XEP-0496: Pubsub Node Relationships

Jérôme Poisson
mailto:goffi@goffi.org
xmpp:goffi@jabber.fr

2024-11-20
Version 0.1.0

Status	Type	Short Name
Experimental	Standards Track	pubsub-relationships

This specification describes how to establish links between pubsub nodes, allowing for optional hierarchical organization.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

There are many cases where pubsub nodes have a relationship: because a feature needs several nodes (e.g., Form Discovery and Publishing (XEP-0346) [1]) or because a node describes information for another one (or its items), like in Pubsub Attachments (XEP-0470) [2].
Sometimes this relationship is hierarchical (e.g., Microblogging Over XMPP (XEP-0277) [3], where comment nodes depend on blog items), or a tree-like hierarchy may be desirable (e.g., a file-sharing feature representing directories and files).
So far, to have this kind of relationship, two ways were used:

- Using a flat organization, with prefixed nodes. For instance, Microblogging Over XMPP (XEP-0277) [4] uses a well-defined prefix for comments ("urn:xmpp:microblog:0:comments/"). This has several major inconveniences:
  - It can quickly become messy, as all nodes appear at the same level. If an end-user inspects the nodes of its pubsub service, a massive number of them may be present.
  - The XMPP client is responsible for updating the nodes' permissions in case of changes, such as if a blog node changes access model from "open" to "whitelist". All comment nodes must be modified accordingly. This is prone to error and can lead to accidentally giving incorrect access to a node.
  - If a node is deleted, other related nodes may not be deleted, resulting in orphan nodes that can be forgotten. For instance: if a parent blog node is deleted, comment nodes or attachment nodes may be forgotten and not deleted, even though they no longer serve any purpose.
- Using PubSub Collection Nodes (XEP-0248) [5]. This specification is an appreciable effort to solve the hierarchical use case but has some major drawbacks:
  - It is not well supported. Only a few Pubsub services or XMPP clients implement it.
  - Its complexity: The specification introduces the notion of collection and leaf nodes, which may be confusing and difficult to handle and requires dedicated logic notably in XMPP clients.
  - It only supports hierarchical relationships and is not adapted to simple relationship use cases as for Form Discovery and Publishing (XEP-0346) [6] or Pubsub Attachments (XEP-0470) [7].
  - The handling of collection node deletion is unspecified, as per 8.2 Handling Collection Node Deletion. While this flexibility may have some use, the unpredictable behavior it can lead to outweighs its benefits.

[1] XEP-0346: Form Discovery and Publishing <https://xmpp.org/extensions/xep-0346.html>.
[2] XEP-0470: Pubsub Attachments <https://xmpp.org/extensions/xep-0470.html>.
[3] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.
[4] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.
[5] XEP-0248: PubSub Collection Nodes <https://xmpp.org/extensions/xep-0248.html>.
[6] XEP-0346: Form Discovery and Publishing <https://xmpp.org/extensions/xep-0346.html>.
[7] XEP-0470: Pubsub Attachments <https://xmpp.org/extensions/xep-0470.html>.

  - Most importantly, a collection node access model overwrites those of its leaf nodes, as explained at XEP-0248 §9.1 Security Considerations/Access Model, which is a showstopper.

This specification proposes another solution, aiming to replace PubSub Collection Nodes (XEP-0248) [8] and is:

- Really easy to implement for XMPP clients.
- Relatively easy to implement for Pubsub services.
- Backward compatible (clients not supporting this specification can still interact as usual with pubsub nodes).
- Simple to understand.
- Hopefully, fixing all the issues mentioned in this introduction.

To make things simple, this specification only treats relationships between nodes. Other features (notably subscription to a hierarchy) will be managed in separate XEPs.

2 Requirements

The design goals of this XEP are:

- to allow simple relationships, indicating related nodes and automatically deleting them when the main node is deleted;
- to enable parent/child relationships to form a tree-like hierarchy;
- in a tree-like hierarchy, permissions must be preserved for child and all parent nodes (i.e., an entity must have permission to access the requested node and all its parents in order to access a given node);
- to be straightforward to implement for XMPP clients;
- to be as easy as possible to implement for Pubsub services;
- to be backward compatible: clients not implementing this specification should still be able to access nodes.

[8] XEP-0248: PubSub Collection Nodes <https://xmpp.org/extensions/xep-0248.html>.

3 Glossary

- **linked node**: Node declared in the ”link” field of the current node.
- **parent node**: Node declared in the ”parent” field of the current node.
- **linking node**: A node having a ”link” relationship to the current node.
- **child node**: A node having a ”parent” relationship to the current node.
- **root node**: A node without any ”parent”.

4 Relationships

4.1 Definitions

This specification defines two types of relationships:

- **link**: a ”link” is a simple relationship between nodes. It indicates that multiple nodes are interconnected, either because a feature requires several nodes to function together (e.g., Form Discovery and Publishing (XEP-0346) [9]), or because one node extends another node or its items (e.g., Pubsub Attachments (XEP-0470) [10]).
- **parent**: a ”parent” is a hierarchical relationship where the referenced node is positioned higher in the hierarchy. It signifies that one or more nodes depend on the parent, such as when comment nodes rely on a microblog node (e.g., Microblogging Over XMPP (XEP-0277) [11]), or when a tree-like structure is required for organizing items (e.g., a file-sharing system that mimics directory/file structure).

4.2 Settings a Relationship

To set a relationship, an XMPP client must first ensure that the pubsub service supports this specification (see Discovering Support below). Then the relationship is established by setting the relevant configuration field parameter to a node as explained in XEP-0060 §8.2 Configure a Node. The var ”{urn:xmpp:pubsub-relationships:0}link” must be used for a ”link” relationship, and the var ”{urn:xmpp:pubsub-relationships:0}parent” must be used for a ”parent” relationship.
When setting a relationship in the pubsub service, the service MUST ensure that the resulting graph does not contain any cycle. This means that it must be impossible to return to an initial node by following the relationships, whether they are labeled as ”link” or ”parent”. By sequentially following these relationships, one must always end at a node without any outgoing ”link” or ”parent” relationships.

[9] XEP-0346: Form Discovery and Publishing <https://xmpp.org/extensions/xep-0346.html>.
[10] XEP-0470: Pubsub Attachments <https://xmpp.org/extensions/xep-0470.html>.
[11] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.

If setting a ”link” or ”parent” relationship would result in a cyclic graph, the service MUST reject the configuration with a <not-allowed/> error, specifying a pubsub-specific error condition of <invalid-option/>, and SHOULD include a human-readable text explaining the problem.
If when setting a ”link” relationship, the linked node has a ”parent” relationship, the pubsub service MUST set the same parent to the linking node. If the linking node has already a parent which is different from the ”parent” of the linked node, the service MUST reject the configuration with a <not-allowed/> error, specifying a pubsub-specific error condition of <invalid-option/>, and SHOULD include a human-readable text explaining the problem.
If a ”parent” relationship is set to a linked node, the ”parent” of all linking node MUST be set to the same node by the service. A service MUST NOT accept a ”parent” relationship set to a linking node: linking nodes' parent relationship are always automatically set by the service itself when the linked node's ”parent” relationship is set. In other terms, ”parent” field can't be set on a node if it has a ”link” field. This is to be sure that linking nodes are always on the same level as the linked node. If a ”parent” is set on a node with ”link” field, the service MUST reject the configuration with a <not-allowed/> error, specifying a pubsub-specific error condition of <invalid-option/>, and SHOULD include a human-readable text explaining the problem.

4.3 Link Relationship Rules

The following rules apply to the ”link” relationship:

- When a linked node is deleted, all linked nodes MUST be automatically deleted by the pubsub service.
- Deleting a linking node does not affect the linked node or any other linking nodes.
- The access model and publish model of nodes with a ”link” relationship are independent.
- When a parent of a linked node is modified, the parents of linking nodes MUST be automatically updated by the pubsub service, as explained in Setting a Relationship.

4.4 Parent Relationship Rules

The following rules apply to the ”parent” relationship:

- When a parent node is deleted, all child nodes MUST be automatically deleted by the pubsub service. This will recursively delete children of children and so on, resulting in the deletion of the entire branch of the parent node.
- To access a node, an entity MUST have access to all parents. In other words, a pubsub service MUST NOT allow access to a node for an entity if that entity is not allowed by the access model of the node or any of its parents up to the root node.

- To publish to a node, an entity MUST have publication rights for all parents. In other words, a pubsub service MUST NOT permit publishing to a node from an entity if that entity is not permitted by the publish model of the node or any of its parents up to the root node.

5 Examples

5.1 Link Relationship Example

Juliet XMPP client links an attachment node to a microblog node.

Listing 1: Client requests link relationship configuration

```
<iq type='set'
   from='juliet@capulet.lit/balcony'
   to='pubsub.example.org'
   id='link1'>
 <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
   <configure node='urn:xmpp:pubsub-attachments:1/xmpp:juliet@capulet.
      lit?;node=urn%3Axmpp%3Amicroblog%3A0;item=balcony-restoration-
      afd1'>
     <x xmlns='jabber:x:data' type='submit'>
       <field var='{urn:xmpp:pubsub-relationships:0}link'>
         <value>urn:xmpp:microblog:0</value>
       </field>
     </x>
   </configure>
 </pubsub>
/iq>
```

Listing 2: Service responds with confirmation

```
<iq type='result'
    from='pubsub.example.org'
    to='juliet@capulet.lit/balcony'
    id='link1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='urn:xmpp:pubsub-attachments:1/xmpp:juliet@capulet
       .lit?;node=urn%3Axmpp%3Amicroblog%3A0;item=balcony-restoration
       -afd1'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='{urn:xmpp:pubsub-relationships:0}link' type='text-
           single'">
␣␣␣␣␣␣␣␣␣␣<value>urn:xmpp:microblog:0</value>
␣␣␣␣␣␣␣␣</field>
```

```
        [...]
      </x>
    </configure>
  </pubsub>
</iq>
```

5.2 Parent Relationship Example

When a comment node for Microblogging Over XMPP (XEP-0277) [12] is created, Juliet's XMPP client configures the parent relationship.

Listing 3: Client requests parent relationship configuration

```
<iq type='set'
    from='juliet@capulet.lit/balcony'
    to='pubsub.example.org'
    id='parent1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='urn:xmpp:microblog:0:comments/some-item-id'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='{urn:xmpp:pubsub-relationships:0}parent'>
          <value>urn:xmpp:microblog:0</value>
        </field>
      </x>
    </configure>
  </pubsub>
</iq>
```

Listing 4: Service responds with confirmation

```
<iq type='result'
  from='pubsub.example.org'
  to='juliet@capulet.lit/balcony'
  id='parent1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='urn:xmpp:microblog:0:comments/some-item-id'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='{urn:xmpp:pubsub-relationships:0}parent' type='
            text-single'">
          <value>urn:xmpp:microblog:0</value>
        </field>
        [...]
      </x>
```

[12] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.

```
    </configure>
  </pubsub>
</iq>
```

6 Business Rules

By default, node discovery as explained at XEP-0060: Discover Nodes is unaffected by this specification. However, it is expected that a future specification will add an optional way to filter out or display linking nodes and/or child nodes, resulting in a much clearer listing of PubSub nodes. That also means that child nodes are not displayed when doing a disco request on a parent node; this is to avoid breaking existing implementations that would not expect to discover nodes on a non-collection node.
This specification is backward compatible: nodes remain accessible normally to unsupporting clients. The main difference for them will be the automatic deletion of linked and child nodes and the propagation rules for access models and publish models. Setting a relationship remains possible even for unsupporting clients, as it involves only a regular node configuration update.
A tree-like structure with "parent" relationships does not prevent node name conflicts: to be backward compatible, nodes are still available normally as they would in a flat structure. This means that names must always be unique within the pubsub service to avoid conflicts, even deep inside the "parent" hierarchy.
If a node is created without any relationship, a Pubsub service MAY automatically create relationships for well-known nodes. For instance, a "parent" relationship can be created to the corresponding microblog node if a Microblogging Over XMPP (XEP-0277) [13] comment node is created, or a "link" relationship can be created if a Pubsub Attachments (XEP-0470) [14] attachment node is detected. However, if a relationship (either "link" or "parent") is set when creating the node, the Pubsub service MUST NOT change it or add other relationships. This is useful for working with non-supporting clients while still maintaining a clean organization of nodes.
For "link" relationships, the first node to be created is the one which is linked. For example: in Form Discovery and Publishing (XEP-0346) [15], the "template" node is the one which is linked, meaning that it's the "submitted" node which must link to it.
If one wants to delete a parent node without deleting all its descendants, the direct child must first be unparented; that is, their "parent" attribute must be set to another node name or removed entirely. Then, the parent node can be deleted.

[13] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.
[14] XEP-0470: Pubsub Attachments <https://xmpp.org/extensions/xep-0470.html>.
[15] XEP-0346: Form Discovery and Publishing <https://xmpp.org/extensions/xep-0346.html>.

7 Discovering Support

If a pubsub service supports the protocol specified in this XEP, it MUST advertise it by including the "urn:xmpp:pubsub-relationships:0" discovery feature in response to a Service Discovery (XEP-0030) [16] information request.

Listing 5: Service Discovery Information Request

```
<iq type='get'
    from='juliet@example.org/balcony'
    to='pubsub.example.org'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 6: Service Discovery Information Response

```
<iq type='result'
    from='pubsub.example.org'
    to='juliet@example.org/balcony'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:pubsub-relationships:0'/>
    ...
  </query>
</iq>
```

8 Security Considerations

The "parent" relationship enhances security by preventing accidental bad synchronization of permission changes. For example, if an Microblogging Over XMPP (XEP-0277) [17] blog node's access model is changed from "open" to "whitelist", without this specification, all comment nodes must be manually updated one by one by the XMPP client, which is error prone. However, with this specification and a "parent" relationship in place, the permissions are automatically propagated according to the Parent Relationship Rules.
The automatic deletion of linked or child nodes might surprise end-users, especially when using non-supporting XMPP clients. If the hierarchy is not visible to the user, it may not be clear that other nodes will also be deleted automatically. Supporting clients should ensure that the automatic deletion is clear to end-user.

[16] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[17] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.

9 IANA Considerations

This document does not require interaction with the Internet Assigned Numbers Authority (IANA) [18].

10 XMPP Registrar Considerations

TODO

11 Acknowledgements

Thanks to NLNet foundation/NGI Zero Core for funding the work on this specification.

[18] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.