MINIDIFF: Templates

I. WECHAT DOMAIN KNOWLEDGE

WeChat Mini-Program. A WeChat mini-program [6] primarily consists of five types of files that work together to create a seamless user experience: (1) `JSON` files for configuration, such as setting page paths, window appearance, and permissions; (2) `WXML` files defining the layout and structure of page views, similar to HTML in Web applications; (3) `WXSS` files responsible for the styling and design of page views, much like CSS in Web applications; (4) `JS` files that contain the logic driving the mini-program's functionality, handling data management, user interactions, and dynamic behaviors; and (5) `WXS` files providing a WeChat-specific scripting language for efficient data processing within page views.

Figure 1 illustrates an example WeChat mini-program that implements a basic accumulator. The `app.json` file (Figure 1a) specifies that the mini-program contains a single `index` page, located in the `pages/index/` directory (Line 1), which serves as the main entry point. The `index.wxml` file (Figure 1b) defines the page view, featuring a text component to display the cumulative sum (Line 7) and a button component to increment the unit value (Line 8). The `index.wxss` file (Figure 1c) controls the appearance of the page, including the styling of the text (Line 10) and button (Line 11) components. The `index.js` file (Figure 1e) initializes the unit value variable (`unit` at Line 16) and defines an event handler (`inc_unit` at Lines 18-22) that increments the unit value. The `index.wxs` file (Figure 1d) defines a cumulative sum variable (`sum` at Line 12) and exports a data processing function (`cal_sum` at Lines 13-14) that calculates the the sum of the unit value and the existing cumulative sum, then displays the result in the page view.

WeChat Mini-Program Framework. The WeChat mini-program framework [9] consists of two distinct layers: the rendering layer and the logic layer. The rendering layer is responsible for displaying page elements by interpreting layouts defined in `WXML` and applying styles from `WXSS` to create the visual presentation, ensuring smooth rendering for an enhanced user experience. In contrast, the logic layer handles the underlying business logic, processing user interactions, managing data, and executing functions defined in `JS` files to control the mini-program's behavior and facilitate communication with back-end services. The scripts in `WXS` enable efficient data processing within the rendering layer, reducing the need for communication with the logic layer.

The rendering layer and the logic layer are designed to operate in separate threads, relying on event binding and data binding for communication. Event binding connects user interactions triggered in the rendering layer to the event handlers registered in the logic layer, allowing developers to create interactive and responsive applications that effectively respond to user actions. Data binding links page elements in the rendering layer to JavaScript variables in the logic layer, ensuring that changes in data are automatically reflected in the user interface without the need for manual updates. Events and data are transmitted between layers through an asynchronous messaging channel.

Figure 2 illustrates the communication of events and data in the example mini-program. Specifically, the event handler `inc_unit` in the logic layer is bound to the `button` component in the rendering layer. Whenever the button is tapped, `inc_unit` is invoked. Additionally, the JavaScript variable `unit` in the logic layer is bound to the `text` component in the rendering layer. Whenever the value of `unit` is changed, the content of the text is updated. Furthermore, event binding and data binding work in conjunction in this example. Event binding triggers the increment of the unit value, while data binding activates the update of the user interface. To optimize performance, virtual DOM [1] is employed, where an update in the logic layer triggers the virtual DOM diffing [2] in the rendering layer. The diffing algorithm compares the current and previous virtual DOM trees to identify changes and update only the modified parts in the real DOM, minimizing unnecessary updates and enhancing rendering efficiency.

Configurations and Platforms. WeChat provides two component architectures to assist developers in structuring their user interfaces and business logic in a modular and efficient manner. The original architectures, Exparser [7], provides a straightforward approach to building reusable page elements, managing their lifecycle, and handling interactions within the mini-program. The more recent GlassEasel [8] architectures is introduced to optimize performance and offer greater flexibility. Both architecturess coexist, allowing developers to select the one that best fits their requirements by configuring the `componentFramework` option in `app.json`. WeChat mini-programs can run on various platforms, including Android, iOS, Windows, Mac and DevTools, with each platform using a different Webview rendering mode. Recently, a custom rendering mode specifically designed for WeChat mini-programs, named Skyline [11], has been introduced to provide a more native-like experience. Developers are allowed to select between the platform-specific Webview and the WeChat-customized Skyline rendering modes by configuring the `render` option in `app.json`. Note that the example mini-program is set to use the Glass-Easel framework and the Webview rendering engine, which is the default configuration.

```
1. "pages": ["pages/index/index"],
2. "renderer": "webview",
3. "componentFramework": "glass-easel",
4. "lazyCodeLoading": "requiredComponents"
```
(a) app.json - app configuration

```
5. <wxs module="md" src="./index.wxs"/>
6.
7. <text>{{ md.cal_sum(unit) }}</text>
8. <button bind:tap="inc_unit">inc</button>
```
(b) index.wxml - page content

```
9.  page { width: 100px; height: 167px; }
10. text { padding: 30%; font-size: large; }
11. button { width: 100%; text-align: center; }
```
(c) index.wxss - page style

```
12. var sum = 0;
13. module.exports.cal_sum =
14.   function(unit){ return sum += unit; }
```
(d) index.wxs - WeChat-specific script

```
15. Page({
16.   data: {unit: 1},
17.
18.   inc_unit:function(){
19.     this.setData({
20.       unit: ++this.data.unit
21.     });
22.   },
23. })
```
(e) index.js - page script

Fig. 1. An example WeChat mini-program that implements a basic accumulator.

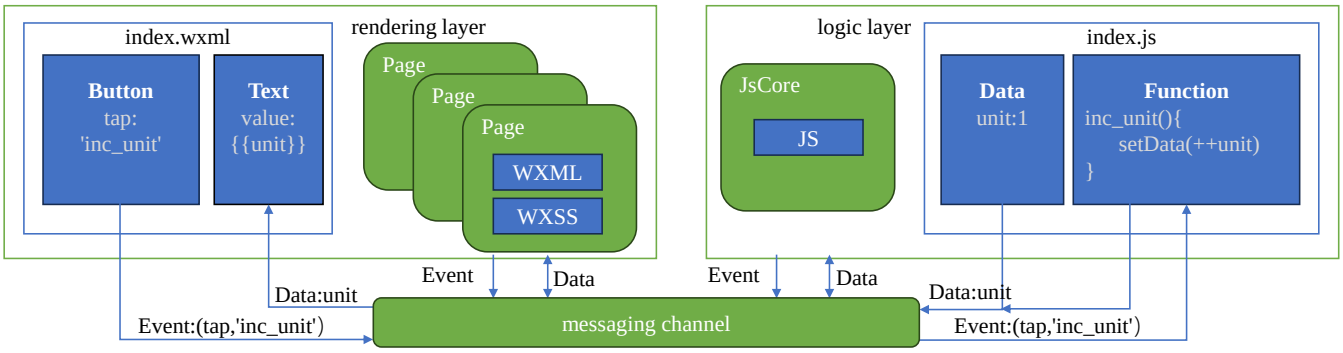

Fig. 2. Communications between the render layer and the logic layer.

II. WeChat Template

Figure 3 illustrates the template for WeChat Mini-Program, which involves the `WXML`, `JS` and `WXS` files. This template is derived from a comprehensive analysis of real-world mini-program samples and the official developer documentation [12]. It encapsulates the key building blocks of data communication between the rendering layer and the logic layer, including data channel, data type, page element, page position, and observation oracle.

• *Data Channel.* The data channel specifies the source (src), destination (dst) and manner (m) of data communication, represented as $src \overset{m}{\rightsquigarrow} dst$ for one-way channels and $src \overset{m}{\leftrightsquigarrow} dst$ for bi-directional channels. There are nine distinct types of data channels, each of which is described in detail below.

① `JS` $\overset{var}{\rightsquigarrow}$ `WXML` (Line 41 $\rightsquigarrow$ `WXML`) and ② `JS` $\overset{fun}{\rightsquigarrow}$ `WXML` (Lines 42 and 47 $\rightsquigarrow$ `WXML`) represent one-way channels, where the values of `JS` variables (var) and the return values of `JS` functions (fun) are transferred to the `WXML` page through initialization at page loading.

③ `JS` $\overset{upd}{\rightsquigarrow}$ `WXML` (Line 56 $\rightsquigarrow$ `WXML`) represent one-way channels, where data defined in the `JS` file is transferred to the page specified in the `WXML` file through initialization ($init$) at page loading and updates (upd) using the `setData` operation. ④ `JS` $\overset{upd}{\leftrightsquigarrow}$ `WXML` (Line 41 $\leftrightsquigarrow$ Line 34) represents a bi-directional channel, where user modifications on the `WXML` page are synchronized back to the corresponding `JS` variables, complementing the one-way channel.

⑤ `WXS` $\overset{var}{\rightsquigarrow}$ `WXML` (Line 3 $\rightsquigarrow$ `WXML`) and ⑥ `WXS` $\overset{fun}{\rightsquigarrow}$ `WXML` (Line 4 $\rightsquigarrow$ `WXML`) represent one-way channels, where the values of `WXS` variables (var) and the return values of `WXS` functions (fun) are transferred to the `WXML` page through initialization at page loading. ⑦ `JS` $\overset{var}{\rightsquigarrow}$ `WXS` $\overset{fun}{\rightsquigarrow}$ `WXML` (`JS` $\rightsquigarrow$ Line 9 $\rightsquigarrow$ `WXML`) represents a one-way transitive channel, where the values of JS variables (var) are passed as parameters to `WXS` functions (fun) and subsequently transferred to the `WXML` page through the functions' return values.

The aforementioned seven data channels are enabled through data binding. Additionally, two more data channels are facilitated through event binding. ⑧ `WXML` $\overset{ev}{\rightsquigarrow}$ `JS` (Line 35,36 $\rightsquigarrow$ Line 53) represents a one-way channel, where the event object and custom properties are transferred from the page element specified in the `WXML` file to the corresponding event handler defined in the `JS` file. ⑨ `WXML` $\overset{ev}{\leftrightsquigarrow}$ `WXS` (Line 35,36 $\rightsquigarrow$ Line 14 and Line 17 $\rightsquigarrow$ `WXML`) represents a bi-directional channel, where the data are transferred to the event handler defined in the `WXS` file (instead of the `JS` file), and the `WXS` event handler can, in turn, update the `WXML` page through direct DOM manipulation. The difference between Channel ⑧ and Channel ⑨ lies in the different capabilities of the scripting environments: `WXS` code has access to the DOM, while `JS` code does not.

• *Data Type.* The WeChat mini-program framework supports both primitive data types (including `number`, `boolean`, `string`) and reference data types (including `array`, `list`, `date`, `regexp`, `function`). Each data type has its corresponding utility methods designed for specific operations or transformations on its values. For example, the `join` method, associated with the `array` data type, returns a new string by concatenating all elements in the array, separated by commas

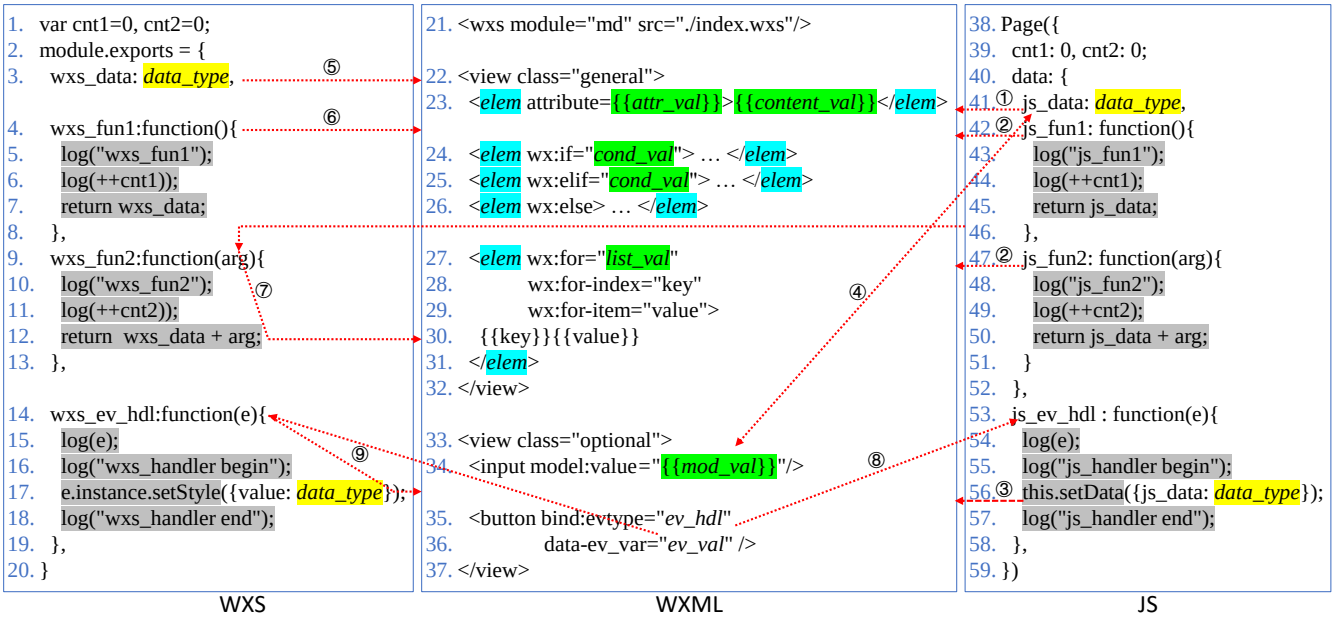

Fig. 3. Simplified template for testcase generation: red lines represent data channels, yellow blocks represent data types, blue blocks represent page elements, green blocks represent page positions, and grey blocks represent observation oracles. Event handler functions are referenced through event bindings, while other data and functions defined in `JS` or `WXS` can be referenced as values placed at any page position.

or a specified separator string. The data being transferred can either be a value of a specific data type or the result of applying a utility method associated with that data type to a value. During cross-layer communication, the data is serialized before transmission and deserialized upon receipt. Different data types adhere to distinct (de)serialization rules.

• *Page Element.* Data binding and event binding are applied to the page elements, which serve as containers or interactive components that display content or capture user input. The WeChat mini-program platform provides various page elements, including `<text>`, `<button>`, `<input>`, etc. Each element comes with a set of attributes that define its appearance, behavior, and interaction with other elements. Some of these attributes are shared by all elements (e.g., "`class`", "`style`"), while others are unique to specific elements (e.g., "`user-select`" for `<text>`, "`bind:tap`" for `<button>`, "`value`" for `<input>`). When selecting a page element, it is important to take its attributes into account.

• *Page Position.* The page position determines where the data transferred from the `JS` or `WXS` code will be displayed on the `WXML` page. The available positions can be specified either as attribute values or as element content (as illustrated in green blocks at Line 23). The transferred data can also serve as inputs for two key directives that control the rendering of dynamic content. The `wx:if` directive (Line 24) conditionally renders elements based on a given condition, with `wx:elif` (Line 25) defining an alternative condition and `wx:else` (Line 26) handling all remaining cases. The `wx:for` directive (Line 27) enables developers to loop through a list and display each item as part of the content of the containing element, with `wx:for-index` (Line 28) and `wx:for-item` (Line 29) specifying the variable names for the key and the value of the current list item, respectively. Additionally, the data can also be transferred to the `<input>` (Line 34) page element.

• *Observation Oracle.* The oracle facilitates the observation of mini-program behaviors for discrepancy analysis. Currently, we support four types of observation oracles, including data value monitor, event object inspector, execution trace logger, and function invocation tracker. Additional oracles can be integrated as needed. The data value monitor (Lines 7, 12, 17, 45, 50, 56) simply outputs the value of the focused data, while the other observation oracles convert non-data behaviors into data values for output. Specifically, the event object inspector (Lines 15 and 54) extracts the dataset from the event object. The execution trace logger (Lines 5, 10, 16, 18, 43, 48, 55, 57) records the order of code execution. The function invocation tracker (Lines 6, 11, 44, 49) counts the number of times a specific function is invoked.

III. Alipay Domain Knowledge

Alipay Mini-Program. A Alipay mini-program [4] primarily consists of five types of files that work together to create a seamless user experience: (1) `JSON` files for configuration, such as setting page paths, window appearance, and permissions; (2) `AXML` files defining the layout and structure of page views, similar to HTML in Web applications; (3) `ACSS` files responsible for the styling and design of page views, much like CSS in Web applications; (4) `JS` files that contain the logic driving the mini-program's functionality, handling data management, user interactions, and dynamic behaviors; and (5) `SJS` files providing a Alipay-specific scripting language for efficient data processing within page views.

```
1. "pages": ["pages/index/index"],
2. "renderer": "native",
3. "useDynamicPlugins": true,
4. "lazyCodeLoading": "requiredComponents"
```

(a) app.json - app configuration

```
5. <import-sjs
6.     name="md" from="./index.sjs"/>
7. <text>{{ md.cal_sum(unit) }}</text>
8. <button bind:tap="inc_unit">inc</button>
```

(b) index.axml - index page content

```
15. Page({
16.   data: {unit: 1},
17.
18.   inc_unit:function(){
19.     this.setData({
20.       unit: ++this.data.unit
21.     });
22.   },
23. })
```

(e) index.js - index page script

```
9.   page { width: 100px; height: 167px; }
10. text { padding: 30%; font-size: large; }
11. button { width: 100%; text-align: center; }
```

(c) index.acss - index page style

```
12. var sum = 0;
13. export function cal_sum (unit){
14.   return sum += unit; }
```

(d) index.sjs - Safe/Subset JavaScript

Fig. 4. An example Alipay mini-program that implements a basic accumulator.

Figure 4 illustrates an example Alipay mini-program that implements a basic accumulator. The `app.json` file (Figure 4a) specifies that the mini-program contains a single `index` page, located in the `pages/index/` directory (Line 1), which serves as the main entry point. The `index.axml` file (Figure 4b) defines the page view, featuring a text component to display the cumulative sum (Line 7) and a button component to increment the unit value (Line 8). The `index.acss` file (Figure 4c) controls the appearance of the page, including the styling of the text (Line 10) and button (Line 11) components. The `index.js` file (Figure 4e) initializes the unit value variable (`unit` at Line 16) and defines an event handler (`inc_unit` at Lines 18-22) that increments the unit value. The `index.sjs` file (Figure 4d) defines a cumulative sum variable (`sum` at Line 12) and exports a data processing function (`cal_sum` at Lines 13-14) that calculates the the sum of the unit value and the existing cumulative sum, then displays the result in the page view.

Alipay Mini-Program Framework. The Alipay mini-program framework [4] also consists of two distinct layers like Wechat Mini-Program: the rendering layer and the logic layer. Figure 2 can also illustrates the communication of events and data in the example Alipay mini-program. Please refer to Chapter Wecha Mini-Program Framework for an introduction to this aspect.

Configurations and Platforms. Alipay provides two render mode: Native Rendering and Web Rendering. According to the official description [10]: Native rendering engine is a new ability to further optimize the performance of Mini programs and provide a near-native app experience. Native rendering is more efficient than WebView rendering and is compatible with most styles and layouts supported by WebView rendering. In addition, Native rendering makes it possible to expand enhancement features, which will further enrich the interactive capabilities of Mini programs and enhance the user experience. Developers can enable Native rendering for all pages/some specified pages of Mini programs, by configuring the render option in app.json. Alipay Mini-Programs can also run on various platforms, including Android, iOS and DevTools.

IV. Alipay Template

Figure 5 illustrates the template for Alipay Mini-Program, which involves the `AXML`, `JS` and `SJS` files. This template is derived from a comprehensive analysis of real-world mini-program samples and the official developer documentation [4]. It encapsulates the key building blocks of data communication between the rendering layer and the logic layer, including data channel, data type, page element, page position, and observation oracle.

• *Data Channel.* The data channel specifies the source (src), destination (dst) and manner (m) of data communication, represented as $src \overset{m}{\rightsquigarrow} dst$ for one-way channels and $src \overset{m}{\leftrightsquigarrow} dst$ for bi-directional channels. Differently there are only seven distinct types of data channels(without ② and ③ in WeChat), each of which is described in detail below.

① `JS` $\overset{var}{\rightsquigarrow}$ `AXML` (Line 42 $\rightsquigarrow$ `AXML`) represent one-way channels, where the values of `JS` variables (var) and the return values of `JS` functions (fun) are transferred to the `AXML` page through initialization at page loading.

② `JS` $\overset{upd}{\rightsquigarrow}$ `AXML` (Line 47 $\rightsquigarrow$ `AXML`) represent one-way channels, where data defined in the `JS` file is transferred to the page specified in the `AXML` file through initialization ($init$) at page loading and updates (upd) using the `setData` operation.

③ `SJS` $\overset{var}{\rightsquigarrow}$ `AXML` (Line 2 $\rightsquigarrow$ `AXML`) and ④ `SJS` $\overset{fun}{\rightsquigarrow}$ `AXML` (Line 3 $\rightsquigarrow$ `AXML`) represent one-way channels, where the values of `SJS` variables (var) and the return values of `SJS` functions (fun) are transferred to the `AXML` page through initialization at page loading. ⑤ `JS` $\overset{var}{\rightsquigarrow}$ `SJS` $\overset{fun}{\rightsquigarrow}$ `AXML` (`JS` $\rightsquigarrow$ Line 8 $\rightsquigarrow$ `AXML`) represents a one-way transitive channel, where the values of JS variables (var) are passed as parameters to `SJS` functions (fun) and subsequently transferred to the `AXML` page through the functions' return values.

The aforementioned seven data channels are enabled through data binding. Additionally, two more data channels are facilitated through event binding. ⑥ `AXML` $\overset{ev}{\rightsquigarrow}$ `JS` (Line 35,36 $\rightsquigarrow$ Line 44) represents a one-way channel, where the event object and custom properties are transferred from the page element specified in the `AXML` file to the corresponding event handler defined in the `JS` file. ⑦ `AXML` $\overset{ev}{\leftrightsquigarrow}$ `SJS` (Line 35,36 $\rightsquigarrow$ Line 13 and Line 16 $\rightsquigarrow$ `AXML`) represents a bi-directional channel, where the data are transferred to the event handler defined in the `SJS` file (instead of the `JS` file), and the `SJS` event handler can, in turn, update the `AXML` page through direct DOM manipulation. The difference between Channel ⑥ and Channel ⑦ lies in the different capabilities of the scripting

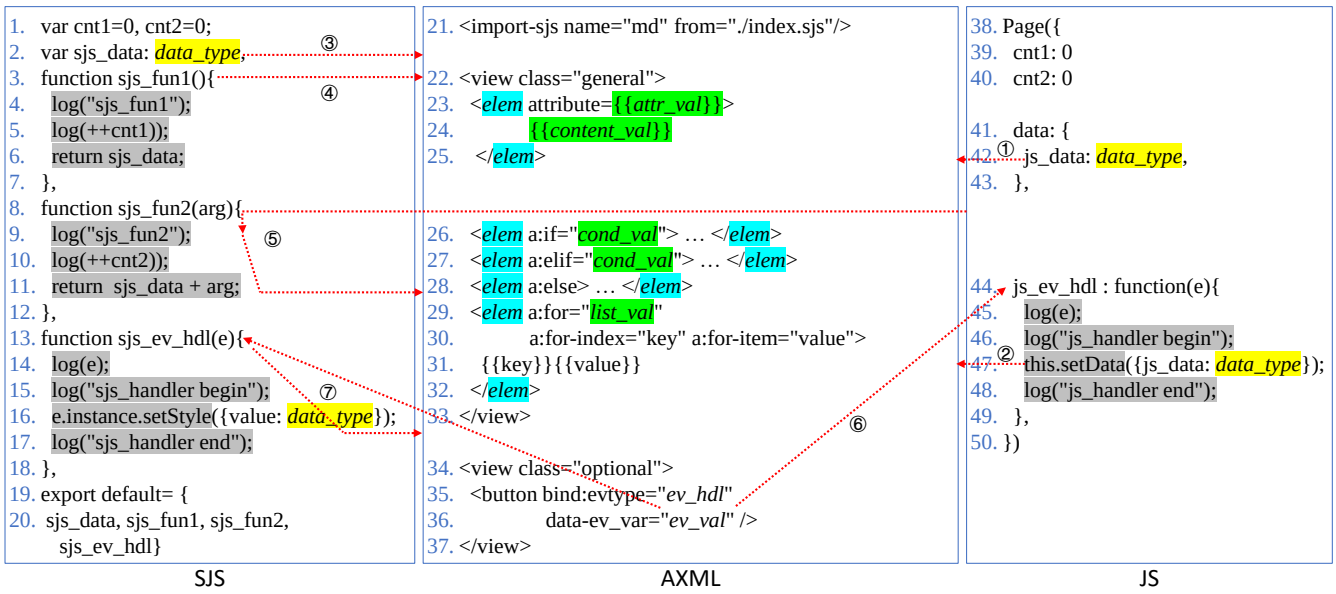

Fig. 5. Simplified template for testcase generation: red lines represent data channels, yellow blocks represent data types, blue blocks represent page elements, green blocks represent page positions, and grey blocks represent observation oracles. Event handler functions are referenced through event bindings, while other data and functions defined in `JS` or `SJS` can be referenced as values placed at any page position.

environments: `SJS` code has access to the DOM, while `JS` code does not.

- *Data Type.* Same as WeChat.
- *Page Element.* Same as WeChat.
- *Page Position.* The page position determines where the data transferred from the `JS` or `SJS` code will be displayed on the `AXML` page. The available positions can be specified either as attribute values or as element content (as illustrated in green blocks at Line 23 or 24). The transferred data can also serve as inputs for two key directives that control the rendering of dynamic content. The `a:if` directive (Line 26) conditionally renders elements based on a given condition, with `a:elif` (Line 27) defining an alternative condition and `a:else` (Line 28) handling all remaining cases. The `a:for` directive (Line 29) enables developers to loop through a list and display each item as part of the content of the containing element, with `a:for-index` and `a:for-item` (Line 30) specifying the variable names for the key and the value of the current list item, respectively.
- *Observation Oracle.* Same as WeChat.

V. TikTok Domain Knowledge

TikTok Mini-Program. A TikTok mini-program [5] primarily consists of five types of files that work together to create a seamless user experience: (1) `JSON` files for configuration, such as setting page paths, window appearance, and permissions(Because TikTok Mini-Programs only have one configuration, there isn't render option); (2) `TTML` files defining the layout and structure of page views, similar to HTML in Web applications; (3) `TTSS` files responsible for the styling and design of page views, much like CSS in Web applications; (4) `JS` files that contain the logic driving the mini-program's functionality, handling data management, user interactions, and dynamic behaviors; and (5) `SJS` files providing a TikTok-specific scripting language for efficient data processing within page views.

Figure 6 illustrates an example TikTok mini-program that implements a basic accumulator. The `app.json` file (Figure 6a) specifies that the mini-program contains a single `index` page, located in the `pages/index/` directory (Line 1), which serves as the main entry point. The `index.TTML` file (Figure 6b) defines the page view, featuring a text component to display the cumulative sum (Line 3) and a button component to increment the unit value (Line 4). The `index.TTSS` file (Figure 6c) controls the appearance of the page, including the styling of the text (Line 6) and button (Line 7) components. The `index.js` file (Figure 6e) initializes the unit value variable (`unit` at Line 12) and defines an event handler (`inc_unit` at Lines 13-17) that increments the unit value. The `index.sjs` file (Figure 6d) defines a cumulative sum variable (`sum` at Line 8) and exports a data processing function (`cal_sum` at Lines 9-10) that calculates the the sum of the unit value and the existing cumulative sum, then displays the result in the page view.

TikTok Mini-Program Framework. The TikTok mini-program framework [3] also consists of two distinct layers like Wechat Mini-Program: the rendering layer and the logic layer. Figure 7 is the TikTok official framework figure, illustrates the communication of events and data in the TikTok mini-program.

Configurations and Platforms. TikTok Mini-Programs can also run on various platforms, including Android, iOS and DevTools. But TikTok Mini-Programs only have one configuration.

```
1. "pages": ["pages/index/index"],
```
(a) app.json - app configuration

```
2.  <sjs module="md" src="./index.sjs"/>
3.  <text>{{ md.cal_sum(unit) }}</text>
4.  <button bind:tap="inc_unit">inc</button>
```
(b) index.ttml - index page content

```
11. Page({
12.  data: {unit: 1},
13.  inc_unit:function(){
14.   this.setData({
15.    unit: ++this.data.unit
16.   });
17.  },
18. })
```
(e) index.js - index page script

```
5.  page { width: 100px; height: 167px; }
6.  text { padding: 30%; font-size: large; }
7.  button { width: 100%; text-align: center; }
```
(c) index.ttss - index page style

```
8.  var sum = 0;
9.  module.exports.cal_sum =
10.   function(unit){ return sum += unit; }
```
(d) index.sjs - index page TikTok script

Fig. 6. An example WeChat mini-program that implements a basic accumulator.

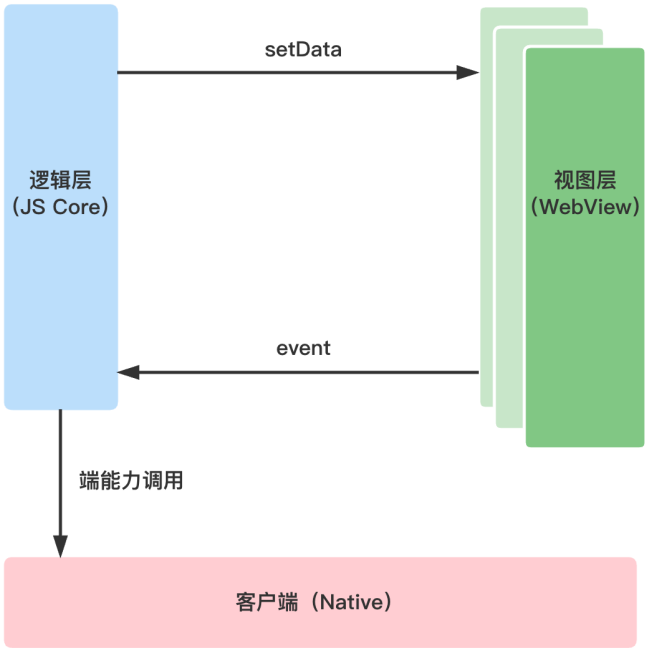

Fig. 7. Communications between the render layer and the logic layer.

VI. TikTok Template

Figure 8 illustrates the template for TikTok Mini-Program, which involves the TTML, JS and SJS files. This template is derived from a comprehensive analysis of real-world mini-program samples and the official developer documentation [5]. It encapsulates the key building blocks of data communication between the rendering layer and the logic layer, including data channel, data type, page element, page position, and observation oracle.

• *Data Channel.* The data channel specifies the source (src), destination (dst) and manner (m) of data communication, represented as $src \overset{m}{\rightsquigarrow} dst$ for one-way channels and $src \overset{m}{\leftrightsquigarrow} dst$ for bi-directional channels. Differently there are only seven distinct types of data channels(without ② and ③ in WeChat), each of which is described in detail below.

① JS $\overset{var}{\rightsquigarrow}$ TTML (Line 42 $\rightsquigarrow$ TTML) represent one-way channels, where the values of JS variables (var) and the return values of JS functions (fun) are transferred to the TTML page through initialization at page loading.

② JS $\overset{upd}{\rightsquigarrow}$ TTML (Line 47 $\rightsquigarrow$ TTML) represent one-way channels, where data defined in the JS file is transferred to the page specified in the TTML file through initialization ($init$) at page loading and updates (upd) using the setData operation.

③ SJS $\overset{var}{\rightsquigarrow}$ TTML (Line 3 $\rightsquigarrow$ TTML) and ④ SJS $\overset{fun}{\rightsquigarrow}$ TTML (Line 4 $\rightsquigarrow$ TTML) represent one-way channels, where the values of SJS variables (var) and the return values of SJS functions (fun) are transferred to the TTML page through initialization at page loading. ⑤ JS $\overset{var}{\rightsquigarrow}$ SJS $\overset{fun}{\rightsquigarrow}$ TTML (JS $\rightsquigarrow$ Line 9 $\rightsquigarrow$ TTML) represents a one-way transitive channel, where the values of JS variables (var) are passed as parameters to SJS functions (fun) and subsequently transferred to the TTML page through the functions' return values.

The aforementioned seven data channels are enabled through data binding. Additionally, two more data channels are facilitated through event binding. ⑥ TTML $\overset{ev}{\rightsquigarrow}$ JS (Line 36 $\rightsquigarrow$ Line 44) represents a one-way channel, where the event object and custom properties are transferred from the page element specified in the TTML file to the corresponding event handler defined in the JS file. ⑦ TTML $\overset{ev}{\leftrightsquigarrow}$ SJS (Line 36 $\rightsquigarrow$ Line

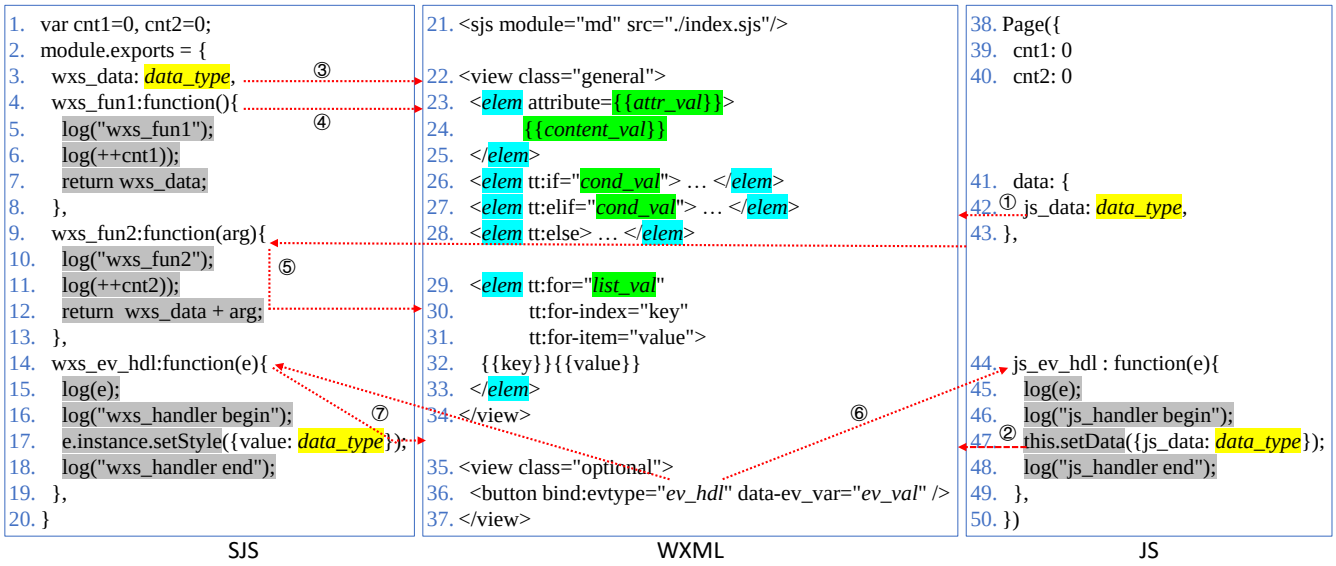

Fig. 8. Simplified template for testcase generation: red lines represent data channels, yellow blocks represent data types, blue blocks represent page elements, green blocks represent page positions, and grey blocks represent observation oracles. Event handler functions are referenced through event bindings, while other data and functions defined in `JS` or `SJS` can be referenced as values placed at any page position.

14 and Line 17 ⇝ `TTML`) represents a bi-directional channel, where the data are transferred to the event handler defined in the `SJS` file (instead of the `JS` file), and the `SJS` event handler can, in turn, update the `TTML` page through direct DOM manipulation. The difference between Channel ⑥ and Channel ⑦ lies in the different capabilities of the scripting environments: `SJS` code has access to the DOM, while `JS` code does not.

- *Data Type.* Same as WeChat.
- *Page Element.* Same as WeChat.
- *Page Position.* The page position determines where the data transferred from the `JS` or `SJS` code will be displayed on the `TTML` page. The available positions can be specified either as attribute values or as element content (as illustrated in green blocks at Line 23 or 24). The transferred data can also serve as inputs for two key directives that control the rendering of dynamic content. The `tt:if` directive (Line 26) conditionally renders elements based on a given condition, with `tt:elif` (Line 27) defining an alternative condition and `tt:else` (Line 28) handling all remaining cases. The `tt:for` directive (Line 29) enables developers to loop through a list and display each item as part of the content of the containing element, with `tt:for-index` and `tt:for-item` (Line 30) specifying the variable names for the key and the value of the current list item, respectively.
- *Observation Oracle.* Same as WeChat.

REFERENCES

[1] Virtual dom. https://en.wikipedia.org/wiki/Virtual$_DOM$, 2024.

[2] A virtual dom and diffing algorithm. https://github.com/Matt-Esch/virtual-dom, 2024.

[3] Alipayminiapp framework. https://developer-open–douyin-com.translate.goog/docs/resource/zh-CN/mini-app /develop/tutorial/miniapp-framework/introduction?_x_tr_sl=zh-CN_x_tr_tl=en_x_tr_hl=zh-CN_x_tr_pto=wapp, 2025.

[4] Code composition of a alipayminiapp. https://opendocs-alipay-com.translate.goog/mini/framework/overview?pathHash=296cdccf _x_tr_sl=auto_x_tr_tl=en_x_tr_hl=zh-CN_x_tr_pto=wapp, 2025.

[5] Code composition of a alipayminiapp. https://developer-open–douyin-com.translate.goog/docs/resource/zh-CN/mini-app /develop/tutorial/beginner/todo-list-app-dev?_x_tr_sl=zh-CN_x_tr_tl=en_x_tr_hl=zh-CN_x_tr_pto=wapp, 2025.

[6] Code composition of a mini program. https://developers.weixin.qq.com/miniprogram/en/dev/framework/ quickstart/code.html, 2025.

[7] Exparser. https://github.com/wechat-miniprogram, 2025.

[8] Glasseasel framework. https://github.com/wechat-miniprogram/glass-easel, 2025.

[9] Mini program host environment. https://developers.weixin.qq.com/ miniprogram/en/dev/framework/quickstart/framework.html, 2025.

[10] Natie rendering for alipayminiapp. https://opendocs-alipay-com.translate.goog/mini/0ai07p?pathHash=01051631 _x_tr_sl=zh-CN_x_tr_tl=en_x_tr_hl=zh-CN_x_tr_pto=wapp, 2025.

[11] Skyline. https://github.com/wechat-miniprogram/awesome-skyline, 2025.

[12] Weixin mini program design guidelines. https://developers.weixin.qq.com/miniprogram/en/design/, 2025.