

Message Set Model for Message Boards

David Manura, Mark Tiefenbruck

October 4, 2004- **Revision Draft v.1a.2**

Abstract

A new “set model” for logically organizing discussions on message boards is presented as an alternative to the traditional forum (flat) and threaded (newsgroup) models. In the set model, sets are the main construct for organizing user messages. Discussions may be categorized naturally in multiple ways, browsing or searching can be performed using set operations, and the data may be mapped onto multiple selectable views that are combinations of flat lists and tree structures. The set model is argued to be more flexible than the other models yet still more intuitive than the threaded model. A prototype is under development, and this is planned for eventual implementation on the Math Message Board at Math2.org (<http://math2.org/mmb>).

History

2004-10-02 - rev.1a - first draft made public

2001-10-04 - rev.1a.2 - largely minor improvements in grammar/wording clarity

1 Introduction

This paper presents a new “set model” for organizing discussions on a message board system, such as a web-based forum or Internet newsgroup. Before looking at the set model, let’s first examine why the traditional models are deficient.

1.1 An Example: Forum Model

Consider the following discussion on a math-related message board. The message board uses a typical “forum” model, in which there are one or more subject areas (e.g. “Elementary Math”), each containing one or more discussion threads (e.g. “Number Problems”), each ideally representing a single topic of discussion and containing a series of messages posted by users on that topic.

```

[Elementary Math]
|[Number Problems]
|- me: Please solve these problems:
(1) 100! = ? (2) 100!/98! = ? (3) Find the pattern 1, 1, 2, 3, 5, 8, ...
|- alice: For #2, we could write (100*99*98*97*...)/(98*97*...).
Notice that most terms cancel, giving 100*99=9900.
|- bob: #3 is the Fibonacci sequence.
|- me: What's Fibonacci?
|- oscar: For #1, we could write a program:
function fib(x) := if x = 0 then 1 else x*fib(x-1)
|- bob: Each term is the sum of the two previous terms.
|- alice: what programming language is that?
|- oscar: It's the XYZ language.
|- bob: The ZYX language is superior in YZW constructs.
|- oscar: Is not.
|- alice: Looks like recursion in some odd language.
|- alice: #1 can be solved with a calculator.

```

There are a number of problems with this model:

- It's not immediately clear who is talking to who.
- The main discussion is getting drowned out by a tangent discussion on programming languages.
- It's not immediately clear what the structure and main topics are or whether all parts of the question have been answered.

1.2 Alternative: Threaded Model

An alternative to the forum model is the threaded model. This model, popularized by Internet newsgroups, consists of a series of discussion threads, each consisting of user messages organized into a tree structure. Message A is a parent of message B if B replied to A. The rationale for this model is that a discussion may have multiple parts or go off into tangents, and this tree model partly captures that the temporal and topical locality.

```

|- me: Please solve these problems:
(1) 100! = ? (2) 100!/98! = ? (3) Find the pattern 1, 1, 2, 3, 5, 8, ...
|- alice: For #2, we could write (100*99*98*97*...)/(98*97*...).
Notice that most terms cancel, giving 100*99=9900.
|- bob: #3 is the Fibonacci sequence.
  |- me: What's Fibonacci?
    |- bob: Each term is the sum of the two previous terms.
|- oscar: For #1, we could write a program:
function fib(x) := if x = 0 then 1 else x*fib(x-1)
  |- alice: what programming language is that?
    |- oscar: It's the XYZ language.
      |- bob: The ZYX language is superior in YZW constructs.
        |- oscar: Is not.
  |- me: Looks like recursion in some odd language.
|- alice: #1 can be solved with a calculator.

```

Still, there are problems:

- deep nesting provides higher complexity. Often, sub-discussions are linear and don't require nesting (e.g. under "alice: what programming language is that?").
- Oscar's and Alice's answers to #1 are not grouped together in any way. This is because the links between tree nodes A and B exist not by topic but only when B replies to A.
- the tangent discussion is just as prominent as the other sub-discussions.
- the tangent discussion is not properly categorized. It really belongs in a Programming subject area, not an Elementary math subject area.
- The model combines (and confuses) temporal locality (the reply-to relationships) and topical locality (the categorization of discussions by topic).

Since the threaded model provides a new set of problems, some argue that the forum model is more advantageous than the threaded model ([1], [2]).

For example, the forum model may enforce better quality and less fragmented discussions.

1.3 A Third Way: The Set Model

We propose in this paper a "set model," in which user messages are organized into multiple, possibly overlapping sets termed message set, and in which the sets can be viewed in a variety of ways, including mapped onto one or more categorical tree structures. Below shows a tree view of the above example using this model.

```

[Elementary math]
  [Number problems]
  - me: Please solve these problems:
    [Problem #1]
    - me: (1)  $100! = ?$ 
      [XYZ language]
      - oscar: For #1, we could write a program:
        function fib(x) := if x = 0 then 1 else x*fib(x-1)
      - alice: what programming language is that?
      - oscar: It's the XYZ language.
      - bob: The ZYX language is superior in YZW constructs.
      - oscar: Is not.
      - alice: Looks like recursion in some odd language.
    - alice: #1 can be solved with a calculator.

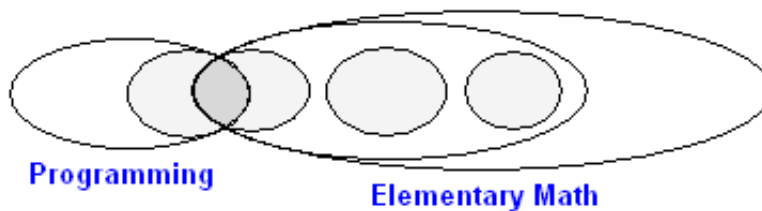
    [Problem #2]
    - me: (2)  $100!/98! = ?$ 
    - alice: For #2, we could write  $(100*99*98*97*...)/(98*97*...)$ .
    - Notice that most terms cancel, giving  $100*99=9900$ .

    [Problem #3]
    - me: (3) Find the pattern 1, 1, 2, 3, 5, 8, ...
    - bob: #3 is the Fibonacci sequence.
    - me: What's Fibonacci?
    - bob: Each term is the sum of the two previous terms.

[Programming]
  [Language Comparisons]
  [XYZ language]
  - oscar: For #1, we could write a program:
    function fib(x) := if x = 0 then 1 else x*fib(x-1)
  - alice: what programming language is that?
  - ...etc....

```

This tree model is just one view of an underlying set model, which may be described using a Venn diagram:



This is also no prohibition against viewing the model in a flat strictly chronological view, as in the forum model.

The above method is better organized and structured, but this alone is only a modest improvement. The set model is in some ways a hybrid of the forum and threaded models, but it also extends both. The model allows discussions to

be categorized in multiple ways (e.g. the programming sub-discussion). Additional advances are achieved by permitting browsing using set operations (e.g. view [Elementary Math] - [Programming]), searching using set operations, and multiple views of the data (e.g. flat forum like or show nodes up to depth 2).

1.4 Goals for the new model / problems addressed

The goals for the set model are

- Better support tangential discussions.
- Allow users to form their own subject areas as needed.
- Categorize discussions in multiple ways.
- More effectively structure parallel, interwoven, and related discussions.
- Support some of the simplicity of the forum model with even more flexibility than the threaded model.

1.5 Observations

A few general observations are stated.

- The set model eliminates the distinction between the forum model's subject areas and contained discussion threads. Both structures are generalized as "message sets," and any remaining distinction between the two is purely conventional. This implies also that a discussion thread (being a message set) can contain other message sets and so forth ad infinitum. A message sets contained within a discussion thread is termed a "sub-thread'," but, again, this terminology is also purely conventional.
- The set model can be mapped to a tree-like model (with the extension that each node can contain multiple possible parents). Specifically, it is a directed acyclic graph.
- The tree representation in the set model represents categorical (topical) relationships. That is, A parent B implies B is a subtopic of A. In the threaded model, the relationship is temporal: B replied to A.

1.6 Related Work

The authors are not aware of identical work, but there are some related works. These works observe that the standard tree model of organizing content (e.g. as in the UNIX file system) is limited because there is no single optimal organization. There are extensions, such as symbolic links, but these these do not easily overcome the tree limitations.

- NEO [3] - allows e-mail messages to be placed in multiple categories (sets) rather than in folders.

- Relink [4] - uses symbolic links to resolve deficiencies in the tree model of the UNIX file system.

The widely used Boolean model for document searching as a set model (by Boolean-set equivalence), but it's used largely for Boolean keyword searches rather than browsing by category (like Yahoo!). Secondly, these sets are defined on the fly, at query time, whereas the sets in the message set model are defined explicitly and before query time.

2 Structural Definitions

The message set model primarily consists of sets called message sets and contained elements called message set elements, which are typically individual messages posted by users. The components of the model are detailed below.

Definition 1 (Message set element) *A message set element is an object. It contains a unique numeric identifier, termed the message ID (MID). A message set element is “abstract,” never used directly but always inherited from.*

Definition 2 (Message) *A message is a specialization of a message set element. It also has a body, posting date, and creator. It typically represents comments by a user at a certain point in time.*

Definition 3 (Invisible element) *An invisible element is a specialization of a message set element. Its purpose will be detailed later, but, generally, it is only a mathematical artifact used to force a desired partial order over the message sets.*

Definition 4 (Message reference) *A message reference is a specialization of a message set element. It also contains the MID of another message set element in order to refer (point) to that message set element.*

Definition 5 (Message set reference) *A message set reference is a specialization of a message set element. It also contains the MSID of another message set in order to refer (point) to that message set.*

Definition 6 (Message set) *A message set A is set containing ~~REMOVED-2004-10-04: an unique invisible element i_A and~~ zero or more message set elements. It has a unique numeric identifier, termed the message set ID (MSID). It has a title, creator, and permissions (FIX:clarify permissions). A message set typically represents a topic of discussion; it is categorical in nature.*

FIX: Are the “reply-to” relationships between messages stored anywhere? I think they should be, for how else can we be certain who is replying to who? Note: the reply-to relationships is a tree-like structure. Without the reply-to relationships, the only temporal information is posting time, which forms a linear structure.

2.1 Notation

The following conventions will be used.

Let M = set of all message set elements.

Let MS = set of all message sets.

Let A, B, C conventionally represent elements in MS , while m represents an element in M .

Note that $MS \subseteq 2^M$, and it is possible—in fact likely—that $\exists X \subseteq M$ such that $\neg(X \in MS)$ and that $\exists X \subset A$ such that $\neg(X \in MS)$. In summary, only certain subsets of M are identified as message sets.

3 Relations between message sets and messages

The relationships between message sets and between message set and messages can be classified as given below.

Definition 7 (Equivalent set) $C = A$ (i.e. sets C and A are equivalent) iff C and A contain all the same elements.

Note: it will become apparent later that two sets sharing all the same elements also have the same MSID due to the placement of the invisible elements.

Definition 8 (Shallow subset/superset) C is a shallow subset of A iff $C \subset A$ and $\neg \exists B$ such that $C \subset B \subset A$. Alternately, we may say that A is a shallow superset of C .

Definition 9 (Shallow member/container) m is a shallow member of A iff $m \in A$ and $\neg \exists B$ such that $m \in B \subset A$. That is, m is not in any shallow subset of A . Alternately, we may say that A is a shallow container of m .

Definition 10 (Deep subset/superset) C is a deep subset of A if $\exists B$ such that $C \subset B \subset A$. That is, C is a proper subset of a shallow subset of A . Alternately, we may say that A is a deep superset of C .

Definition 11 (Deep member/container) m is a deep member of A if $\exists B$ such that $m \in B \subset A$. That is, m is a member of a shallow subset of A . Alternately, we may say that A is a deep container of m .

Definition 12 (Intersection set) B is an intersection set of A iff $0 < |A \cap B| < |A|, |B|$.

Theorem 1 Two sets A and C are related in exactly one of the following ways: deep superset, shallow superset, equivalent set, shallow subset, deep subset, intersection set, or disjoint.

Definition 13 (Subset depth) The depth of a set C with respect to a set A , denoted $\text{depth}(C, A)$, is the minimal value of n such that $C \subset B_0 \subset B_1 \subset \dots \subset B_{n-2} \subset A$. The depth of C with respect to C is zero.

Definition 14 (Member depth) The depth of an element m with respect to a containing set A , $\text{depth}(m, A)$, is the minimal value of n such that $m \in B_0 \subset B_1 \subset \dots \subset B_{n-2} \subset A$. The depth of m with respect to a set it is a shallow message of is 1.

3.1 One shallow container rule

The following constraint is currently imposed on the model:

Constraint 1 *Each message has exactly one shallow container.*

This can alternately be stated as

- The intersection of two message sets contains no shallow members.
- All sets of shallow members are disjoint.

(FIX:proof).

This constraint is not strictly necessary, but it is imposed for simplicity.

For example, you cannot have

```

-----
|  |m1|  |          A = {m1, m2, m3}
|  ---- |          B = {m1, m4, m5}
|m2|  |m4|
|m3|  |m5|
--   --

```

But rather must do

```

-----
| m1      |          A = {m1, m2, m3}
|  ----- |          B = {m2, m3}
| |m2 m3| |          C = {m4, m5}
|  ----- |
|  ----- |
| |m4 m5| |
|  ----- |
-----

```

Neither can you have

```

-----
|m1|    A = {m1,m2,m3}
|m2|    B = {m3,m4,m5}
|--|
|m3|
|--|
|m4|
|m5|
-----

```

And you cannot have crazy things like

```

A = {m1, m2}
B = {m2, m3}
C = {m1, m3}

```

We will see later that this constraint provides a way to make replying intuitive, so a reply m_4 to message m_1 will be placed in all sets that m_1 is a member of:

A = { m_1 , m_2 , m_4 }
B = { m_2 , m_3 }
C = { m_1 , m_3 , m_4 }

and a subsequent reply m_5 to m_3 results in

A = { m_1 , m_2 , m_4 }
B = { m_2 , m_3 , m_5 }
C = { m_1 , m_3 , m_4 , m_5 }

Such behavior is argued to be non-intuitive, or at least no scheme has been devised to make it intuitive.

Note that the analogous constraint “Each message set has exactly one shallow superset” is in general **not true**. For example, Algebra and Geometry may each be a shallow superset of Algebraic Geometry.

3.2 Reduced set membership form

To specify the supersets of A, we could simply list them: $supersets(A)$. However, a more compact and equally expressive form is to list only the shallow supersets of A: $shallow_supersets(A)$. $shallow_supersets(A)$ can be deduced from $supersets(A)$ as described in Appendices A-B. Conversely, $supersets(A)$ can be deduced from $shallow_supersets(A)$ by transitivity.

The utility of the compact form is realized in the GUI. For example, when re-categorizing a message set, it is far easier to list the changes to be made to the shallow supersets than to the supersets, and it is less error prone as well (e.g. removal from a certain set but not the proper subsets of that set). Secondly, as explained in the following Logical tree view section, the shallow supersets are generally where a set is situated hierarchically, so the identification of shallow supersets is intuitive for the user.

Similarly, to describe the container sets of m , we could simply list them: $containers(m)$. However, a more compact and equally expressive form is to list the shallow containers of m : $shallow_containers(m)$. A similar rationale applies as above.

4 Logical tree view

The subset relation (and its contained shallow subset relation) forms a partial order over MS . This provides one way to organize message sets hierarchically.

For browsing message sets within a hierarchical display, each message set A can be viewed as a node within a tree with multiple parents and multiple children per node. Specifically, it is a directed acyclic graph (DAG).

The mapping from the sets to the graph is

- The parent nodes of A are the shallow supersets A.

- The child nodes of A are the shallow subsets and shallow members of A .

Theorem 2 *The presence of an intersecting set to A has no effect on the local graph structure at A .*

Theorem 3 *For each $m \in A$, $\text{depth}(m, A)$ is finite. For each $C \subset A$, $\text{depth}(C, A)$ is finite.*

That is, all members and proper subsets are reachable via the graph model.

5 Invisible elements

The purpose of the invisible elements is, in special cases, to enforce a desired logical hierarchy (partial order) between message sets. For example, with no invisible elements, a message set could be a null set, in which case it would be a subset every set and therefore be a child of every set in the tree view, and each message posted to an empty set would be inserted into every set. With invisible elements, one could take three otherwise empty sets:

$$A = \{i_A\}, B = \{i_B\}, C = \{i_C\}$$

and force an order on these sets by inserting the invisible elements into the other message sets:

$$A = \{i_A, i_B, i_C\}, B = \{i_B, i_C\}, C = \{i_C\}$$

It is now clear that $C \subset B \subset A$.

The invisible elements also resolve the case where two message sets contain the same elements (e.g. a subject area containing a single discussion thread). Without invisible elements, these two message sets would be identical in their contained elements and therefore hierarchically identical as well.

The usage of invisible elements also leads to the following theorem:

Theorem 4 *Two equivalent sets (i.e. sharing all same elements) also have the same MSID. That is, $A = C \Leftrightarrow \text{msid}(A) = \text{msid}(C)$.*

An alternative to using invisible elements would be to treat the null set as a special case (e.g. be a subset of no set or be a subset of the top-most set hierarchically). This solution is less clean and general than the use of invisible elements. It may also be undesirable to require a message set (e.g. a subject area) to contain at least two disjoint proper subsets (e.g. two discussion threads).

The invisible elements are a mathematical artefact, and we will see that they can be largely ignored in certain implementations.

6 Actions

This section describes the changes that may be made to the data in the message set model.

Constraint 2 *A message m posted in reply to a message p is by default inserted into every message set that p is a member of.*

Constraint 3 *A message m posted as a reply into a message set A is by default also inserted into all nonproper supersets of A .*

The above two constraints imply the following:

Theorem 5 *The partial order on MS is unchanged by merely posting a reply.*

The situation is different when creating a new message set. In this case, a new element is being added to MS , so the partial order on MS does change.

A user can create a new message set and properly place it by specifying a desired reduced set membership form.

If a message set B is deleted from a set A , the shallow subsets and shallow members of B respectively become shallow subsets and members of A .

On deleting a message set B , the user may optionally specify to delete all contained members and subsets of B as well (not including those within an intersection set of B).

7 Security Model

A security model needs to define things such as who can read messages sets, add messages to message sets, and delete messages or message sets in other message sets (e.g. moderators).

There's two natural options of where to assign permissions: on messages or on message sets. Although users may have special permissions on their own messages, it seems generally simpler and sufficient that permissions be assigned to message sets and propagate to the contained subsets and member messages.

Constraint 4 *Each message set contains a list of permissions (i.e. who can do what). Permissions include who can read message, post messages, or delete other people's messages. Some or all permissions may be left unspecified for a given message set (in which case permissions propagate as explained later).*

Consider for example that there is a small group of people that need to hold discussions restricted to only themselves. For example, a mathematical journal might only want editors to be able to post new message sets (write permissions) within a certain area. Alternately, the moderators of a subject area might want to hold discussions private to themselves (read permissions).

Let us consider the case of a fictitious mathematical journal named "Analytic Geometry News". This journal concerns topics related to both calculus and geometry. Occasionally it may include some more off-topic discussions (e.g. teaching methods for analytic geometry) that intersect other subject areas.

Now, threads posted within the intersection of "Analytic Geometry News" and "Teaching" should have all the security restrictions of the former. Someone having full permissions to "Teaching" should not necessary have permissions to post new threads in the Teaching subsection of "Analytic Geometry News".

Furthermore, we may allow, just for illustration, Teaching to be restricted to postings by teachers so that the subsection will be restricted to users who are both editors and teachers.

Constraint 5 *In order to read a message, the user must must have read permissions in **every** message set that the message is contained in.*

Constraint 6 *Permissions propagate from supersets to subsets, but permissions explicitly defined in subsets override propagating permissions when those permissions are overridable.*

For example, the “Analytic Geometry News” and Calculus areas are both subsets of All. The permissions on All propagate to the Calculus area. However, the “Analytic Geometry News” area overrides certain permission given in All.

Constraint 7 *There may be restrictions in whether a given user can create a superset of a certain subject area, for the permissions of the new superset might propagate to the subset (if the subset doesn’t override such permissions). There may even be some harm if no permissions are defined on the new superset, such as if a mischievous user creates a message set named “my super-duper message set” that is a superset of Calculus, in which case Calculus will now display nested within “my super-duper message set”.*

As an alternative to propagating permissions by subset/member relationships, there may also be “templates” (similar to CSS) that apply based on different relationships (e.g. message sets having a certain attribute or keyword in the title).

8 Fuzzy Set Extensions

There are ways that the message set model could be extended to support fuzzy set like concepts such as partial membership of messages in message sets or or partial containment of message sets in other message sets. Warning: this section is highly experimental.

8.1 Fuzzy scores between messages and contained messages

Using the “one shallow container” rule, we can’t just go categorization happy and categorize each message in a thread into a different subject area. For example, consider a thread with the following messages:

```
m1: How do I compute the area of a circle?
m2: a = pi r^2
m3: a statistical approach would be to use monte carlo
    simulation of a dart board. go google it.
m4: you could use monte carlo, but it’s better to integrate
    the circle as follows in this very detailed explanation...
m5: integration works great!
```

It would violate the one shallow container rule to just do

```
Geometry = {..., m1, m2, m4, m5}
Statistics = {..., m3, m4}
Calculus = {..., m4, m5}
```

unless we create a couple message sets containing m1..m5 and then place these message sets into the subject areas. This may be overkill since such message set may contain only one or two messages and do not form any type of coherent discussion.

So, what the users would likely end up doing is assigning m1..m5 all to the same subject areas:

```
Geometry = {...,m1,m2,m3,m4,m5}
Statistics = {...,m1,m2,m3,m4,m5}
Calculus = {...,m1,m2,m3,m4,m5}
```

That is, all shallow messages in a thread are categorized identically. However, this scheme does not, for example, capture the fact that m2 is unrelated to Calculus. Here's where the fuzzy set concept comes in. Yes, we will assign all m1..m5 to each above subject area so that the above constraint is satisfied and so that the partial order over the message sets is defined exactly the same way as in the non-fuzzy set model. However, a non-zero fuzzy "relevance" score could also be recorded and assigned to each message ↔ message set link:

```
Geometry = {..., m1 : 1.0, m2 : 1.0, m3 : 0.2, m4 : 0.5, m5 : 0.5}
Statistics = {...,m1 : 0.1, m2 : 0.1, m3 : 1.0, m4 : 0.3, m5 : 0.1}
Calculus = {..., m1 : 0.1, m2 : 0.1, m3 : 0.4, m4 : 1.0, m5 : 0.9}
```

These scores may be used strictly for searching, ranking, and display purposes. So, if one does a search related to the subject area "statistics", then this particular thread might be assigned a relatively low ranking in the search results since the quantity of statistics content (as defined by the fuzzy scores) is low in the thread. Similar applications of such ranking could also be stated (e.g. send user e-mail alerts on statistics threads provided the fuzzy-inspired rank is high enough).

ADDED:2004-10-04. One downside is that this could involve a lot of work to maintain the categorizations. For this reason, similar to as in the security model, it may be more worthwhile to assign fuzzy scored instead on and between message sets, as described in the next section.

8.2 Fuzzy scores between message sets

There might be a use of fuzzy scores between message sets (in addition or replacement of between message sets and messages). Consider the following message set with two message subsets:

```
Message set #1
m1: How do I compute the volume of a cube?
m2: A = a*a*a - 1 + 1
m2: Or, more simply, A = a*a*a.
```

m3: Here's an example $A = 2*2*2 = 8$ for a $2x2x2$ cube.

Message set #2

m4: Don't forget the units: 8 units^3 .

m5: The units are implicit.

m6: No they, aren't. You need to specify the units.

Message set #3

m7: I once wrote a song on computing the volume of a cube.

m8: The volume of the cube...The volume of the cube...

m9: Hey, so did I! The cube has a volume...The cube has a volume...

Note that message set #2 is more related to message set #1 than is message set #3. This might be expressed by $\text{score}(1, 2) \geq \text{score}(1, 3)$. Further, these scores might be used in the display. For example, instead of specifying "show me a certain message set and all its message subsets of depth no greater than 2," you could more accurately say "show me a certain message set and all its message subsets whose product of scores along the parent-child path is no less than 0.75." A query such as that would weed out off-topic threads and sub-threads regardless of their depth.

9 Relational database structure

There are quite a few ways that the message set model can be mapped onto a relational database. The choice of representation depends on the relative goals of efficiency of queries v.s. updates, minimizing the amount of redundant data, reducing the possibility of representing inconsistent states (e.g. $A \supset B \supset A$), and the generality of the model (e.g. the "one shallow container" constraint). No method is without defect.

Method #1 Deduce the partial order from set membership alone.

Strictly speaking, only a single relation may be needed to represent the model:

$$R1 = \{(A, m) | m \in A\}$$

Using this relation, it is easy to list all the messages in a given message set or all the message sets containing a given message. Computing the shallow super/subsets and members/containers at run time is a little more challenging. One way is to build from the above relation a second relation:

$$R2 = \{(A, B, n) | n = |A \cap B| > 0\}$$

Appendix A.1 provides several SQL statements can then be composed using this new relation to compute the shallow super/subsets and members/containers.

Besides computational complexity, a possible downside is that the set hierarchy is determined from set membership alone. So, a subject area containing a single thread will be non-distinguishable hierarchically from its contained thread because both contain the same members and are therefore equal. The workaround is to sprinkle the message sets with invisible elements in appropriate places.

The details of implementing R1 and R2 are examined in Appendix A.1.

A slight alternative is to maintain a physical copy of R2. Here R2 is stored in the database and maintained incrementally on each database update. This improves query performance at a minor cost to update complexity. However, hierarchy is still determined by set membership alone.

Method #2 Store the set hierarchy explicitly.

One alternative to determining set hierarchy from membership is to maintain a third relation (and drop the second relation, R2):

$$R3 = \{(A, B) | A \supseteq B\}$$

It still takes some work to deduce the hierarchy, but the hierarchy is fully determined without knowing the contained messages. This also eliminates the need to explicitly store the invisible elements, and the invisible elements can be largely forgotten in practice. A possible downside is that the data storage is increasingly redundant, and it is difficult to recognize or fix inconsistent data at a glance (e.g. $A \supset B \supset A$).

R3 is examined in Appendix B.1.

An slight alternative would be

$$R3b = \{(A, B) | A \supset B\}$$

However, the downside is that it seems to make the queries more complex, visually at least, since the queries generally require more left-joins rather than inner-joins.

R3b is examined in Appendix B.2.

Additional relational information between message sets could be stored as well:

$$R4 = \{(A, B, d) | |A \cap B| > 0 \wedge d = \text{depth}(A, B)\}$$

This could make updates more complex, but the exact extent of this has not been examined. In fact, the exact depth (rather than merely identifying shallow v.s. deep superset relationships) may be unused.

A simpler variation is

$$R4b = \{(A, B, t) | |A \wedge B| > 0 \dots\}$$

where $t \in \{deepsuperset, shallowsuperset, equalset, shallowssubset, deepsubset\}$ identifies the relationship between A and B.

A slight additional simplification is

$$R4c = \{(A, B, s) | A \supseteq B, s = (A \text{ shallow_superset } B)\}$$

which can be used in conjunction with a similar relation for the message set-message relationships:

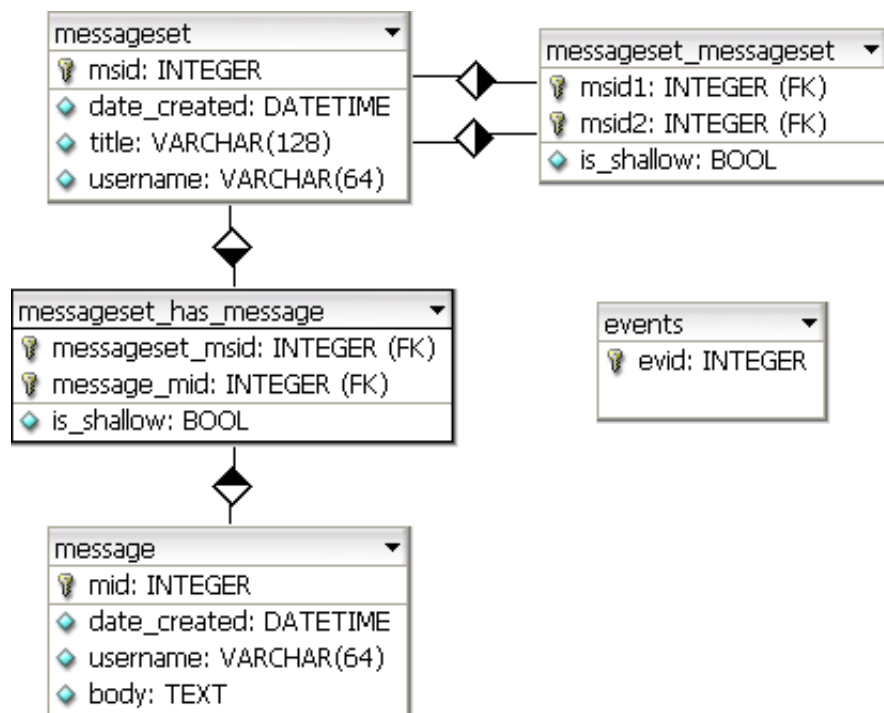
$$R5c = \{(A, m, s) | m \in A, s = (A \text{ shallow_container } m)\}$$

R4c and R5c are used in the current prototype of our software. This approach is a reasonably elegant compromise. It makes it trivial to list all supersets/subsets and members/containers as well as all shallow super/subsets and members/containers. There is a cost of complexity to updates, but this is kept manageable and not as complex as R4. Although redundant information is stored, the tuples having s is false can be considered as simply cached data because they can be regenerated from the the tuples with s is true. Therefore, if there is any corruption or discrepancy in the data, the former tuples can be discarded and regenerated.

R4c and R5c are examined in Appendix B.3.

9.1 Details

The relational model used is shown below.



10 Appendix A

The following Appendices A-B evaluate various database implementations of the set model. These sections be be skipped, and maybe they should be removed from this paper entirely. Appendix A examines solutions that deduce the partial order over message sets automatically. Appendix B examines solutions that store the partial order more explicitly.

10.1 Appendix A.1

In this subsection, we consider the case where

$$R1 = \{(A, m) | m \in A\}$$
$$R2 = \{(A, B, n) | n = |A \cap B| > 0\}$$

At a minimum, the relation R1 between messages and message sets could be stored as a single table within a relational database:

```
/*
 * Many-Many relation between message sets and contained
 * messages.
 */
CREATE TABLE messageset_message (
  /* message set ID */
  msid INT UNSIGNED NOT NULL,

  /* message ID */
  mid INT UNSIGNED NOT NULL,

  INDEX messageset_message_msid_idx(msid),
  INDEX messageset_message_mid_idx(mid)
);
```

Mathematically, `messageset_message` $\subseteq MS \times M$.

However, an SQL query that computes the shallow members and shallow subsets of a given message set is not easily expressed in MySQL. In fact, it seems to require more than one SQL statement or SQL mixed with Perl code (no, I don't have a proof!). The difficulty might be attributed to the lack of subselects, stored procedures, and other more advanced features in the current production version of MySQL (4.0).

FIX: I could put one of my algorithms on this here for historical interest.

It has been found that such queries can be expressed much more naturally if an additional relation R2 is used to store the number of messages shared between each pair of messages sets that share at least one message:

```
/*
 * 1-1 relation between message sets.
```

```

*/
CREATE TABLE messageset_messageset (
  /* first message set ID */
  msid_1 INT UNSIGNED NOT NULL,

  /* second message set ID */
  msid_2 INT UNSIGNED NOT NULL,

  /* number of messages shared by first and second message sets */
  cnt INT UNSIGNED NOT NULL,
  INDEX messageset_messageset_msid_1_idx(msid_1),
  INDEX messageset_messageset_msid_2_idx(msid_2)
);

```

This can be constructed using

Query 1 (messageset_messageset populate)

```

INSERT INTO messageset_messageset
  SELECT b.msid, c.msid, COUNT(b.msid) as cnt
  FROM messageset_message as b, messageset_message as c
  WHERE b.mid = c.mid
  GROUP BY b.msid, c.msid
;

```

Mathematically, $\text{messageset_messageset} \subseteq MS \times MS \times Z^+$
 $\text{messageset_messageset} = \{(A, B, n) : A, B \in MS \wedge n = |A \cap B| > 0\}$.

Some databases might define `messageset_messageset` as a view over the `messageset_message` relation.

Now, the queries for finding shallow members and shallow subsets of a given set can be expressed as single SQL queries and done quite naturally using joins and groupings.

Query 2 (shallow_subsets) *To find all shallow subsets of a given message set.*

```

SELECT mm_bc.msid_2
FROM
  messageset_messageset as mm_bc,
  messageset_messageset as mm1,
  messageset_messageset as mm2,
  messageset_messageset as mm_bb,
  messageset_messageset as mm_cc,
  messageset_messageset as mm_ab,
  messageset_messageset as mm_ac,
  messageset_messageset as mm_aa
WHERE
  mm_aa.msid_1 = ? AND mm_aa.msid_2 = mm_aa.msid_1 AND

  mm_bc.msid_1 = mm1.msid_2 AND mm1.msid_1 = mm_aa.msid_1 AND
  mm_bc.msid_2 = mm2.msid_2 AND mm2.msid_1 = mm_aa.msid_1 AND

```

```

mm_bb.msids_1 = mm_bc.msids_1 AND mm_bb.msids_2 = mm_bb.msids_1 AND
mm_cc.msids_1 = mm_bc.msids_2 AND mm_cc.msids_2 = mm_cc.msids_1 AND
mm_ab.msids_1 = mm_aa.msids_1 AND mm_ab.msids_2 = mm_bc.msids_1 AND
mm_ac.msids_1 = mm_aa.msids_1 AND mm_ac.msids_2 = mm_bc.msids_2 AND

mm_ac.cnt = mm_cc.cnt AND mm_cc.cnt < mm_aa.cnt
GROUP BY
mm_bc.msids_2
HAVING
SUM(mm_ab.cnt = mm_bb.cnt AND mm_bb.cnt < mm_aa.cnt AND
mm_cc.cnt = mm_bc.cnt AND mm_cc.cnt < mm_bb.cnt) = 0
;

```

Mathematically, $\{C : |A \cap B| > 0 \wedge |A \cap C| > 0 \wedge C \subset A \wedge \neg \exists B \text{ such that } C \subset B \subset A\}$

Note that each SQL expression of the form $|A \cap C| = |C| \wedge |C| < |A|$ is another way of writing $C \subset A$.

Query 3 (shallow_members) *To find all shallow members for a given message set:*

```

SELECT a.mid
FROM
  messageset_message as a,
  messageset_message as b,
  messageset_messageset as mm_ab,
  messageset_messageset as mm_aa,
  messageset_messageset as mm_bb
WHERE
  a.msids = ? AND a.mid = b.mid
  AND mm_ab.msids_1 = a.msids AND mm_ab.msids_2 = b.msids
  AND mm_aa.msids_1 = a.msids AND mm_aa.msids_2 = a.msids
  AND mm_bb.msids_1 = b.msids AND mm_bb.msids_2 = b.msids
GROUP BY
  a.mid
HAVING
  SUM(mm_ab.cnt = mm_bb.cnt AND mm_bb.cnt < mm_aa.cnt) = 0
;

```

Mathematically, $\{m : m \in A \wedge \neg \exists B \text{ such that } m \in B \subset A\}$.

FIX: I could add code for supersets, intersection sets, etc... The concepts are similar.

10.2 Efficiency

The messageset_messageset relation does require the storage of redundant data (with respect to the messageset_message relation). So, any time a message is added to or removed from the messageset_message relation, the messageset_messageset relation will have to be updated correspondingly. Luckily, the

update can be done incrementally, and only a limited number of records (corresponding to the message sets that the message is a member of) need to be updated. FIX: could describe algorithm or SQL for this.

FIX: The efficiency and scalability of the above SQL queries has not been tested for real sized data sets (e.g. +100000 messages), but they do look reasonably efficient.

11 Appendix B

In this section, we consider cases where the non-proper supersets relation is stored explicitly.

11.1 Appendix B.1

In this subsection, we consider the case

$$R3 = \{(A, B) | A \supseteq B\}$$

```
/*
 * M-N Relation between message sets and contained
 * messages.
 * {(A, m) | m in A}
 */
CREATE TABLE messageset_message (
  /* message set ID */
  msid INT UNSIGNED NOT NULL,

  /* message ID */
  mid INT UNSIGNED NOT NULL,

  PRIMARY KEY (msid, mid)
);

/*
 * M-N Relation between message sets and contained
 * message sets.
 * {(A, B) | A nonproper superset B}
 */
CREATE TABLE messageset_messageset2 (
  msid_1 UNSIGNED INT NOT NULL,
  msid_2 UNSIGNED INT NOT NULL,
  PRIMARY KEY (msid_1, msid_2)
)
*/

Query 4 (shallow_subsets)
SELECT ac.msid_2
```

```

FROM
    messageset_messageset2 as ac,
    messageset_messageset2 as bc,
    messageset_messageset2 as ab
WHERE
    ac.msids_1 = ? AND
    bc.msids_2 = ac.msids_2 AND
    ab.msids_2 = bc.msids_1 AND
    ab.msids_1 = ac.msids_1
GROUP BY ac.msids_2
HAVING COUNT(bc.msids_1) = 2

```

Query 5 (shallow_supersets)

```

SELECT ac.msids_1
FROM
    messageset_messageset2 as ac,
    messageset_messageset2 as ab,
    messageset_messageset2 as bc
WHERE
    ac.msids_2 = ? AND
    ab.msids_1 = ac.msids_1 AND
    bc.msids_1 = ab.msids_2 AND
    bc.msids_2 = ac.msids_2
GROUP BY ac.msids_1
HAVING COUNT(ab.msids_2) = 2

```

Query 6 (shallow_members)

```

SELECT am.mid
FROM
    messageset_message as am,
    messageset_messageset2 as ab,
    messageset_message as bm
WHERE
    am.msids = ? AND
    ab.msids_1 = am.msids AND
    bm.msids = ab.msids_2 AND
    bm.mid = am.mid
GROUP BY am.mid
HAVING COUNT(ab.msids_2) = 1

```

Query 7 (shallow_containers)

```

SELECT
    am.msids
FROM
    messageset_message as am,
    messageset_messageset2 as ab,
    messageset_message as bm
WHERE
    am.mid = ? AND
    ab.msids_1 = am.msids AND
    bm.msids = ab.msids_2 AND

```

```

        bm.mid = am.mid
GROUP BY
    am.msid
HAVING COUNT(ab.msids_2) = 1

```

Query 8 (intersection_sets)

```

SELECT cb.msids_1
FROM
    ((messageset_messageset2 as ab,
     messageset_messageset2 as cb)
 LEFT JOIN
     messageset_messageset2 as ac
   ON ac.msids_1 = ab.msids_1 AND ac.msids_2 = cb.msids_1)
 LEFT JOIN
     messageset_messageset2 as ca
   ON ca.msids_1 = cb.msids_1 AND ca.msids_2 = ab.msids_1
WHERE
    ab.msids_1 = ? AND
    cb.msids_2 = ab.msids_2 AND
    ab.msids_1 <> cb.msids_1
GROUP BY cb.msids_1
HAVING SUM(ac.msids_2 IS NOT NULL OR ca.msids_2 IS NOT NULL) = 0

```

One possible downside is that some of the queries (particularly those in locating shallow super/subsets and containers/members), though not higher complex, are not trivial either. This can be rectified using the scheme described in Appendix B.3.

11.2 Appendix B.2

This section considers a database structure very similar to as in Appendix B.1 except that we define

$$R3b = \{(A, B) | A \supset B\}$$

Query 9 (shallow_subsets)

```

SELECT ac.msids_2
FROM
    messageset_messageset2 as ac,
    messageset_messageset2 as ab LEFT JOIN
    messageset_messageset2 as bc
ON
    bc.msids_2 = ac.msids_2 AND ab.msids_2 = bc.msids_1
WHERE
    ac.msids_1 = ? AND
    ab.msids_1 = ac.msids_1
GROUP BY ac.msids_2
HAVING COUNT(bc.msids_1) = 0

```

Query 10 (shallow_members)

```
SELECT am.mid
FROM
  (messageset_message as am,
   messageset_message as bm) LEFT JOIN
  messageset_messageset2 as ab
  ON
    ab.msids_1 = am.msids AND ab.msids_2 = bm.msids
WHERE
  am.msids = ? AND
  bm.msids = am.msids
GROUP BY am.msids
HAVING COUNT(ab.msids_2) = 0
```

Query 11 (intersection_sets)

```
SELECT cb.msids_1
FROM
  ((messageset_messageset2 as ab,
   messageset_messageset2 as cb)
  LEFT JOIN
    messageset_messageset2 as ac
    ON ac.msids_1 = ab.msids_1 AND ac.msids_2 = cb.msids_1)
  LEFT JOIN
    messageset_messageset2 as ca
    ON ca.msids_1 = cb.msids_1 AND ca.msids_2 = ab.msids_1
WHERE
  ab.msids_1 = ? AND
  cb.msids_2 = ab.msids_2 AND
  ab.msids_1 <> cb.msids_1
GROUP BY cb.msids_1
HAVING SUM(ac.msids_2 IS NOT NULL OR ca.msids_2 IS NOT NULL) = 0
```

The disadvantage of this method over that in Appendix B.1 is that it requires left joins, which make the SQL appear more complicated.

11.3 Appendix B.3

The approach described in this section is similar to that in Appendix B.1 except that

$$R_{4c} = \{(A, B, s) \mid A \supseteq B \wedge s = (A \text{ shallow_superset } B)\}$$

```
/*
 * M-N Relation between message sets and contained
 * message sets.
 * {(A, B, s) | A nonproper superset B \wedge s = (A shallow_superset B) }
 */
CREATE TABLE messageset_messageset2 (
  msids_1 UNSIGNED INT NOT NULL,
  msids_2 UNSIGNED INT NOT NULL,
```

```

        is_shallow UNSIGNED TINYINT(1) NOT NULL DEFAULT 0,
        PRIMARY KEY (msid_1, msid_2)
    )
*/

```

The “is_shallow” column can be reconstructed in the messageset_messageset2 relation by the following SQL.

Query 12 (is_shallow populate)

```

CREATE TEMPORARY TABLE temp AS
SELECT ac.msid_1, ac.msid_2
FROM
    messageset_messageset2 as ac,
    messageset_messageset2 as bc,
    messageset_messageset2 as ab
WHERE
    bc.msid_2 = ac.msid_2 AND
    ab.msid_2 = bc.msid_1 AND
    ab.msid_1 = ac.msid_1
GROUP BY ac.msid_1, ac.msid_2
HAVING COUNT(bc.msid_1) = 2
;
UPDATE messageset_messageset2 as ab, temp
SET is_shallow = 1
WHERE
    ab.msid_1 = temp.msid_1 AND
    ab.msid_2 = temp.msid_2
;
DROP TABLE temp;

```

The reverse can be performed by repeated invocations of an insert join statement that uses the transitivity relationships.

shallow_subset and shallow_superset queries are trivial compared to those in Appendix B.1.

shallow_member and shallow_container are not as trivial unless relation R5c is similarly maintained:

$$R5c = \{(A, m, s) | m \in A, s = (A \text{ shallow_container } m)\}$$

intersection_sets is not known to be simplifiable.

References

- [1] Linear v.s threaded discussion. <http://support.discusware.com/center/resources/essays/thread.php>, 2003.
- [2] Andy roberts’ blog. <http://blog.ultralab.net/~blogger/andy/archives/000651.html>, 2004.

- [3] Nelson email organizer - neo - the microsoft outlook email software add-on.
<http://emailorganizer.com>, 2004.
- [4] Relink. <http://relink.sourceforge.net>, 2004.