The Attribute-Driven Full Regeneration Model
Official Specification

Working title: Sercrod

Chapter 1. Introduction

This document formalizes a new user interface model, the **attribute-driven full regeneration model**, which is built entirely on top of existing Web standards: HTML, the DOM, and browsers and DOM parser execution environments, including headless and virtual browsers.

The model is not a manual for a particular library or tool. Instead, it is defined as an abstract design principle whose core transformation is

data (scope) → attributes → DOM

as a single, one-way pipeline.

This document uses "Sercrod" as the reference implementation for illustration. The model itself is intended to be independent of any specific implementation.

In contemporary Web development, many approaches have been proposed to abstract DOM updates, including virtual DOM-based and declarative UI systems. These often maintain an internal virtual structure and compute differences to apply to the actual DOM.

By contrast, Sercrod does not maintain a virtual structure. It treats HTML attributes and the DOM itself as the primary carriers of UI description, and expresses UI changes as a linear flow:

change in data ⇒ update of attributes ⇒ reconstruction of the DOM

Here, **attribute-first** means that the official representation of UI state is concentrated in attributes and the DOM, without maintaining any separate mirrored UI tree (for example, a virtual DOM).

Sercrod uses Custom Elements (Web Components), but it does not rely on hidden rendering optimizations or special partial repaint behavior provided by Web Components. The main observations are:

- Browsers, DOM parsers, and virtual browsers execute layout and paint according to JavaScript-driven DOM operations.
- Using Web Components does not, by itself, provide special differential rendering semantics.

Sercrod is characterized by the way it encapsulates an attribute-first UI model inside Custom Elements, and then runs it directly on top of the standard DOM engine of the browser and DOM runtime.

The purpose of this document is not to advertise Sercrod as a product, but to record the design and observational results as:

- a conceptual description of an attribute-driven model
- a formalization of the internal model (scope, attributes, DOM)
- a structured description of the update procedure and processing flow
- a classification of an alternative UI model that does not rely on virtual DOM

The scope of this document is:

- to organize the rendering model centered on attributes
- to describe the observed behavior of rendering and updates in standard environments
- to abstract the internal structure and procedures of Sercrod
- to position the model as a UI approach distinct from virtual DOM-based systems
- to discuss the academic and practical implications of an attribute-centric model

In what follows, "Sercrod" refers to the reference implementation used for illustration.

Chapter 2. Scope and Positioning of the Model

This chapter clarifies the reach of the model and the conceptual relationship to existing UI models.

2.1 Targeted model

The document targets UI update models that satisfy all of the following conditions:

1. The primary representation of state is concentrated in HTML or DOM attributes.
2. Updates are performed by reconstructing the DOM from a combination of template and attributes.
3. No virtual DOM or independent mirrored tree is maintained.
4. All mechanisms are implemented using standard Web APIs only, such as Custom Elements, DOM API, and the event model.

A model with these properties is called here the **attribute-driven full regeneration model**.

Sercrod implements this model in the following way:

- UI logic is described as attributes inside HTML.
- Each Custom Element holds its own template.
- Every call to update() reconstructs the DOM from the template.
- Data is represented as a scope object and reflected into the DOM via attributes.

2.2 Conceptual distinction from virtual DOM-based models

Virtual DOM-based models generally have the following structure:

- They maintain an internal virtual tree structure.
- State changes are reflected in the virtual structure.
- Differences between virtual structures are computed and then applied to the real DOM.

This creates a layered pipeline of

state → virtual structure → diff → DOM.

In contrast, the attribute-driven full regeneration model (as illustrated by the reference implementation, working title: Sercrod) adopts a more direct three-step path:

state (scope) → attributes → DOM.

The important point is not that Sercrod is faster than virtual DOM-based approaches, but that Sercrod **does not maintain a virtual DOM at all**.

In virtual DOM-based approaches, developers must often consider both internal state and the real DOM, and the diffing logic exists as a separate layer from the core UI logic.

The attribute-driven full regeneration model avoids this layer entirely: under the attribute-first principle, the DOM is treated as the unique target structure and attributes are the official interface between state and DOM.

2.3 Role of Web Components

In the model described here, Web Components (Custom Elements) play the following roles:

1. **Scope boundary**
   Each Sercrod element is a Custom Element and holds its own data scope, template, and lifecycle. This provides a natural boundary for UI segments.
2. **Attribute change hook**
   Through attributeChangedCallback and related mechanisms, the element can receive notifications when specific attributes change, and then invoke its own internal logic such as update().

The important point is that Web Components are not used for special rendering optimization, but simply as a standard framework for reacting to attribute changes and structuring scopes.

2.4 Positioning of the model

With this in mind, the attribute-driven full regeneration model can be positioned conceptually as follows:

- It is not a virtual DOM-based model.
- It is not merely a collection of imperative DOM operations.
- It is a UI language model whose main descriptive medium is HTML attributes.
- It is a scope control model whose unit is the Custom Element.

This document defines the model as an independent UI design framework, and organizes its internal structure and semantics.

Chapter 3. Fundamental Principles of the Attribute-Driven Model

This chapter explains why the Sercrod model adopts the attribute-first principle and how it defines the flow of state.

3.1 Treating HTML attributes as state

HTML is commonly used as a markup language to describe structure and presentation. In actual implementations, however, attributes are tightly coupled with DOM state:

- Adding, changing, or removing an attribute immediately changes the state of the corresponding DOM node.
- Many attributes are reflecting attributes that are synchronized with DOM properties through the Element interface.
- The presence and values of attributes affect style, conditional display, and behavior.

From this, the model treats attributes not as auxiliary information but as

the primary medium for expressing UI state.

Sercrod's attribute-first principle explicitly adopts this position.

3.2 A one-way flow from attributes to DOM

The DOM is a dynamic object representation of HTML as a tree structure. Attribute changes propagate to the node and, in principle, to its subtree.

The Sercrod model explicitly defines this one-way flow:

scope values → attribute values → DOM node state

and essentially disallows updates that bypass attributes.

At first glance this constraint might appear to reduce flexibility, but it offers significant benefits:

- The flow of state becomes structurally and visually traceable.
- Divergence between internal state and DOM is less likely.
- External tools or scripts that modify attributes do not easily break the semantic consistency of the UI.

3.3 Relationship between templates and attributes

Each Sercrod element holds an internal template. This template is **not** a separate state world. Instead, it is treated as

the static original form of HTML structure that includes attributes.

At runtime, the template and the current scope are used together to reconstruct a DOM that includes attributes and text content.

In this relationship:

- Attributes are the primary carriers of semantics.
- The template is a container that stores HTML with embedded attribute-based logic.

Therefore, this document assumes attribute-first as the core design principle and regards template-based reconstruction as a concrete method to maintain structural consistency.

3.4 Why attribute-first is a necessary choice

Treating attributes as the primary representation of state has several kinds of necessity:

1. **Alignment with browser and DOM implementations**

   Attributes and the DOM are coupled by design, so placing state in them follows the original

architecture of Web platforms.

2. **Avoidance of double bookkeeping**
   By not maintaining additional state trees such as virtual DOM structures, the model avoids the double bookkeeping between internal state and the visible DOM.
3. **Consistency in the description layer**
   Encoding state, conditions, iterations, and input bindings all as attributes gives the UI a single, unified descriptive vocabulary.

For these reasons, attribute-first is not just a stylistic preference but a design choice that aligns with the structure of the platform.

3.5 DOM as the single source of truth

In the Sercrod model, the DOM together with its attributes is the only authoritative representation of rendered UI state.

- The scope is input used during reconstruction.
- The template is the original form that prescribes the shape of the DOM.
- The actual state presented to the user is defined entirely by the DOM.

This ensures a tight binding between:

- HTML as specification
- the template as static representation
- the DOM as runtime state

and forms the basic conceptual foundation of the model.

Chapter 4. Web Standards Underpinning the Model

This chapter summarizes the Web standards and behaviors on which the Sercrod model depends.

4.1 Correspondence between HTML and the DOM

HTML documents are reconstructed by parsers into DOM trees.

Elements, attributes, and text nodes are represented as DOM objects. Attribute changes are exposed through the Element interface.

The model takes this standard mapping as a given, and adds another mapping on top:

scope → attributes → DOM

HTML remains the language that describes the UI. The model extends this by structuring how state flows into the existing HTML and DOM.

4.2 Reflecting attributes and attribute operations

Many attributes are defined as reflecting attributes, synchronized with DOM properties. As a result:

- Attribute operations are effectively equivalent to certain DOM operations.
- DOM state can be inspected via attributes.
- External scripts and internal implementations can share the same vocabulary for state.

Sercrod treats these facts as part of its base, and gives attributes the meaning of state containers.

4.3 Custom Elements and attribute change hooks

Custom Elements define attributeChangedCallback and related extension points to detect attribute changes.

Sercrod uses this to:

- receive notifications when certain attributes change
- trigger internal processes such as update() in response to these changes

The place where an attribute is changed does not matter; it could be:

- user input
- an external script
- a data fetch from a server

All such changes appear uniformly as attribute changes and are handled through the same mechanism.

4.4 Shadow DOM (optional)

Shadow DOM can be used, when needed, to localize structure and styles. It is not required for the core attribute-first model, but can help:

- visually demarcate scope boundaries
- avoid style collisions

The presence or absence of Shadow DOM does not affect the fundamental attribute-first principle.

4.5 DOM recomputation and rendering

When the DOM changes, the environment (browser and DOM runtime) executes layout, paint, and compositing as needed.

Detailed internal implementations differ by engine and are not fully specified, but observations show that:

- DOM operations are recorded at the level of individual elements.
- The region of layout and paint is determined internally based on style and structure.
- Whether a node is part of a Custom Element usually does not change the units of reflow or repaint.

The Sercrod model does **not** try to control this internal pipeline.

update() reconstructs the DOM, and layout or paint is then delegated to the standard behavior of the environment.

4.6 Event model

The standard event model uses a three-phase propagation (capture, target, bubble) based on the DOM tree. Sercrod attaches handlers defined as attributes (such as *input) to DOM nodes. These handlers modify the scope if necessary, and then trigger update().

This yields two consistent flows:

- Event propagation follows the DOM structure.
- State change follows **scope → attributes → DOM**.

4.7 A model that stays within Web standards

All features discussed so far are part of existing Web standards and implementations.

The Sercrod model requires no new native APIs. It constructs an attribute-first, full regeneration UI model on top of:

- attribute-first semantics
- full regeneration of the DOM from templates
- a unified scope, template, and DOM model

within the boundaries of the standard platform.

Chapter 5. Rendering Behavior and Observational Results

This chapter describes how the model behaves in real environments, and clarifies several points that are often misunderstood.

5.1 Observation setup

The following tools and conditions were used in observation:

- MutationObserver to track changes in the DOM.
- Tracking of node identities to distinguish between reuse and re-creation.
- Paint flashing or similar features in developer tools to visualize repaint regions.
- Bulk operations on structures of ten thousand to one hundred thousand nodes, using textContent and attribute updates.

These allowed observation of:

- which nodes are created and destroyed
- which regions of the display are repainted

5.2 Observations under the setup

The following observations were obtained under the setup described in 5.1. They are empirical results from specific environments and do not constitute a universal guarantee across engines, versions, hardware, or workloads.

The observations can be summarized as follows:

1. **DOM operations faithfully follow JavaScript commands**
   When innerHTML, textContent, or setAttribute is called, the environment creates and destroys DOM nodes according to those operations.
   No special differential update behavior was observed solely due to the use of Web Components.
2. **Layout and paint regions are decided internally**
   Even when an entire element subtree is updated, the visible effect may appear localized. This is due to standard optimizations inside the engine, not because of Web Components specific behavior.
3. **Large DOMs still complete updates within practical times**
   In structures with tens of thousands of nodes, bulk updates of textContent or attributes completed under the tested environments, but latency varied substantially with engine, hardware, DOM size, and style complexity.
   No universal performance claim is made here. The result is interpreted as evidence that the underlying DOM engine can handle large-scale mutations, not as a guarantee of interactive latency for all cases.

5.3 Clarifying common misunderstandings

Based on these observations under the setup in 5.1, the following interpretations are supported:

- It is inaccurate to assume that “using Web Components automatically yields local rendering only”.
- UI updates in Sercrod remain completely within the standard DOM processing pipeline.
- Local-looking repaints are due to the internal strategies of the environment, not to custom rendering algorithms implemented by Sercrod.

5.4 The assumptions adopted by Sercrod

With these observations as a base, the Sercrod model adopts the following stance:

- Rather than optimizing rendering itself, the model focuses on clearly defining **how** the DOM is described and **how** it is updated.
- Performance depends largely on how directly and simply the model uses the DOM engine, and on how much additional abstraction is avoided.

The full regeneration update model, without a virtual DOM, fits naturally with this stance.

Chapter 6. Distinctiveness of Sercrod and the Attribute-Driven Control Model

This chapter explains where the Sercrod model is distinctive, not in terms of rendering tricks, but in terms of its descriptive and control structure.

6.1 A UI description system that treats attributes as primary

Sercrod provides a structured system for describing UI logic via HTML attributes:

- **Conditional branches**: *if, *elseif, *else
- **Iteration**: *for
- **Scope extension**: *let
- **Input binding**: *input
- **Display reflection**: *print, :class, :style, and others

All of these appear as attributes on DOM nodes. As a result, questions such as:

- which node depends on which data
- under which condition a node is shown or hidden

can be answered directly by reading the HTML.

6.2 Systematization via template and scope

Each Sercrod element holds:

- an internal template
- a data scope
- an update procedure (update())

The template is the static original HTML with embedded attribute logic, and the scope is the set of values that can be referenced inside that template.

On update, the following sequence is executed:

scope evaluation → attribute computation → DOM reconstruction from the template

6.3 An update model without diffs

update() in Sercrod does not compare the existing DOM with the new structure. Instead it follows a simple procedure:

- clear the internal DOM of the host element
- evaluate the template and construct a new DOM
- call update() recursively on child Sercrod elements if needed
- reflect attributes and text nodes

Because the update model does not carry its own diffing layer, all update rules are fully described in terms of the template and attributes.

6.4 A conceptual answer to virtual DOM-based models

Virtual DOM-based approaches gain flexibility from their virtual structures, but as a result they introduce:

- two parallel views of state (internal and DOM)
- a diffing layer that exists independently of UI logic

The Sercrod model does not attempt to modify such architectures. Instead it belongs to a different family that:

- does not maintain a virtual DOM
- treats attributes and the DOM as the only official state representation

- concentrates UI logic and update rules in attributes

This is not an argument that one family dominates the other. It is a classification into distinct conceptual lineages.

6.5 Scope separation based on Web Components

Using Custom Elements, each Sercrod component has its own scope and lifecycle. This yields:

- clear separation boundaries in deep UI trees
- avoidance of accidental interference between parent and child scopes

Such structure simplifies reasoning about data and behaviors that belong to each segment of the interface.

6.6 Summary of distinctiveness

The distinctiveness of the Sercrod model lies in the combination of:

- an attribute-first UI description system
- a full regeneration update model
- a scope structure based on Custom Elements

The model does not manipulate rendering internals. Instead, it systematizes how attributes and the DOM are used.

Chapter 7. Internal Structure and Processing Steps

This chapter reformulates update() and the internal model of Sercrod as abstract procedures independent of any concrete language.

7.1 Overview of update()

The update() procedure can be divided into the following stages:

1. pre-locking and state preparation
2. scope evaluation and template interpretation
3. DOM reconstruction
4. post-processing and hooks

Details of this staged model appear in Appendix B.

7.2 Pre-locking and state preparation

At the start of update(), an internal flag such as _updating is set to prevent reentrancy.

The system then performs:

- synchronization of temporary state controlled by attributes such as stage or buffer
- branching between full update and children-only update according to attributes such as lazy
- selection of the target template and scope

No DOM changes occur in this stage.

7.3 Template evaluation

In the template evaluation stage, directives are resolved in a fixed order:

- conditional attributes: *if, *elseif, *else
- iteration structures: *for
- scope extension: *let
- reflection attributes: *print, :class, :style, and others

The result is an abstract structure that states:

- which nodes are to be generated
- which attributes are to be attached to those nodes

7.4 DOM reconstruction

DOM reconstruction follows these steps:

- clear the internal DOM content of the host element
- construct new nodes according to the evaluated template
- for each nested Sercrod element, call its update() as needed
- attach attributes and text nodes

The key point is that there is no diffing: the DOM is rebuilt based solely on the template and the current scope.

7.5 Post-processing

After reconstruction:

- indices of directive-containing nodes are rebuilt
- post-update hooks such as updated are invoked
- input values and focus are restored when possible
- the _updating flag is cleared and pending updates are processed if they exist

This prepares the component for subsequent update cycles.

7.6 Consistency properties

From this structure, the model guarantees:

- **determinism**: given the same template and scope, the same DOM is generated
- **one-way semantics**: the flow of state is **scope → attributes → DOM**
- **idempotence**: repeated calls to update() with the same scope do not change the DOM
- **local scope isolation**: local scopes do not modify outer scopes

Appendix C formalizes these properties in a more model-like style.

Chapter 8. Advantages and Systematic Positioning

This chapter summarizes the advantages of the Sercrod model as derived from the preceding structure.

8.1 Structural stability by template primacy

Reconstructing the DOM from a template keeps structural drift under control:

- No matter how complex the state transitions, update() restores the DOM to a shape that matches the template.
- Unlike approaches that continuously apply local DOM manipulations, there is always a guarantee that the runtime DOM is isomorphic to the template.

This property becomes more important as the UI grows in complexity.

8.2 Clear update rules based on attribute systems

Since conditions, loops, input bindings, and display reflections are all expressed as attributes:

- dependencies between nodes and data can be read directly from HTML
- display conditions can be inspected without descending into a separate logic layer

This improves the inspectability of the UI behavior.

8.3 Predictability of the full regeneration model

By foregoing diffing and adopting full regeneration, the model gains:

- simple update rules, easy to read and reason about
- fewer edge cases tied to specific diffing strategies

The choice is deliberate: it emphasizes predictable behavior over maximum optimization.

8.4 Alignment with DOM engine optimizations

The Sercrod model uses the DOM engine directly, without extra layers such as virtual DOM or custom diff structures.

Thus:

- the path from logic to DOM operations is short
- optimizations inside the DOM engine can be leveraged as they are

The focus of this document is on the structure of the model, not on absolute performance, but the alignment with the underlying engine is an important part of the design.

8.5 Long-term maintainability via HTML-centric description

Sercrod code is defined entirely within HTML documents. This yields:

- UI behavior that can be explained by HTML files alone
- easier integration with external tools such as static site generators and content management systems
- a better fit for educational scenarios where HTML, the DOM, events, and state management are taught together

8.6 Summary of systematic positioning

Taken together, the Sercrod model can be classified as a UI design framework whose core is:

- template primacy
- attribute-first semantics
- full regeneration updates

It is a family distinct from virtual DOM-based and imperative DOM manipulation-based families.

Chapter 9. Academic Positioning and Future Applications

This chapter considers the Sercrod model as an abstract concept and outlines its potential applications.

9.1 Three conceptual pillars

The model can be viewed as resting on three conceptual pillars:

1. **Attribute-centered semantics**
   Attributes are treated as the semantic units of the UI. Conditions, loops, and input bindings are described as attributes.
2. **Template primacy**
   The template functions as the canonical structure, and updates restore the DOM to a template-conformant state.
3. **Full regeneration**
   Updates are carried out by reconstructing the DOM from the template, not by computing diffs.

The combination of these three pillars distinguishes this model from others.

9.2 Systematic difference from virtual DOM-based families

Virtual DOM-based models:

- maintain state in internal virtual structures
- compute diffs between successive virtual structures
- apply those diffs to the real DOM

The Sercrod model:

- maintains no virtual structure
- uses attributes and the DOM as the only state representation
- adopts full regeneration as the update method

This difference is not only technical but also systematic in terms of how models are classified.

9.3 Target problem domains

The Sercrod model fits well to problem domains characterized by:

- hierarchical UI structures
- a desire to unify specification and implementation in HTML
- long-term maintenance and educational or documentation focus

Because templates and attributes effectively serve as a UI specification, the design intent is easier to express explicitly.

9.4 Applications in education and documentation

Attribute-based UI design has several advantages in education:

- HTML, the DOM, events, and state management can be taught as parts of a single narrative.
- Attributes can be used as a textbook vocabulary for describing UI behavior.

In documentation, the attribute system can be transcribed almost directly into technical specifications, suggesting the possibility of:

- standardized UI description languages based on attribute systems

9.5 Academic significance

The Sercrod model offers a unique framework in UI design theory by combining:

- template primacy

- attribute-centered semantics
- full regeneration

These characteristics place the model in a category that has not been clearly named or described in existing classifications of UI models.

Chapter 10. Conclusion - A Systematic Framework for the Sercrod model

Throughout this document, using Sercrod as a reference, we have organized:

- an attribute-first UI model
- template primacy as a structural principle
- full regeneration as the update procedure

The Sercrod model:

- concentrates state in attributes and the DOM
- reconstructs the DOM from templates
- does not maintain a virtual DOM

Rendering optimizations are delegated to the standard DOM engine of the environment. Sercrod itself focuses on the principles for using attributes and the DOM.

This document is not a usage manual for a specific implementation, but:

- a conceptual description of the attribute-driven full regeneration model
- a definition of the internal model (scope, attributes, DOM)
- a structured description of procedures and semantics

The model has potential applications in education, documentation, template generation, and beyond.

Appendix A. Glossary (Basic Vocabulary of the Sercrod model)

A.1 Template related terms

- **Template**
  The original HTML structure of the UI. It includes attributes and is reevaluated on each update.
- **Template primacy**
  The design principle that the runtime UI structure must remain isomorphic to the template after updates.
- **Template reconstruction**
  The procedure of rebuilding the DOM based on the template during updates.

A.2 Attribute system

- **Attribute-centric model**
  A model that describes UI behavior such as conditions, loops, and input bindings as HTML attributes.
- **Behavioral attributes**
  Attributes that define behavioral rules of the UI, for example *if, *for, *let, *input.
- **Structural attributes**
  Attributes that define template structure and generation behavior, such as *template, *include.
- **Render attributes**
  Attributes that control rendering aspects such as classes, styles, and links, for example :class, :style, :href.
- **Scope**
  The set of data accessible inside a template. Expressions in attributes are evaluated with respect to a scope.

A.3 Update and rendering

- **Reconstruction model (update by template reconstruction)**
  An update method that reconstructs the DOM from the template on every update rather than performing diffs.
- **Internal state**
  The data of the UI (objects, arrays, and so on). It is evaluated as a scope and reflected into the DOM via attributes.
- **Update rules**
  The set of procedures and semantics defined by templates, scopes, and attributes.
- **Update unit**
  The DOM region to which update() applies. Typically this is the inside of one Custom Element.

A.4 Input binding and data flow

- **Input binding**
  A mechanism such as *input that synchronizes user input with internal state.
- **Data flow**
  The one-way flow of **scope → attributes → DOM** together with the reverse flow of **input → scope**.
- **Scope extension**

The introduction of temporary variables inside templates using constructs such as *let.

A.5 Logical constructs and structural control

- **Conditional attributes**
  Attributes such as *if, *elseif, *else that describe display conditions for DOM blocks.
- **Iteration**
  The expansion of repeated structures using *for over arrays or iterable objects.
- **Template expansion**
  The insertion or expansion of templates via attributes such as *include.

A.6 Reflection on the display surface

- **Reflection attributes**
  Attributes that compute and reflect display values into the DOM, such as *print, :class, :style.
- **Static reflection**
  Reflection that is evaluated only during the initial render.
- **Dynamic reflection**
  Reflection that is reevaluated on each call to update().

A.7 Meta structure

- **Directive**
  A generic term for attributes that control UI behavior.
- **Structural rules**
  Internal rules applied when constructing and reconstructing the DOM from templates.
- **Execution model**
  The overall pipeline of template extraction, scope evaluation, DOM reconstruction, and rendering.

A.8 Vocabulary specific to the Sercrod model

- **Attribute-first**
  The principle that state is represented primarily by attributes and the DOM, without additional parallel state trees.
- **Reconstructive update**
  An update procedure that reconstructs the DOM from the template without computing diffs.
- **Template-first behavior**
  The behavior by which update() always restores the DOM to a structure that conforms to the template.

Appendix B. Process Flow (Staged Model of Updates)

This appendix defines the update procedure of the Sercrod model as a staged flow.

B.1 Overall structure

The update process has four stages:

1. pre-processing
2. template evaluation
3. DOM reconstruction
4. post-processing

B.2 Pre-processing

Pre-processing includes:

- setting a reentrancy prevention flag
- preparing temporary state for stage, buffer, and similar attributes
- selecting the template and scope to be used in this update
- deciding whether to update only children or the entire component based on attributes such as lazy

B.3 Template evaluation

Template evaluation includes:

- evaluating conditional attributes (*if, *elseif, *else)
- expanding iteration structures (*for)
- extending scopes (*let)
- evaluating reflection attributes (*print, :class, :style, and so on)

The result is an abstract structure describing which nodes to generate and which attributes to attach.

B.4 DOM reconstruction

During DOM reconstruction:

- the internal DOM of the host is cleared
- new nodes are generated according to the evaluated template
- nested Sercrod elements, if any, receive calls to update()
- attributes and text nodes are applied

B.5 Post-processing

Post-processing includes:

- rebuilding indices of directive-containing nodes
- invoking post-update hooks such as updated
- restoring input values and focus if possible
- clearing the reentrancy flag and processing pending updates if they exist

B.6 Role of the staged model

The staged model supports:

- structural stability through template primacy
- behavioral stability through attribute-centered rules
- predictability by using a reconstruction model without diffs

Appendix C. Internal Model

This appendix defines the internal structure of the Sercrod model in a more formal style.

C.1 Purpose

The internal model is defined as three stages:

1. **scope evaluation**
2. **attribute computation**
3. **DOM construction and update**

C.2 Scope model

C.2.1 Scope hierarchy

- **Root scope**
  The base data obtained through operations such as data or fetch.
- **Local scope**
  Temporary bindings introduced by constructs such as for and let.
- **Stage scope**
  Temporary values maintained by constructs such as stage or buffer.

C.2.2 Evaluation rules

- Name resolution proceeds from inner scopes to outer scopes.
- Unresolved identifiers are treated as undefined.
- Evaluations with side effects are restricted to the range permitted by the model.

C.3 Attribute model

C.3.1 Attribute categories

- **Binding attributes**
  Attributes such as :value, :class, :style that map scope values to strings or fragments.
- **Behavioral attributes**
  Attributes such as *input that mediate between DOM events and the scope.
- **Control attributes**
  Attributes such as *if, *for, *let, *template that determine the generation or omission of DOM structures.

C.3.2 Evaluation timing

- During update(), all dynamic attributes are reevaluated based on the current scope.
- Control attributes affect structural decisions, while binding attributes determine attribute values on generated nodes.

C.4 Template expansion model

- The template is initially obtained as a string.
- Optional pre-processing, such as comment stripping, takes place as needed.
- The template is converted to an internal representation akin to an abstract syntax tree.

Expansion rules:

1. resolve conditionals (if family)
2. expand loops (for)
3. apply scope extensions (let)

4. evaluate attributes
5. recursively expand child nodes

The resulting structure serves as input to DOM reconstruction.

C.5 Update model

C.5.1 Role of update()

update() is responsible for:

- updating scope values when needed
- re-expanding the template
- reconstructing the DOM
- invoking hooks and resetting internal state

C.5.2 Execution steps

1. set a reentrancy prevention flag
2. synchronize stage and buffer scopes
3. decide the update target according to attributes such as lazy
4. clear the internal DOM
5. expand the template and reconstruct the DOM
6. rebuild directive indices
7. invoke updated hooks
8. clear the flag and process pending updates

C.6 Consistency guarantees

The internal model guarantees:

- **determinism**: the same template and scope always generate the same DOM
- **one-way semantics**: state flows only from scope to attributes and from attributes to DOM
- **idempotence**: repeated calls to update() with the same scope leave the DOM unchanged
- **scope isolation**: local scopes do not affect outer scopes

C.7 Limitations of the model

The model explicitly does not guarantee:

- diff-based optimization of updates
- control over low-level rendering strategies such as layout and paint
- elimination of all possible update-event cycles

The cost of updating large DOM structures is delegated to the environment that executes the DOM operations.

This internal model makes explicit the assumptions and guarantees of the Sercrod model and provides a formal basis for describing the attribute-driven full regeneration UI approach.