



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.151

Corrigendum 1
(04/2012)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE
ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – User Requirements
Notation (URN)

User requirements notation (URN) – Language
definition

Corrigendum 1

CAUTION !

PREPUBLISHED RECOMMENDATION

This prepublication is an unedited version of a recently approved Recommendation. It will be replaced by the published version after editing. Therefore, there will be differences between this prepublication and the published version.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU [had/had not] received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2012

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Modifications introduced by this corrigendum are shown in revision marks. Deleted text is struck-through and red, thus: ~~deleted text~~. Inserted text is underlined and blue, thus: inserted text. A figure is only shown in this corrigendum, if it replaces the original. Unchanged text is replaced by ellipsis (...). Some parts of unchanged text (clause numbers, etc.) have been kept to locate the correct insertion points.

Recommendation ITU-T Z.151

User requirements notation (URN) – Language definition

Corrigendum 1

2 References

...

[ITU-T Z.150] Recommendation ITU-T Z.150 (2011~~03~~), *User Requirements Notation (URN) – Language requirements and framework*.

...

8.2 UCM maps and path nodes

...

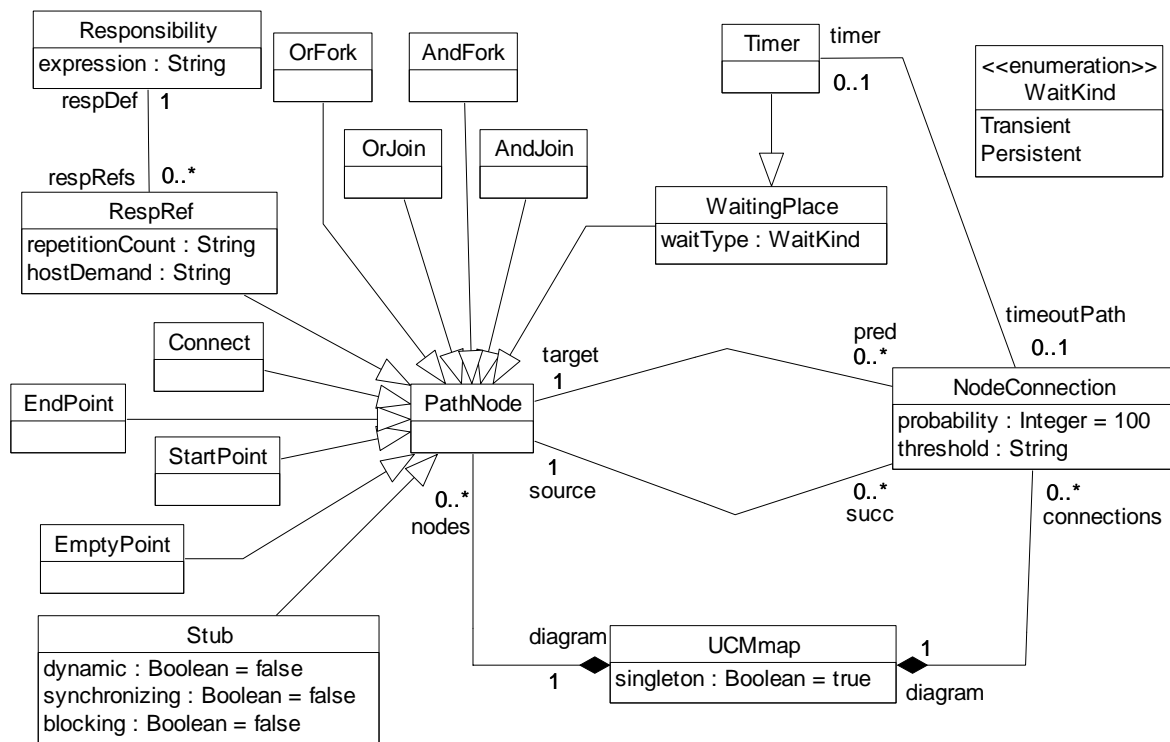


Figure 54 – Abstract grammar: UCM paths and path nodes

8.2.1 UCMmap

UCMmap serves as a container for all path nodes and component references of a map. A map may be a singleton, i.e., only one [runtime](#) instance of it ~~shall exist in the UCM specification~~. A map may be reused as a plug-in map for stubs (see Figure 54, Figure 68, and Figure 76).

a) *Abstract grammar*

Attributes

...

- singleton (Boolean): Indicates whether one (true) or more (false) runtime instances of a *UCMmap* ~~shall may~~ exist ~~in the UCM specification~~. Default value is true (i.e., only one [runtime](#) instance ~~shall may~~ exist).

...

c) *Semantics*

A *UCMmap* may contain zero or many UCM paths consisting of *NodeConnections* and *PathNodes* which may optionally as well as partially be bound to structural elements called components via *ComponentRefs*. A singleton map exists only once ~~at runtime in the UCM specification~~, i.e., if the same singleton map is used as a plug-in map (see *PluginBinding*) for two different stubs (see *Stub*), the same (and only) [runtime](#) instance of the map in the UCM specification is used for both stubs. If, however, the map is not a singleton, a different map [runtime](#) instance is used for each different [runtime](#) instance of a stub that uses the map as its plug-in map. For a more detailed discussion on [runtime](#) instances of maps in the UCM specification, see clause 8.3.1.

...

8.2.4 Responsibility

...

a) *Abstract grammar*

...

Relationships

...

- Association with *RespRef* ([01..*](#)): A *Responsibility* definition may be [referenced by](#) ~~the definition for at least one~~ responsibility references.

...

8.2.13 WaitingPlace

...

c) *Semantics*

...

Table 2 gives an overview of the decision process for continuing past the waiting place (PWP) or generating a warning (WAR) based on the condition of the waiting place (CWP) and the trigger

counter (TC). The overview assumes that the traversal of the UCM path has already reached the ~~timer~~ [waiting place](#) via the waiting path (i.e., waiting counter > 0). Table 2 is read row by row. For example, the last row says that if the CWP evaluates to false (first column), then a warning is generated if TC equals zero (second column) and traversal continues past the waiting place if TC is greater than zero (third column).

...

8.3.1 Stub

...

c) Semantics

...

Types of stubs

...

A *visit* in the context of synchronizing and blocking stubs is defined by how often an in-path of the stub has been traversed. If an in-path is traversed the first time, then it is the first visit of the stub. If the same in-path is traversed the n^{th} time, then it is the n^{th} visit of the stub. If another in-path of the stub is traversed for the first time, then it is the first visit of the stub. Plug-in maps that have been instantiated because of a visit are said to belong to the visit.

The concept of a visit also applies to unconnected start points on non-singleton plug-in maps regardless of the type of the plug-in map's stub. When such an unconnected start point is traversed for the n^{th} time, it is the n^{th} visit of the plug-in map and hence the plug-in map instantiated for the n^{th} visit is traversed.

Plug-in map [runtime](#) instances

Plug-in maps that are plugged into a stub are instantiated when the stub is reached the first time during the traversal of a UCM path. The fact that stubs are often used to restructure a complicated map implies that a stub [runtime](#) instance must contain not more than one [runtime](#) instance of a plug-in map at any time (with the exception of replicated plug-in maps which require that the specified number of [runtime](#) instances is created). This also applies to a stub that is used in a loop. The "not more than one map [runtime](#) instance per stub [runtime](#) instance" rule ensures the equivalence of a plug-in maps-based UCM specification with its flattened representation that uses only one single map. Synchronizing stubs are an exception for this rule and are discussed later on in this clause.

...

- Map G is a singleton and therefore the same [runtime](#) instance of this map is used by the stubs on Map A, Map B, and Map C. The same applies to Map H and the stubs on Map D, Map E, and Map F.
- Map I, on the other hand, is not a singleton. Therefore, the stubs on Map G and Map H use different [runtime](#) instances of Map I.
- Finally, if a group of stubs ~~are may want~~ to use the same [runtime](#) instance of a plug-in map, ~~T~~ this is achieved with an intermediary layer of singleton maps. For example, the group of stubs on Map A, Map B, and Map C uses the same [runtime](#) instance of Map I but the group of stubs on Map D, Map E, and Map F uses a different [runtime](#) instance of Map I.

...

Figure 72 – Example: UCM plug-in map runtime instances

Flattened UCM models

...

The semantics for a dynamic stub is similar to static stubs in that a dynamic stub shall~~may~~ contain only one runtime instance of each of its plug-in maps at a time. The semantics differs as the purpose of a dynamic stub is to model AND-forks and OR-joins in addition to simple hierarchical structuring (see Figure 73). Each in-path is equivalent to an AND-fork that is connected to the flattened plug-in maps according to the specified *InBindings*. Analogously, each out-path corresponds to an OR-join connected to the flattened plug-in maps based on the specified *OutBindings*. Plug-in bindings are indicated in the example below by labelling in-paths, out-paths, start points, and end points with *iN* and *oN*. Preconditions of plug-in maps are indicated in square brackets next to the name of the plug-in map.

...

The semantics for a synchronizing stub in terms of runtime instances of maps, however, is slightly different from the semantics for static and dynamic stubs, because the plug-in maps bound to a synchronizing stub have to act as one group. If an in-path of the synchronizing stub is visited for a second time, a second group of plug-in maps have to~~must~~ be created. Therefore, synchronizing stubs can contain more than one runtime instance of a plug-in map at the same time. This behaviour, however, is equivalent to one runtime instance with tokens flowing between AND-forks and AND-joins that can only synchronize if they were created by an AND-fork during the same visit. The synchronizing stub is therefore still conceptually equivalent to its flattened counterpart (see Figure 74) with each in-path converted to an AND-fork and each out-path converted to an AND-join. The connections of the AND-fork and AND-join to the flattened plug-in maps are again based on the specified *InBindings* and *OutBindings*.

...

e) Examples

Multiple plug-in map runtime instances for a synchronizing stub with multiple in-paths are created as follows. If an in-path is visited for the first time after a different in-path was visited the first time, then no new plug-in map runtime instances need to be created, because both traversals belong to the same visit. See the UCMs in Figure 75 and the first, second, and third column of Table 5 for an example. Given a synchronizing stub with three in-paths *i1*, *i2*, and *i3*, two plug-in maps bound to the stub as indicated in the example below, and a traversal order of the in-paths (*i1*: 1st, 4th, 5th; *i2*: 2nd, 6th; *i3*: 3rd, 7th, 8th, 9th), there will be four visits where runtime instances of both plug-in maps P1 and P2 are created. The first runtime instances of P1 and P2 are created at the first traversal of in-path *i1*. The second and third traversals do not cause new runtime instances to be created because these in-paths have not yet been used for the first runtime instances, and hence are part of the same visit. The fourth traversal creates the second set of runtime instances (second visit) because in-path *i1* is traversed for the second time. The fifth traversal creates the third set of runtime instances because in-path *i1* is again traversed. The sixth and seventh traversals use the runtime instances of the plug-in maps that belong to the second visit because the events go to the longest-waiting runtime instance of a plug-in map. The eighth traversal uses the third set of runtime instances. Finally, the ninth traversal causes the fourth set of runtime instances to be created because in-path *i3* is traversed for the fourth time.

The traversal of in-paths is important for the creation of plug-in map runtime instances. ~~but not the traversal of start points on the plug-in map (e.g., P1's start point is traversed five times because in-~~

~~paths i1 and i2 are both bound to the start point).~~ however, is not important except in the case of an unconnected start point on a non-singleton plug-in map. Such a start point may also cause the creation of a plug-in map runtime instance. Consequently, if the needed runtime instance has already been created by the arrival at the unconnected start point, the traversal of an in-path then must not trigger the creation of a plug-in map runtime instance.

NOTE – Such a start point does not exist in the plug-in maps in Figure 75 as all of the start points are connected to the stub via plug-in bindings.

Furthermore, if a replication factor is defined for a plug-in map, then ~~as many~~not one runtime instances of the plug-in map ~~is created each time but as many~~ as specified by the replication factor are created.

NOTE – It is only possible to define a~~A~~ replication factor other than the default value~~can only be defined~~ for plug-in maps of dynamic stubs.

Table 5 – Runtime iInstances and synchronizing stubs

#	In-path	Resulting Action {specified synchronization threshold}	Resulting Action {default synchronization threshold}
...			
2	i2	continue with i2 on 1 st P1 (for the second time on that <u>runtime</u> instance) and 1 st P2	continue with i2 on 1 st P1 (for the second time on that <u>runtime</u> instance) and 1 st P2 {the synchronization threshold is reached and traversal continues past the stub}
...			
6	i2	continue with i2 on 2 nd P1 (for the second time on that <u>runtime</u> instance) and 2 nd P2	continue with i2 on 2 nd P1 (for the second time on that <u>runtime</u> instance) and 2 nd P2 {the synchronization threshold is reached and traversal continues past the stub}
...			

If the synchronization threshold is not specified in the example in Figure 75, then the default behaviour stipulates that as many plug-in map runtime instances must arrive at the out-path as are traversed in parallel before the traversal is allowed to continue. The fourth column in Table 5 explains the behaviour of the synchronizing stub in this case assuming that both plug-in maps are selected.

The synchronization threshold is always specified upon first arrival at a stub during each visit. Subsequent arrivals during the same visit along other in-paths do not change the synchronization threshold for that visit, even if the number of plug-in map runtime instances that are being traversed changes.

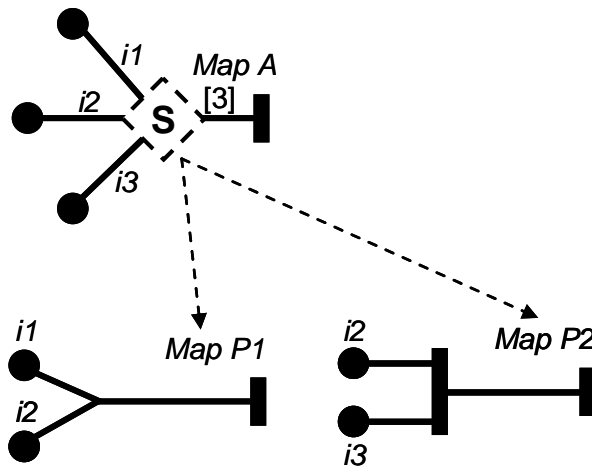


Figure 75 – Example: UCM synchronizing stub with threshold

8.3.2 PluginBinding

PluginBinding defines the binding (i.e., connection) of behavioural and structural specifications on a parent map to behavioural and structural specifications on a plug-in map with the help of *ComponentBindings* (covered in clause 8.4), *InBindings*, and *OutBindings*. A plug-in binding is contained by a stub and has a precondition that defines when the plug-in map is to be selected. Furthermore, a replication factor can be defined for a plug-in map, specifying how many [runtime](#) instances of the plug-in map are to be traversed in parallel. Finally, a plug-in map has a probability value stating the likelihood with which the plug-in map is selected in the UCM specification (see Figure 68 and Figure 76).

a) Abstract grammar

Attributes

...

- *replicationFactor* (String): The replication factor is an Integer expression that indicates how many [runtime](#) instances of the plug-in map are used.

...

Constraints

...

- The *UCMmap* containing the *Stub* of a *PluginBinding* is the same as the *UCMmaps* that contain the *NodeConnections* that belong to the *OutBindings* of the *PluginBinding*.

[i1. A *UCMmap* must not be associated with more than one *PluginBinding* of the same *Stub*.](#)

...

c) Semantics

...

Second, a *replicationFactor* defines for a plug-in map how many [runtime](#) instances of the plug-in map are to be traversed in parallel. A replication factor other than the default value may be defined for dynamic stubs but not for static stubs. Replicated maps for a dynamic stub are conceptually the

same as copying one UCM map many times and plugging all copies with the same preconditions and the same bindings into the same stub.

...

8.3.3 InBinding

...

a) *Abstract grammar*

...

Constraints

- a. The target *PathNode* of the *NodeConnection* of an *InBinding* is the *Stub* that contains the *PluginBinding* of the *InBinding*.

a1. A *StartPoint* must occur only once in all *InBindings* of a *PluginBinding*.

a2. Each in-path of a stub must be in at least one *InBinding*.

...

8.3.4 OutBinding

...

a) *Abstract grammar*

...

Constraints

- a. The source *PathNode* of the *NodeConnection* of an *OutBinding* is the *Stub* that contains the *PluginBinding* of the *OutBinding*.

a1. An *EndPoint* must occur only once in all *OutBindings* of a *PluginBinding*.

a2. Each out-path of a stub must be in at least one *OutBinding*.

...

c) *Semantics*

The traversal of a UCM path utilizes the *OutBindings* of a *Stub*'s plug-in map to move from the plug-in map back to the parent map. The traversal shall only return to the same *Stub* through which the plug-in map was entered. The traversal, however, does not always return to only one *Stub*. For example, if the traversal of two UCM paths entered a plug-in map through two different *Stubs* S1 and S2 and the two UCM paths subsequently synchronize in the plug-in map and reach an end point E, then the traversal returns to S1 and S2 as long as an *OutBinding* exists from E to one out-path in each *Stub*.

...

8.4.1 Component

...

a) *Abstract grammar*

...

Constraints

...

- d. The *Component* containment hierarchy established by the *includedComponents* relationship does not contain any cycles (i.e., a *Component* definition must not appear more than once on a path from a top node to a leaf node in the containment hierarchy).

d1. The context attribute of the *Component* definition of a *pluginComponent* in a *ComponentBinding* must be true.

d2. The kind attribute of the *Component* definition of a *pluginComponent* in a *ComponentBinding* must be *Team*.

d3. The *Component* definition of a *pluginComponent* in a *ComponentBinding* must not have a *ComponentType*.

...

8.4.4 ComponentRef

...

b) *Concrete grammar*

...

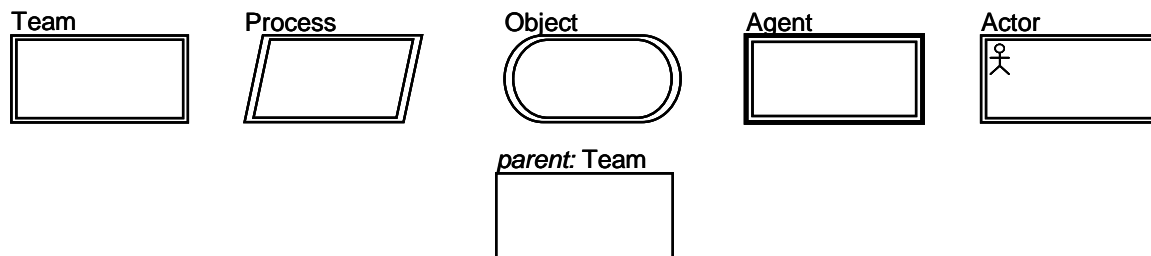


Figure 78 – Symbol: UCM protected and context-dependent component reference

...

8.4.5 ComponentBinding

...

a) *Abstract grammar*

...

Constraints

...

- b. The *UCMmap* of the *ComponentRef* associated as *pluginComponent* is the *UCMmap* of the *PluginBinding* of the *ComponentBinding*.

b1. Let M be a non-singleton *UCMmap* and let PG be a *pluginComponent* group where each *pluginComponent* is contained in M, has the same *Component* definition, and belongs to a *ComponentBinding* in the same *PluginBinding*. Then, each *Component* definition of a *parentComponent* of a *ComponentBinding* of the *pluginComponents* in PG must be the same.

b2. Let M be a singleton *UCMmap* and let PG be a *pluginComponent* group where each *pluginComponent* is contained in M, has the same *Component* definition, and belongs to a *ComponentBinding* in any *PluginBinding*. Then, each *Component* definition of a *parentComponent* of a *ComponentBinding* of the *pluginComponents* in PG must be the same.

- ~~c. The context attribute of the *Component* definition of a *pluginComponent* must be set to true.~~
~~d. The kind attribute of the *Component* definition of a *pluginComponent* must be set to Team.~~
~~e. The *Component* definition of a *pluginComponent* does not have a *ComponentType*.~~

...

10 URN interchange format

...

- **Enumeration class:** A *simpleType* element is declared for the enumeration class with the name attribute set to the class name. A restriction element is generated with base set to string. Each of the class attributes are appended to the restriction element as XSD enumeration elements with value set to the attribute name. They are presented first in the schema, in alphabetical order.

...

11.2.2 Requirements for path traversal mechanism

...

Table 13 – Requirements for path traversal mechanism

<i>ID</i>	<i>Requirement</i>
...	
29	Upon arrival at a persistent waiting place, transient waiting place, persistent timer, or transient timer along the waiting path <u>WP</u> , PT shall increase the number of arrivals <u>along WP</u> ed waiting paths by 1 (the initial number of arrivals <u>along WP</u> ed waiting paths is 0).
30	Upon arrival at a persistent waiting place or persistent timer along the trigger/release path <u>TRP</u> , PT shall increase the number of arrivals <u>along TRP</u> ed trigger/release paths by 1 (the initial number of arrivals <u>along TRP</u> ed trigger/release paths is 0).
31	Upon arrival at a transient waiting place or transient timer along the trigger/release path <u>TRP</u> , PT shall set the number of arrivals <u>along TRP</u> ed trigger/release paths to 1, if the number of arrivals <u>along the</u> ed waiting paths is greater than 0 (the initial number of arrivals <u>along TRP</u> ed trigger/release paths is 0).

ID	Requirement
32	The PCC for a waiting place W shall be fulfilled if <ul style="list-style-type: none"> a) W's condition evaluates to true or b) <u>the number of arrivals along the waiting path of W is</u> at least one waiting path and <u>the number of arrivals along the trigger/release path of W is</u> at least one trigger path has arrived at W.
...	
34	The PCC for a regular path <u>RP</u> of a timer T shall be fulfilled if the condition of T's regular path <u>RP</u> evaluates to true.
35	The PCC of the regular path <u>RP</u> of a timer T shall be fulfilled if <ul style="list-style-type: none"> a) the condition of RP T's regular path evaluates to false, and b) the condition of T's timeout path evaluates to false, and c) <u>the number of arrivals along the waiting path of T is</u> at least one waiting path and <u>the number of arrivals along the trigger/release path of T is</u> at least one trigger path have arrived at T.
...	
37	PT shall decrease <u>NWP</u> , the number of arrivals <u>along the</u> ed waiting paths _i by 1 when continuing past the waiting place or timer unless the number of arrived waiting paths <u>NWP</u> is already 0.
38	PT shall decrease <u>NTRP</u> , the number of arrivals <u>along the</u> ed trigger/release paths _i by 1 when continuing past the waiting place or timer unless the number of arrived trigger/release paths <u>NTRP</u> is already 0.
...	
41	Upon arrival at a dynamic stub S, PT shall traverse in parallel the number of <u>runtime</u> instances of a plug-in map M of S as specified by the replication factor of the plug-in binding of M.
42	For each plug-in map <u>runtime</u> instance M of stub S, PT shall move in parallel from S to start points SP ₁ , SP ₂ , ... and SP _N of plug-in map <u>runtime</u> instance M if <ul style="list-style-type: none"> a) the traversal is currently visiting path element S, and b) the traversal has reached the stub via a direct node connection <u>NC</u> from path element A to S, and c) there is an in-binding of M from <u>NC</u> to SP₁, from <u>NC</u> to SP₂, ..., and from <u>NC</u> to SP_N, and d) the PCC of the plug-in map <u>runtime</u> instance M is fulfilled.
43	The PCC for the plug-in map <u>runtime</u> instance of a static stub shall be always fulfilled.
44	The PCC for a plug-in map <u>runtime</u> instance M of a dynamic stub shall be fulfilled if the precondition of the plug-in binding for M evaluates to true.
45	The PCC for an out-path NO from the non-synchronizing stub S to a path element shall be fulfilled if <ul style="list-style-type: none"> a) the traversal is visiting an end point E on a plug-in map <u>runtime</u> instance M of S, and b) an out-binding <u>OB</u> from E to NO exists for M <u>and OB belongs to the same plug-in binding used to arrive at M</u>.
46	The PCC for an out-path NO from the synchronizing stub S to path element B shall be fulfilled if <ul style="list-style-type: none"> a) the traversal has visited end points E₁, E₂, ... or E_N on a plug-in map <u>runtime</u> instance M of S as often as specified by <u>ON</u>'s synchronization threshold during the same <u>visit</u>, and b) out-bindings <u>OB₁</u> from E₁ to O, <u>OB₂</u> from E₂ to O, ... and <u>OB_N</u> from E_N to NO exist <u>and OB₁, OB₂, ... and OB_N belong to the same plug-in binding used to arrive at M</u>.
47	PT shall synchronize a synchronizing stub's plug-in map <u>runtime</u> instances, only if they belong to the same <u>visit</u> .

<i>ID</i>	<i>Requirement</i>
48	Once for each visit upon first arrival at a synchronizing stub S with the default synchronization threshold for an out-path <u>O</u> , PT shall set the synchronization threshold of OS's out-path <u>O</u> to the number of S's plug-in map <u>runtime</u> instances with preconditions evaluating to true.
49	PT shall ignore the arrival of plug-in map <u>runtime</u> instances at an out-path <u>O</u> of a synchronizing stub S during a visit , if the synchronization threshold of the S's out-path <u>O</u> has been reached for the visit .
50	Upon arrival at a synchronizing stub S with blocking enabled, PT shall allow an in-path of S to be traversed another time when all plug-in map <u>runtime</u> instances of S have been traversed.
51	When all plug-in map <u>runtime</u> instances of a synchronizing stub S have been traversed in the N th visit , PT shall treat an in-path of S as having been visited N times, if the in-path was visited less than N times.
52	Upon arrival at a singleton map M, PT shall traverse the only <u>runtime</u> instance of M that exists in the UCM model.
53	Upon arrival at an unconnected start point S of a non-singleton traversal root map M, PT shall traverse the N th <u>runtime</u> instance of M where N is the number of times S has been visited in the current traversal.
54	Upon arrival at a non-singleton plug-in map M of a non-synchronizing stub S, PT shall traverse <ul style="list-style-type: none"> a) a different <u>runtime</u> instance of M per different <u>runtime</u> instance of a stub and b) the same <u>runtime</u> instance of M for the same <u>runtime</u> instance of S.
55	Upon arrival at a non-singleton replicated plug-in map M of a non-synchronizing stub S, PT shall traverse <ul style="list-style-type: none"> a) a different set of replicated <u>runtime</u> instances of M per different <u>runtime</u> instance of a stub and b) the same set of replicated <u>runtime</u> instances of M for the same <u>runtime</u> instance of S.
56	Upon arrival at a non-singleton plug-in map M along an in-path of a synchronizing stub S, PT shall traverse <ul style="list-style-type: none"> a) a different <u>runtime</u> instance of M per different <u>runtime</u> instance of a stub, and b) the Nth <u>runtime</u> instance of M for this <u>runtime</u> instance of S during the Nth visit, and c) the same <u>runtime</u> instance of M for the same <u>runtime</u> instance of S in the same visit.
57	Upon arrival at a non-singleton replicated plug-in map M along an in-path of a synchronizing stub S, PT shall traverse <ul style="list-style-type: none"> a) a different set of replicated <u>runtime</u> instances of M per different <u>runtime</u> instance of a stub, and b) the Nth set of replicated <u>runtime</u> instances of M for this <u>runtime</u> instance of S during the Nth visit, and c) the same set of replicated <u>runtime</u> instances of M for the same <u>runtime</u> instance of S in the same visit.
58	Upon entering a protected component reference C along a path P, PT shall start the traversal of P when no other path is being traversed in <u>any component reference of the component definition of</u> C.
59	PT shall interleave path nodes of parallel branches that are bound to the same component reference C _i if the component definition of C is of kind Object.
60	Upon arrival at a component reference C on a plug-in map <u>runtime</u> instance with a component plug-in binding to a component reference CP on the parent map <u>runtime</u> instance, PT shall use the component definition of CP as the component definition of C.
...	

12 Compliance statement

...

Table 14 – Compliance statement of this Recommendation against [ITU-T Z.150] language requirements

<i>ID</i>	<i>Z.150 Language Requirement</i>	<i>Type</i>	<i>E/D</i>	<i>Z.151 Clauses</i>	<i>Explanations</i>
...					
5	Specify satisficeability of goals and NFRs.	NFR	E	7.5	Evaluation strategies
...					
10	Specify business, organizational, and system objectives.	NFR	E	7.2	GRL actors
...					
27	Support performance indicators and mappings to satisfaction levels.	NFR	E		Not supported in this version
27 28	Support the mapping of input events and preconditions to output events and postconditions in various degrees of detail.	FR	E	8.2	UCM maps
28 29	Specify the set of input events at a scenario start point.	FR	E	8.2.6	Start point
29 30	Specify the set of output events at a scenario end point.	FR	E	8.2.7	End point
30 31	Specify preconditions at scenario start points.	FR	E	6.1.6, 8.2.6	Preconditions of start points
31 32	Specify postconditions at scenario end points.	FR	E	6.1.6, 8.2.7	Post-conditions of end points
32 33	Specify input sources (human or machine).	FR	E	8.4	Components
33 34	Specify output sinks (human or machine).	FR	E	8.4	Components
34 35	Specify responsibilities and references to these responsibilities.	FR	E	8.2.4, 8.2.5	Responsibilities and responsibility references
35 36	Specify system operations as causal flows of responsibilities (paths).	FR	E	8.2.2, 8.2.3	Path nodes and node connections
36 37	Specify alternative paths.	FR	E	8.2.8	Or-forks
37 38	Specify common paths.	FR	E	8.2.9	Or-joins
38 39	Specify condition-based decision-making at branching points.	FR	E	6.1.6, 8.2.8	Conditions on Or-forks
39 40	Define a data model and expression evaluator to express and evaluate conditions at branching points.	FR	E	9, 6.1.6, 8.2.8	URN data language, with use in conditions on Or-forks
40 41	Specify parallel or concurrent paths.	FR	E	8.2.10	And-forks
41 42	Specify synchronization of paths within a scenario.	FR	E	8.2.11	And-joins
42 43	Specify synchronization between paths from multiple scenarios.	FR	E	8.2.13, 8.2.15, 8.2.16	Waiting places, wait kinds, and connects
43 44	Specify timed synchronization, with a timeout path.	FR	E	8.2.14, 8.2.15, 8.2.16	Timers, wait kinds, and connects
44 45	Specify repetitive actions within a scenario.	FR	E	8.2.5	Responsibility reference with repetitionCount attribute

<i>ID</i>	<i>Z.150 Language Requirement</i>	<i>Type</i>	<i>E/D</i>	<i>Z.151 Clauses</i>	<i>Explanations</i>
45 46	Support hierarchical decomposition of scenarios.	FR	E	8.3	Stubs and plug-ins
46 47	Specify sub-scenarios as scenarios.	FR	E	8.3	Stubs and plug-ins
47 48	Specify sub-scenario preconditions and postconditions.	FR	E	8.3.2, 6.1.6	Conditions on plug-in bindings
48 49	Specify scenario containers with multiple sub-scenarios.	FR	E	8.3.1	Dynamic stubs
49 50	Define a data model and expression evaluator to select sub-scenarios in dynamic containers.	FR	E	9, 6.1.6, 8.3.1	URN data language, with use in conditions on dynamic stubs
50 51	Group-related scenarios.	FR	E	8.2, 8.5.1	UCM maps and scenario groups
51 52	Extract individual scenarios from grouped scenarios.	FR	E	8.5.2, 11.2	Scenario definitions and UCM path traversal mechanism
52 53	Specify individual scenarios using a data model and initializations.	FR	E	8.5.3, 9	Initializations and URN data model
53 54	Express desirable feature interactions in scenarios.	FR	E	8.5.2, 11.2	Scenario definitions and successful UCM path traversal mechanism
54 55	Detect undesirable feature interactions in scenarios.	FR	E	11.2	UCM path traversal mechanism errors and warnings
56	Specify scenario cancellation situations with scope.	FR	E		Not supported in this version
57	Specify scenarios describing recovery mechanisms.	FR	E		Not supported in this version
58	Specify qualitative time-dependent behaviour in scenarios.	FR	E		Not supported in this version
59	Specify timer types in time-dependent behaviour.	FR	E		Not supported in this version
55 60	Specify components and references to these components.	FR	E	8.4.1, 8.4.4	Components and component references
56 61	Specify scenarios without reference to components.	FR	E	8.2.2	Path nodes do not need to be bound to components
57 62	Specify scenarios where scenario elements are allocated to components.	FR	E	8.2.2, 8.4.4	Path nodes may be bound to components
58 63	Specify abstract components and COTS.	FR	E	8.4.2, 8.4.5	Component types and component bindings
59 64	Specify dynamic entities.	FR	E	8.4.3, 8.4.5	Component kinds and component bindings
60 65	Specify system boundaries.	FR	E	8.4.1, 8.4.3	Components of kind other than Actor
61 66	Specify the behaviour of the system's environment.	FR	E	8.2.2, 8.4.4	Path nodes not bound to components
62 67	Specify actors external to the system.	FR	E	8.4.1, 8.4.3	Components of kind Actor

<i>ID</i>	<i>Z.150 Language Requirement</i>	<i>Type</i>	<i>E/D</i>	<i>Z.151 Clauses</i>	<i>Explanations</i>
63 68	Support backward traceability from URN to source documents.	URN	E	6.1.2, 6.1.4	Unique model element identifier, metadata
64 69	Support forward traceability from URN to the other models used in the development process.	URN	E	6.1.2, 6.1.4	Unique model element identifier, metadata
65 70	Support facilities to connect URN elements to external requirements objects.	URN	E	6.1.2, 6.1.4	Unique model element identifier, metadata
66 71	Enable transformations to elements of other languages in the ITU-T family of languages and of UML.	URN	D	11.2	UCM path traversal mechanism
67 72	Support traceability between operational aspects of goal/NFR models and responsibilities/scenarios in scenario models.	URN	E	6.1.3	URN links
68 73	Support traceability between performance constraints in NFR models and responsibilities/scenarios/response-time requirements in scenario models.	URN	E	6.1.3	URN links
69 74	Support the testing of requirements.	URN	E	8.5	Scenario definitions
70 75	Support testing based on requirements.	FR	E	8.5	Scenario definitions
76	Support the evaluation of the satisfaction of goals and NFRs.	NFR	E	7.5, Appendix II	Evaluation strategies, GRL Evaluation Algorithms
71 77	Enable preliminary analysis of performance properties.	URN	E	8.6	UCM performance annotations
72 78	Attach performance/workload annotations to scenario elements.	FR	E	8.6	UCM performance annotations
73 79	Specify the environment's processing capacity, network delays, and services provided.	FR	E	8.6	UCM performance annotations
74 80	Specify response times in terms of target fragments of scenarios.	FR	E	8.5.2, 8.6	Scenario definitions, UCM performance annotations
75 81	Specify identifiers for the model elements.	URN	E	6.1.2	URNmodelElement with unique id attribute
76 82	Specify document versions.	URN	E	6.2.1	specVersion
83	Support textual annotations traceable to graphical elements.	URN	E	7.6.13	Comments
77 84	Support a graphical representation of requirements.	URN	E	6.2, 7.6, 8.7	Concrete grammar metaclasses and graphical concrete syntax
78 85	Support a tool-oriented interchange format.	URN	E	10	URN interchange format
79	Support textual annotations traceable to graphical elements.	URN	E	7.6.13	Comments
80 86	Support textual annotations displayable on conventional media.	URN	E	7.6.13	Comments
87	Support the grouping of any elements in a model.	URN	E	6.1.5	Concern
88	Support annotating any model element with name-value pairs.	URN	E	6.1.4	Metadata

<i>ID</i>	<i>Z.150 Language Requirement</i>	<i>Type</i>	<i>E/D</i>	<i>Z.151 Clauses</i>	<i>Explanations</i>
89	Support linking any pair of model elements.	URN	E	6.1.3	URN links

...

Annex A

URN Interchange Format: XML Schema

(This annex forms an integral part of this Recommendation)

The following XML schema defines the URN interchange format. It is explained in clause 10.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
== XML Schema for the User Requirements Notation (Recommendation ITU-T Z.151)
== Version: 20120112080903
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
            elementFormDefault="qualified">

...
<!-- ~~~~~ -->
<!-- ComponentBinding -->
<!-- ~~~~~ -->
<xsd:complexType name="ComponentBinding">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:ID" /> <!-- ADDED because ComponentBinding is not a URNmodelElement (no ID) -->
    <xsd:element name="parentComponent" type="xsd:IDREF"/> <!-- ComponentRef -->
    <xsd:element name="pluginComponent" type="xsd:IDREF"/> <!-- ComponentRef -->
  </xsd:sequence>
</xsd:complexType>

...
<!-- ~~~~~ -->
<!-- InBinding -->
<!-- ~~~~~ -->
<xsd:complexType name="InBinding">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:ID" /> <!-- ADDED because InBinding is not a URNmodelElement (no ID) -->
    <xsd:element name="startPoint" type="xsd:IDREF"/> <!-- StartPoint -->
    <xsd:element name="stubEntry" type="xsd:IDREF"/> <!-- NodeConnection -->
  </xsd:sequence>
</xsd:complexType>

...
<!-- ~~~~~ -->
<!-- OutBinding -->
<!-- ~~~~~ -->
<xsd:complexType name="OutBinding">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:ID" /> <!-- ADDED because OutBinding is not a URNmodelElement (no ID) -->
    <xsd:element name="endPoint" type="xsd:IDREF"/> <!-- EndPoint -->
    <xsd:element name="stubExit" type="xsd:IDREF"/> <!-- NodeConnection -->
  </xsd:sequence>
</xsd:complexType>

...
<!-- ~~~~~ -->
<!-- PluginBinding -->
<!-- ~~~~~ -->
<xsd:complexType name="PluginBinding">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:IDstring" /> <!-- ADDED because PluginBinding is not a URNmodelElement (no ID) -->
    <xsd:element default="100" name="probability" type="xsd:nonNegativeInteger"/>
    <xsd:element name="replicationFactor" type="xsd:string"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="in" type="InBinding"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="out" type="OutBinding"/>
    <xsd:element name="plugin" type="xsd:IDREF"/> <!-- UCMmap -->
    <xsd:element minOccurs="0" name="precondition" type="Condition"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="components" type="ComponentBinding"/>
  </xsd:sequence>
</xsd:complexType>

...
```

```

<!-- ~~~~~ -->
<!-- Responsibility -->
<!-- ~~~~~ -->
<xsd:complexType name="Responsibility">
  <xsd:complexContent>
    <xsd:extension base="UCMmodelElement">
      <xsd:sequence>
        <xsd:element name="expression" type="xsd:string"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="demands" type="Demand"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="respRefs" type="xsd:IDREF"/> <!-- RespRef -->
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
...
<!-- ~~~~~ -->
<!-- Variable -->
<!-- ~~~~~ -->
<xsd:complexType name="Variable">
  <xsd:complexContent>
    <xsd:extension base="UCMmodelElement">
      <xsd:sequence>
        <xsd:element default="Boolean" name="type" type="DatatypeKindxsd:IDREF" /> <!-- DatatypeKind -->
        <xsd:element minOccurs="0" name="enumerationType" type="xsd:IDREF"/> <!-- EnumerationType -->
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
...

```

Appendix I

Summary of the URN Notation

(This appendix does not form an integral part of this Recommendation)

I.1 Summary of Abstract Metamodel

...

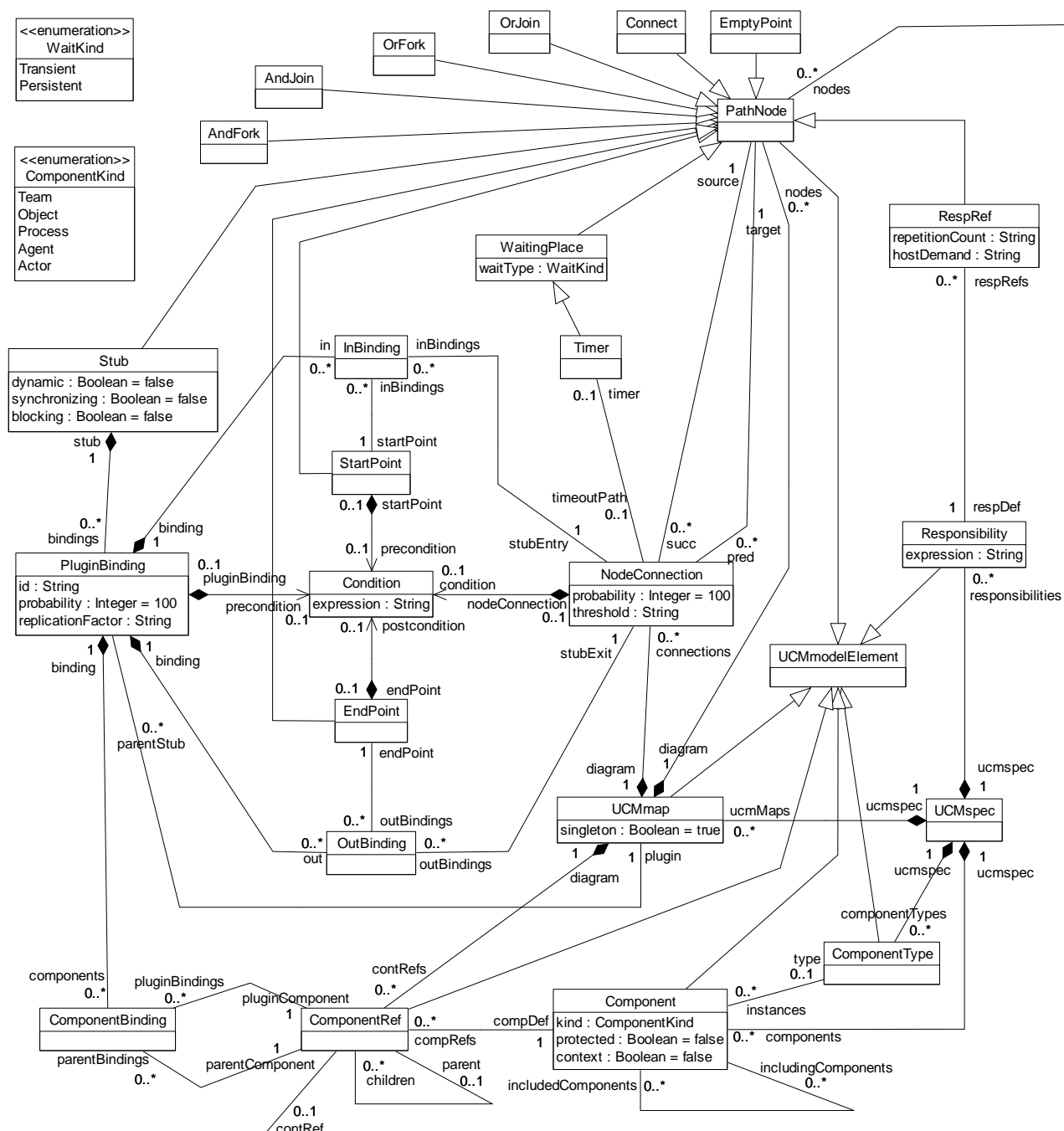


Figure 92 – Abstract grammar: UCM core overview

...

Appendix II

Examples of GRL Model Evaluation Algorithms

(This appendix does not form an integral part of this Recommendation)

...

II.2.2 Calculating quantitative evaluations for contribution links

...

Algorithm CalculateContributions

Inputs element:IntentionalElement, decompValue:Integer

Output contribValue:Integer

```

tolerance:Integer           // predefined tolerance, between 0 and 49
oneCont:Integer             // one weighted contribution
totalCont:Integer = 0       // weighted sum of the contribution links
hasSatisfy:Boolean         // a weighted contribution of 100 is present
hasDeny:Boolean            // a weighted contribution of -100 is present
hasSatisfy = (decompValue == 100)
hasDeny = (decompValue == -100)

// compute the weighted sum of contributions
for each link:Contribution in element.linksDest
{
    oneCont = link.src.quantitativeVal × link.quantitativeContribution
    totalCont = totalCont + oneCont
    if (oneCont == 100) hasSatisfy = true
    if (oneCont == -100) hasDeny = true
}
totalCont = totalCont / 100
contribValue = totalCont + decompValue

// contribution value cannot be outside [-100..100]
if (|contribValue| > 100)
    contribValue = 100 × (contribValue/|contribValue|)

// take tolerance into account if a weighted contribution of 100 or -100 is not present
if ((contribValue ≥ 100 – tolerance) and not(hasSatisfy))
    if (totalCont > 0) // positive contribution
        contribValue = max (decompValue, 100 – tolerance)           // case A
        // else there is nothing to do, contribValue remains unchanged.
    else if ((contribValue ≤ -100 + tolerance) and not(hasDeny))
        if (totalCont < 0) // negative contribution
            contribValue = min (decompValue, -100 + tolerance)       // case B
            // else there is nothing to do, contribValue remains unchanged.

return contribValue

```

Figure 103 – Example: Quantitative CalculateContributions algorithm

...

Table 15 – Example: Calculating contribution values with different tolerance values

<i>Case in Figure 103</i>	<i>hasSatisfy</i>	<i>decompValue</i>	<i>totalCont</i>	<i>tolerance limit</i>	<i>contribValue</i>
A	false	95	3	$100 - 10 = 90$	$\max(\text{decompValue}, 90) = 95$
hasSatisfy	true				$95 + 3 = 98$
below tolerance limit	false			$100 - 1 = 99$	$95 + 3 = 98$
B	false	95	-3	$100 - \underline{10} = 90$	$95 - 3 = 92$
hasSatisfy	true				$95 - 3 = 92$
below tolerance limit	false			$100 - 4 = 96$	$95 - 3 = 92$
A	false	85	13	$100 - \underline{10}8 = 902$	$\max(\text{decompValue}, \underline{902}) = 902$
hasSatisfy	true				$85 + 13 = 98$
below tolerance limit	false			$100 - 1 = 99$	$85 + 13 = 98$

...

II.2.4 Calculating quantitative evaluations for actors

...

$$\text{actor.quantitativeVal} = \left(\sum_{i=1}^n \text{elem}_i.\text{quantitativeVal} \times \text{elem}_i.\text{importanceQuantitative} \right) / \sum_{i=1}^n \text{elem}_i.\text{importanceQuantitative}$$

...

$$((100 \times 100) + (100 \times 29) + (-75 \times 60)) / (100 + 29 + 60) = 44.28$$

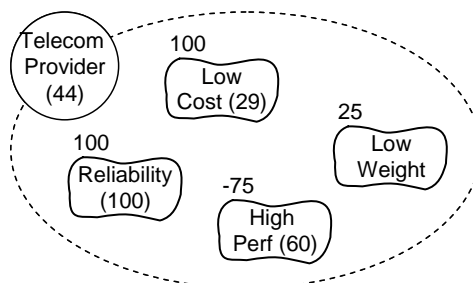


Figure 106 – Example: Quantitative evaluation of actors

...

II.3.2 Calculating qualitative evaluations for contribution links

...

```
Algorithm CalculateContributions
Inputs element:IntentionalElement, decompValue:QualitativeLabel
Output contribValue:QualitativeLabel

oneCont:QualitativeLabel           // one weighted contribution
ns:Integer = 0                     // number of Satisfied weighted contributions
nws:Integer = 0                    // number of WeaklySatisfied weighted contributions
nwd:Integer = 0                    // number of WeaklyDenied weighted contributions
nd:Integer = 0                     // number of Denied weighted contributions
nu:Integer = 0                     // number of Unknown weighted contributions
weightSD:QualitativeLabel          // partial weighted contribution from ns and nd
weightWSWD:QualitativeLabel        // partial weighted contribution from nws and nwd

// adjust the weighted contribution counters according to decompValue
AdjustContributionCounters(decompValue, ns, nws, nwd, nd, nu)

// compute the numbers of weighted contributions for each kind
for each link:Contribution in element.linksDest
{
    oneCont = WeightedContribution(link.src.qualitativeVal, link.contribution)
    AdjustContributionCounters(oneCont, ns, nws, nwd, nd, nu)
}

// check for the presence of unknown weighted contributions
if (nu > 0)
    contribValue = Unknown
else
{
    weightSD = CompareSatisfiedAndDenied (ns, nd)
    weightWSWD = CompareWSandWD (nws, nwd)
    contribValue = CombineContributions (weightSD, weightWSWD)
}

return contribValue
```

Figure 109 – Example: Qualitative CalculateContributions algorithm

...