October 2019

USER INTERFACE BASED APPROACH TO USER DATA COLLECTION FOR BUSINESS-FOCUSED AND CONTEXTUAL ENRICHMENT OF BROWSER PAGE PERFORMANCE DATA

Puneet Anand

USER INTERFACE BASED APPROACH TO USER DATA COLLECTION FOR BUSINESS-FOCUSED AND CONTEXTUAL ENRICHMENT OF BROWSER PAGE PERFORMANCE DATA

AUTHORS:
Puneet Anand

ABSTRACT

Techniques are described for capturing business-data values automatically using a configuration User Interface (UI) on a browser Real User Monitoring (RUM) product. These techniques may allow customers to save time and also enhance product adoption. The data obtained from the UI may be analyzed independently and may provide context to performance data.

DETAILED DESCRIPTION

One of the main problems in setting up front-end monitoring is the challenging/difficult configuration of the JavaScript® agent. This requires JavaScript knowledge, access to modifying source code documents, and coding skills to instrument applications. To extract and report contextual data from their applications, customers have to write some JavaScript instrumentation code using a JavaScript Agent and then update their web pages with this code. Oftentimes these customers are Information Technology Operations (IT OPS) specialists or in other roles that do not have access to the code like their development counterparts.

Described herein is a configuration User Interface (UI) to enable customers to specify data to be collected. Pages on customer interfaces may also be selected to collect the data using the same configuration UI. As a result, customers may add and improve the configuration and management of custom contextual data without requiring assistance from software development teams. Users may capture data that adds user context (and, in turn, business context when aggregated) to the already-available performance-oriented data captured by the same agent. This may enable improved decision-making and more efficient use of time and effort. Today, on the browser Real User Monitoring (RUM) side, there is no web browser or JavaScript-based custom data collection that exists in the industry that allows remote automatic instrumentation.

For example, a customer in the insurance field may wish to capture the branch identifier, which is passed along from the server to their application interface. Or a customer in the financial space may wish to capture member tier or status and/or transaction value for analysis with statistics regarding high error rates. Or in still another example, a customer who has a web site with thousands of daily users may wish to determine the different types of business-relevant data values that are being displayed on user screens as they load the interface. That customer may also wish to understand the context of these business-relevant data values to understand the impact of different performance issues in the application UI or by means of business transaction correlation, into the back-end.

Today, this data is obtainable only by instrumenting every single such data point. Depending on how the organization is structured, it may be necessary to navigate through a large number of organizational layers to obtain the instrumentation code running live on the application. This can often take days or even weeks.

Described herein are techniques to add (e.g., create, read, update, delete, manage, etc.) dynamic and custom data instrumentation on the configuration UI. Upon saving, this configuration may be transmitted in a matter of seconds to deploy JavaScript agents on the users' UIs. The JavaScript agents may then start collecting the data (e.g., custom business context data points) immediately.

There may be many ways to capture custom business relevant data. Figure 1 below illustrates an example Document Object Model (DOM) element selector. The UI may enable customers to add an arbitrary DOM selector using the DOM identifier, for example. The JavaScript agent may be enabled to read and capture values on those selectors.

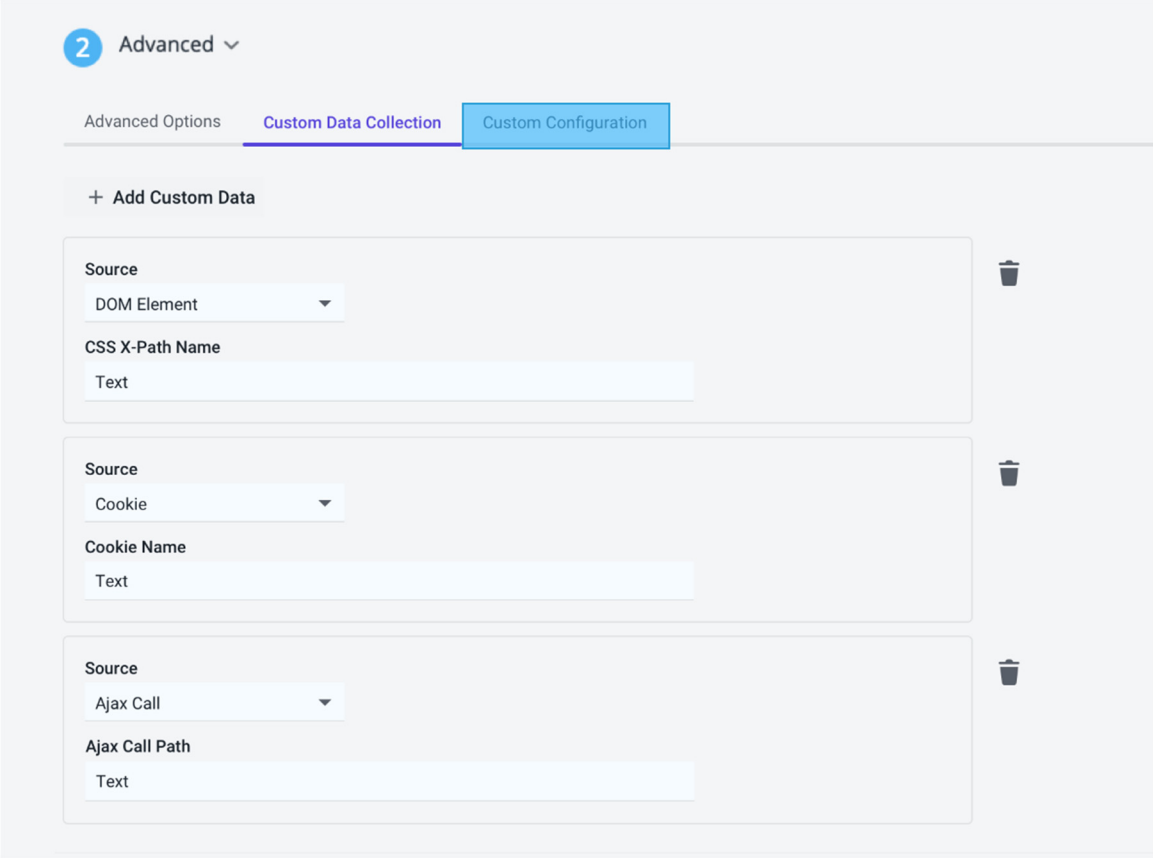

Figure 1

Figure 2 below illustrates how DOM element selectors may be captured from an example live website using browser plugins and then fed into the aforementioned configuration.

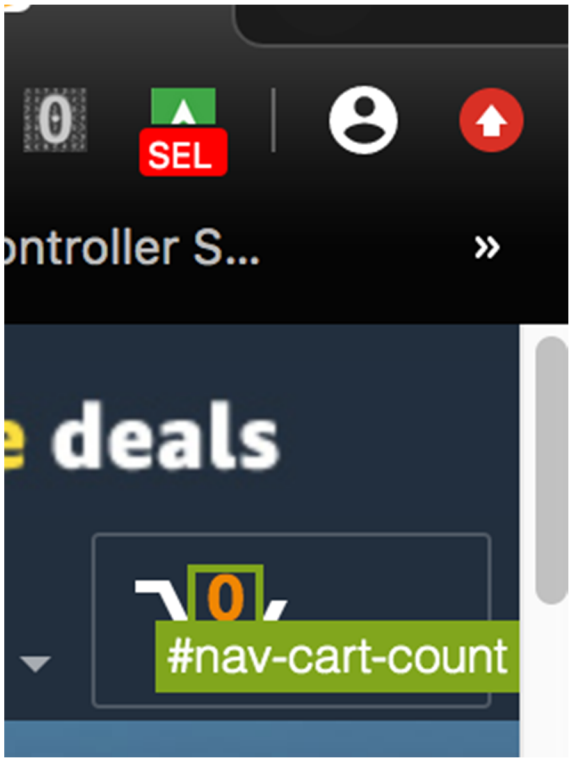

Figure 2

Apart from DOM selectors, modern browsers allow loaded applications to store state information in browser memory and browser applications commonly make use of cookies. The JavaScript agent may read from the storage or cookie values given the aforementioned settings including the appropriate key.

Modern applications deal with asynchronous data fetching using Asynchronous JavaScript and eXtensible Markup Language (XML) (AJAX). The system may select the correct data values from the returned payloads given particular calls and properties.

Techniques described herein may allow customers to edit the JavaScript agent configuration and data collection function through a controller UI. Figure 3 below illustrates an example overview of the high-level system that can serve this purpose.

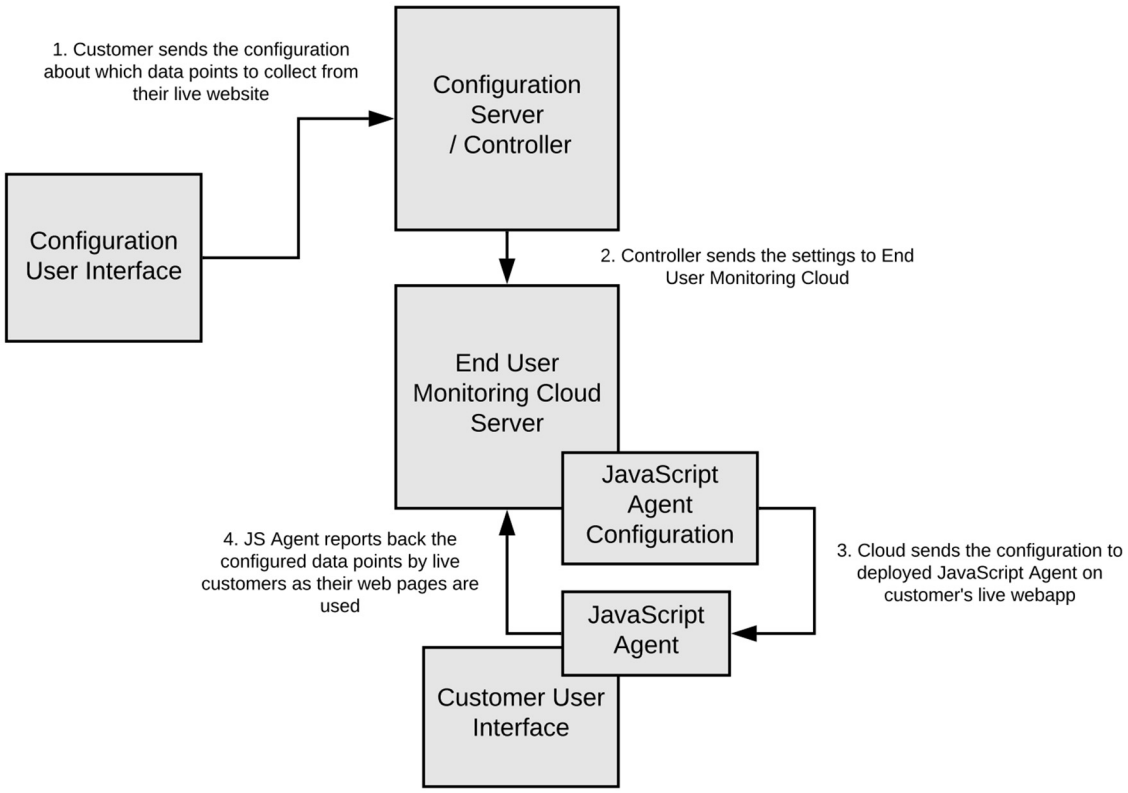

Figure 3

A browser RUM customer may configure the JavaScript agent configuration properties and data collection function through the controller UI (Step 1). The configuration may be stored in a controller database and pass it on to the End User Monitoring (EUM) cloud (Step 2). The customer may update their web page(s) through server driven automatic or manual injection of the JavaScript agent. In case of automatic instrumentation, the agent is hosted in their web server with the customer configuration and is loaded with the customer-specified web application pages (Step 3). When an end user visits and uses the customer web site, the agent automatically captures the values in the DOM selectors / cookie values / AJAX payloads / local memory and generates beacons containing those values that are sent over to the EUM cloud, which processes the browser beacons and collects and aggregates the values sent over in a database (Step 4).

The beacon format may include a page type value for a customer event. The customer event may have an event identifier field and a parent identifier field that points to the base page / virtual page as the parent of the customer event. The customer data may be in the same field of the base page / virtual page / iframe / AJAX custom data fields. Beacon parsing and validation may enable the system to detect and accept the page type and, for customer events, relax requirements on metrics.

The system may further accept, validate, and persist customer events/data into a store, and allow customers to search and view customer events/data in the controller's web analytics or sessions features.

In summary, techniques are described for capturing business-data values automatically using a configuration UI on a browser RUM product. These techniques may allow customers to save time and also enhance product adoption. The data obtained from the UI may be analyzed independently and may provide context to performance data.