A Distributed Blogging System Based on Independent HTML Fragments

Andy Wang

September 16, 2025

This paper proposes a distributed blogging system [1] [2] [3] [4] [5] [6] based on independent HTML fragments, which we term TOM (Time Object Model) [7]. The core characteristics of this blogging system are: ”individual authors have full autonomy over managing their own blog content, using independent HTML fragments (in JSON format, with unique HTTP addresses) as the content carrier, and supporting other blogs to embed these fragments without notifying the original author.”

Part 1: Decentralized Blogging System Based on Independent HTML Fragments

Absolute Independence in Creation and Management for individual Authors

Complete Autonomy in Content Creation: Each author possesses absolute creative and managerial rights over every article in their blog. The entire process requires no collaboration with other authors nor third-party review, fully adhering to the core requirement of “full management by a single author.”

Autonomous Control over Updates and Takedowns: Authors can modify or remove their article fragments at any time. All blogs that have embedded the fragment will automatically load the updated content upon the next request. This “the embedding end continues to display old content after the original is taken down,” enduring content responsibility and ownership remain entirely with the author.

“Authorization-Free Freedom” and “Low Technical barrier” in Cross-Blog Embedding

Freedom to embed Without Notifying the Original Author: When other blogs include an author’s article, they do not need prior authorization from or need to notify the original author. Embedding is achieved simply by obtaining the HTTP address of the TOM document, significantly reducing the collaboration cost for cross-blog content integration.

The most contentious aspect of this process is whether embedding an author’s article into one’s own blog without consent constitutes theft. We argue is does not; it should not be viewed as “stealing” but rather as “distribution.”

Our reasoning is as follows: A TOM document is not merely a data document; it contains not only all content and styling but also complete control information. A complete TOM document is an entry point and, in essence, a piece of program logic. Regardless of who embeds the TOM document, control remains with the original author; the referrer cannot alter this fact. YouTube allows other websites to embed its videos because control (and thus rights and benefits) remains in its hands. Anything that belongs to you, no matter where it is in the world, remains your property as long as full control over it remains in your hands.

For authors, publishing a blog post in the TOM format implies a willingness to share the link with the world. For referrers, embedding a TOM document created by someone else implies agreement that the original author retains control over the content.

Low Technical Barrier for Implementation: Cross-blog including requires no complex technical adaptation; it can be accomplished through standard AJAX requests and runtime parsing. The referencing blog does not need to develop special interfaces or adapt to the original author's technology stack. It only requires three steps: First, load the runtime (a JavaScript file), which only needs to be loaded once. Second, prepare a container for loading the TOM, typically a <div> element. Finally, call the runtime's API to load the specified TOM document. Even non-technical blog authors can perform this operation by pasting a few simple lines of code, representing an extremely low technical barrier. Embedding other documents within a TOM document itself requires merely setting an attribute.

Decentralized Cross-Blog Collaborative Ecosystem

All blogs form a decentralized collaborative ecosystem through fragment includes, eliminating reliance on a centralized platform: each blog acts as an independent "content production node" and "aggregation node" – it can both create and provide fragments for other blogs to include, and also embed fragments from other nodes for content aggregation. The entire ecosystem has no core control platform; connections between nodes are realized solely through fragment URLs. Compared to traditional centralized blogging platforms, this approach is more flexible and less susceptible to the rules of any single platform.

The boundaries of content responsibility between the referrer and the original author are clear: The original author is responsible for the legality and authenticity of the fragment content. The referrer is only responsible for "correctly embedding and displaying" the content and does not bear responsibility for the quality of the original content. If issues exist with the original fragment, the referrer must cease referencing promptly upon discovery, and the original author bears the primary responsibility for rectification.

Traffic Sources: Decentralized Expansion from a "Single Blog" to "Multiple Reference Nodes"

Traffic Import from Referrer Blogs: When other blogs include an author's TOM fragment, they essentially create "traffic entry points" for the author's original blog. The author can leverage their control over the embedded content to guide users back to their own blog.

SEO Enhancement: Each of the author's articles has a unique HTTP address, which can be directly crawled and indexed by search engines. Search engines consider "external links from other websites pointing to the target domain" as an "endorsement of content quality." A higher number of such backlinks, especially from referrer domains with high authority, leads to a more significant improvement in the target domain's SEO weight. Under the fragment reference model, referrer blogs necessarily contain links pointing to the original blog's fragment URL, which will contribute to the SEO weight of the primary domain.

Part 2: Principle and Implementation of Embedded HTML fragments Based on TOM

Now we describe the foundation of this system: the embeddable HTML fragment – the TOM document.

TOM stands for Time Object Model. It adds a structured timeline – a timetree [8] – on top of the DOM, enabling TOM to express a dynamic information flow.

TOM is designed following a content – presentation separation pattern but differs significantly from other Headless CMSs [9] [10] [11]. The content format of TOM is fixed and does not adapt to specific requirements, only undergoing version updates. The front-end display layer (runtime) of TOM is also unique, cannot be customized, and only versioned.

This design ensures TOM documents are presented consistently and maximizes security but raise a question: how to adapt to different needs? Our solution is – atomizing the content. A TOM document is essentially an HTML document in JSON format, containing four types of information: HTML information (currently supporting only three elements: <div>, <pre>, <img>), CSS information (supporting only a subset of CSS properties), control information, and time information. Since TOM documents do not contain any scripts, we adopt a unique approach – making the media itself the program.

Media as Program

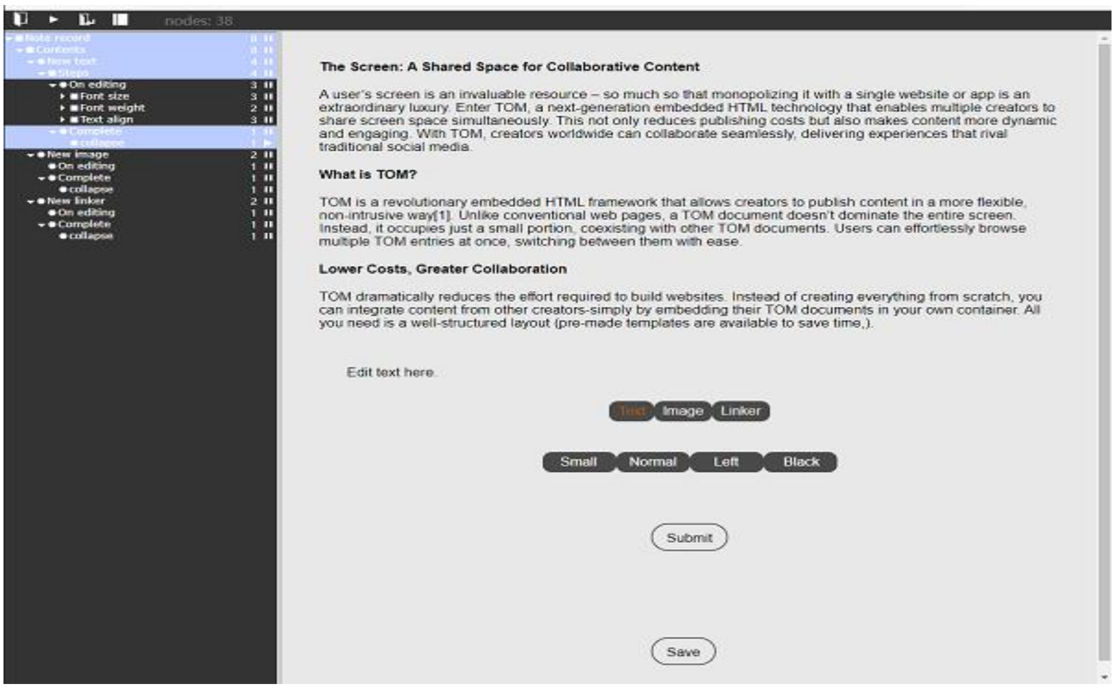
The origin of TOM stemmed from an idea: Can we merge media and program into one? This is possible. The timeline expressing a dynamic media stream and the code sequence expressing a program are both linear, unidirectional data flows with many commonalities. With appropriate adjustments, we can create a structure that can express both media and logic. We adopted the following approach: First we redesigned the data structure for expressing time. Traditionally, the industry uses timelines, but timelines are unstructured data. Therefore, we replaced the timeline with a timetree. A tree is an excellent control structure; using a time tree allows describing not only time but also logic.

The next challenge was how to construct programs using the timetree. Programs require three basic structures: sequence, loop, and branch. The timetree has an inherent forward-playback characteristic, naturally providing the sequence structure. For the loop structure, we designed an attribute playback for each node, with three possible values: play, stop, loop. Selecting ‘loop’ defines a loop structure. Finally, for the branch structure, we implemented it through events with conditions. This is not perfect but functional. Please refer to the help documentation for more details [12].

We also needed to express values. How can a timetree express a number? The timetree itself cannot express numbers, but each parent node has a current child node. For serial nodes, this current child node is unique; this unique current child node represents the current value of the parent node. Thus, we can easily define variables: we can define a value as the current child node of a specific parent node. Since the current child node of that parent node is uncertain, we obtain a variable. However, the current version of TOM does not yet support variables; it can only use constants.

We also needed to implement functions or subroutines, indispensable in modern programming. Function calls in TOM are very simple: embedding one TOM document into another TOM document in real-time constitutes a function call. From a media perspective, this is very similar to embedding an iframe into an HTML document. But from a program perspective, it equates to calling a function because it actually returns! We can perform a collapse operation on an embedded TOM document, which corresponds to a return operation in programming. The collapse operation destroys the TOM’s timetree, returns all currently visible elements to the parent TOM document, and replaces the TOM document element that executed the collapse operation. The replaced TOM document is then destroyed.

This process can be performed recursively, allowing us to design editable templates based on TOM that work in real-time. Tomestream.com demonstrates such an editable template – the world’s first TOM-based editor [13]. It can edit text, images and TOM document URLs. Each time a user submits an edit, this editor recursively calls itself, allowing the editing process to continue. This editor is very rudimentary; it only allows limited adjustments and, in fact, only additions – users cannot modify or delete already submitted content, which greatly limits its practicality.



The Screen: A Shared Space for Collaborative Content

A user's screen is an invaluable resource – so much so that monopolizing it with a single website or app is an extraordinary luxury. Enter TOM, a next-generation embedded HTML technology that enables multiple creators to share screen space simultaneously. This not only reduces publishing costs but also makes content more dynamic and engaging. With TOM, creators worldwide can collaborate seamlessly, delivering experiences that rival traditional social media.

What is TOM?

TOM is a revolutionary embedded HTML framework that allows creators to publish content in a more flexible, non-intrusive way[1]. Unlike conventional web pages, a TOM document doesn't dominate the entire screen. Instead, it occupies just a small portion, coexisting with other TOM documents. Users can effortlessly browse multiple TOM entries at once, switching between them with ease.

Lower Costs, Greater Collaboration

TOM dramatically reduces the effort required to build websites. Instead of creating everything from scratch, you can integrate content from other creators-simply by embedding their TOM documents in your own container. All you need is a well-structured layout (pre-made templates are available to save time,).

Edit text here.

We can improve its performance using parameterized functions. We have not yet implemented this feature in the current TOM, but we can explain it here preliminarily. We could add a linker attribute to any HTML element, pointing to a TOM document link, typically triggered by an event like click. Once this attribute is triggered, the element would be replaced by the TOM document it points to, and the element itself would be passed as a parameter to that TOM document. The TOM document would contain an element of the same type, which would be initialized using the passed parameter.

Using parameterized functions, we could enhance this editor to not only add but also modify or delete submitted content, and even insert content, though moving content would still be unavailable.

Using TOM

HTML fragments based on TOM can be independently deployed and used like HTML pages. We believe this will bring more possibilities to the internet. Embeddable, independent TOM fragments facilitate information aggregation and provide a continuous reading experience. If a blogger aggregates a lot of the latest information on their blog, all their readers can access this information directly without extra effort. We believe if many bloggers do this, this human-powered information aggregation method will not be inferior to algorithm-based aggregation and will have advantages in categorization. Bloggers who put in the effort to aggregate information will be rewarded accordingly. Search engines can also be used to aggregate and embed TOM documents, making them another way to read TOM content.

Security of TOM

Embedding HTML faces many challenges, primarily security. Embedding unverified webpage fragments can lead to uncontrollable security risks. However, if content must be trusted and verified, it usually leads to a centralized solution. Therefore, loaded TOM documents are always assumed to be unverified, making security the primary challenge.

TOM's response is the separation of content and presentation; content does not contain scripts. Furthermore, all interactive elements, such as <input>. Are prohibited. HTML links are not supported; readers cannot jump from a TOM document to an HTML page. TOM documents can only load other TOM documents; TOM documents can only link to other TOM documents.

After these restrictions, the risk of active intrusion caused by embedding TOM documents is significantly reduced, remaining within basically safe and controllable limits. Risks still exist, such as personal information leakage during resource loading or the loaded resources (e.g., images) potentially being malformed or even malicious files. However, these risks are relatively low and typically do not lead to active intrusion. At this stage, we consider the risks of loading unverified TOM documents to be within safe and controllable limits. If you find we are overly optimistic or have missed critical aspects, please point out the shortcomings.

Publishing TOM

TOM documents are stored in a TOM repository responsible for their storage and distribution. TOM Sever is a PHP-based storage service that can be used to run TOM repository services. Download link: https://tomstream.com/data/download/TOMServer.zip. To run TOM Server, you need your own domain and storage space.

Professional Backend-as-a-Service (BaaS) content delivery services are also well-suited for TOM storage and distribution, though this may incur additional costs.

Reference

[1] Weblogs: A History and Perspective. 2000. Rebecca Blood
https://www.rebeccablood.net/essays/weblog_history.html

[2] Weblogs as a bridging genre. 2005. Susan C. Herring, Lois Ann Scheidt, Elijah Wright, Sabrina Bonus
https://www.researchgate.net/publication/220436990_Weblogs_as_a_bridging_genre

[3] Enabling decentralized microblogging through P2PVPNs. 2013. Pierre St Juste, Heung Sik Eom, Kyung Yong Lee
https://ieeexplore.ieee.org/document/6488465

[4] Twittering by Cuckoo: Decentralized and Socio – Aware Online Microblogging Services. 2010. Tianyin Xu, Yang Chen, Xiaoming Fu
https://user.informatik.uni-goettingen.de/~ychen/papers/Cuckoo_sigcomm10.pdf

[5] https://wordpress.org/plugins/activitypub/

[6] https://joinplu.me

[7] Time Object Model: A New Model for HTML. 2024
https://www.techrxiv.org/doi/full/10.36227/techrxiv.170846827.75951549/v1

[8] Timetree: A New Way for Representing Time. 2024
https://www.techrxiv.org/doi/full/10.36227/techrxiv.170629845.50870587/v1

[9] Eleventy by Example: Create Powerful, Performant Websites with a Static – First Strategy. 2023. Bryan Robinson
https://ieeexplore.ieee.org/document/10251297

[10] Headless CMS: Everything you need to know – Hygraph
https://hygraph.com/learn/headless-cms

[11] Design and Development of a Headless Content Management System.
https://mail.irjet.net/archives/V9/i7/IRJET-V917133.pdf

[12] https://tomstream.com

[13] https://tomstream.com/data/TOMData/tomIntroduction.tom