10/18/2017

MEISTER SMART™ BUSINESS APPLICATIONS FOR SAP WHITE PAPERS : SMART SINGLE CALL

Product Document v1.0

Andre Rosenthal

www.s4meister.com

Gateway Architects Product Document

Document Name:	Meister Suite Whie Papers – Smart Single Call
Document ID:	WHPSR0001
Document Owner:	Andre Rosenthal
Document Version:	V1.0
Document Date:	10/18/2017
Document Status:	Released

Table of Contents

Document Control

Deliverable Information

Deliverable Name	***Meister Suite White Papers – Smart Single Calls***
Author	***Gateway Architects***
Status	V1.0
Location	

Revision History

Version No.	Date	Author	Revision Description
V1.0	10/18/2017	ARosenthal	Initial version

Authorisers for sign off

Role	Name	Date Signed Off in SharePoint
CTO	Andre Rosenthal	10/18/2017

Distribution - in addition to Authorisers

Name	Role
Yaron Meefeld	CEO
William Mays	Chairman
Craig Twiss	President, US

Reference Documents

Document	Location

Overview

The notion of a single call capable of relating back to the UX, in case of users, or consumer application, in case of integration, is on itself present at SAP natively from the standpoint of consumption of SAP designed BAPIs and custom-made function modules. Due to the nature of the RFC protocol used by SAP, the best that could be achieved is the return of discrete tables with referential integrity foreign keys spread about these tables leaving the consumption end with the burden of understanding the relationships between the primary and foreign keys as well as the nature of the data needed to be either transposed or altered.

SAP Gateway 2.0's OData protocol-bound services are enabled with a function that allows the product to glue back together the referential keys and return a complex payload as if it was a single call, but that is far from being optimal, both in case of performance as well as computational gains.

This white paper describes Meister's Smart Single Call, hereinafter SSC, and the processes taken by the Meister Suite to totally remove the need for protocol-bound efforts and return a response payload which is on itself, fully referential, complete, and ready to be use by the consumption end without loss of computation power and performance.

Streamlining the call structure

For SSC to become powerful, it ought to be both computational and performance oriented. Natively the SAP Gateway 2.0 provides code hooks for programmers to glue back discrete calls done at the backend and create associations where complex structures like Purchase Order Header and its line items could be consumed.

In order to understand the loss of computational power and performance of the association method we need to explore further the ways association takes place.

For that, we translate the association process into a mathematical equivalent as follows:

Let $B(I,E,T,C)$ be a set describing an SAP BAPI, where each element of the set is itself a set $S(i)$ such that $\forall\, i \,\exists\, x \rightarrow y$. Let the set S represent the mapping between each individual structure being used by the BAPI such that for each component of the structure there exist a map between x as the field name as y as the field type.

Let

$$\boldsymbol{S = \sum_{i>0}^{n} B(I,E,T,C)} \qquad \textbf{Equation 1}$$

be the total cost of invoking a given BAPI where there is a mapping between the Imports, Exports, Tables, and Change structures defined according to $S(i)$.
By simple replacement, we can see that equation 1 above is transformed into:

$$\boldsymbol{S} = \sum_{i>0}^{n} \boldsymbol{B}(\sum \boldsymbol{I}(\boldsymbol{i})), \sum \boldsymbol{E}(\boldsymbol{e})), \sum \boldsymbol{T}(\boldsymbol{t}), \sum \boldsymbol{C}(\boldsymbol{c})) \qquad \textbf{Equation 2}$$

which indicates that, for each element found on individual $S(i)$ such as $I(i)$ there could be one or many associations mapped by the developer. When the content of discrete $S(i)$ are referencing yet another $S(j)$ the process gets the asymptotic runtime of $O(n^m)$ which on the average case represents a loop within a loop within yet another loop, and so on.

As defined by SAP, the OData protocol returns each individual $S(i)$ on a discrete call represented by an Entity Set, which is then stored as a temporary table at the Gateway 2.0 node, and yet another call is needed (mostly repeating the exact same code done before) to retrieve another subset $S(i)$ of S.

In the nutshell, the association process repeats the call to the backend BAPI a number of times depending on how deep the association is and how complex the call may be. The loss of computation power is clearly visible now given that a non-polynomial (exponential) asymptotic runtime is being executed. As the depth of the original association grows, say 4 levels deep, the original call is repeated by the depth, in our example 4, where only one such call would have been sufficient had the protocol be capable to returning anything besides an Entity Set.

Gluing the pieces together: Meister

Meister eliminates all the extra iterations of the same BAPI at the backend by allowing the ABAP developer to iterate the referential keys as needed by the consumption end, and create a complex structure directly in ABAP where table of tables could be defined. The way to eliminate the single table requirement is the elimination of the RFC dependency, which Meister replaces with a simple call to a non-RFC function module. In Meister, the calls are always sent and received as Json Documents rather than OData Entity Sets and Entities. With this shift in engineering, Meister eliminates the need for Entity Sets altogether and replaces these with Json Documents that are easily consumed by any receiving end, from UX engines to script languages.

Meister improves the response time in both aspects:

- By removing the need for association, it changed the asymptotic runtime from exponential to polynomial.

- By removing the dependency on OData Entity Sets and Entities, it changed number of calls to the backend from the depth of the association to a single call.

