3 Implementation Strategy - Transport level Details

Each node contains the following -

- The maximum number of nodes that are allowed in the cluster.
- The IP-discrimination scheme employed by the nodes within the cluster. This scheme is used to determine if a node can be part of the cluster. One of the reasons why we employ such a scheme is that, nodes within a cluster would communicate using UDP. Losses are very low, and communication very fast while using UDP between nodes on the same token ring in the network. In the case of a node not on the same token ring as the rest of the cluster, losses incurred due to UDP communication would be inordinately high. These high losses would serve to divert the rest of the nodes, towards error correction and handling of messages over the erring link, from their basic task of effiecient dissemination of messages received at the cluster gateways.
- There is also a connection set up vector, which specifies the maximum number of concurrently active connections that a node can maintain at any given time. Associated with every connection between nodes (server or client) there is a receiver thread per transport protocol, and sender threads propotional to the number of active connections and the number of transport protocols supported by the node and nodes connected to it. There is thus a price to be paid (CPU-dollars) for maintaining active concurrent connections to multiple nodes. In addition using UDP, Multicast require regular polling since there are no outtage notifications (akin to ICMP provided by the TCP core porotocol set) in response to socket failures. A node hosted on a slower machine could thus tune its processing to the capabilities existing in the underlying system and also on the number of processes active on the system.

In addition to this every node, can communicate using TCP, UDP, Multicast. Every node publishes the TCP port, UDP port and the Multicast group over which it monitors connection requests and message interchange. In the case of UDP and Multicast, though the communication is fast the communication is unreliable. UDP and Multicast communication provide no guarantees on the delivery and ordering of message packets. As a result messages received could contain holes in the intended sequence and the messages could also be out of order. Thus we employ an error detection and correction mechanism atop the UDP and Multicast transports for our protocol needs. The other disadvantage of UDP communication is that there are no outtage notifications.

4 The DTD for the event

Events conform to XML DTDs. Not all fields within the DTDs need to be present, some fields are however mandatory. At every server node hop, the DTD definition for the event needs to be referenced. There are two ways for this information to be included within the XML event

(a) Include the DTD definition within the event itself. This is ruled out as the information contained within the XML event would increase.

(b) Include a pointer to the DTD definition. This would entail a lot of network traffic with every arrived event resulting in a network operation to fetch the document definition.

To work around items (a) and (b) we employ the following approach. The first time that an event type is encountered at a server node, the DTD definition is fetched[1] and cached at the server node. Thus we circumvent the network operation.

An event exists within the context of a stream, thus the specification of an event includes the stream that this event is a part of, this is specified by the StreamId. Every event needs to have an Id, `Mspaces:EventId`, that is unique in space and time. Events also should be able to specify the linkages that exist between them and events within other streams, this constitutes the `Mspaces:EventLinkage`.

[1]This DTD definition could be fetched either from the pointer contained within the event or from the node which routed the event to this node in the first place.

Resolution of the event linkage is a precursor to the creation of merged streams. We also need an indication of the type of event that this event is, i.e. live or recovery and the security constraints contained within the event. This is included in `Mspaces:EventType`. Events could also possibly specify zero or applications that it is a part of. The event summary, which could occur once or not at all, provides a synopsis of the event itself. Thus an Event definition could be the following.

```
<!ELEMENT Mspaces:Event (Mspaces:StreamId, Mspaces:EventId,  Mspaces:EventType,
                         Mspaces:EventLinkage, Mspaces:ApplicationType*,
                         Mspaces:Summary?)>
```

This specification dictates that the various elements should appear in the order `Mspaces:StreamId` first, then `Mspaces:EventId` and so on. The `StreamId` representation is a simple (`#PCDATA`) representation.

```
<!ELEMENT Mspaces:StreamId         (#PCDATA)>
```

The `Mspaces:EventId`, needs to be unique in space and time. Having a unique Client Id, `Mspaces:ClientId` reduces the uniqueness problem to a point in space. `Mspaces:TimeStamp` provides the uniqueness in the time domain, while the sequence number (contained in `Mspaces:SequenceNumber`) scheme ensures issue rates which are higher than that dictated by the constraint imposed on uniquely identifiable events by the granularity of the underlying clock. `Mspaces:IncarnationNumber`'s are essential to avoid conflicts when an issuing client initiates a roam. The duplicate detection loop hole exists since no process has access to a global clock and also since the clocks on individual machines are never synchronized. Even if the clocks were synchronized, the rates at which these individual clocks tick are different. The following definition for the eventId specifies a an ID unique in space and time.

```
<!ELEMENT Mspaces:EventId (Mspaces:ClientId, Mspaces:TimeStamp,
                           Mspaces:SequenceNumber, Mspaces:Incarnation)>
<!ELEMENT Mspaces:ClientId         (#PCDATA)>
<!ELEMENT Mspaces:TimeStamp        (#PCDATA)>
<!ELEMENT Mspaces:SequenceNumber   (#PCDATA)>
<!ELEMENT Mspaces:Incarnation      (#PCDATA)>
```

Earlier we discussed our approach to circumventing network operations while parsing the XML events. DTD's could however change, and the cache rendered useless, to account for this scenario we need to include the concept of version Number within the DTD fields. When the event is parsed a look at the `Mspaces:versionNumber` field could tell us if the cache needs to be updated. If the DTD definition for the event is changed the clients interested in the events conforming to the old DTD definition need to be notified about this change. These clients could then decide if their profiles need to be updated to reflect this change. This notification of the change in the DTD of the event that a client is interested is included in the field `Mspaces:LatestVersionNumber`. Also nodes need to maintain the DTD definitions for different versions of the same DTD. It is concievable that there are events being published within the system or there are recovery events which would conform to the old versions of the DTD. Information regarding these version numbers along with the security constraints and liveness indicator consitute `Mspaces:EventType`.

```
<!ELEMENT Mspaces:EventType (Mspaces:VersionNum, Mspaces:LatestVersionNum?)
<!ATTLIST Mspaces:EventType
         Securitylevel    (low | med | high) ‘‘med’’
         Liveness         (live|recovery)  ‘‘live’’>
<!ELEMENT Mspaces:VersionNum       (#PCDATA)>
<!ELEMENT Mspaces:LatestVersionNum (#PCDATA)>
```

If an `Mspaces:EventType` created does not specify values for the `SecurityLevel` and `Liveness` attributes, the `EventType` is assumed to be a "live" event of "med" security. Recovery events are the events which clients have missed either during a *roam* operation or during a prolonged disconnect.

The `Mspaces:EventLinkage` specifies the dependencies that exist between events in multiple streams. The linkage should provide for resolution of the spatial and timing dependencies in an implicitly or explictly specified order. Besides these we also need the ability to create *bundles* of events within a given stream. The bundles that we create need an identifier, this is provided by `Mspaces:BundleId`. However, there could be situations where the bundle we consider is the stream itself. Bundles need to also indicate the methodology that needs to be in place to decide upon the merging schemes. This is provided by the enumeration of `Mspaces:TimeConstraint` and `Mspaces:MergeScheme`. Some bundles however, may not impose any scheme on the merging of bundles. We account for such a scenario by including "None" in the enumeration for the linkage schemes which we mentioned earlier. Events within a bundle also have monotonically increasing sequence numbers assigned to events within the bundle. This is in addition to the sequence numbers that events possess to determine a uniqueID. The `Mspaces:BundleNumber` however, comes into play only in the presence of a `Mspaces:BundleId` within the event stream. The `Mspaces:BundleOrder` specifies the ordering scheme that should be in place for events which are "concurrent" based on the merging methodology that is specified by `Mspaces:BundleLinkage`.

```
<!ELEMENT Mspaces:EventLinkage ((Mspaces:BundleId, Mspaces:BundleNumber)? ,
                                 Mspaces:BundleLinkage, Mspaces:BundleOrder)
<!ELEMENT Mspaces:BundleId         (#PCDATA)>
<!ELEMENT Mspaces:BundleNumber     (#PCDATA)>
<!ELEMENT Mspaces:BundleLinkage     NONE | (Mspaces:TimeConstraint?, Mspaces:MergeScheme?)>
<!ELEMENT Mspaces:TimeConstraint   (#PCDATA)>
<!ELEMENT Mspaces:MergeScheme      (#PCDATA)>
<!ELEMENT Mspaces:BundleOrder      (Mspaces:StreamId+ | Mspaces:BundleId+)>
```

A brief note about the `Mspaces:EventLinkage` is in order. If an event is allowed to be part of multiple bundles within the same stream with multiple BundleNumber's the `?` should be `*` in the `Mspaces:BundleId, Mspaces:BundleNumber` grouping. The listing of the DTD in section 4.1 and element analysis in table 1 assumes the `?` occurance operator.

4.1 The complete DTD

The event routing information as specified by the event routing protocol (ERP) and the information contained within the event during recoveries are not included within the definition for the DTDs. The event itself is encapsulated within an XML document, however the routing is not. Below we include the complete definition of the event, which follows from our discussions so far.

```
<!ELEMENT Mspaces:Event (Mspaces:StreamId, Mspaces:EventId,  Mspaces:EventType,
                         Mspaces:EventLinkage, Mspaces:ApplicationType*,
                         Mspaces:Summary?)>

<!ELEMENT Mspaces:StreamId         (#PCDATA)>

<!ELEMENT Mspaces:EventId (Mspaces:ClientId, Mspaces:TimeStamp,
                           Mspaces:SequenceNumber, Mspaces:Incarnation)>
<!ELEMENT Mspaces:ClientId         (#PCDATA)>
<!ELEMENT Mspaces:TimeStamp        (#PCDATA)>
<!ELEMENT Mspaces:SequenceNumber   (#PCDATA)>
<!ELEMENT Mspaces:Incarnation      (#PCDATA)>

<!ELEMENT Mspaces:EventType (Mspaces:VersionNum, Mspaces:LatestVersionNum?)
<!ATTLIST Mspaces:EventType
         Securitylevel    (low | med | high) ‘‘low’’
         Liveness         (live|recovery)  ‘‘live’’>
<!ELEMENT Mspaces:VersionNum       (#PCDATA)>
<!ELEMENT Mspaces:LatestVersionNum (#PCDATA)>
```

```
<!ELEMENT Mspaces:EventLinkage ((Mspaces:BundleId, Mspaces:BundleNumber)? ,
                                 Mspaces:BundleLinkage, Mspaces:BundleOrder)
<!ELEMENT Mspaces:BundleId         (#PCDATA)>
<!ELEMENT Mspaces:BundleNumber     (#PCDATA)>
<!ELEMENT Mspaces:BundleLinkage    NONE | (Mspaces:TimeConstraint?, Mspaces:MergeScheme?)>
<!ELEMENT Mspaces:TimeConstraint   (#PCDATA)>
<!ELEMENT Mspaces:MergeScheme      (#PCDATA)>
<!ELEMENT Mspaces:BundleOrder      (Mspaces:StreamId+ | Mspaces:BundleId+)>

<!ELEMENT Mspaces:ApplicationType  (#PCDATA)>
<!ELEMENT Mspaces:Summary          (#PCDATA)>
```

Table 1 depicts the various elements, the nested elements and occurance bounds for the nested elements within a specific element. The table also snaphots our discussions so far with brief descriptions of the purpose of each element with the event element hierarchy.

Element	**Allowed Nested Elements**	**Number**	**Purpose of the Element**
`Mspaces:Event`	Mspaces:StreamId Mspaces:EventId Mspaces:EventType Mspaces:EventLinkage Mspaces:ApplicationType Mspaces:Summary	1 1 1 1 0 or more 0 or 1	Overall root element of the Event
`Mspaces:StreamId`	None		Stream the event belongs to
`Mspaces:EventId`	Mspaces:ClientID Mspaces:TimeStamp Mspaces:SequenceNumber Mspaces:Incarnation	1 1 1 1	The unique event ID.
`Mspaces:ClientID`	None		The issuing Client ID.
`Mspaces:TimeStamp`	None		Time Stamp usually in milliseconds
`Mspaces:SequenceNumber`	None		Issue events at rate greater than the granularity of the timeStamp.
`Mspaces:Incarnation`	None		Allows for duplicate detection during a issuing client *roam* .
`Mspaces:EventType` (*attributes* : live, secure)	Mspaces:VersionNum Mspaces:LatestVersionNum	1 0/1	Information about the versioning and liveness of an event.
`Mspaces:VersionNum`	None		The version number of the DTD that XML event conforms to
`Mspaces:LatestVersionNum`	None		Inform clients about the version change to a DTD.
`Mspaces:EventLinkage`	Mspaces:BundleId Mspaces:BundleNumber Mspaces:BundleLinkage Mspaces:BundleOrder	0/1 0/1 1 1	Specification for the linkage of events in multiple streams.
`Mspaces:BundleId`	None		Indentifies a specific bundle within the stream.
`Mspaces:BundleNumber`	None		Specifies the numbering with in the bundle of a stream. Depends on the presence of the Bundle.
`Mspaces:BundleLinkage` *(Enumeration)*	Mspaces:TimeConstraint Mspaces:MergeScheme	0/1 0/1	Specifies the method for merging streams/bundle.
`Mspaces:TimeConstraint`	None		Specifies merging based on time.
`Mspaces:MergeScheme`	None		Specifies a merge scheme.
`Mspaces:BundleOrder` *(Enumeration)*	Mspaces:StreamId Mspaces:Bundle	1 or more 1 or more	Specifies ordering for concurrent events

Table 1: Mspaces:Event Hierarchy