Algorithm 1 Evolve Configuration. It creates a new model with the evolved configurations of all the tenants.

Inputs: V: Evolved variability model.
R_{pre}: Previous configuration model.
Req: New requirements.
Output: $(R_{new}, Boolean)$: New evolved configuration model and a Boolean value *full*.

$R_{new} \leftarrow V$ /* Configuration model mirrors variability model */
/* Copy previous configurations */
$R_{new}.VSPEC_{res} \leftarrow \{(v \in R_{pre}.VSPEC_{res} : R_{pre}.resolved(v) \in V.VSPEC\}$
$R_{new}.resolved \leftarrow \{(r, v) \in R_{pre}.resolved : r \in R_{new}.VSPEC_{res}\}$
$R_{new}.decision \leftarrow \{(r, x) \in R_{pre}.decision : r \in R_{new}.VSPEC_{res}\}$
$R_{new}.value \leftarrow \{(r, x) \in R_{pre}.value : r \in R_{new}.VSPEC_{res}\}$
$R_{new}.instance \leftarrow \{(r, v) \in R_{pre}.instance : r \in R_{new}.VSPEC_{res}\}$
/* Add new requirements */
$R_{new}.VSPEC_{res} \leftarrow R_{new}.VSPEC_{res} \cup Req.VSPEC_{res}$
$R_{new}.resolved \leftarrow R_{new}.resolved \cup Req.resolved$
$R_{new}.decision \leftarrow (R_{new}.decision \setminus \{(r, x) \in Req.decision : r \in R_{new}.VSPEC_{res}\}) \cup Req.decision$
$R_{new}.value \leftarrow (R_{new}.value \setminus \{(r, x) \in Req.value : r \in R_{new}.VSPEC_{res}\}) \cup Req.value$
$R_{new}.instance \leftarrow R_{new}.instance \cup Req.instance$
/* Add mandatory features */
$R_{new}.decision \leftarrow (R_{new}.decision \setminus \{(r, x) \in Req.decision : V.mandatory(Req.resolved(r))\}) \cup \{(v, true) \in V.mandatory\}$
/* Check if all needed configurations are present */
$full \leftarrow$ fullConfigurationModel(R_{new}, V)
return $(R_{new}, full)$

Algorithm 2 Difference Configuration. It calculates the differences between the previous configuration and the new evolved configuration model.

Inputs: R_{pre}: Previous configuration model.
R_{new}: New evolved configuration model.
Output: $Diff$: Differences.

$Diff \leftarrow R_{new}$ /* Guarantee well-formed resolution */
$CHANGES \leftarrow Diff.VSPEC_{res} \cup R_{pre}.VSPEC_{res}$
$CHANGES \leftarrow CHANGES \setminus \{r \in R_{pre}.CHOICE_{res} : R_{pre}.decision(r) = R_{new}.decision(r)\}$ /* Only choices that have changed */
$CHANGES \leftarrow CHANGES \setminus \{r \in R_{pre}.VARIABLE_{res} : R_{pre}.value(r) = R_{new}.value(r)\}$ /* Only variables that have changed */
$Diff.decision \leftarrow Diff.decision \setminus \{(r, x) \in Diff.decision : r \notin CHANGES\}$
$Diff.variable \leftarrow Diff.variable \setminus \{(r, x) \in Diff.variable : r \notin CHANGES\}$
$Diff.instance \leftarrow Diff.instance \setminus \{(r, v) \in Diff.instance : r \notin CHANGES\}$
$Diff.VSPEC_{res} \leftarrow CHANGES$
return $Diff$

Algorithm 3 Create Weaving/Unweaving Model. It generates a new weaving model that specifies the elements will be incorporated and/or removed from the previous deployed architecture.

Inputs: $NewConf$: New evolved configuration model.
$Diff$: Differences.
$EvoArch$: Evolved base software architecture.
$AppArch$: Previous deployed software architecture.
Output: $V_{weaving}$: Weaving model. /* Well-formed model */

$V_{weaving} \leftarrow NewConf$
/* Incorporate differences */
$V_{weaving}.VSPEC_{res} \leftarrow V_{weaving}.VSPEC_{res} \cup Diff$
$V_{weaving}.decision \leftarrow V_{weaving}.decision \cup \{(v, true) : v \in Diff.UNSELECTIONS\}$
$vps \leftarrow (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing)$
for all $v \in Diff$ **do**
  $vp \leftarrow v \oplus$ `''VP''` /* String concatenation */
  $vps.VP \leftarrow vps.VP \cup \{vp\}$
  $vps.binding \leftarrow vps.binding \cup \{(vp, v)\}$
  $elements_{pre} \leftarrow$ getMOFRefs($v, AppArch$) /* Get MOF elements associated with functionality v in previous model */
  $elements_{evo} \leftarrow$ getMOFRefs($v, EvoArch$) /* Get MOF elements associated with functionality v in evolved model */
  $vps.refs \leftarrow vps.refs \cup \{(vp, elements_{pre} \cup elements_{evo})\}$
  if $v \in V_{weaving}.CHOICE_{res} \wedge Diff.SELECTIONS$ **then**
    if $elements_{pre} \neq \varnothing \wedge elements_{pre} \subseteq M_{App}$ **then** /* Update existing MOF elements */
      $vps.type \leftarrow vps.type \cup \{(vp,$ `FragmentSubstitution`$)\}$
    else /* Weave new MOF elements */
      $vps.type \leftarrow vps.type \cup \{(vp,$ `OpaqueVariationPoint`$)\}$
      $ovpType \leftarrow$ getSpecialSubstitution(v) /* Get weaving pattern type */
      $vps.ovpType \leftarrow vps.ovpType \cup \{(v, ovpType)\}$
      $spec \leftarrow$ getSemanticSpecification($ovpType$) /* Get M2M transformation */
      $vps.semantic \leftarrow vps.semantic \cup \{(ovpType, spec)\}$
    end if
  else if $v \in V_{weaving}.VARIABLE_{res} \wedge v \in Diff.MODIFICATIONS$ **then** /* Update MOF variables */
    $vps.type \leftarrow vps.type \cup \{(vp,$ `ParametricSlotAssignment`$)\}$
  else /* Unweaving existing MOF elements */
    $vps.type \leftarrow vps.type \cup \{(vp,$ `OpaqueVariationPoint`$)\}$
    $ovpType \leftarrow$ getSpecialSubstitution(`''unweaving''`) /* Get unweaving pattern type */
    $vps.ovpType \leftarrow vps.ovpType \cup \{(v, ovpType)\}$
    $spec \leftarrow$ getSemanticSpecification($ovpType$) /* Get M2M transformation */
  end if
  associateMOFReferences($vps.refs(vp)$, $vps.type(vp)$) /* Set up source and target references according the type of variation point */
end for
$V_{weaving}.variationPoints \leftarrow vps$
return $V_{weaving}$