BRFplus - Performance and Sizing

Applies to:

Business Rule Framework plus (BRFplus) shipped with SAP NetWeaver 7.0 Enhancement Package 2.For a detailed introduction to BRFplus please visit Business Rules Management at the SDN network and select 'BRFplus' in the Related Content section.

Summary

This paper gives an overview about performance and sizing in BRFplus. It explains how BRFplus works and how it can be used in high performance scenarios. A list of do's and don'ts is provided.

Authors: Carsten Ziegler, Hans Beuter

Company: SAP AG

Created on: May 14th, 2010

About the Authors

Carsten Ziegler is the Lead Architect and Product Owner of Business Rule Framework plus. He joined SAP in 2000. Since then he has been working in various projects as a developer, development architect and project lead.

Hans-Georg Beuter is a senior developer in the BRFplus team. He is the performance expert and responsible for the code generation framework in BRFplus. Hans joined SAP in 1992 and since then he has been involved in various projects in application and framework development.

Table of Contents

Introduction

The motivation to use BRFplus is based on different aspects depending on the role of the involved persons:

- Business experts
- IT experts

The main interest of a business analyst is to empower the business users. This refers mainly to the intuitive graphical approach which is used in BRFplus to allow business users compose rules and define business logic without the need to create program code. Business logic includes data validation, calculation steps, triggering actions, decision services, process automation, etc. Also important is the ability to understand the rule evaluation or have legal evidence for automated decisions performed with BRFplus.

Usually, business analysts do not care for technical details such as sizing and performance considerations. They just take it for granted that these questions are answered by the IT personnel installing and operating the software. The purpose of this document is to help IT experts understand the implications of using BRFplus for various use cases in their IT landscape in terms of performance and sizing.

BRFplus as a Part of SAP NetWeaver Application Server

BRFplus is part of the SAP NetWeaver Application Server. It is optimized for the ABAP stack supporting all of the proven and well-known concepts of the ABAP server such as

- ABAP Objects API (for rule composition and execution)
- Generation of ABAP code, RFC function modules, and Web Services
- DDIC integration (reuse and binding to DDIC types, rule access to database tables)
- Change and Transport System (automatic and manual transport recording, evaluation of client settings)
- Client concept (local, client-dependent, and client-independent rule content)
- User interface in WebDynpro for ABAP

Being part of SAP NetWeaver Application Server allows for direct communication with the applications using BRFplus. Installation of software is not required, neither on client nor on server side. The same holds for setting up network connections. Instead, any ABAP-based application can directly connect with BRFplus using a local API. For remote calls, the RFC generator may be used. RFC-enabled function modules are very efficient compared to Web Services.

Business Rule Composition

Business rule composition in BRFplus is done with the BRFplus workbench (transactions BRF+ or BRFplus). The BRFplus workbench is a WebDynpro-based user interface that can be used with a web browser. Additional installation of client software is not required.

The rule content can be versioned. In case versioning is switched on, all changes to an active object leads to a new inactive version being created for that specific object. The subsequent changes made by a user are applied to the inactive version until activation. After activation, again, a new inactive version is needed for any changes. The objects in BRFplus are versioned independently. For example, consider a ruleset containing a rule that calls a formula. Changes in the formula will lead to new versions of the formula but not of the ruleset or the rule. In case you have a lot of rule content that is versioned and has been continuously changed over time, you may have lots of historic versions of rule artifacts in your database. Although BRFplus provides tools to remove historic versions or to archive content by exporting it as XML files, our assumption is that in most use cases the database size will not exceed the range of some megabytes. We therefore do not expect that database size will become an issue.

Business Rule Execution

Business rule execution is done with the API (package SFDT). There are several possibilities to call BRFplus (see following chapters). BRFplus uses a code generation approach for maximum performance. The first call per session retrieves the appropriate class name from the database and loads the class into the ABAP memory. In case there is no generated class yet, a new one will be generated automatically. Then, a method for rule evaluation in the generated class is called. Once the class is loaded into memory, no database access is needed anymore during the session. Instead, the method is called directly by the BRFplus API. In

mass data scenarios, this approach is not much different than calling a hand-coded ABAP method multiple times. The memory consumption of the generated class is usually below 100 kilobytes. BRFplus does not need any additional memory. Using BRFplus very close to the data on which the rules operate makes sure that no additional memory and runtime is needed for data retrieval and copying steps.

Approach

The example (see chapter *Example*) used for the measurements is very simple. However, even more complex examples do not necessarily lead to significant changes in the measurement results. It is hardly possible to measure the runtime of 50 more lines of code with ABAP IF or CASE statements. This corresponds to some 10 additional lines in a decision table or 20 rules in a ruleset. However, it is very easy to double and triple the runtime by just adding a rule with a SELECT statement bypassing buffers and indices. Also, the way how the BRFplus API is used for the processing of a function can make a big difference. Therefore, chapter *Influencing Factors* explains the do's and don'ts for optimal performance.

The result of any performance measurement is highly dependent on available hardware, network capacity and other processes running on the system. Therefore, the measurements in chapter *Benchmarks* do not only show the absolute numbers with unit of measures, but they also compare different ways of calling BRFplus or even compare a BRFplus call with a SELECT statement.

Example

The example is about a simplified price calculation. Based on several inputs the final price of an item is returned. For the example, we use one function, one ruleset, several data objects, three rules, two decision tables, and two formulas.

Function

The function "Price Calculation" returns the value "Final Price" based on 4 inputs.

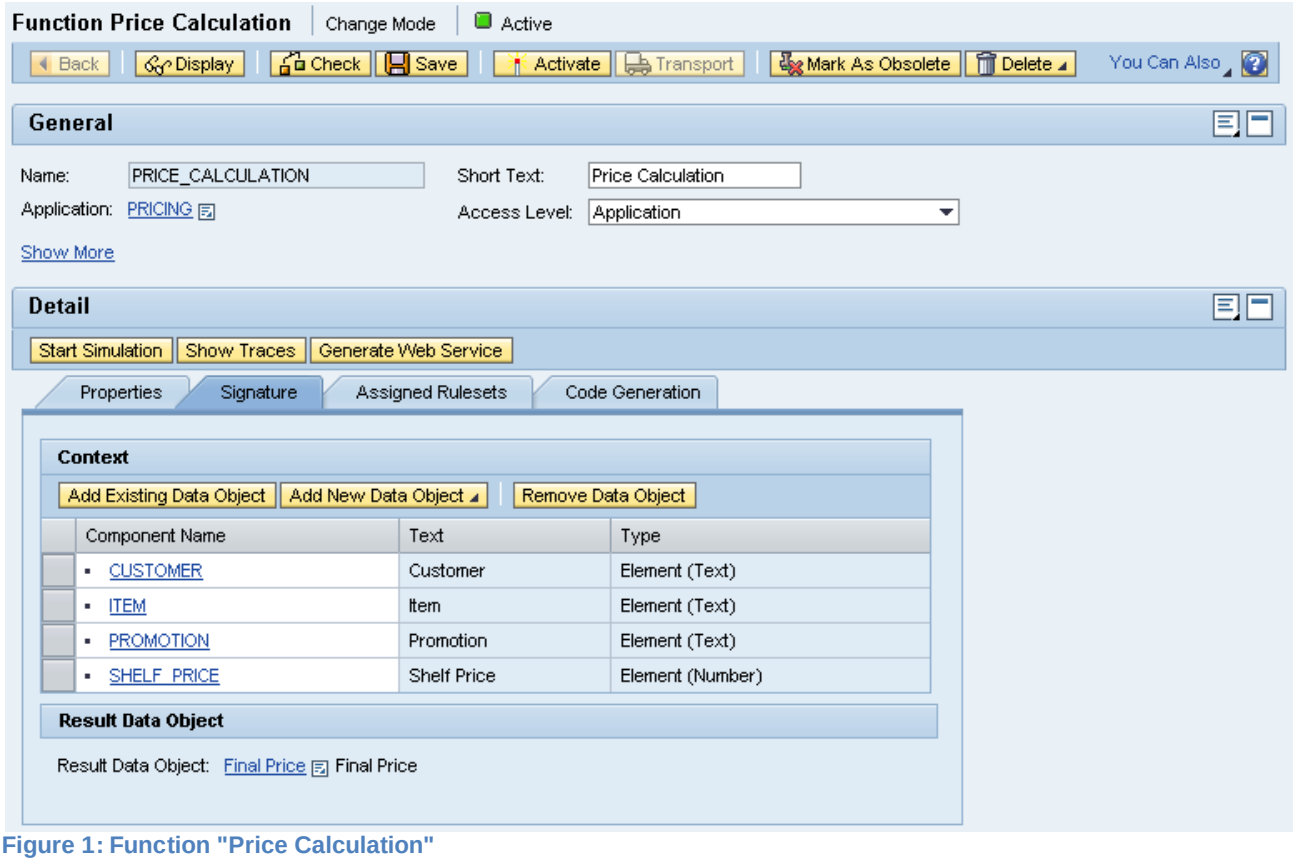

Figure 1: Function "Price Calculation"

Ruleset

The ruleset “Price Calculation Rules” contains the rules for the execution of function “Price Calculation”. There are three rules in the ruleset.

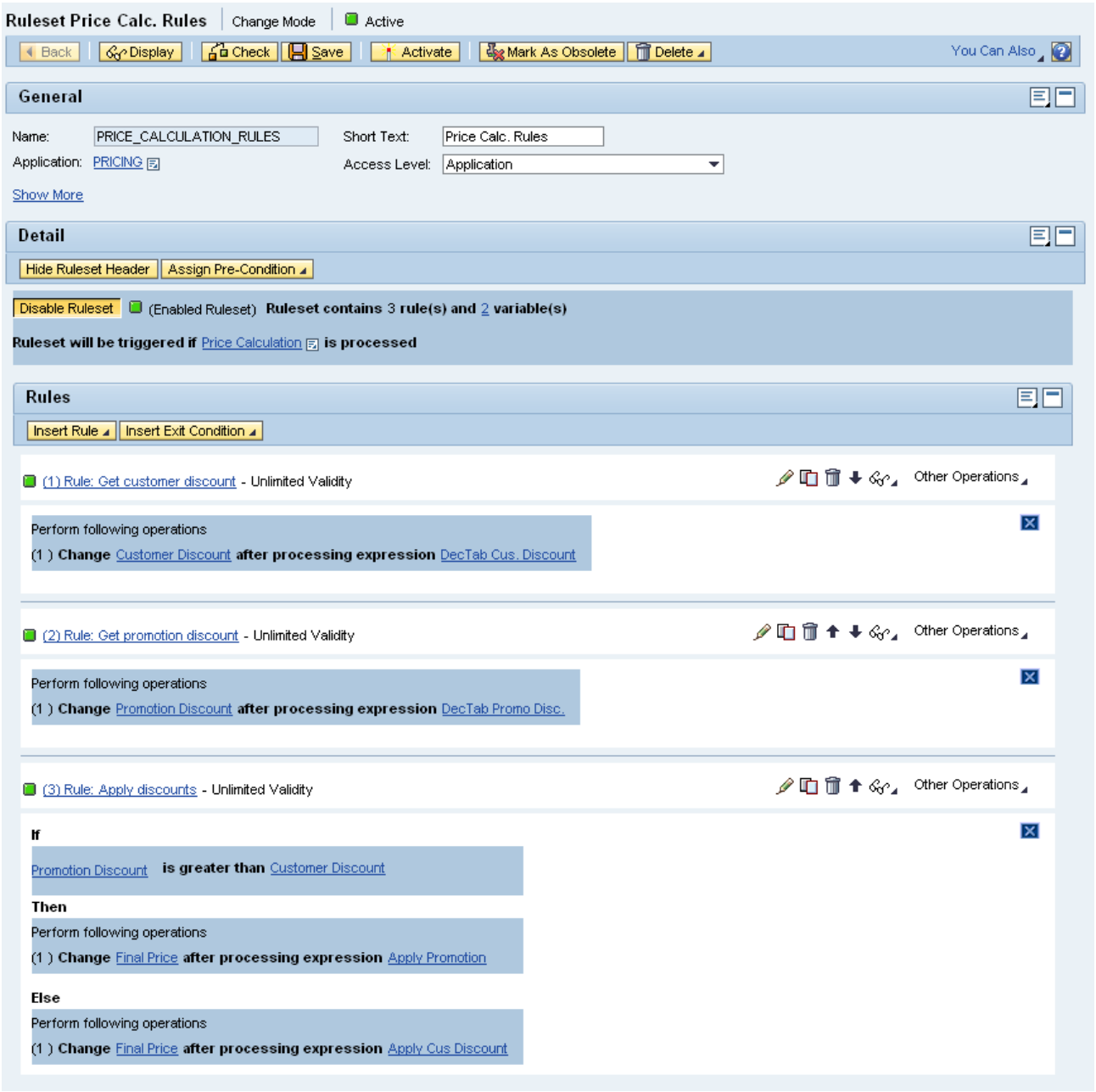

Figure 2: Ruleset "Price Calculation Rules"

The Ruleset also defines two ruleset variables.

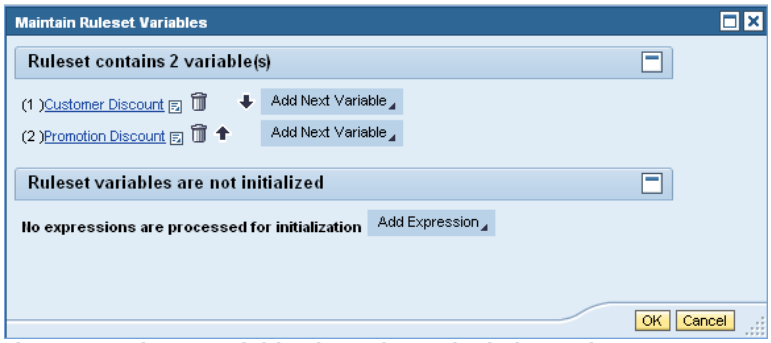

Figure 3: Ruleset Variables in "Price Calculation Rules"

Decision Tables

There are two decision tables: “Decision Table Customer Discount” and “Decision Table Promotion Discount”.

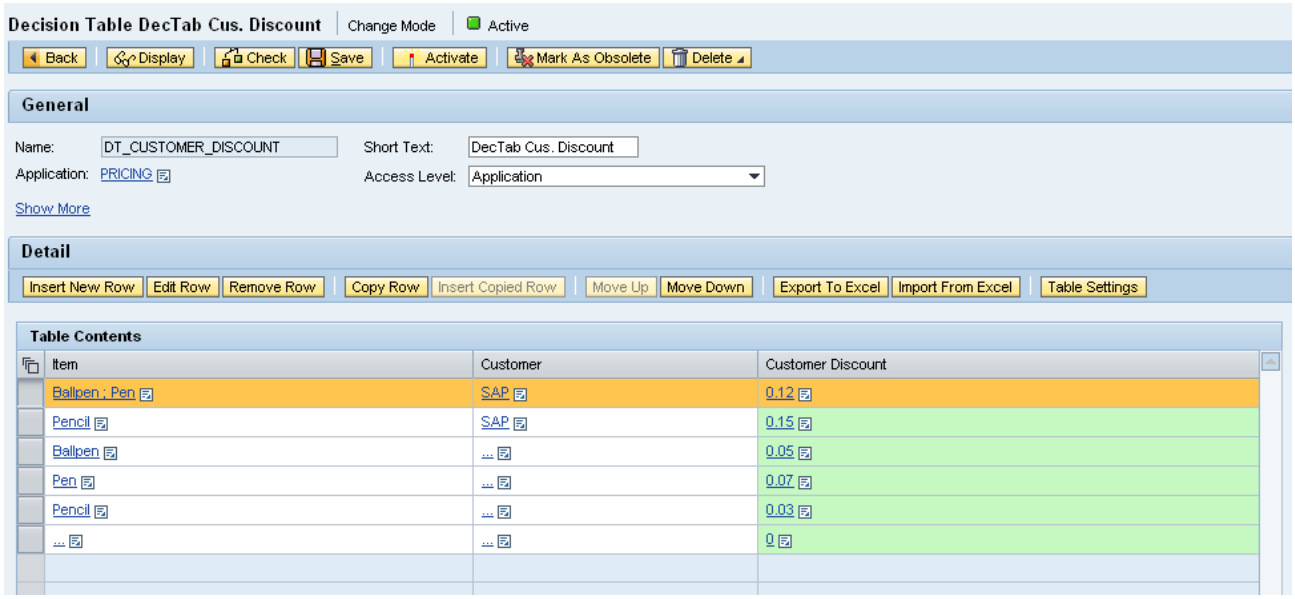

Figure 4: Decision table “Decision Table Customer Discount”

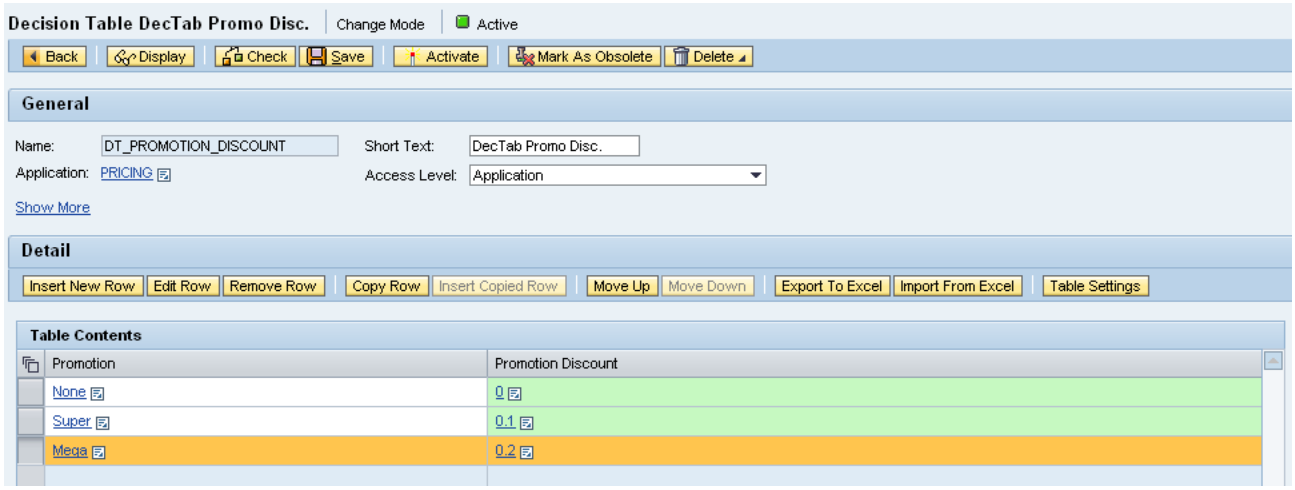

Figure 5: Decision table “Decision Table Promotion Discount”

Formulas

There are two formulas: “Apply Customer Discount” and “Apply Promotion Discount”.

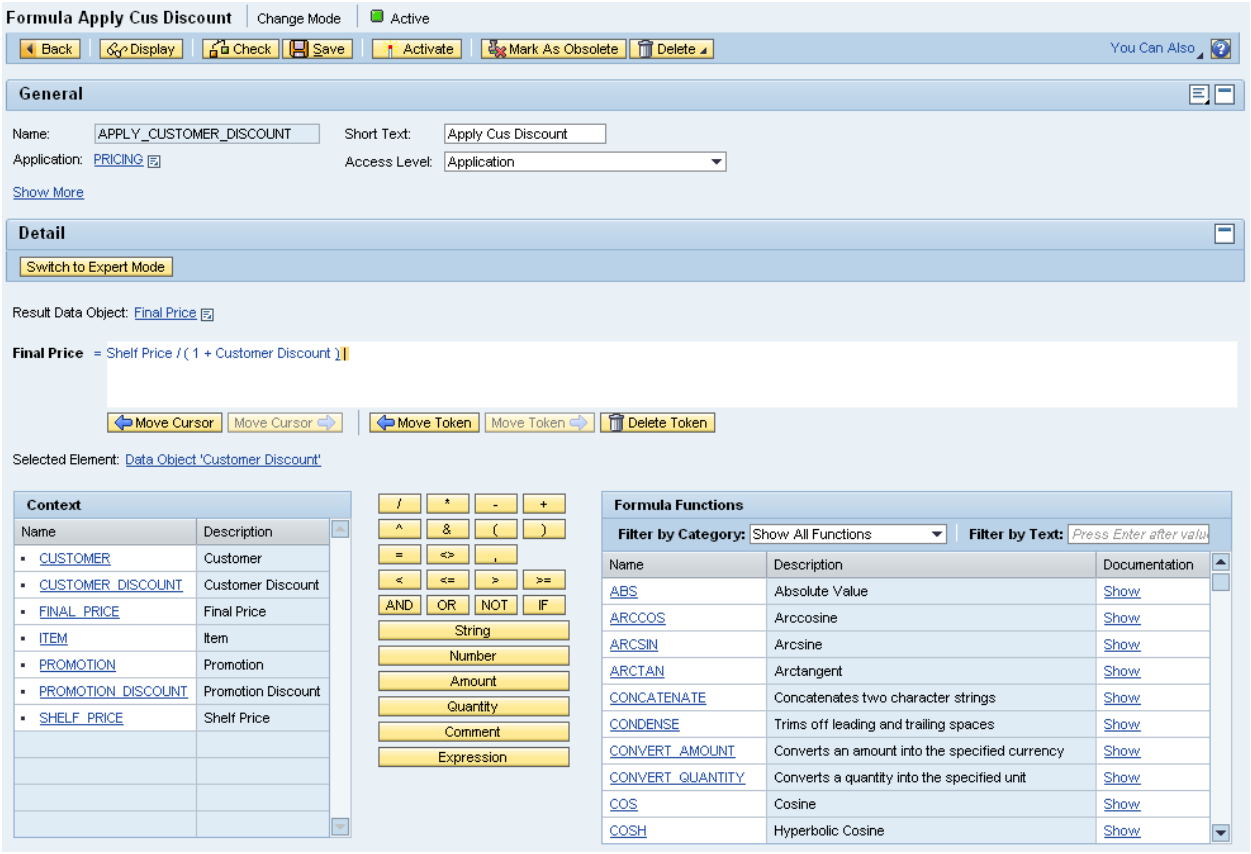

Figure 6: Formula “Apply Customer Discount”

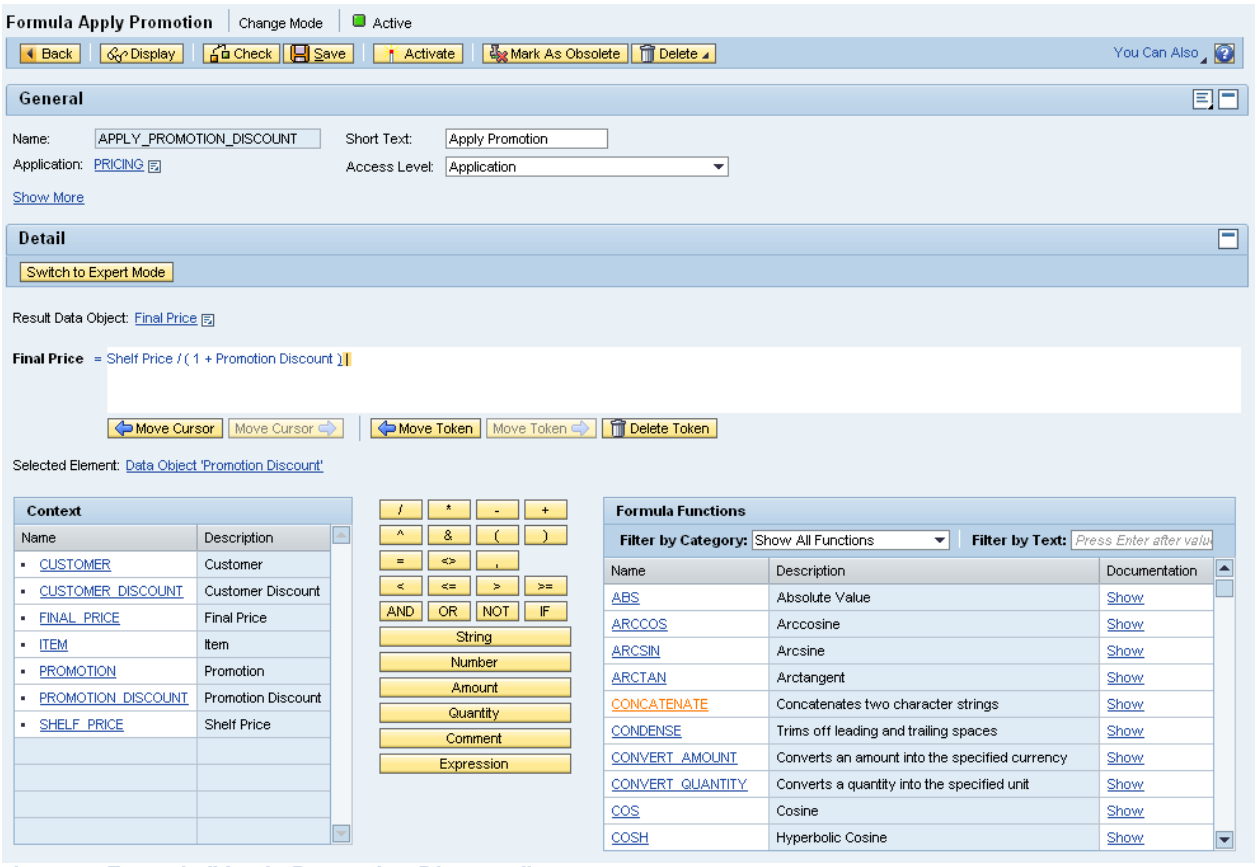

Figure 7: Formula "Apply Promotion Discount"

Influencing Factors

Call Types

BRFplus provides several options for a function process call. Depending on the situation, it may be necessary to do a remote call with a generated RFC-enabled function module or a web service.

Some users find it more convenient to use our nice object-oriented way of calling a function. However, for the sake of performance it is best to execute a procedure call and hand over a set of references.

Call Type: Instance

The following code snippet shows how a BRFplus function can be processed with an ABAP OO instance method.

```
DATA: lo_function TYPE REF TO if_fdt_function,
      lo_context  TYPE REF TO if_fdt_context,
      lo_result   TYPE REF TO if_fdt_result.

TRY.
    lo_function = cl_fdt_factory=>get_instance(
                ->get_function( gv_function_id ).
    lo_context = lo_function->get_process_context( gv_timestamp ).

    lo_context->set_value( iv_id =: gv_cus_id ia_value = gv_customer ),
                                    gv_itm_id ia_value = gv_item ),
                                    gv_prc_id ia_value = gv_price_in_eur ),
                                    gv_prm_id ia_value = gv_promotion ).
```

```
    lo_function->process( EXPORTING io_context   = lo_context
                                    iv_timestamp = lv_timestamp
                          IMPORTING eo_result    = lo_result ).

    lo_result->get_value( IMPORTING ea_value = gv_final_price_in_eur ).
  CATCH cx_fdt .
ENDTRY.
```

Method SET_VALUE in IF_FDT_CONTEXT also offers a parameter IV_NAME that can be used to set the values with the data object name instead of the identifier.

Processing a function with this approach is easy to read and understand. First a function instance is retrieved. The function provides a context instance that can be used to set values. When done the function is processed with the context and subsequently a result is returned. However, all objects that need to be instantiated require database selects. This is the big disadvantage of this approach.

Call Type: Static

The following code snippet shows how a BRFplus function can be processed with an ABAP OO static method.

```
DATA: ls_name_value         TYPE abap_parmbind,
      lt_name_value         TYPE abap_parmbind_tab,
      gv_customer           TYPE if_fdt_types=>element_text,
      gv_item               TYPE if_fdt_types=>element_text,
      gv_price_in_eur       TYPE if_fdt_types=>element_number,
      gv_promotion          TYPE if_fdt_types=>element_text,
      gv_final_price_in_eur TYPE if_fdt_types=>element_number.

ls_name_value-name = 'CUSTOMER'.
GET REFERENCE OF gv_customer INTO ls_name_value-value.
INSERT ls_name_value INTO TABLE lt_name_value.

ls_name_value-name = 'ITEM'.
GET REFERENCE OF gv_item INTO ls_name_value-value.
INSERT ls_name_value INTO TABLE lt_name_value.

ls_name_value-name = 'PRICE_IN_EUR'.
GET REFERENCE OF gv_price_in_eur INTO ls_name_value-value.
INSERT ls_name_value INTO TABLE lt_name_value.

ls_name_value-name = 'PROMOTION'.
GET REFERENCE OF gv_promotion INTO ls_name_value-value.
INSERT ls_name_value INTO TABLE lt_name_value.

TRY.
    cl_fdt_function_process=>process(
      EXPORTING iv_function_id = gv_function_id
                iv_timestamp   = gv_timestamp
      IMPORTING ea_result      = gv_final_price_in_eur
      CHANGING  ct_name_value  = lt_name_value ).
  CATCH cx_fdt .
ENDTRY.
```

It is possible to hand over a list of value and reference pairs in CT_NAME_VALUE. In case the names of the data objects are not stable, you may also use parameter ITS_ID_VALUE for identifier and value pairs.

For optimal performance this is the best way of processing a BRFplus function. In contrast to the instance method call described above, no SELECT statement is needed for object instantiation. Only the class name with the rules is retrieved and loaded into the ABAP memory during the first call.

At the time of writing this paper it was necessary to type the variables handed over in CT_NAME_VALUE according to the BRFplus types (see interface IF_FDT_TYPES and code example above). Strong BRFplus typing is not necessary for call type Instance. Improvements are planned so that this call type allows comparable flexibility.

Call Type: RFC

Remote Function Call (RFC) is an SAP procedure for calling function modules in remote systems. It is used for remote communication in the languages ABAP, Java, and C/C++. RFC can be used as a synchronous or asynchronous communication procedure.

Each remote communication introduces some overhead for data packaging and un-packaging, and the network traffic. According to our experience, a local synchronous call of an RFC-enabled function module introduces additional costs of 15+ milliseconds. Asynchronous calls need about 90+ milliseconds.

Call Type: Web Service

The SAP NetWeaver Application Server ABAP can be used both as a web services provider and as a service consumer. A web service is a modularized, executable unit. Web services can be compared with function modules. Based on the given input parameters, output is determined and passed back to the caller. Web Services are implemented with the Web Service Framework that ensures that various standards are supported, such as SOAP, WSDL and UDDI.

The overhead of the web service communication is comparable with the RFC overhead. However, in most use cases there is additional processing time spent in ABAP because of different data formats so that web service calls are usually slower than Remote Function Calls.

Note: In both cases, web service and RFC, communication overhead can be reduced significantly by bundling of calls. Instead of doing a remote call per data set, a call for a package of 100, 1000 or even more data sets should be considered.

Trace

Usage of trace always leads to a performance decrease because of the additional trace data that has to be calculated and written to the database. In generated code, it is possible to use the so-called Lean Trace.

```
DATA: lo_trace TYPE REF TO if_fdt_trace.

cl_fdt_function_process=>process(
  EXPORTING iv_function_id = gv_function_id
            iv_timestamp   = gv_timestamp
            iv_trace_mode  = if_fdt_constants=>gc_trace_mode_lean
  IMPORTING ea_result      = gv_final_price_in_eur
            eo_trace       = lo_trace
  CHANGING  ct_name_value  = lt_name_value ).
```

Using lean trace provides the possibility to record the rule execution and write it to the database. It is in the caller's responsibility to call the methods for the database access. By this option the caller may write any set of aggregated trace information (such as a set of 10 or 100 or 1000 calls) to the database to optimize performance.

```
DATA lo_lean_trace TYPE REF TO if_fdt_lean_trace.

TRY.
    lo_lean_trace->save( iv_update_task = abap_true ).
  CATCH cx_fdt.
ENDTRY.
```

The input parameters of the method provide further options to optimize the runtime. The best performance is achieved when saving bigger chunks of data in the update task (IV_UPDATE_TASK). Switching off authorization checks (IV_AUTH_CHECK) further reduces the overall runtime.

The usage of the technical trace IF_FDT_CONSTANTS=>GC_TRACE_MODE_TECHNICAL will switch the rule processing into interpretation mode. For details of the interpretation mode see chapter Interpretation. The technical trace is used to get a simulation protocol but is not suitable for performance-critical scenarios.

Parallel Processing

When processing large data volumes, you may consider using parallel processing. Parallel processing as implemented in ABAP provides options to split data and distribute it over several work processes in parallel.

Parallel processing is implemented with a special variant of asynchronous RFCs:

CALL FUNCTION STARTING NEW TASK DESTINATION IN GROUP

BRFplus is prepared to be used in parallel processing scenarios.

Currency and Unit Conversion

Working with data objects of type Amount or Quantity allows to define rules with reference to different currencies. For example, you may define something like:

```
If
   Income is greater than 1000 EUR
Then
   Do something
```

During runtime, the Income field contains a value in USD. BRFplus converts the value of the Income field into EUR as a preparation for the comparison. Therefore, conversion services are used. The conversion services have to look up exchange rates and maybe additional logic. In case this happens only once per function that is processed, the negative effect on performance may still be tolerable. However, think of examples where you have tabular input. One function call may trigger hundreds of currency conversion service calls to be able to complete. Quantities behave in a similar way. The difference to currency conversion is that a lookup is often not needed because the conversion logic is only a simple calculation step (e.g., conversion from miles into kilometers). For best performance, try to avoid using elementary data objects of type Amount and Quantity but only of type Number.

Interpretation

There are mainly three reasons for BRFplus to switch from generated code into interpretation:

- Custom action and expression types that do not support code generation.
- Expression type “Dynamic Expression”
- Usage of technical trace (simulation protocol)

Interpretation mode is also enforced when the function is processed for the first time (no code has been generated yet) and the function is locked for changes (e.g. from UI). As soon as generated code exists, this generated code is used for processing, regardless whether the function is locked or not. Hence, this option is mainly relevant for non-productive environments.

Whenever BRFplus is forced to switch to interpretation mode, you will encounter a substantial decrease in performance. Interpretation involves multiple SELECT statements and lots of highly dynamic code to be processed. To avoid this drawback, adhere to the following rules:

- Do not use the expression type “Dynamic Expression”.
- Do not use any custom expression or action type that does not support code generation.
- Use the lean trace (generated into the rule code) instead of the technical trace.

For high-performance scenarios, rule interpretation is not acceptable. Rule interpretation is often slower by a factor of 50 to 500 than execution of generated code.

Expression Type Database Lookup

With expression type Database Lookup, it is possible to define expressions that select data from database. There are different modes of execution:

- Aggregation
- Data Retrieval
- Existence Check

Access to database tables can be optimized with indices. A database access performed by BRFplus does not differ from an access being done in any hand-written ABAP code. If lots of data is copied from the database to the application server or if no index or no buffer can be used for the selection, the performance can be very bad. A business expert has virtually no chance to assess the performance implication when using DB lookup expressions. As these expressions are more complex to set up, IT experts are usually involved. It is the task of the IT expert to define the expression in an optimized way or maybe even to create a Procedure Call expression instead. In the ABAP code that is used in the Procedure Call expression, a developer may implement an optimized selection or reuse buffered data.

Expression Type Dynamic Expression

Using this expression type is by definition a threat to performance. For more information, see the comments in chapter *Interpretation*.

Expression and Action Type Procedure Call

The expression and action type to call a procedure facilitates the usage of ABAP methods and function modules during rule processing. Often this is used to retrieve additional data needed by the rules or to perform actions implemented in ABAP code. BRFplus calls the method or function module with the parameters defined in the expression. Hence, the performance implication depends solely on the quality of and the tasks done in the code that is called and if buffering mechanisms can be used.

Expression Types with Explicit and Implicit Loops

There are two expression types that may result in generated code with ABAP loops:

- Loop (processes an array of data in a loop and performs operations)
- Table Operations (operates on tables to carry out aggregations, existence checks, line counts…)

What is true for ABAP is of course also true for BRFplus which generates ABAP. Nesting of loops can have a negative impact on performance. This is especially the case when you loop over big internal tables and it is getting even worse if loops are nested. In case nested loops cannot be avoided consider using loop conditions to reduce the number of loop iterations to be performed.

Number of Rules

The number of rules in a ruleset or the lines of a decision table are not as critical to performance as many users intuitively assume. Of course, more rules content will lead to more code to be generated and potentially to be processed. However, the graphical approaches such as text rules and decision tables are translated into IF statements and simple data assignments in the ABAP code. These parts are nearly invisible in the performance analysis. In many cases it is not relevant if a method contains 100 or 1000 IF statements compared with the impact of other influencing factors.

Action Types

Expression types define a self-contained computational unit with a well-defined logic. The expressions use context data or nest other expressions to calculate, determine or derive a result. In contrast to this, action types are often used to perform activities outside of BRFplus. Usually, various APIs are involved (such as the SAP Business Workflow or Business Communication Services) that may not be designed for mass data scenarios. In fact, some may even perform a COMMIT WORK or a database Commit. We therefore highly recommend that prior to the usage of actions in mass data scenarios, the performance of the involved actions should be checked thoroughly.

Benchmarks

In the following section, you find the results of our performance measurements that we carried out in a controlled environment. The hardware used for the measurements consists of a server with 4 CPUs of type AMD Opteron 852 and 16 Gigabyte RAM.

This is the installed software:

- SUSE Linux Enterprise Server 10 SP2 (x86_64)
- MaxDB 7.7.04.032
- SAP NetWeaver 7.0 Enhancement Package 2 SP4

The measurements were done with the sample application described in the *Example* chapter in this paper.

A runtime of 100 microseconds per call means that it is possible to have 10.000 calls per second or 600.000 calls per minute. With parallel processing, it is possible to further improve the numbers easily.

Different Call Types

In chapter Influencing Factors, Call Types you have learnt about the different ways of rule invocation through the call of a BRFplus function. In chart Processing Time by Call Types you can see the performance of 1, 10 and 100 calls for the various call types. The measurements show that a static call is the fastest way of rule processing in BRFplus. Even when using lean trace the runtime is by far better than the runtime of instance-based or RFC invocation.

The following table shows the **overall processing time** for the various call types in milliseconds.

Call Type	1 Call	10 Calls	100 Calls	1000 Calls	10000 Calls
Static	0.78	1.39	7.26	46.37	484.65
Static with Trace	1.21	3.07	22.83	151.58	1500.76
Instance	8.88	13.28	40.68	354.52	3425.81
RFC	13.47	24.24	133.42	1118.43	11406.82

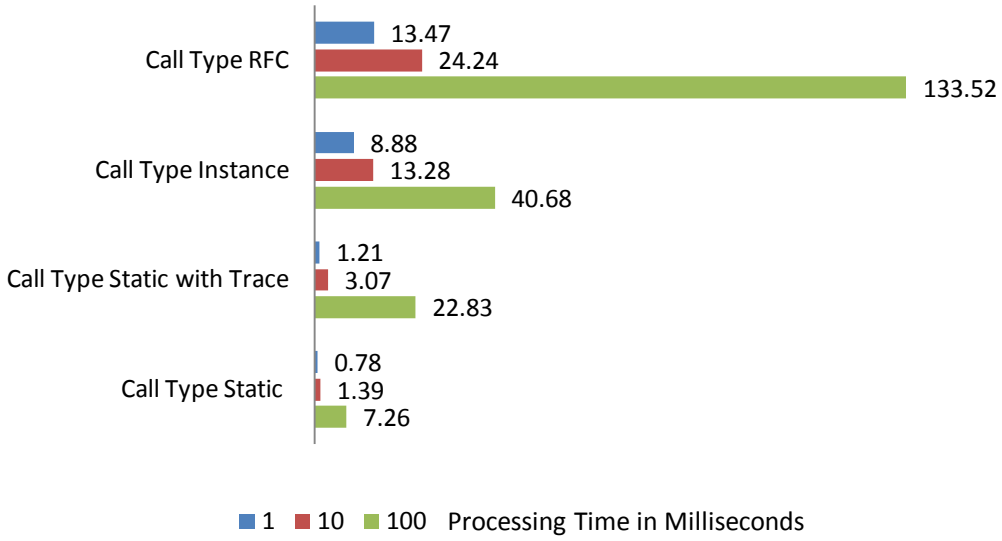

Figure 1: Processing Time by Call Types and Call Iterations

For mass data scenarios it is important to understand the delta added per call to assess the overall processing time. The first call requires loading the generated class and some preparatory work in case of call type Instance or RFC. Consecutive calls in the same session do not have to load the class again but only to re-run the generated method(s).

The measurements show a stable and slightly declining delta runtime that is stable as of about 100 calls. In the following table you can see all the **delta calls** measurements with higher precision. Slight changes in the deltas between 100, 1000 and 10000 calls are due to inaccuracy in measurements and the missing ability to have an exactly same runtime environment in all tests.

Call Type	1 Call	10 Calls	100 Calls	1000 Calls	10000 Calls
Static	0.78	0.07	0.07	0.05	0.05
Static with Trace	1.21	0.21	0.22	0.15	0.15
Instance	8.88	0.49	0.34	0.35	0.34
RFC	13.47	1.32	1.23	1.11	1.14

With a delta runtime of 0.05 milliseconds we can have 20 function calls per millisecond or 20k calls per second or 1.2 Million calls per minute.

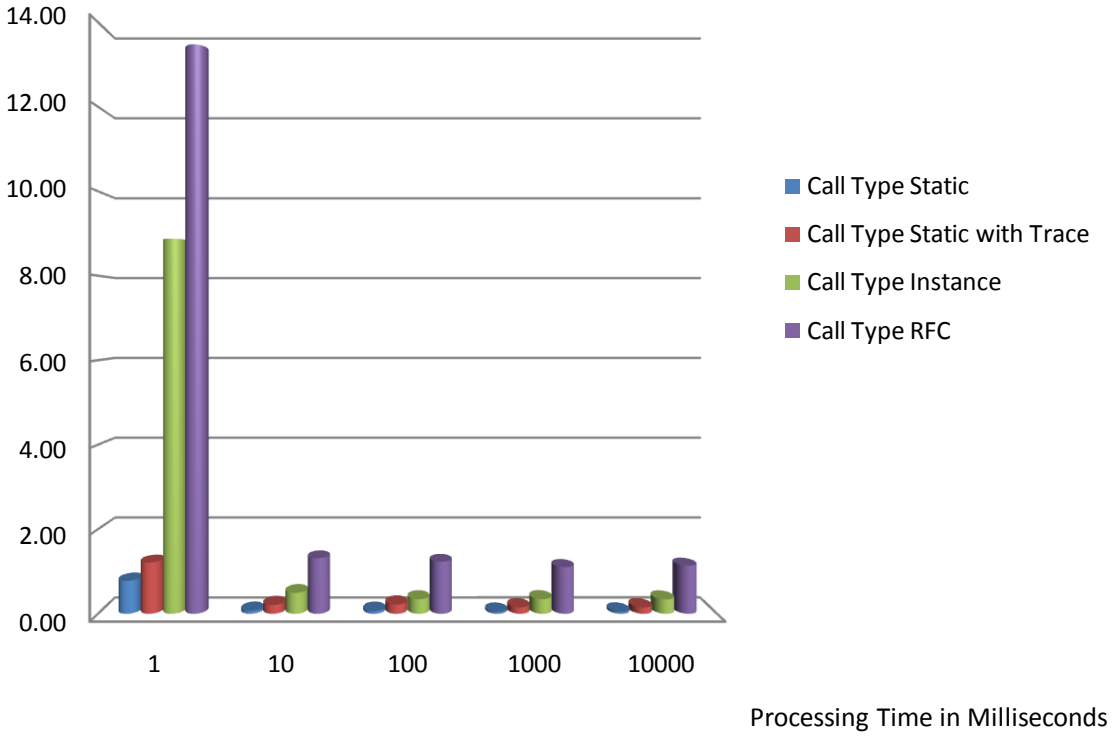

Figure 2: Delta Processing Time by Call Type

BRFplus versus Database SELECT Statements

Knowing how BRFplus performs, it is interesting to compare a BRFplus function call with a database SELECT statement. This comparison may also help you estimate the performance of BRFplus in your system as most likely you will have a different configuration than the one given above. To avoid any misunderstandings of the following discussion of the measurement results, let us again stress the fact that we will compare a BRFplus function call (the example from this paper) with a database table access via a SELECT statement, i.e, two different operations. We have decided to do so because a SELECT statement is such a common operation that hopefully everyone can get an idea of how BRFplus performs.

For the comparison we have chosen the two database tables T100 (Messages) and SFLIGHT (Flight) which are available in all NetWeaver installations. Table T100 contains the messages used in ABAP statement MESSAGE. The table is buffered with single-record buffering (see transaction SE11 for details). A select will therefore not necessarily access the database but answer the request from the buffer. Table SFLIGHT is not buffered. That's why for accessing SFLIGHT, a database access is always required. For the measurements we made sure the buffers for T100 are filled and requests are answered without database access.

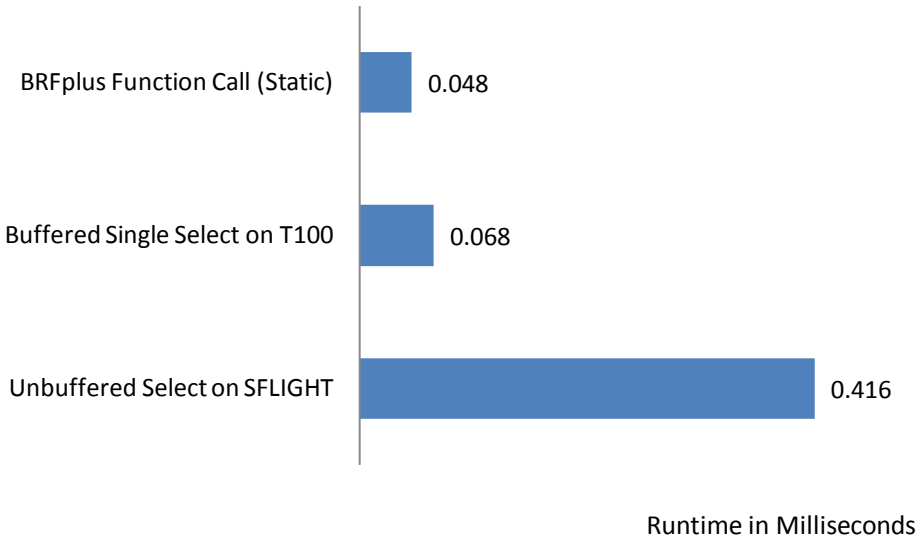

Figure 3: BRFplus versus Database

The results show that a delta call of the BRFplus function takes about 0.048 milliseconds compared with 0.068 milliseconds for the select single in table T100 and 0.416 milliseconds for the select single in table SFLIGHT. The BRFplus call is about 30% faster than a select single on a buffered database table.

Summary

In ABAP based solutions, BRFplus offers an unbeatable performance compared with other rule engines. This is mainly due to the possibility of local execution and the unique code generation approach. Overhead in form of network traffic and data packaging may take more time than the processing of a complete function in BRFplus. Sometimes even data collection and preparation steps to call into BRFplus consume more processing time than actual rule processing. But even when called from remote, e.g. by RFC, BRFplus can be an interesting alternative in high performance scenarios. Its built-in code generation framework translates the rule metaphors in form of text rules, decision trees, decision tables etc. into ABAP code. This is a big advantage compared with any interpreting approach as the benchmarks clearly show.

SAP is committed to invest into BRFplus in the future. There are many ideas to further improve the performance for the benefit of all the applications using it. Not all ideas will become reality and some changes may take more than a release. Finally, users will experience continuous improvements in performance and memory consumption.

Related Content

- BRFplus
- Business Rules Management

Copyright