MyVision on MMBase.

I am involved in the MMBase project for several years now and have developed my own view on it. The last release introduced several new features which are implemented in a way we think MMBase should be in the future. Some side-effect are coming to the surface which makes me believe we shouldn't stay in this code state too long.
This document is going to contain my ideas on MMBase and how I think MMBase should be. This document is food for thought to start discussing the shortcomings of MMBase and how we could resolve some of them.

A discussion on the developers list pointed out that my idea of "what MMBase is" is different from others so here are the definitions I use.

MMBase community - Everyone who participates on the mailinglists or builds an MMBase web application.
MMBase web application - An web application which uses the binaries from mmbase.org
MMBase community application – Group of files which implement some functionality and which is maintained by the MMBase developers. These files make it easier to use the MMBase core in a web application.
MMBase core – the only jar which is required to use the data management features.

This document uses the word MMBase as substitute for the MMBase core. This document only discusses the MMBase core.

Before going into the details of the MMBase core, it is important to ask the question what the scope is and what the requirements are. These can be derived from the problem domain MMBase solves? In my view MMBase solves the domain of a data repository which is backed by a database. MMBase has many data management features a data repository requires. Examples of these are: search, notification, access control and type, object and relationship management. A data repository consists of a repository engine and an API to interact with the engine. The engine has to solve every aspect of the issue like storage, configuration, object model, security, search, events, etc

The above problem domain can not be solved by a pojo-based application which has hard coded the types and relations in a domain model. A data repository is responsible for type and relationship management and this does not have to be visible in the database. A pojo is always mapped to a (set of) database tables.

It is time to zoom in on MMBase. There are several subsystems in MMBase and each solves a different part of the problem domain.

Configuration
: Reading, writing and parsing of configuration resources is the primary task of this subsystem.

Storage
: CRUD actions on the database. The storage layer hides the differences between database dialects.

Cache
: The cache subsystem reduces the load on the database. An average MMBase ` application reads more than it updates.

Search

An API which can create select queries in a database independent way.

Model

MMBase has a dynamic structure with node types and allowed relations

Security

The security implementation answers all questions about the rights of a user or system.

Module

A module adds service functionality to the system. It could perform task like indexing, cleaning, etc. It could also start a service which listens to remote requests.

Events

This subsystem notifies listeners when data is changed.

Some of the above subsystems are already visible in the code by its own package. Others are hidden in packages like module and util. The order in which the subsystems are mentioned is important, because lower systems are dependent on higher ones. MMBase could start the subsystems in this order.

Configuration

The sources which implement this subsystems are all marked as util, Examples of these are ResourceLoader, FileWatcher and the xml readers. A class as entry point (Facade) to this subsystem is missing. The ResourceLoader is a good start to centralize configuration management, but it is still low level. Other subsystems should interact with the configuration system on a higher level. Other subsystems are only interested in settings and objects.

Storage

This layer has been rewritten for several times now. The current version meets the requirements. The storage has an abstraction in what type of systems it stores its data, but that is overkill in my opinion. MMBase is a data repository backed by a database. A batch update and delete interface is missing.

Cache

MMBase has several types of caches. They can be grouped into three categories.

- Resource caches – The BlobCache, CKeyCache, JumpersCache are all in this categorie
- Model caches – These caches contain the model of MMBase. The NodeCache and RelatedNodeCache are both easy to update when a node changes in MMBase.
- Query caches – These caches contain result which are retrieved by a search query.

The last two cache types are to reduce the load on the database. They can be used by the storage subsystem. The cache system shouldn't be seen by other subsystem then the storage. The storage subsystem has all the information to keep the cache system up to date with the database. The storage and cache systems should be in a stable state after each action from another subsystem. This is currently not the case with MMBase, because the caches are updated by the event system.

IMO, we should also consider when to use Query caches, because they usually contain queries which are just as fast in the database as the cache is. Just to make a small comparison, hibernate does not recommend to use their query cache. Hibernate is used in application which have equal or more updates than reads.

Another thing MMBase is missing is a batch update interface to prevent heavy cache invalidation. The cache and storage system should be in a stable state after each action which means that only cache validation should happen at the end of the batch update.

Search
MMBase has a search api to develop code which is database dialect independent. The api currently has two issues. First it is part of the storage layer so it is duplicated in the bridge. The api should get a generic part in its own package and a part (QueryHandlers) should stay in the storage layer. Second the api is elegant in design, but not easy to use in development. Query code requires now too many lines of code to make a simple query.

Model
This is the interesting part of MMBase and which makes it different from other products. During the years I have identified the following concepts.

The MMBase model is based on a concept similar to the class/object pattern. The builder or node type corresponds to the class concept. The node instance represents the state of the node at runtime.

MMBase uses a base data model which is stored in five tables. These five node types are the base of the model. Everything is derived from this.

- object: instances of persisted information
- insrel: instances of relations between persisted objects
- typedef: definition of persisted objects (object type/node type)
- reldef: definition of relation between persisted objects
- typerel: defines which relation definition is allowed between object types

Another interesting node type to mention is the oalias (Object alias) which maps a name to a node number.

MMbase uses internal several types of object instances. These types are different from node types, because they are used for different purposes.
The first is MMObjectNode which is used to represent the object instances listed above. An MMObjectNode can be persisted in the database.
The second is the VirtualNode which removes the persistent feature from the MMObjectNode. A VirtualNode is usually derived from real MMObjectNodes. For example, typerel nodes are stored in the database, but there are more possible relations then stored in the database. MMBase implemented node type inheritance and subtypes inherit the relations of their super types. These allowed relations are stored in a VirtualNode.
The third are ClusterNode and ResultNode. These nodes remove the restriction of one node type per node. Both nodes can have multiple node types corresponding to the tables used in the query which loaded the instances. Instances of these types are not fully loaded instances of one node type. They combine the fields which are mentioned in the query. The real difference between ClusterNode and ResultNode is not clear to me. The ResultNode is in the storage search package and might be introduced for the same purpose as the old ClusterNode for multilevel queries.

Every node type has fields and functions. Every field has a datatype attached to it. The way fields work are great. Still a lot needs to be done in the implementation to make it a clean and lean implementation.
There are three type of fields in MMBase. Again, these are not datatypes, but conceptually different types.
One, the persistent field which is mapped to a column in the database.

Two, the virtual field which can be used to calculate a value based on external data like persistent fields or values in an external system.
Three, temporary fields. These fields are hidden from outside the system, but can be used to do some administration. Temporary fields start with an underscore.

A node type can have a java class attached to it to extend some methods. A lot of old features implemented with these extensions could now be moved to FieldSet and FieldCommit processors. I can still come up with some use cases where a builder extension is still nice, but extending the MMObjectBuilder and overriding the core methods is a little dangerous. You could easily forget to call the super methods. Another solution would be great to add functionality to a node type.

Security

Security is handled by the bridge at the moment. Some security features in web applications are done by the taglib (eg login methods). MMBase does not use the j2ee security api and does not work well together with the application server container security. You always have to write your own Authentication class when you don't administer users in the MMBase system.

Module

A MMBase module can be used to start additional services for an application. In previous releases I usually used modules to start threads for maintenance tasks, but the crontab feature is better suited for this. Other ways of usage are to start listeners like rmi, smtp, irc, etc.
The only peculiar thing about the modules is that MMBase is a module of its own. IMO, this should change. MMBase should be the system which starts all subsystems including the module sub system.

Events

This is a very nice feature to detect all node and relation changes. The events should fire after the storage layer is in a stable state again and all node type extensions are processed.

The only thing which is still not discussed is the api to interact with MMBase. In MMBase terms this is called the bridge. The bridge has been added to MMBase to make it possible to use several templating languages with MMBase. The bridge was and in most cases still is a wrapper layer with security features. This has been sufficient for some years, but not anymore as the fieldtypes project has shown. The wrapper class Field and Node in the bridge have now features which should have been added to a class which could be used in the bridge and core.
I like to change the role of the bridge in the MMBase system. The bridge should become a user session for the MMBase core system. Nodes are loaded into the user session and the session tracks changes. A user session can be committed back to the MMBase core. The cloud object should become a session like hibernate or jsr-170 uses to let code interact with the system.
Changing the role of the bridge to the above will make the core system responsible for the repository data stored in the database. The core is not responsible to track changes in field

values. The core is responsible to handle the commit of changed field values in an ACID way. The core system should return in a stable state after each commit has finished.
The cloud/user session loads a node from the repository into its own space. After the load, the cloud has a snapshot of the repository when it was in a certain state. The cloud will always return the snapshot version instead of the version in the repository. A search with criteria in the repository could return the node even when the snapshot version does not match anymore.

The above means a big change in the core code, Most of the functionality is currently in the MMObjectNode class. This change does not affect the way a developer uses the bridge. The developer already has to call commit on nodes and transactions to persist the changes in MMBase.

When we make this change then all custom builders and modules (which are using MMObjectnode) won't work anymore, because the changes are not tracked by the MMOBjectNode class. We need a new way to create a custom builder.