Timetree: A New Way for Representing Time

Hui Wang

We always use timeline[1] to describe time, but timeline unable to describe dynamic time. Because dynamic time is on changing, so timeline must be changing too, but it is impossible. Timeline includes layers, relationships existed between layers. If we changed a layer, the others need to be fixed too. Timeline can not make it up automatically, human being must take the work, therefore real-time-changing becomes impossible. It is no matter for film making, but internet[2] needs instant responding, timeline can not cover it. Here we show a new structure called timetree, it is an auto-balanced hierarch structure. Its structure always be complete during changing without help from human being. It is a challenge for making dynamic interactive contents on internet, timetree is born for it. We have tried timeline before[3] [4] now it is the turn of timetree.

Introduction

Timeline born with film industry. It looks naturally taking time as a linear flow. But today, internet asks for interactive dynamic contents. It needs time to do more, we need time to be scalable, controllable, independent. In other words, we need time to be structured. We are familiar with structured plane, for example, web page[5]. We are benefited from structured plane, so why not do the similar work with time to retrieve the same advantage? It does work, the answer is timetree.

Timetree is hierarch structure for describing time. We know each timeline represents a single time thread, comparatively each timetree includes multiple time threads as many as you wanted.

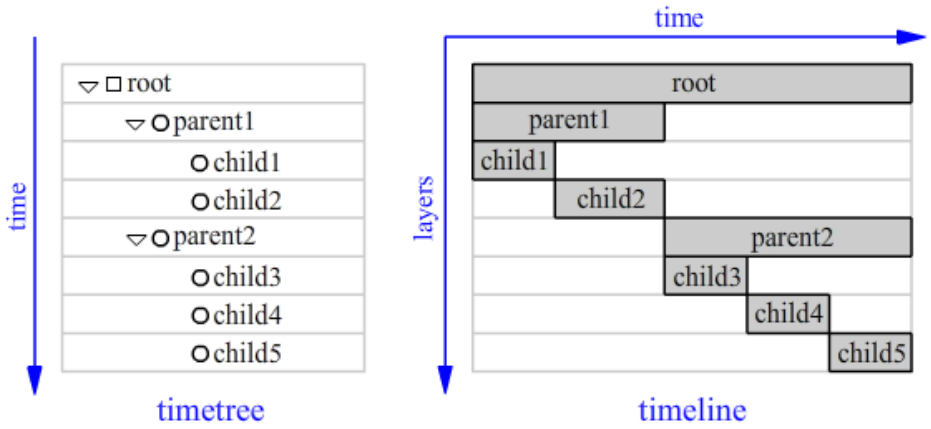
Each node of timetree is a container which contains a period of time, its time is not only a linear line but also a self-closed time closure. Timetree includes two kinds of node: Serials Node, and Parallel Node. Parallel Node focus on creates new time thread. Serials Node focus on creates details of thread which be created by Parallel Node.

It is law of timetree each node only allowed to include one kind of children at any time. That means, if a node contains a parallel child, all of its children must be Parallel Nodes too. A node can keep its children as Serials Nodes or Parallel Nodes at any time, but not both.

Single Time Thread

First, we discuss Single Time Thread (STT) of timetree. Timetree includes multiple time threads, here we focus on one of them. Normally STT is a branch of timetree led by a Parallel Node, but sometime STT includes the entire timetree if timetree only with single time thread. STT defined by a Parallel Node, so its root must be a Parallel Node,

the rests are all Serials Nodes. Essentially, STT is a structure for dividing time, it is similar to using a tree to divide a plane. The children always divide the time its parent owns, so the hierarchy structure allows us manipulating time in different scales. It looks more intuitive if we convert a STT to a timeline [Fig]. It looks like a structured timeline.



We using a circle to indicate a Serials Node, using a box to identify a Parallel Node.

Each parent node contains a period of time, the length is always equal to the sum of its children's timespan. We use "**L**" to represent the time length of a node, if the parent node contains children for count "**n**", the equation as following:

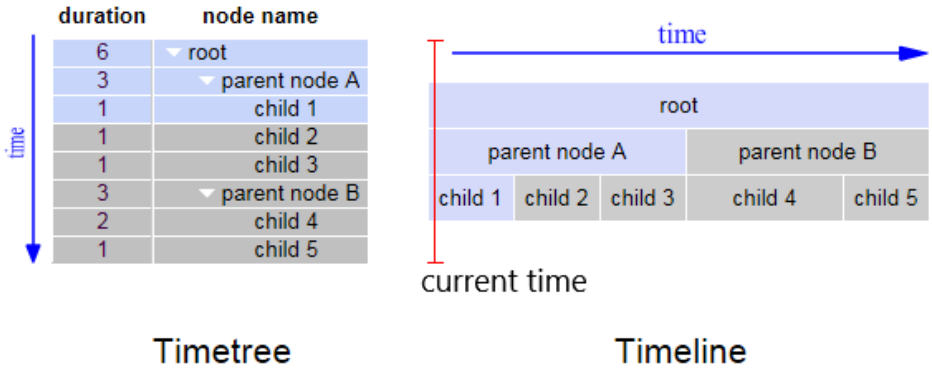
L(parent) = L(child 1) + L(child 2) + ... + L(child n).

The equation is dynamic, the time length of parent node is decided by its children. In order to keep the equation, the time length of parent node always being adjusted if we inserted a child in or removed a child away or changed the time length of a child. The way it works from bottom to top. Only leaf allowed to customize its time length. The time of branch is a variable, we unable to change it, because it depends on its children.

The timespan of child is always included in its parent's timespan. The children of same parent node share no time with each other.

The structure of STT is always balanced whatever we editing on it, it supports real time editing.

Second, we discuss how to play STT, convert it to a timeline makes it clearer [Fig].



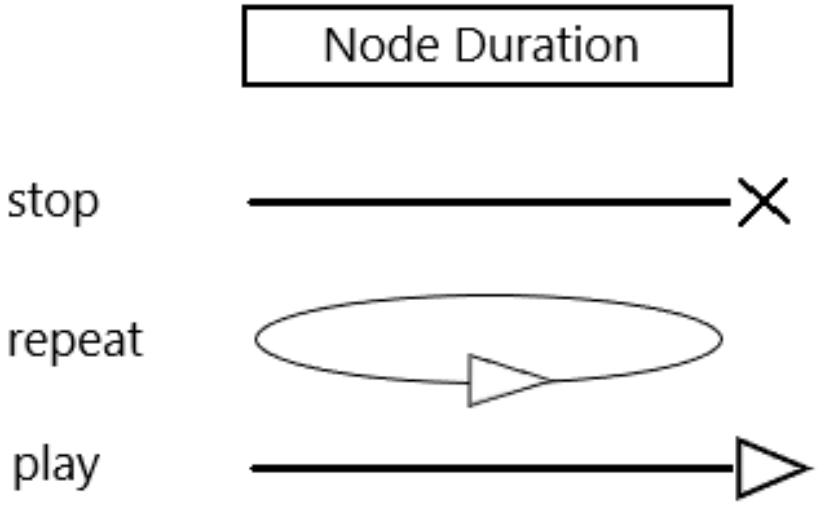
Timetree Timeline

STT owns its local time, a node of STT is visible if the current local time inside the period of its time, or it is invisible. Obviously if a node is visible, its parent node must be visible too. If it owns children, one of its children also be visible.

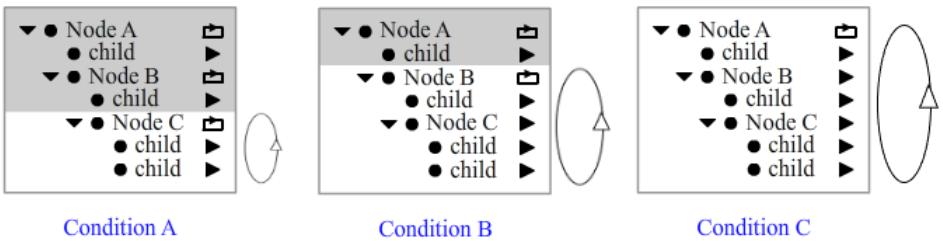
STT looks like a compressed timeline, playing STT is also similar to playing timeline, but STT is not a structured timeline. The difference is: Each node has chance to control the play, the controlment is taking place when the current time reaches the end line of node.

Each node keeps attribute of “playback”, it has three options: “stop”, “repeat”, “play”. The attribute decides how a node to play its self.

First, we discuss how to play a leaf. If the current time reached at the end line of the leaf, we call the leaf got play-completed. If the “playback” set to “stop”, the current time will stop at the end line of the leaf. If the “playback” set to “repeat”, the current time will go back to the start line of the leaf to restart the next cycle. And most importantly, if the “playback” set to “play”, the current time will go through the end line of leaf to play forward, just like play on timeline. In this condition, we say the node got play-out [Fig].



Second, if a node has children, it always shares the end line of time with its last child. In this condition, the node is considered as got play-completed only when its last child got play-out. The play order is always inside-out, similar to loop of programming [Fig]. The rest part is equal to leaf playing.



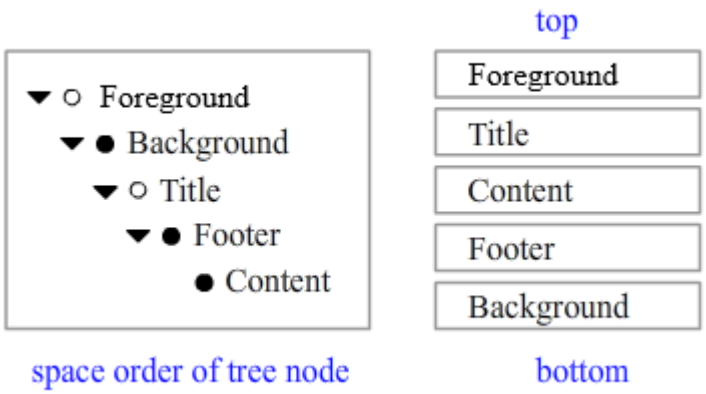
The model we discussed here is a simplified timetree. The real timetree, not only each STT, but also each branch, each node owns its local time whatever it is visible or not.

Let's look back the discussion here: STT is the plan for dividing time. We are benefited with the hierarchy structure for not only time assignment but also time play. Let each node at each level controlling time play is not only let us control time more precisely, but also takes STT as a structure for semantic usage. Animation is also programming, that is it.

Third, we discuss how to set space order on STT. Timeline owns layers for describing space order. Timetree only 1D, so we use the way called Binary-Method to set the space order between nodes. First, only visible nodes need to be considered with. Second, because STT only contains Serials Node (except root), so in fact only the relationship between ancestors and descendants needs to be considered. Here is the

method: We defined an attribute of “node-order” for each node, it has two options: “ahead of descendants”, “behind of descendants”. The ancestor always allowed to be outside of its descendants, so the space structure of STT looks like an onion [Fig]. If a node got its value of “node-order”, its space order also be confirmed whatever how changing the timetree it belongs to will be in future.

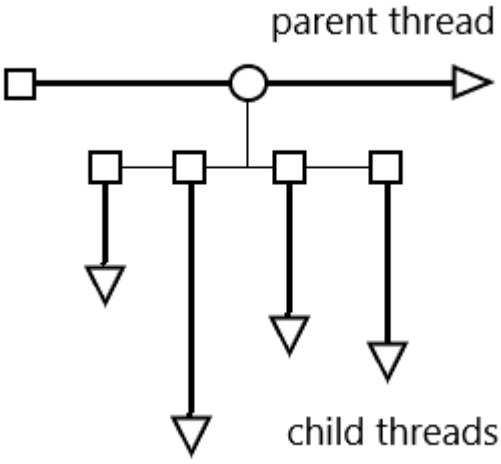
We using filled icon to indicate “behind of descendants”, using framed icon to identify “ahead of descendants”.



Yes, we lost plenty of flexibility which timeline owns. That is the cost we must pay, real-time-edit is not a free launch. The question is: Is it enough for requirement? For movie making, no, it is not. For internet, yes, it is.

Multiple Time Threads

It is useless if timetree only with STT, so we need Multiple Time Threads (MTT). In timetree, every thread created by its parent thread except the root-thread. The parent thread creates its child threads at a time point inside its duration, we use a node to represent the time point, the node is the parent node of all child threads. Each child thread defined by a Parallel Node [Fig].



The parent thread keeps on being suspended while its child threads on running, the parent thread will not play forward until its child threads got play-out. That means the duration of the parent node in fact is zero. Then we discuss the details of MTT.

First, no doubt the parent node with its parallel children must be visible or invisible simultaneously. Inside, each child thread is independent, owns its local time. No thread will get effected if we insert a thread in or remove a thread away or change a thread.

Second, we discuss the play of MTT. It is similar to STT, but still has some difference. Each node of timetree owns attribute of "synchronize", it works only the node owns parallel children. It has two options: "synchronize", "compete". If it set to "compete", the parent node got play-completed when one of its children got play-out. If it set to "synchronize", the parent node got play-completed when all of its children got play-out. The parent node got play-out if its "playback" set to "play" while it got play-completed, then the parent thread goes on. If the parent node no longer be visible, all of its child threads are invisible too.

Third, we discuss the space order of MTT. The Binary-Method is still available here, but now we need to consider the space order between parallel siblings. It is simple, their orders on timetree also determined the space orders for them.

Real time operation of timetree

Timetree supports some operations in real time. Some timetree parameters are customizable in real time. The local time of STT branch is customizable in real time. The node attribute of "playback" and "synchronize" are also changeable in real time. These operations already got tested.

Timetree also supports construction or destruction in real time. Includes change the time duration of leaf, insert a branch in timetree, or remove a branch away from timetree.

These operations have got tested in limited situation.

Practice: Time Object Model

Here we just describe the principle without involving details. We know web page is important part of internet. It is created by language HTML[6], its structure represented as Document Object Model (DOM)[7]. DOM is a hierarchy structure, similar to timetree. The difference between them is: timetree is the structure for time dividing, DOM is for plane dividing. But they can work together. The combination of them called Time Object Model (TOM). TOM represents a dynamic stream, rather than a static page.

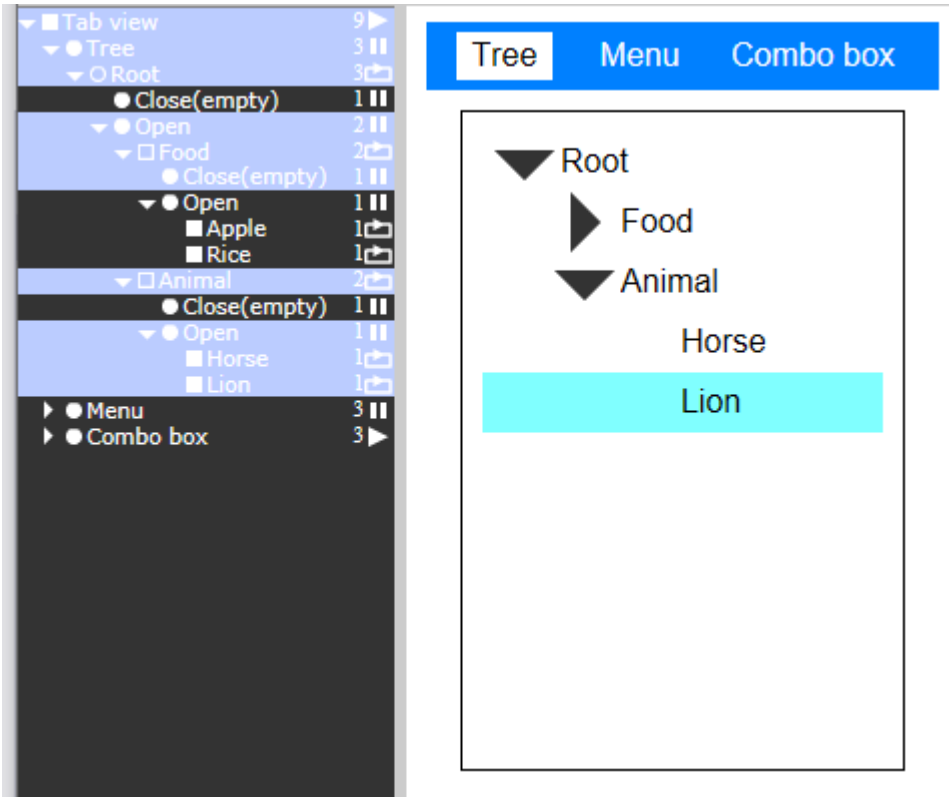
Although web pages we saw normally are dynamic, but that is the magic of script[8], DOM its self is static. That means the html elements already be there when web page got established, they always be there until web page got removed. Time is permanent to these elements. In order to change it, we need to embed a timetree in, then map each html element to one and only one timetree node. Make sure the element will be visible when the mapped node is visible, the element will be invisible when the mapped node is invisible. Then only we need to do is to play the timetree. If a node got visible, all elements mapped to it get to be inserted in DOM. If a node got invisible, all elements mapped to it get to be removed away from DOM. The principle is simple.

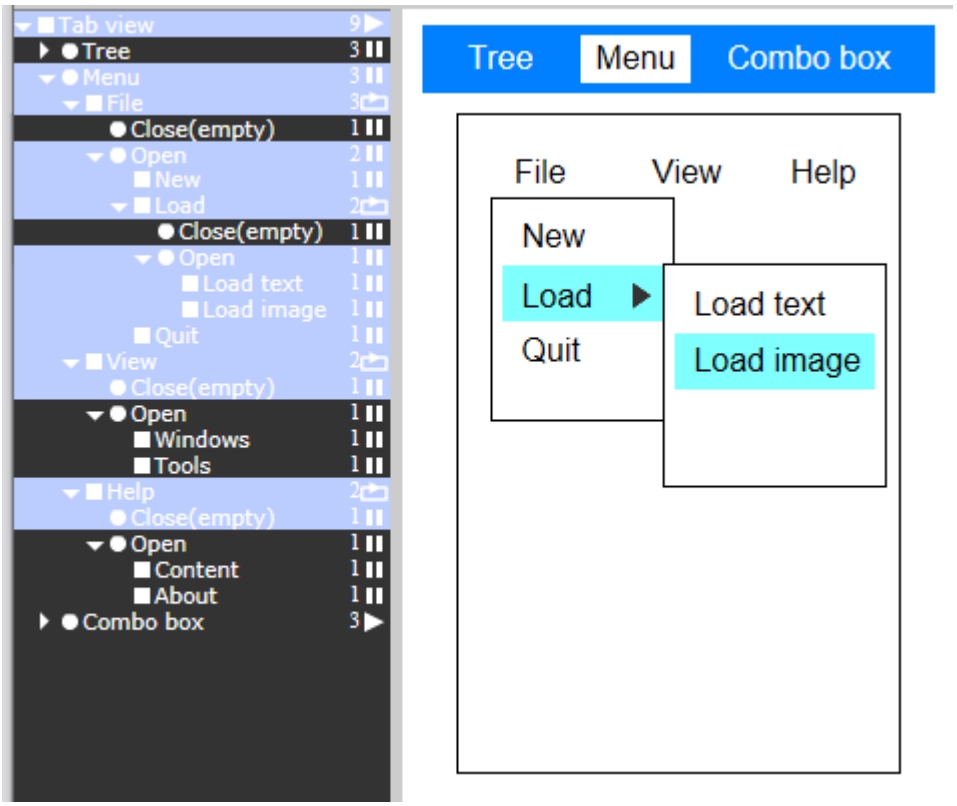
Still has a question, how to make sure the inserted in elements will be placed in right place? The answer is using “node-order”, we skip the details here.

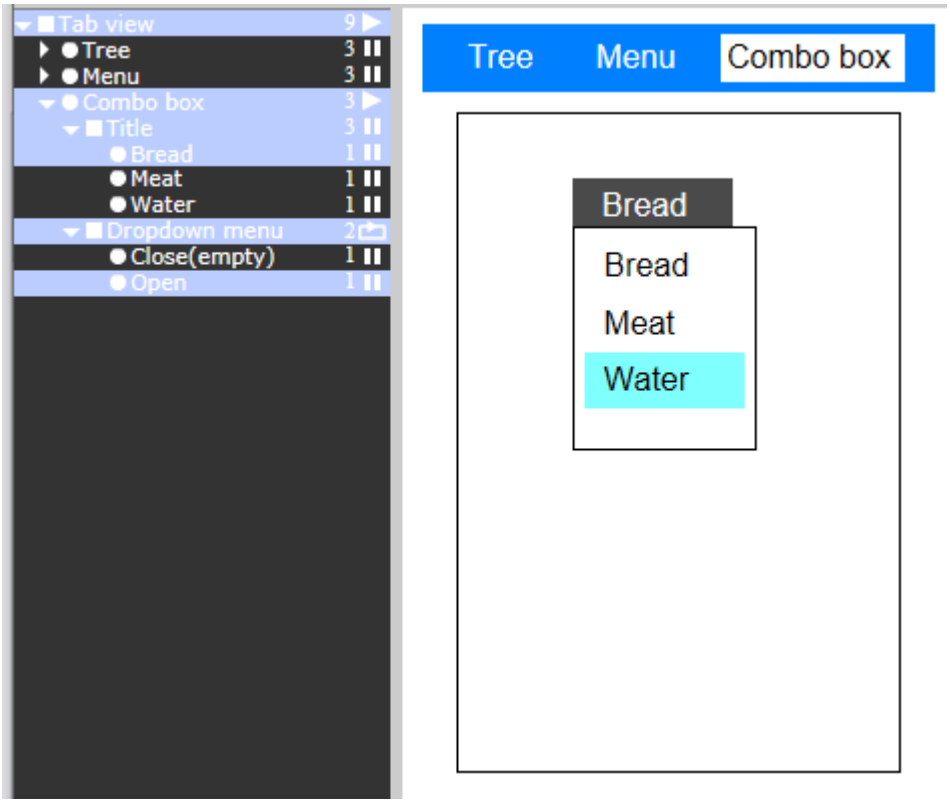
Sample of TOM

Normally we create UI controls by do programming[9], or use special tools[10], or use widgets[11]. Now we have a new way to make it. The following is the snapshot of a TOM based html document[12]. It represents some functional UI controls [Fig].

The left side is timetree panel. The right side is html document.


Tab view 9
Tree 3
Root 3
Close(empty) 1
Open 2
Food 2
Close(empty) 1
Open 1
Apple 1
Rice 1
Animal 2
Close(empty) 1
Open 1
Horse 1
Lion 1
Menu 3
Combo box 3
Tree
Menu
Combo box
Root
Food
Animal
Horse
Lion


Tab view 9
Tree 3
Menu 3
File 3
Close(empty) 1
Open 2
New 1
Load 2
Close(empty) 1
Open 1
Load text 1
Load image 1
Quit 1
View 2
Close(empty) 1
Open 1
Windows 1
Tools 1
Help 2
Close(empty) 1
Open 1
Content 1
About 1
Combo box 3
Tree
Menu
Combo box
File
View
Help
New
Load
Quit
Load text
Load image



Practice: Structured concurrency with timetree

Timetree is a good choice for organizing Structured Concurrency[13] [14] [15]. We get to briefly introduce the concept at below.

Programming is different than designing, here timetree is formed with texts. First, Serials Node is no longer needed to be represented because instructions are already ordered serials line. Second, we still need to point out Parallel Nodes. We use a parallel-block for wrapping parallel part, and the Parallel Nodes are inside it. For example, it looks maybe like this:

```
function parallelSample(paraA, paraB){
    var I;
    parallel(compete){
        thread 0: I = func01(paraA, paraB);    break;
        thread 1: I = func02(paraA, paraB);    break;
        thread 2: I = func03(paraA, paraB);    break;
        thread 3: I = func04(paraA, paraB);    break;
    }
    Return I;
}
```

In the sample above, keyword “parallel” defined a parallel block, inside it four parallel sub threads are running simultaneously. The keyword of “compete” points out that the parallel block is running as competition mode. That means if one of sub threads is completed, the whole block will get over, the rest unfinished sub threads will be abandoned. The parent thread will go on, the next instruction is “*Return I;”.*

At above we picked mode “compete”. Another alternative mode is of “synchronize”, as the sample below shows:

```
function parallelSample(paraA, paraB){
    var A, B, C, D;
    parallel(synchronize){
        thread 0: A = func01(paraA, paraB);      break;
        thread 1: B = func02(paraA, paraB);      break;
        thread 2: C = func03(paraA, paraB);      break;
        thread 3: D = func04(paraA, paraB);      break;
    }
    Return A + B + C + D;
}
```

In the sample above, the parallel block is defined as mode “synchronize”. That means the block will not get over until all sub threads finished their job.

Nested parallel blocks are supported as case of designing. See the sample below:

```
function parallelSample(paraA, paraB){
    var A, B, C, D, I;

    parallel(compete){
        thread 0:
          parallel(synchronize){
             thread 0: A = func01(paraA, paraB);      break;
             thread 1: B = func02(paraA, paraB);      break;
          }
          I = A + B; break;

       thread 1:
          parallel(synchronize){
             thread 0: C = func03(paraA, paraB);      break;
             thread 1: D = func04(paraA, paraB);      break;
          }
          I = C + D; break;
    }
    Return I;
}
```

Conclusion

It is a challenge to create dynamic interactive contents for internet. Today the only reliable way is programming. That is not the situation we hope for because many people lack the skills of programming. The fundamental difficulty is: dynamic interactive contents include complicated information, it is hard to represent through a simple way. Timetree is a new approach. It still has many works to do. But we already took the first step.

Reference:

[1] "What is A Timeline -Explain with Examples", https://www.edrawmax.com/timeline

[2] "Internet | Description, History, Uses, &Facts | Britannica", https://www.britannica.com/technology/internet

[3] "Rich Media Applications development using HTML5, AJAX, ⋯", https://dartinnovations.com/rich-media-applications

[4] Adobe Flash, https://www.adobe.com

[5] "What is Web Page? -Computer Hope", https://www.computerhope.com/jargon/w/webpage.htm

[6] "HTML Living Standard", https://html.spec.whatwg.org/multipage/

[7] "DOM Living Standard", https://dom.spec.whatwg.org

[8] "JavaScript", https://developer.mozilla.org/en-us/docs/web/javaScript

[9] https://vuejs.org, https://react.dev, https://angular.io

[10] "Programming Without Code: The Rise of No-Code Software Development", https://spectrum.ieee.org/progrmming-without-code-no-code-software-development

[11] "What is SaaS (Software as a Service)?", https://aws.amazon.com/what-is/SaaS/

[12] TOMStream, https://store.steampowered.com/app/2495810/TOMStream

[13] EDSGER W. DIJKSTRA, "Notes on structured Programming", https://www.iicseonline.org/structured-programming2.pdf

[14] Nathaniel J. Smith, “Notes on structured concurrency, or: Go statement considered harmful”, https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful

[15] Roman Elizarov, https://medium.com/@elizarov/structured-concurrency-722d765aa952