FALSE. The constraint expression is currently false, and will stay false at least until a clock's attribute or scalar variable used within the expression is changed.
- TRUE. The constraint expression is currently true, and will stay true at least until a clock's attribute or scalar variable used within the expression is changed.
- WBT. The constraint expression is currently false, but will become true in a predictable amount of time given it current state.

We also recognize that WBT also has an inverse *Will Become False* (abbreviated WBF). However WBF has not been implemented for two reasons:

- The semantic meaning of the `When` command would become ambiguous. Will the enforcement action be invoked when the constraint expression becomes true or when it becomes false?
- Any temporal constraint which needs the notion of WBF can simply invert its relational operator and use WBT.

In order to evaluate the expression stack, the top element

Table 1: **Truth table for temporal AND**

AND	**TRUE**	**FALSE**	**WBTy**
TRUE	TRUE	FALSE	WBTy
FALSE	FALSE	FALSE	FALSE
WBTx	WBTx	FALSE	WBT max (x,y)

Table 2: **Truth table for temporal OR**

OR	**TRUE**	**FALSE**	**WBTy**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	WBTy
WBTx	TRUE	WBTx	WBT min(x,y)

is popped from the stack and each of its operands is recursively evaluated. Each operand is evaluated according to the following rules:

- Constants are promoted to a temporary clock object with a speed of zero, and with a value set equal to the constant.
- Scalar variables first have their value retrieved using the Tcl `subst` command. The variable value is then promoted to a temporary clock object in the same manner as constants.
- Arithmetic operators (+, -, *, /) are evaluated by applying the operator to the *value* attributes of the clock objects*. Because of constant and scalar variable promotion, each operand of the arithmetic operator is guaranteed to be a clock object.
- The inequality and equality operators (>, >=, <, <=, ==) work similar to the arithmetic operators, except when either of the clock object's speed is not equal to zero (each operand will be a clock object due to previous promotions). When the speed of each clock is zero, the clock object values are compared using the appropriate operator and TRUE or FALSE is returned. However, when one of the speeds is non-zero, then a prediction algorithm is invoked which may return TRUE, FALSE, or WBT along with a predicted time. The following code fragment represents the computation performed for the > operator between two clock objects. This code sample produces the correct result for any combination of actual media clocks and temporary clocks resulting from promotions.

```
set value1 [$clock1 cget -value]
set speed1 [$clock1 cget -speed]
set value2 [$clock2 cget -value]
set speed2 [$clock2 cget -speed]
if {$value1 > $value2} {
    #clock1 value is already > clock2 value
    return TRUE
} elseif {$speed1 <= $speed2} {
    #value1 < value2 and impossible for the
    #two clocks to cross
    return FALSE
} else {
    #value1 < value2, but clocks will cross,
    #predict that future time
    set prediction [expr (($value2 - $value1)
                     / ($speed1 - $speed2))]
    return [list WBT $prediction]
}
```

- The boolean AND (&&) and OR (||) binary operators are summarized by the truth tables given in Tables 1 and 2 above. As Table 1 shows, if both operands currently have the temporal boolean value WBT, the AND operator returns WBT along with the *later* (max) of the two predicted values because that is when *both* of the operands will be true. Similarly, as Table 2 shows, the OR operator will return WBT along with the *sooner* (min) of the two predicted values because that is the time when the *first* of the two operands will be true.

Every time the constraint expression stack is evaluated, the Evaluator applies the above rules to determine the status of the constraint expression. If the constraint expression evaluates to

- FALSE. No action is taken.
- TRUE. The Scheduler is called upon to invoke the enforcement action immediately.
- WBT. The Scheduler is called upon to invoke the enforcement action at the predicted time.

5.3 Change Monitor

As shown in Figure 3, logical time progresses at a pre-

*When applied to two clocks with non-zero speed, the result is undefined.

dictable rate until one of the attributes, *value*, *speed*, or *offset* is changed. If one these clock attributes or any other variable used within the constraint expression changes, then all expression stacks referencing either of these needs to be re-evaluated. TclProp watches each of the necessary clock attributes and variables, and notifies the Scheduler if any changes occur.

5.4 Scheduler

The Scheduler is responsible for invoking the constraint enforcement action at the future time predicted by the Evaluator. To perform this function, the Scheduler needs to maintain a timed event queue. Fortunately, Tcl already provides the required functionality with the `after` command. The Tcl `after` command arranges for an arbitrary Tcl command to be executed after a specified number of milliseconds have elapsed*. The command returns an identifier which can later be used to cancel the delayed command. To perform its task, the Scheduler simply calls the `after` command with the predicted time and enforcement action as parameters. Once the predicted amount of time has elapsed, the corresponding constraint expression will have just changed from FALSE to TRUE, and the enforcement action should, and will be invoked. However, if the Change Monitor notifies the Scheduler that a dependent clock attribute or variable has changed, then the Scheduler removes all appropriate `after` commands and notifies the Evaluator. The Evaluator then re-evaluates any expression stack referencing the changed clock attribute or variable and the whole process repeats itself.

6 Discussion

Although Tcl is in most respects a very simple language, it has proven capable of building a complex, yet efficient application. In summary, Tcl provided several mechanisms useful for the development of the Nsync multimedia synchronization toolkit:

- *Array and list support*. Used to quickly implement the constraint expression parse tree and equivalent postfix expression stack.
- *Dynamic code evaluation*. Provided great flexibility when specifying the enforcement action of each constraint.
- *Existing tools*. Reusing or extending existing tools such as OAT, TclProp, and CMT greatly increased our productivity.
- *Packages*. The Tcl package mechanism facilitated good source code modularity.
- *Rapid code development*. Nsync was originally implemented in less than 4 weeks of programming effort.
- *Timed event queues*. The Tcl `after` command provided an excellent interface for invoking delayed commands.

Nsync has been implemented in about 3,000 lines of Tcl code. The entire source distribution is available at http://www.cs.umn.edu/Research/GIMME

7 References

[1] Blakowski, G., J. Huebel, and U. Langrehr. "Tools for Specifying and Executing Synchronized Multimedia Presentations," *2nd Int'l Workshop on Network and Operating System Support for Digital Audio and Video*. 1991. Heidelberg, Germany.

[2] Blakowski, G. and R. Steinmetz. "A Media Synchronization Survey: Reference Model, Specification, and Case Studies," *IEEE J. Select. Areas Commun.*, Volume 14, no. 1, January 1996.

[3] Borning, A. and R. Duisberg. "Constraint-Based Tools for Building User Interfaces," *ACM Transactions on Graphics*, 5:4, Oct. 1986, pp. 345-374.

[4] Herlocker, J. and J. Konstan. "Tcl Commands as Media in a Distributed Multimedia Toolkit," P*roceedings of the 1995 Tcl/Tk Workshop* (Usenix Association).

[5] Iyengar, S. and J. Konstan. "TclProp: A Data-Propagation Formula Manager for Tcl and Tk," P*roceedings of the 1995 Tcl/Tk Workshop* (Usenix Association).

[6] Little, T.D.C. and A. Ghafoor. "Interval-Based Conceptual Models for Time-Dependent Multimedia Data," *IEEE Transactions on Knowledge and Data Engineering*, 1993. 5(4): p. 551-563.

[7] Ousterhout, J.K. "Tcl and the Tk Toolkit," Addison-Wesley Publishing Company, 1994.

[8] Pazandack, P., "Multimedia Language Constructs and Execution Environments for Next-Generation Interactive Applications," PhD Thesis. University of Minnesota, 1996.

[9] Rothermel, K. and T. Helbig. "Clock Hierarchies: An Abstraction for Grouping and Controlling Media Streams," *IEEE J. Select. Areas Commun.*, volume 14, no. 1, January 1996.

[10] Rowe, L. and B. Smith. "A Continuous Media Player," *Network and Operating Systems Support for Digital Audio and Video, Third Int'l Workshop Proceedings*. 1992.

*The `after` command does not guarantee any deterministic boundaries on the actual invocation time versus the asked for time.

[11] Safonov, A. "Extending Traces with OAT: an Object Attribute Trace package for Tcl/Tk," P*roceedings of the 1997 Tcl/Tk Workshop* (Usenix Association).

[12] Schnepf, J., J. Konstan, and D. Du. "Doing FLIPS: FLexible Interactive Presentation Synchronization," *IEEE J. Select. Areas Commun*., volume 14, no. 1, January 1996.