SAP S/4HANA for Advanced Variant Configuration

Improvement List

THE BEST RUN SAP

TABLE OF CONTENTS

HANDLING OF CHARACTERISTICS 4
All Characteristics are Restrictable 4
Single-Valued Characteristics and Alternative Values 4
Semantics of Multi-Valued Characteristics 4
Handling of “Additional Values Allowed” 4
Numeric Characteristics 5
Distinction “Integer” vs “Float” 5
Numeric Range of Numeric Characteristics 5
DEPENDENCY PROCESSING 6
Stronger Inferences on Individual Syntax Elements 6
Stronger Inferences on Constraints 6
Choosing the Right Modeling Technique 6
Modeling with “FALSE IF” 7
Set-Based Semantics of Multi-Valued Characteristics in Dependencies 7
Preconditions on Value 8
Preconditions on Overlapping Value Ranges 8
Termination of Procedures 9
Uniform Behavior of Procedure Assignments 9
Procedures on BOM Item 9
Numeric Computations 10
Decimal Floating Point 10
Float-Rounding Behavior 10
Integer Behavior 10
Mixed Numeric Expressions 10
No Usage of Arbitrary Characteristics 10
Behavior in Case of Inconsistent Model 11
Excluding Mandatory Characteristics 11
Read-only Reference Characteristics 11
Key Access in Variant Tables 11
No Ambiguous Table Statements 11
Evaluation of Complex Logical Expressions 11
Constraints Belonging to Inactive Instances 11
Enforcing the Correct Length of String Values 11
Consistent Behavior of Case-(In)Sensitive Characteristics 12
DEFAULT VALUES 13
Sticky defaults 13
Dynamic Defaults and Other Procedure Assignments 13
Default correction to avoid inconsistencies 13
MULTI-LEVEL BOM EXPLOSION 14
BOM Explosion Strategy 14
Build-From-Top 15
Inconsistency During BOM Explosion 15
Dependency Processing in Multi-Level Configuration Models 16

PLANNED IMPROVEMENTS 17
BOM explosion strategy 17
High-level dependencies on BOM items 17
Default handling 17
Syntax enhancements 17

This document describes the cases where the AVC engine behaves differently to the LO-VC engine. It does not describe which LO-VC functionality is still missing in the AVC. For the current AVC scope, please see the AVC documentation.

HANDLING OF CHARACTERISTICS

All Characteristics are Restrictable

In AVC, all characteristics are treated as restrictable, even the multi-valued ones. In LO-VC, for non-restrictable characteristics the master-data value domains are always displayed for the user to choose from. For restrictable characteristics (in LO-VC and AVC), the configuration engine determines which values can still be chosen without causing an inconsistency, and only those values are displayed, resulting in a better user experience.
Characteristic value domains can be restricted in AVC by using constraints and preconditions on values.

Single-Valued Characteristics and Alternative Values

A major drawback of restrictable characteristics in LO-VC is that, once assigned, the user needs to manually un-assign them, to be able to view the alternative values. In the AVC, however, the alternative values are always displayed to the user upon focusing the characteristics. These alternative values are exactly those, which the engine would display if the respective characteristic were not assigned. These are not necessarily the same values as the master data values; typically, this will be a subset thereof: the one that is still consistent with the rest of the configuration.

Semantics of Multi-Valued Characteristics

For multi-valued characteristics AVC adopts the following semantics: Any subset of values from the master data value domain can be assigned to the characteristic (including no value at all). Hence a characteristic with three possible values “A”, “B”, “C” can have eight different assignment sets (no value, one of them, all of them or any combination of two). Individual elements can be assigned to / excluded from the characteristic individually. In our example, after assigning “A” and excluding “B”, the characteristic has value “A” assigned, and value “C” is still possible. In this example, “A” is displayed as an assigned value and “C” as an alternative value.

Different to single-valued characteristics, excluding all values from a multi-valued characteristics domain does not cause an inconsistency. If the characteristic represents, for example, a set of extras available for the product, not choosing any extras should not constitute an error.

This behavior is in line with that of LO-VC in classical dependencies. However, in LO-VC, one can only exclude values of multi-valued characteristics by preconditions, and those exclusions only hide the value on the UI. With restrictable multi-valued characteristics in the AVC, value exclusion can also be done by constraints, and other dependencies can react on the domain changes (see also section Preconditions on Value).

LO-VC uses two different semantics for multi-valued characteristics, one for classical dependencies and one for constraints. AVC uses only one, described in section Set-Based Semantics of Multi-Valued Characteristics in Dependencies.

Handling of “Additional Values Allowed”

In LO-VC, the flag “additional values allowed” can only be set for non-restrictable characteristics. AVC interprets this as follows for restrictable characteristics: On top of the predefined values all values, matching the characteristic template are allowed. For example, for a numeric characteristic where the template requires a three-digit number, no decimal places, and allows negative values, the domain is -999 – 999. For string characteristics, the domain consists of all Unicode strings respecting the defined length.

For numeric characteristics, the (possibly restricted) set of additional values is always displayed to the user on the valuation UI, because choosing a value outside of this set would cause an inconsistency. Values defined in the master data are shown on the UI above the additional values. For string characteristics, the additional values cannot be shown on the UI because of representational issues.

Please note that characteristics without any predefined values are treated as if “additional values allowed” were set. This is in line with LO-VC behavior.

Numeric Characteristics

For the AVC, the numeric engine behavior has been redesigned with the goal of achieving greater accuracy. This is achieved by

- introducing a distinction between integer and floating-point characteristics; and
- improved handling of floating-point characteristics.

Distinction “Integer” vs “Float”

Pure integer characteristics are sufficient or even desirable for many customer use cases. Computing with integers allows for stronger results and avoids the computational overhead needed for the correct treatment of floating-point numbers.

Therefore, AVC distinguishes between integer and float characteristics, while in LO-VC, all numeric characteristics are floats. As AVC also has improved floating-point handling (see section Decimal Floating Point), AVC generally provides much more accurate numeric results than LO-VC.

The distinction in AVC is based on the characteristic template (which is assumed to refer to the characteristic’s base unit):

- integer: numeric characteristics with no decimal places
- float: numeric characteristics with decimal places

While AVC distinguishes between integer and float characteristics, the two types can be mixed in numeric expression (see section Mixed Numeric Expressions).

Numeric Range of Numeric Characteristics

The numeric range for float characteristics in AVC is ±1 * 10^15 (as in LO-VC). For integer characteristics it is ~ ±2.1 * 10^9 for single-value characteristics and ~ ±1.05 * 10^9 for multi-value characteristics in AVC.

DEPENDENCY PROCESSING

The AVC engine has been designed to compute the set of possible user choices far more accurately than LO-VC. The fundamental problem is the following: If the user gets to choose a value on the UI, which is not actually feasible, then choosing this value results in an inconsistency, which then needs to be resolved. The AVC engine attempts to determine upfront that a value cannot consistently be included in the configuration. At the same time the engine has been designed with a clear focus on performance. This means, for example, that it does not attempt to identify all values violating a complicated mathematical constraint. In such cases, the engine identifies a subset of the impossible values, that can be obtained at a reasonable computational cost. Summing up, the AVC engine should always find at least as many impossible values as LO-VC, and often it should find more.

The AVC engine processes only the dependencies, which are not attached to any low-level objects (e.g. BOM items or to operations in routings). For these latter dependencies, the AVC utilizes the LO-VC engine, which is tightly integrated with BOM / routing explosion. Please bear this in mind when considering the below information.

Stronger Inferences on Individual Syntax Elements

Here are some examples where AVC does stronger dependency processing than LO-VC:

- Assume you have a restriction “X = Y”, where X is unrestricted, and Y has value domain {1, 3, 5}. Then LO-VC will restrict X to the domain {1 – 5} forgetting about the holes in Y’s domain. AVC, on the other hand, correctly intersects the two variable domains.
- The AVC engine determines all the values which are still possible according to a variant table, whereas LO-VC uses an approximation, leaving the user to choose values which are not actually supported by the table (see note https://launchpad.support.sap.com/#/notes/367002).
- With all characteristics being restrictable, AVC considers restricted domains when testing conditions. For example, a comparison “X < 4”, where X has the restricted domain {1,2,3}, will evaluate to true in AVC. The rationale is that, no matter which choice the user still makes for X, the comparison will always evaluate to true. In LO-VC the comparison will only be evaluated once X gets assigned. If this comparison is used as a trigger for a procedure assignment, for example, then the user will see this assignment earlier in AVC than in LO-VC.

Stronger Inferences on Constraints

The LO-VC engine supports inferences only for a restricted set of syntax elements. For example, inequalities, the negation of tables or IN-lists, non-linear arithmetic like multiplication or division of characteristics, and trigonometric operations all cannot be inferred. Moreover, inferences are limited to characteristics on the left-hand side of the IF in a restriction. The AVC engine, on the other hand, can do inferences for all syntax elements. To shield the user from avoidable inconsistencies, it always does inferences for all characteristics in a constraint restriction.

Example: Consider a restriction “X = 1 IF Y = 2”. Assume 1 is not valid for X, for example, because X is assigned to 3. Then the AVC engine will exclude 2 from the domain of Y, because choosing 2 would cause an inconsistency. For both LO-VC and AVC, as long as the value 1 is valid for X, nothing happens, until Y is assigned to 2.

This behavior leads to a better user guidance: Fewer values that would necessarily lead to an inconsistency are displayed to the user.

Choosing the Right Modeling Technique

As in LO-VC, the RESTRICTION of a constraint only gets activated in AVC once the CONDITION evaluates to true.

Same as LO-VC, AVC provides several ways of expressing that a specific condition should hold for the configuration in question. For one, one could write a procedure:

a) someAssignment IF someCondition

Next one could write a constraint using the CONDITION and the RESTRICTION part:

b) CONDITION: someCondition
   RESTRICTION: someAssignment

Finally, one could write a constraint using the RESTRICTION section only:

c) RESTRICTION: someAssignment IF someCondition

You can then choose the appropriate modeling technique in AVC as follows:

1. If you want the assignment to be made, only if the condition is fulfilled at a specific point in time, you choose a) and make the procedure run at that point in time.
2. If you want the assignment to be made whenever the condition is fulfilled, you choose option b).
3. If, moreover, you want to prevent the user from triggering the assignment, when it will result in an inconsistency, then you choose option c).

When choosing which modeling technique is appropriate for your model, there is no need to worry about performance. Constraint processing is driven by a very efficient change monitoring algorithm so that the differences between options a) – c) should be negligible.

Modeling with "FALSE IF"

The AVC engine interprets the keyword "FALSE" differently from LO-VC. A constraint "FALSE IF X = 1" will cause an inconsistency in LO-VC when X gets assigned to 1. In the spirit of preventing inconsistencies, the AVC will rewrite that constraint to "X <> 1" and exclude 1 from X's domain. More generally, a constraint of the form "FALSE IF CONDITION" is equivalent in AVC to "NOT CONDITION". Using this pattern, one can express, for example, the exclusion of values from a list (via "FALSE IF X IN (1,3,5)") or negative tables (via "FALSE IF TABLE …"), and the AVC engine will perform the desired domain restrictions.

As in LO-VC, a constraint

- CONDITION: someCondition
  RESTRICTION: FALSE

will simply cause an inconsistency as soon as the condition becomes true.

Set-Based Semantics of Multi-Valued Characteristics in Dependencies

The LO-VC engine uses two different semantics for multi-valued characteristics, one for constraints, and one for the classical dependencies. For example, if you have a characteristic X with assigned values 1 and 2, then a comparison "X <> 1" will fail in classical dependencies but succeed in constraints. In classical dependencies, the complete set of values is considered, whereas in constraints, the semantics are based on finding one element of the set for which the comparison holds.

The AVC always considers the complete set of assigned and possible values when processing multi-valued characteristics.

Example: If you have a multi-valued characteristics X assigned with values 1 and 2, and a constraint Y = 'A' IF X NE 1, then LO-VC will assign Y to 'A', and AVC will not.

Some functionality in LO-VC crucially depends on the value-based semantics of multi-valued characteristics in constraints: for example, arithmetic operations on multi-valued characteristics or the concatenation of their values. These operations are not meaningful in the context of the new AVC semantics.

An equal comparison between a single-valued term and a multi-valued characteristic ("SV = MV") in AVC always means the same as "SV IN MV". An unequal comparison becomes "SV NOT IN MV2". The greater and smaller comparisons between a single-valued term and a multi-valued characteristic mean that all the values of the multi-valued characteristic must fulfil the comparison relation with the single-valued term. Equal and not-equal relations between two multi-valued characteristics always denote the (dis-)equality of sets in the AVC.

When comparing multi-valued characteristics, there are two different views one can take:

- Only consider the values which are assigned to the characteristic at the point in time when the comparison is executed.
- Consider both the assigned and the possible values.

The first approach is unsuited for constraints, because for these the modeler cannot control the point in time when they are executed. Sometimes the AVC engine might execute another constraint, making an assignment to the multi-valued characteristic before the constraint containing the comparison, and sometimes it might only execute it afterwards, making the whole configuration result unpredictable. Hence the AVC adopts the second approach.

Example: A comparison “1 IN X” on a multi-valued characteristic will evaluate to

- true if X has assigned values {1, 3, 5} and possible values {2, 4, 6};
- unknown if X has no assigned values and possible values {1 – 6};
- false if X has assigned values {2} and possible values {3 – 6}.

A comparison “1 NOT IN X” will evaluate as follows:

- false if X has assigned values {1, 3, 5} and possible values {2, 4, 6};
- unknown if X has no assigned values and possible values {1 – 6};
- true if X has assigned values {2} and possible values {3 – 6}.

Assignments to and restrictions for multi-valued characteristics behave as follows: “X = 1” makes “1” an element of X (but does not equate X with the set {1}, leaving other values possible). A restriction “X <> 1” excludes “1” from the set of possible values.

Preconditions on Value

In LO-VC, a violated precondition on value excludes the value only on the UI, not in the engine. That is, even though the value is excluded via precondition, other dependencies still see the value as available. The AVC overcomes this weakness by treating preconditions on values like constraints: They are evaluated perpetually and exclude values in the engine, not just on the UI. More precisely, a precondition CONDITION on value “A” of characteristic X is interpreted in AVC as if it were written as:

RESTRICTION:
X <> A IF NOT CONDITION.

Historically, preconditions on values have been the only modeling technique for restricting multi-valued characteristics in LO-VC models. Multi-valued characteristics are now restrictable in the AVC, and restrictions via preconditions on values fit to this concept.

While this behavior has the advantage of enabling stronger dependency processing, it has a drawback when it comes to the handling of SPECIFIED. In LO-VC and AVC, a SPECIFIED-check in a constraint evaluates to true once values are assigned.[1] However, it never evaluates to false; this now also holds for preconditions. This means that the use of SPECIFIED in preconditions on value effectively disables the dependency. This is also the reason why NOT SPECIFIED is not supported in LO-VC (and AVC) constraints.

Preconditions on Overlapping Value Ranges

LO-VC and AVC adopt a different reading of overlapping numeric intervals. Assume we have a characteristic with two domain ranges 1 – 10 and 5 – 15, and both have preconditions attached to them. In LO-VC, once a precondition excludes the range 1 – 10 the complete range 5 – 15 is still available. In the AVC, however, what remains after excluding 1 – 10, is 11 – 15. The rationale is that for the numeric engine, it should not matter which master data domain value a specific number was defined in.

[1] This is due to the declarative nature of constraint, which essentially means that the inferences a constraint makes, must not depend on a particular point in time – they must be absolute truths.

Termination of Procedures

LO-VC and AVC have a different behavior when it comes to terminating procedures. Consider a procedure consisting of several subprocedures:

```
$self.X = 1,
$self.Y = $root.C,
$self.Z = 2.
```

If characteristic C is not assigned when processing the procedure, in LO-VC the procedure will terminate at that statement, meaning that X gets assigned, but Z does not. In AVC, the procedure does not terminate. This makes the above procedure equivalent to the following set of individual procedures:

$self.X = 1.
$self.Y = $root.C.
$self.Z = 2.

In LO-VC, these two modeling techniques are not equivalent.

Uniform Behavior of Procedure Assignments

In LO-VC, the restrictable flag on a characteristic determines the behavior of procedure assignments. For restrictable characteristics, procedures also perform domain restrictions, whereas for non-restrictable characteristics, only concrete value assignments are performed. AVC adopts the latter behavior. For example, if characteristic Y is not assigned, then the procedure assignment X = Y in AVC has no effect, no matter whether X is restrictable or not. In LO-VC, if X is restrictable, the value domain of Y is assigned to characteristic X.

Procedures on BOM Item

Instead of adapting the VC behavior of processing any procedure on BOM item multiple times during high-level configuration (see note 1755434), AVC aims for a clear separation between high and low-level dependencies. Procedures on BOM items are only evaluated during BOM-explosion, and the calculated values are not considered in high-level configuration. See section

Planned improvements for future enhancements.

Numeric Computations

For the AVC, the numeric engine behavior has been redesigned with the goal of achieving greater accuracy. As already mentioned this is achieved by

- introducing a distinction between integer and floating-point characteristics; and
- improved handling of floating-point characteristics.

Decimal Floating Point

The floating-point computations in LO-VC are based on the binary floating-point standard (IEEE'754). This standard has the drawback that most non-integer quantities cannot be represented exactly. For example, of the prices 0.00 – 0.99 only 0.00, 0.25, 0.5, and 0.75 can be represented exactly. Consequently, rounding has to be applied in many computations where one would not expect it. For example, the result of 0.1 + 0.2 is very close, but not exactly equal to 0.3.

For this reason, the IEEE'754-2008 standard introduces decimal floating-point arithmetic with the goal of better satisfying business needs. In this standard, all non-integer master data value of a configuration model can be represented exactly. The AVC is based on this standard: It uses the IEEE'754 DecFloat34 data type, meaning that values and computed results requiring not more than 34 decimal digits can always be represented exactly.

Float-Rounding Behavior

Obviously, even 34 digits are not enough to represent all computational results exactly. A simple example would be computing 1/3. To control the inevitable rounding errors, the AVC engine uses a combination of ideas from the IEEE'1788 standard and a small epsilon in the order of 10^-22. Consequently, numeric computations and comparisons can be expected to work as in standard high-school mathematics.
With rounding error under control, repeatedly restricting the same characteristics to smaller and smaller domains by the same numeric constraint is safe for the AVC: It does not increase the rounding error.
LO-VC, on the other hand, is based on the old IEEE'754 standard and its rounding behavior. Because of this, it employs much more liberal comparisons, accepting deviations of up to 0.5 in extreme cases.
Repeatedly refining characteristic domains via some equation is unsafe also: Each further restriction potentially increases the rounding error. Overall, this may result in numerically inconsistent configurations being accepted as consistent in LO-VC.

Integer Behavior

The behavior of numeric computations with integer characteristics follows standard mathematics in the AVC engine.

Mixed Numeric Expressions

While AVC distinguishes between integer and float characteristics, the two types can be mixed in numeric expressions. For example, an expression X = Y + Z is legitimate, regardless of the numeric types of X, Y, and Z. If X is an integer characteristic while Y and Z are float characteristics, the model becomes inconsistent if Y + Z is evaluated to a non-integer value: X = 1.3 + 1.8 always leads to an inconsistency, while X = 1.3 + 1.7 is a legal assignment.

No Usage of Arbitrary Characteristics

In LO-VC, characteristics that are not part of the configured class / material can be used in classical dependencies: Any characteristic available in the system can be used. This is not supported in AVC. This also holds for reference characteristics. Including all addressed characteristics in the configuration model increases transparency and overall consistency.

Behavior in Case of Inconsistent Model

LO-VC tries to keep configuring in case of an inconsistency (continued dependency evaluation and BOM explosion). In order to provide predictable and reliable behavior, AVC stops dependency evaluation and BOM explosion in case of an inconsistency (see also chapter Inconsistency During BOM Explosion).

Excluding Mandatory Characteristics

In AVC, whenever a characteristic, which is excluded by a precondition, becomes mandatory, an inconsistency is caused. In LO-VC, only the static mandatory flag causes an inconsistency, not the dynamic one via selection conditions. Thereby, AVC unifies the behavior of both the static and dynamic mandatory flag.

Read-only Reference Characteristics

LO-VC does not distinguish between read-only and read-write reference characteristics. All reference characteristics can be overwritten by dependencies. The embedding application may refuse to accept the changed value, but by then the changed value may already have impacted other characteristics via dependencies. The AVC respects the distinction. Please note that the distinction is enforced only if they are already assigned by the embedding application, or manually by a user in the simulation environment.

Key Access in Variant Tables

In LO-VC, the distinction restrictable / non-restrictable characteristics determines whether a constraint restriction via a variant table results in domain restrictions, or in an inference via a table key (assignment alternative). This means that the behavior of a restriction “TABLE TAB(col1 = X, col2 = Y)” on LO-VC can only be inferred from the properties of the characteristics X and Y.

AVC, on the other hand, does not know about the distinction between restrictable and non-restrictable characteristics. Presently, AVC always does domain restrictions in constraints, and this certainly works for assigned key values as well.

No Ambiguous Table Statements

No column needs to be used twice in AVC for variant-table statements in dependencies. In LO-VC, this leads to semantic ambiguities. See also SAP note 1007381. The same note also contains a simple modeling workaround, avoiding the semantic ambiguities.

AVC also does not support restricting the same characteristic over multiple columns in one table statement, again because of unclear semantics.

Evaluation of Complex Logical Expressions

In LO-VC, the logical expressions in dependencies are transformed to disjunctive normal form prior to evaluation (https://en.wikipedia.org/wiki/Disjunctive_normal_form). In AVC, dependencies are not transformed into disjunctive normal form, but processed as entered by the modeler. This saves a potential exponential blowup, and the corresponding performance impact.

Constraints Belonging to Inactive Instances

In LO-VC, constraints belonging to instances that have been active once in a configuration session, remain active until the end of the configuration session (even if the instance is no longer included in the configuration). In AVC, only the constraints of the currently active instances are considered.

Enforcing the Correct Length of String Values

When defining a string characteristic, the modeler specifies the maximal length for this characteristic’s values. AVC ensures that this specification is respected for any assigned value, and if a computed value is

too long, this causes an inconsistent model. In the LO-VC engine, predefined values are checked to meet the length constraint, but computed values for characteristics with additional values allowed may violate it.

Consistent Behavior of Case-(In)Sensitive Characteristics

In LO-VC and AVC, it is possible to mix characteristics, which are case-sensitive, with those which are not in one and the same expression. For example, one could ask whether “C1 = C2” holds, where C1 is case-sensitive and assigned to ‘a’, and where C2 is not case-sensitive and assigned to ‘A’. The problem with this is that no single and obviously clear semantics for such statements exists.
The treatment of this case in LO-VC is not consistent. Sometimes lower-case values are converted to upper case, sometimes an equality ‘a’ = ‘A’ causes an inconsistency.
The AVC takes the following approach throughout: All values of case-insensitive characteristics must be in upper case. Assigning a lower-case value to a case-insensitive characteristic always leads to an inconsistency, there is no implicit case conversion. The same holds for comparisons.

DEFAULT VALUES

Sticky defaults

In the LO-VC dynamic defaults are sticky (i.e. a default always comes back unless the user has replaced it by a different value; leaving the characteristic blank is not possible). Static defaults are not: They can be blocked, but not re-enabled.

Currently, in AVC all defaults are sticky. For future enhancements, see section Planned Improvements.

Dynamic Defaults and Other Procedure Assignments

In the LO-VC engine, dynamic defaults remain valid across roundtrips. That is, different from other procedure assignments, the dynamic defaults from the last roundtrip are not retracted at the start of a new roundtrip. They must be explicitly removed using $del_default.
LO-VC example: A procedure “$self.X ?= 1 IF CONDITION” assigns a dynamic default, if the condition is fulfilled in any roundtrip. The default value is then kept for all following roundtrips. This even applies if the condition is not fulfilled anymore.
In the AVC engine, dynamic defaults are recalculated in every roundtrip, just like ordinary procedure assignments, making most uses of $del_default obsolete.

Default correction to avoid inconsistencies

Defaults, like any other value assignment, may cause inconsistencies. One can argue that merely setting a default value should not leave the end user with an inconsistent model. Both LO-VC and AVC follow this approach.

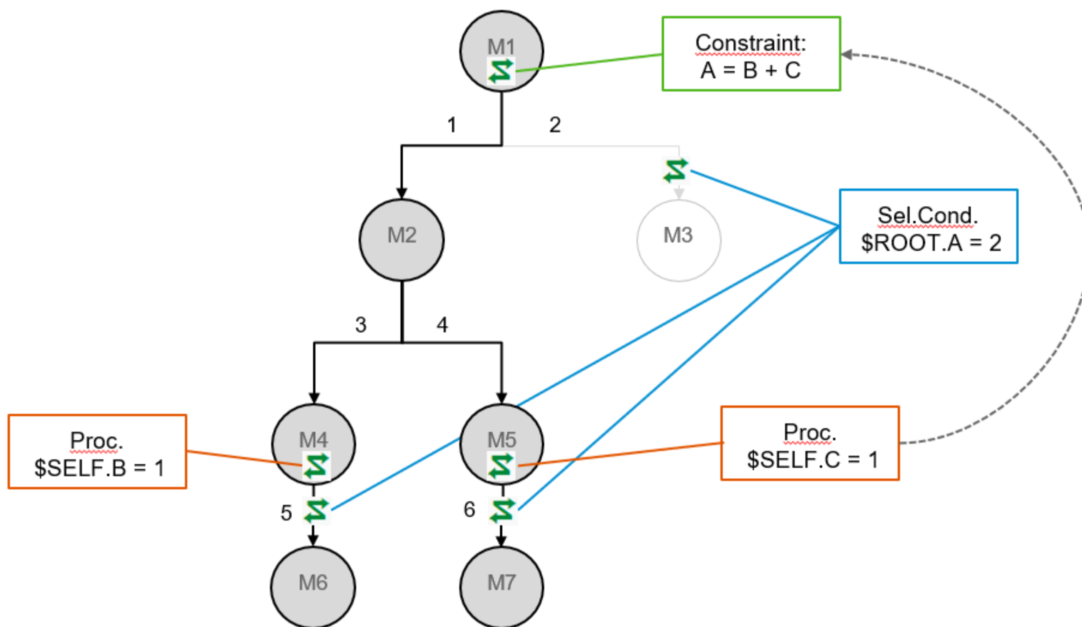
LO-VC auto-deletes such conflicting defaults if the conflict is detected in an assignment operation. The default is then overwritten. One problem with the approach is that defaults can cause inconsistencies in many more ways than just by failing assignment operations. Whether a conflict due to a default can be resolved then also depends on the order dependencies are evaluated in (which cannot be controlled by the modeler): Did the failing assignment operation run first or was it the equally failing smaller-comparison?
The AVC, on the other hand, resolves more inconsistencies than LO-VC, because it uses a different algorithm, and can delete more than one default value simultaneously to restore consistency. Please note that the characteristics where defaults are deleted are not necessarily assigned a value afterwards.

MULTI-LEVEL BOM EXPLOSION

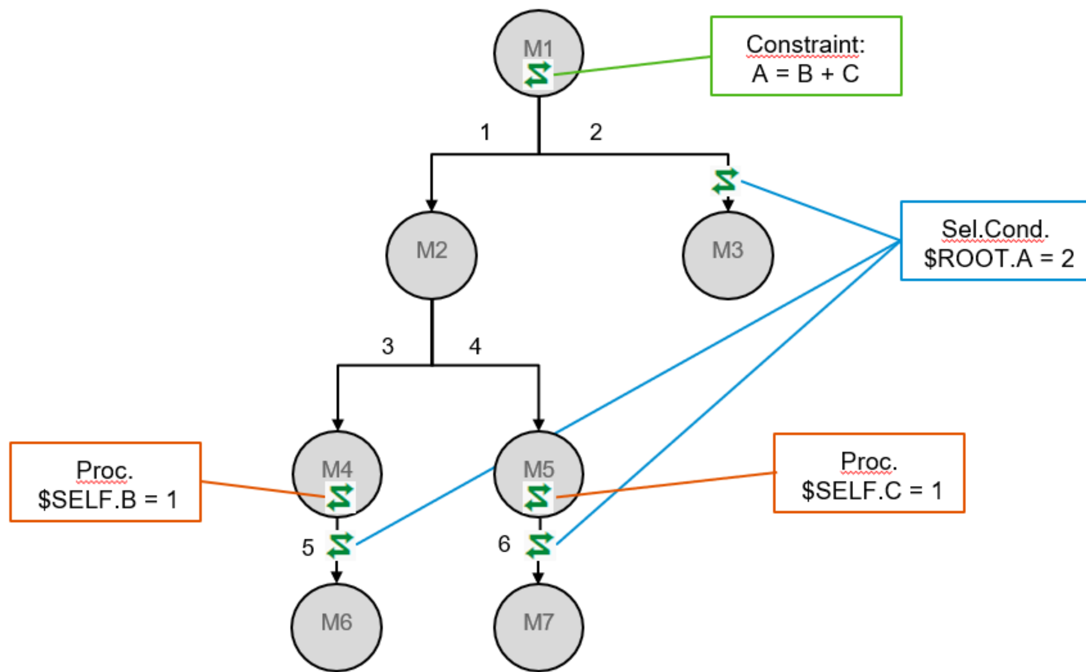
The interaction between multi-level BOM explosion and dependency processing has been redesigned for AVC. The main goals here were ease of understanding and predictability. It should be noted that the AVC engine presently supports a single BOM explosion strategy only. See section Planned Improvements for future enhancements.

BOM Explosion Strategy

The LO-VC explodes the BOM in a breadth-first manner (see https://en.wikipedia.org/wiki/Breadth-first_search). During a BOM explosion, it interleaves processing the dependencies attached to the selected instance's configuration profile, and the dependencies attached to BOM items. Consider the following example: When entering the configuration, first the root instance M1 is processed, running its constraint, which has no effect, because the characteristics B and C belong to instances, which are not active yet. Next M1's BOM is exploded. M2 has no selection condition, so it becomes active. M3 has a selection condition, which is not fulfilled, so it is not active. As M2 does not bring further dependencies, its BOM is exploded next, making M4 and M5 active. Then the dependencies attached to M4 are run, assigning B to 1. M5 is next, resulting in the assignment C = 1. Now all the characteristics needed for M1's constraint are there, so the constraint assigns A to 2. Because of this, exploding the BOMs of M4 and M5 results in the selection of M6 and M7.



Now LO-VC detected that there were upward changes to M1 requiring a new BOM explosion. During this next iteration of BOM explosion, the already selected components are kept active in the background with their current configuration until their parents become deactivated. Because of this, in a second iteration M3 is also selected. Apart from that, there are no further changes in the model (as depicted below).



In this example, the BOM explosion of LO-VC terminates after passing the model twice.
In general, models might need many iterations for reaching a state where no more upward changes occur. To avoid infinite loops, a cut-off after a certain number of iterations is enforced.

AVC currently provides a single-pass strategy and does not react to upward changes. The first picture already shows the BOM explosion result for the example. This means that the result of BOM explosion does not change when executed multiple times (see also section Build-From-Top). Eventually, loops shall be made available for the AVC as discussed in section Planned improvements.

Build-From-Top

In AVC, the BOM structure is re-calculated from top to bottom by first de-activating all instances below root, and then iteratively re-activating and validating them ("Build-From-Top approach"). In LO-VC, lower instances remain active when the BOM explosion starts and their assignments are considered. Therefore, repeated execution of the BOM explosion may lead to different results – in some cases executing a BOM explosion twice on identical user inputs may lead to two different results. The Build-From-Top approach of AVC guarantees that results are stable. Additionally, the LO-VC approach cannot prevent sequence effects during the interactive configuration, which means that the configuration result depends on the order in which the user visits the instances: Configuring instance A first and instance B second may yield different results than doing it in the reverse order, even if the overall set of user choices is identical for instances A and B. Such sequence effects are resolved by AVC when the BOM explosion is executed. Moreover, this approach guarantees that a reload from DB results in exactly the saved configuration (presuming that the model did not change). Overall, in the AVC engine, transparency and predictability for the modeler are increased.

Inconsistency During BOM Explosion

If an inconsistency occurs during BOM explosion, the result of continuing computation is not reliable. Therefore, AVC interrupts the BOM explosion in case of an inconsistency, and gives the user a clear guidance which instances were already validated to be part of the exploded BOM, and on which instance the inconsistency was detected. This behavior is predictable for the modeler and allows the user to revise those instances where user assignments might have caused the inconsistency, and to change user value settings before re-exploding the BOM to achieve a consistent and validated state.

Dependency Processing in Multi-Level Configuration Models

In principle, it would be necessary to perform a complete BOM explosion, and to re-evaluate all dependencies, in order to compute all consequences of a user's input. Since this is highly performance intensive, especially for large models, both configurators offer the possibility to validate the user input of a single instance without complete BOM explosion ("instance validate"). In principle, this instance validate is a trade-off between performance and correctness: To some extent, BOM explosion and dependency results are assumed to be unchanging.

The instance validate of LO-VC considers those instances as active that were active after the last BOM explosion. When the user enters a new value-assignment, the classical dependencies are re-calculated only on the instance focused by user. On all other instances, the classical dependencies are not processed. Their configuration is only input for the dependency processing on the focus instance. Constraints are processed globally, so that they can affect any active instance.

In AVC, the instance validate also operates on the set of instances that was active after the last BOM explosion. But on these instances, the complete approach described in section Build-From-Top is performed, including re-calculation of dependencies on all instances, but without performing an actual BOM explosion. That is, the set of active instances is assumed to be unchanged in the instance validate, but otherwise the results are fully accurate.

PLANNED IMPROVEMENTS

All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon when making purchasing decisions.

BOM explosion strategy

We plan to enhance the advantages of the build-from-top approach by combining this strategy with reacting on upward changes (loops). The goal is to guarantee that identical inputs always yield identical results.

High-level dependencies on BOM items

We are currently working on a design for a more generic solution to attach dependencies to BOM items.

Default handling

We have a plan for the user to be able to block and re-enable both static and dynamic defaults on characteristic level. To avoid inconsistencies, deletion, or overwriting of defaults on non-focus instances - at least in a BOM-explosion-with-loops-scenario - is currently under discussion, which then would be combined with a UI feature that allows to display recent changes to the user.

Syntax enhancements

We plan to improve the AVC syntax, for example by providing key access to tables, a richer syntax for multi-valued characteristics, or improved string processing. There shall also be a new extension concept for AVC (variant functions).

www.sap.com/contactsap

© 2018 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See www.sap.com/copyright for additional trademark information and notices.

THE BEST RUN