# Position on Review of MPS Information Model Draft D

The following notes give the position taken by the MPS WG during the telecon of 04/09/2018 on both the outstanding action items from the review of Draft C, and comments received from DLR. This forms an attachment to the MoM for the telecon.

Conclusions are inserted below in blue text.

# Action Points from Review of Draft C

1. Global Constraints / Request Subtypes

We agreed to consider options for representation of Global Constraints:

*Global Constraints*

*This will be considered off-line.*

*Renaming is OK, but it is to be considered off-line how to represent Global Constraints.  Two proposals were made:*

*1.            There should be a single Global Constraints collection to which Constraints can be added or deleted.*

*2.            There should be a sub-type of Request to represent persistent standing orders or rules.  This could contain Constraints - equivalent to the Global Constraint.  It could also be used to express the instantiation of an Activity in response to an Event.*

I have looked into this and concluded that Option 2 is more flexible.  In practice, the existing representation of Planning Requests already provides the means to specify both Global Constraints and Standing Orders (the latter using Repetitions):

However, there are three distinct use case that would require constrained usage of the  current attribute set:

1. A one-shot “single” request may have constraints and a set of Activity Details – it *may* include [limited] repetition that allows a series of activities to be planned.
2. A standing order request may have constraints and a set of Activity Details, that must at the root comprise an [unbounded] repetition.
3. A Global Constraint is a standing order that has no Activity Details, only constraints.

I have updated the model to reflect 3 sub-classes of Request as above:



It is noted that currently Request Template/Instance has a set of Activity Details, but a single Constraint Node.

Currently, Repetitions can only contain a single Activity Detail (which itself can have children).  This means a standing order cannot invoke multiple root activities in response to the same trigger time/position/event.

If instead we allowed Repetitions to contain a set of Activity Details, then we could replace the Activities set by a Repetition in much the same way as we have a single Constraint Node for the constraints.  This would make the one-shot and standing order cases look the same – the Repetition becoming the equivalent of the Constraint Node.

I have also deleted the Global Constraint classes as it is no longer needed.

***WG Position 04/09/2018***

*The similarity between the three sub-classes was noted and a preference expressed to keep the model simple.*

*Repetition (subclass of Activity) is to be reformulated as an Activity Node that supports the specification of both single shot and standing order requests with defined repetition criteria, and supporting. Further detail on this is given in the position on DLR Comment 1 below.*

*Subtypes of Request are to be merged into a single class that has an Activity Node rather than a set of Activities as an attribute, similar to the way it has a Constraint Node instead of a set of constraints.*

*A Global Constraint will simply have no Activity Node associated with it.*

1. Activity Triggers

*It was proposed that a common structure should be defined that defines the Trigger (or Trigger Reference) for an Activity in the Activity Update.  This may be Time-based, Position-based or a Planning Event reference.  This will be considered off-line and a proposal made for discussion by the WG.*

Previously we had Start and End Time plus Start and End Position for all Activity Updates.

I propose replacing this with a combination of an Activity Trigger and a Duration.

The Activity Trigger structure has three subtypes as follows:



Start Time is always present – it may be the actual planned Start Time of the Activity (for a Time Trigger), or the predicted time for other sub-classes.  Once the Activity has actually been triggered by Plan Execution it will contain the actual Start Time.

Nothing else is required for a Time Trigger – the end-time can be derived from Start Time + Duration.

For a Position Trigger, the Start Position is given.  I have not included an End Position, as it does not seem necessary – but it could be added.

For an Event Trigger, a reference to the Trigger Event is given, together with a Time Offset.

It should, however, be noted that Position and Event Triggers may effectively be resolved to Time at any stage in a distributed planning system.  Event Triggers are normally only necessary if the Event is updated/detected downstream of the Planning function that generates the Plan – linkage to the Event is already provided in the Activity Instance through the Related Event.

***WG Position 04/09/2018***

*DLR proposal (see comment #2) to separate predicted and actual time on the base class was agreed. Actual time is optional as is only relevant in Activity Updates following execution. Triggers should be neutral on whether they are Start or End – they are simply a trigger point, name of attributes to be updated accordingly. Subtypes will contain the trigger time, position or event (with offset) respectively.*

1. Data Validation on Argument Values

*The WG concluded that data validation on Argument values is needed.  RT will propose an approach to extend the Argument Definition to support this.*

See extension to Argument Definition below:



This is similar to the approach on Resources, except that these used resource profiles that had evolution over time.

Question: should Status type arguments be supported?

***WG Position 04/09/2018***

*See additional points in DLR Comment 4.*

*Status type arguments are to be supported. There was debate on whether these should be strings or enumerated integers. Agreed preference was for enumerated integers, amongst other reasons considering the overhead of transferring strings on a space link.*

1. Argument Constraints

*It was also proposed that there should be the possibility to express constraints on the value of an Argument of a referenced object (Event or parent Activity).  We do not currently have a constraint type that supports this, but this could look very similar to Resource Constraints.  RT will consider and propose a solution to the WG.*

Additional Constraint Type added to support constraints on Argument Value.  This follows the pattern of the Simple Resource Constraint.

Note that the referenced Object will typically be the current Object “Me”, its parent “Source” or another related object e.g. “Related Event”.



***WG Position 04/09/2018***

*Following some debate, the inclusion of argument constraints was agreed.*

1. Related Events / Event Groups

From the telecon minutes:  *There is a potential requirement relating to the expression of constraints that reference related Events (e.g. AOS and LOS).  Agreed that this requires definition of related Events, as well as the instance of related Events. Will be taken off-line to consider how this should be represented.  Options: add Related Events (definitions) to Event Definition; define Event Groups that will also have Instances as a container for multiple related events.*

The approach I took was to add related events to Event Definitions, rather than defining Event Groups.  Both solutions are however, feasible.

My rationale was that in most cases the Events are an external input into the Planning System, which does not typically represent these as a layered model of Event Groups and Events.

If we were to have Event Groups, then the RelatedEvent attributes of Events and Activities would need to be able to reference both Events and Event Groups.

In expressions that reference a related Event, the means of referencing the Related Event would have an additional level of indirection.  Activity/Event.Related Event > Event Group > Event.  Either way we need a means of specifying which Related Event within the group – this probably means something along the lines of the Event in the set with id/name X.  Hence with the current model, the constraint would reference the Activity and then be expressed as something along the lines of Me.RelatedEvent(X).  If we introduce Event Groups this would have to be expressed in terms of Me.RelatedEvent.Member(X) – noting that RelatedEvent may be a set of both Events and Event Groups.



***WG Position 04/09/2018***

*The general preference of the WG was for the introduction of Event Groups as a distinct entity, as this seems a more elegant solution. No firm decision was taken, but RST will model this approach in advance of the Fall meeting to support discussion and a final decision.*

*In order to avoid complexity in the inclusion of Events and Event Groups in the related event argument of Activities, it was proposed that Event Groups could be a sub-class of Event (to be considered).*

# DLR Comments on MPS Information Model Draft D

1. Diagram: Activity Details and Repetitions

In order to support variable durations, we need a slider and an offset for the Repetition in order to determine the repetitionTime, which corresponds to an ActivityInstance.
We could omit the offset, however as sliders and offsets often come together, we might keep them together.

The offset is mandatory if we follow Roger’s suggestion and allow multiple activities to be part of the same Repetition.

For ease of use, we might replace *Tolerance: DurationExpression* by *Tolerance: DurationInterval*, which would allow e.g. *up to 5 minutes earlier or up to 10 minutes later*

***WG Position 04/09/2018***

*See also Action point 1 above.*

*There was concern that this approach may make the solution more complex and harder to understand for the majority of cases where the assumption is that repetitions refer to activity start time. It was nevertheless agreed that the slider+offset construct should be associated with each root activity listed in the repetition [now activity node], provided that these default to activity start time.*

1. Diagram: Activity Triggers
* *StartTime*We might distinguish in between
	+ *predictedStart: TimeInterval*
	+ *occurredStart: Time[0 .. 1]*

where occurredStart is provided, once the trigger has occurred (distinguish in between planned and executed), or even better

* + *predictedStart: TimeDistribution*
	+ *occurredStart: Time[0 .. 1]*

where *TimeDistribution* provides a probability distribution over time, which specifies when we expect the Trigger to occur.
Simple example for *PositionTrigger* or *EventTrigger*: even distribution over a finite *TimeInterval*.
*TimeTrigger* would override *predictedStart* to have probability 1 at *StartTime*

* *EventTrigger*In case a *TriggerEvent* has a duration, we need a slider

***WG Position 04/09/2018***

*See Action point 2 above.*

*Inclusion of separate predicted and option occurred Start Time was agreed.*

*Extension to include probability distribution was not agreed (considered too complex).*

*Events do not have duration so no slider is required.*

1. Diagram: Planning Activities
* *ActivityUpdate*
We could replace ‘Duration: Duration’ by ‘End: ActivityTrigger’. A duration can be represented by a relative trigger, which references the start and adds a Duration

***WG Position 04/09/2018***

*See also Action 2 above.*

*Agreed to add an optional End as a trigger, as well as Start.*

*Duration to be retained, but made optional.*

1. Diagram: Arguments

*We also need to be able to*

* *Check string arguments whether they obey a given Regex*
* *Check numeric values for accuracy (e.g. rounded to 0.1)*

***WG Position 04/09/2018***

*Agreed to add to Argument data validation. See also Action 3 above.*

1. Diagram: Argument and Resource Types, Comparators and Operators
* *Status*We’d prefer using *String (Enumerated)* over *Integer (Enumerated)*
* We’d prefer *!=* over */=*
* Do we need String as Resource Type?
* Is it correct that the resource type ‘Reference’ shall allow pointing at a model object, e.g. until 12 o’clock use activityA afterwards use activityB?

***WG Position 04/09/2018***

*See Action 3 above. Statuses to remain enumerated integers.*

*No strong views, so can change symbol for not equal.*

*It was previously agreed to retain string resource types.*

*It is correct that References are to allow the passing of pointers to Planning objects (Activities, Events, Resources.*

1. Diagram: Conditional Constraints

We introduced ArgumentConstraints in order to be able to verify input the planning system receives from an external interface. Constraints on the other hand shall define rules, how a plan may look like.

From this perspective, we should remove ArgumentConstraints and put more effort in ArgDef, where these restrictions should be located.

Unless we agree that we expect the planning process not only to create timeline entries but also to adapt the underlying model. However in this case I fear that loads of other constraints might seem useful, e.g. ‘maximum number of children to create’, ‘all children must have n children themselves, one called uplink, another datatake the third downlink, which have the constraints …’

***WG Position 04/09/2018***

*See Action 4 above. There are use cases for Argument Constraints, so they should be retained.*

1. Diagram Planning Requests
* *StandingRequest*should be removed as more information about how to repeat the request is required. We should use Repetition instead (if necessary, extend it)
* *GlobalConstraint* is just a *SingleRequest* without Activities and therefore can be removed

We therefore only need one *RequestTemplate* and one *RequestInstance*, which correspond to the remaining *SingleRequestTemplate* and *SingleRequestInstance* in the current diagram.

* regarding the new *Planning Request Validity*:

Shouldn’t we replace [0..1] by [0..\*] as there might be more than one window or a window with gaps…?

***WG Position 04/09/2018***

*See Action 1 above. Agreed to remove Request subtypes following introduction of an Activity Node that incorporates the functionality of the existing Repetition.*

*Agreed that Planning Request validity may have multiple windows.*

1. Diagram: Planning Events

We might also replace

1. *RelatedEvent: ObjectRef[0..\*]*

*by*

1. *RelatedEventGroup: groupId[0..\*]*
where each groupId represents one *EventGroup*, to which all events belong to, whose *RelatedEventGroups* contain the groupId.
Technically it is not even necessary to have a table of groupIds, which define those Ids, which may occur here.

**Benefit of a)**

we have a directed graph, which e.g. allows a chain like

 Event(myGs>5°) 🡪 Event(myGsMaxElevation) 🡪 Event(myGs<5°)

A constraint therefore may easily navigate through this graph, as long as we have only one relatedEvent per event, in particular if the constraint always refers to the next event before or after itself, e.g.:

* time dependency in between Event(myGsMaxElevation) and its preceding Event is XXX
* time dependency in between Event(myGsMaxElevation) and its succeeding Event is YYY

In order to support this properly, we propose, to allow just one RelatedEvent. (However, having only one RelatedEvent on the other hand would lead to having to introduce some artificial hierarchy for having cascading RelatedEvents instead of a flat group of potentially independent events referenced here which is not our intention of course…)

**Benefit of b)**

We have a proper grouping, which allows navigating from one Event to another within the same group just by indicating that we refer to a related event and by specifying e.g. a name, which picks the right event from all related events, e.g.:

* time dependency in between Event(myGS, maxElevation) and its related event with name myGs>5° is XXX
* time dependency in between Event(myGS, maxElevation) and its related event with name myGs<5° is YYY

This decision should be discussed. Currently we tend to prefer b).

***WG Position 04/09/2018***

*See Action Point 5 above.*