Decision register

Who do we serve stakeholders and users.

Decision

Focus on User roles

i. User roles

1. Site builder / Integrator
2. Extension developer
3. Backend user (administrator / manager)
4. Site owner (decision maker)

ii. Contributor

iii. Site visitor

iv. Hoster

Reason

We focus on user roles, the other stakeholders need to be informed but are outside of our scope. Contributors also typically have a secondary user role.

How do we represent our target audience to make valuable decisions?

Decision

Personas will be created for each class of stakeholder. They should also cover the combined "Site owner / Site builder / Backend user" sufficiently. The personas are published to the community to use them as "User Patterns", similar to "Design Patterns" in software development.

Reason

Personas help to focus on the needs of our users and other stakeholders.

References

See https://en.wikipedia.org/wiki/Persona_%28user_experience%29 for more detailled information.

What is the Joomla! 4 timeline

Decision

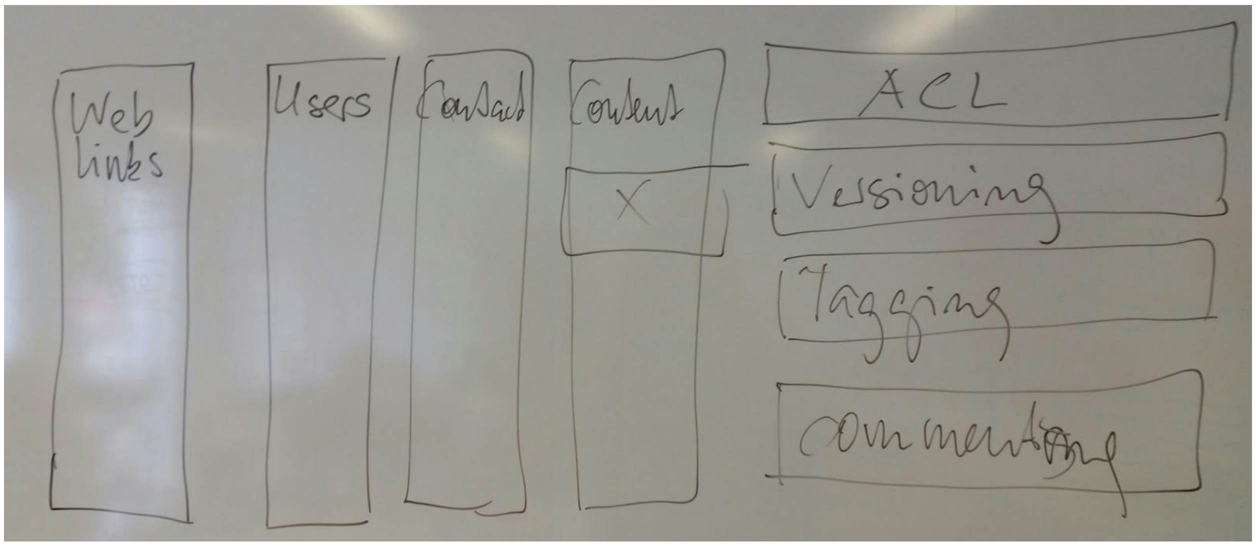
Release at JAB 2016

Reason

We need to timebox based on a end-date

Orthogonal Component Structure

Decision



An orthogonal system is introduced, where two different kinds of components are distinguished:

- Vertical: Weblinks, Contacts, Users, Content, …
- Horizontal: Versioning, translating, tagging, commenting, ...

Any horizontal component works with any vertical component *out of the box* being themselves agnostic about other components. This way, com_contact, com_weblinks, even com_users would automatically have ‘inherited’ tagging and versioning in 3.2 / 3.3 without the need to change a single bit of code in these components.

Reason

This approach allows any component to take advantage of new capabilities that are designed to be used across extensions. The orthogonal structure is a design supporting change.

CQRS (Command Query Responsibility Segregation)

Decision

The separation of read and write models will be implemented

Reason

CQRS allows for scaling, by supplying multiple slave databases for read access and thus increase performance on large sites.

References

http://martinfowler.com/bliki/CQRS.html

Event Sourcing

Decision

Event Sourcing was deemed of interest to the attendees. The library “broadway” was proposed as a potential solution although it would need a good amount of integration with the CMS/Framework (the eventdispatcher, use of DBAL etc.).
We need performance tests on this.
Complexity for extension developers must be taken into account

Performance Testing Information

There were also potential performance issues raised. A complete cycle for storing an article using Event Sourcing shall not take more than 150% of the time than it does using the current JModelLegacy in traditional 3.x

Reason

Using events instead of states for internal communication allows other parts of the software to react on these events in an adequate way. Storing is lossless.The impact of replaying the events on the last snapshot should not have much impact (will get tested first), but give a native versioning instead. Event sourcing is a design supporting change.

References

https://github.com/qandidate-labs/broadway

Minimum Joomla! 4.0 PHP Version

Decision

Working minimal version PHP 5.5.9
Ultimate minimal PHP version will be defined at the moment of Joomla release. It cannot be changed within the major release after 4.0.0.

Reason

End of support

a. PHP 5.4 14 Sep 2015
b. PHP 5.5 10 Jul 2016
c. PHP 5.6 28 Aug 2017

Want to use native password hashing, instead of libraries, and misc language features

Use of namespacing

Decision

We will use namespacing with the following convention:

a. Core Components

   Joomla\Component\<Component>\<App>

   - Joomla\Component\Content\Site

        - Joomla\Component\Content\Admin

b. 3rd party Component

   <Vendor>\<Component>\<App>

      - Akeeba\Backup\Site
      - Akeeba\Backup\Admin

c. Other extensions (modules, plugins)

   - Module

     <Vendor>\Module\<App>\<ModuleName>

      - Akeeba\Module\Site\SubscriptionStatus
      - Akeeba\Module\Admin\LastBackupStatus

   - Plugin

     <Vendor>\Plugin\<Group>\<PluginName>

      - Akeeba\Plugin\System\BackupOnDemand

   The vendor part of Joomla core extensions is ‘Joomla’.

d. Framework Library:

   Joomla\<Package>

   - Joomla\Image
   - Joomla\Controller

e. CMS Library:

   Joomla\Cms\<Package>

   - Joomla\Cms\Mail

It is recommended that the JED marks extensions not adhering to this scheme.

Reason

This scheme covers the need of the Joomla project(s). 3rd party developers use their own vendor namespace, so collisions are effectively avoided.

Use of database abstraction layer

Decision

Our intention is to use a DBAL. We will investigate using Doctrine 2’s database abstraction layer. We are not going to put an abstraction layer in front of the DBAL.

Reason

The DBAL allows us to target all of its supported database types without having to write queries specifically for the database type. It supports testing by making database mocking easy to accomplish.

Its performance is near to what we can do by manually optimising queries and generally better than our under-optimised queries, something we have in common occurrence in Joomla!. In case a developer needs to use native queries, for example to get the best out of performance optimisation, the DBAL does allow them to do that and even lets them know the database type to target it with the correct query format.

Finally, the DBAL has data importers and exporters that work which can create and update both the schema and the data of the database.

We are not going to put an abstraction in front of the DBAL because of performance reasons and because it has no added value.

References

Doctrine 2 DBAL https://github.com/doctrine/dbal/
http://doctrine-dbal.readthedocs.org/en/latest/

Using composer for 3rd party extensions

Decision

Composer isn’t usable in mass distributed Joomla because of issues with the time it takes to update dependencies and the inability of users to debug failed dependency resolutions. Most users also do not have SSH abilities on Shared Hosting and requiring users to build their sites on a local machine is not feasible.
Nevertheless, a composer-like dependency management is wanted for Joomla’s installer. Dependencies, which cannot be resolved automatically, are represented to the user (admin) as a choice of possible resolutions.

Reason

The design for change encourages components to only provide their core functionality and pull in other features as needed from other components. As an example, a forum component provides the thread management and pulls in any commenting component to manage follow-ups. It is up to the site builder to decide *which* commenting system to use.

References
White paper by Nic about improving the Joomla Installer system:
https://dl.dropboxusercontent.com/u/5168399/Joomla%21%20Extensions%20Installer%20TNG.pdf
Drupal is dealing with composer for D8 between core and third party and how it all affects distro, site management, etc. https://www.drupal.org/node/2002304 has some useful resources.

Filesystem abstraction

Decision
We will use flysystem as our filesystem abstraction layer.

Reason
Framework already has planned to create a wrapper for the version 2 Flysystem package. It has existing adapters for FTP, SFTP and Local wrappers. The AWS adapter might prove to be useful when rebuilding the media manager.

References
Flysystem: https://github.com/thephpleague/flysystem

Universal Content Model

Decision
Content uses the Composite Pattern. There is no difference (internally) between categories and content items. *Any* component producing output implements the same Content Interface. Output is created using channel specific Renderers.
According to Herman, PHPCR implements that concept already.
Performance of pagination, edit, delete with PHPCR has to compared the same with current Joomla 3.x.

Reason
Content should be handled as content, not just articles. In reality, content is made up as a non-cyclic graph: paragraphs, images, or other content types can have multiple parents, so content gets re-usable. For each node, the site owner can choose to render its children as a list (like categories do now), or to compose a page from the children.
Performance: Pagination, edit, delete, are hurting performance-wise in J3 currently. If PHPCR does not worsen things, it is the way to go.

References
Content structure: http://nibralab.github.io/joomla-architecture/content-structure.html
PoC for Renderer:
https://github.com/nibralab/joomla-architecture/blob/master/poc/dynamic-renderer.php

GPL3

Decision

The attendees prefer to stay with GPL v2 or later, i.e., not to change the license. However it is needed to consult with lawyers.

Reason

Majority of intended functionalities (doctrine2 DBAL, flysystem etc.) use the MIT license (X.11). We need to check with the Legal Team regarding distribution of MIT-licensed code with GPL2. Otherwise we will need to upgrade the CMS license to GPL3.

Consideration

Contributors that have had a “significant contribution” (? > 100 lines) to the code that have not signed JCA need to be consulted.
Extensions should be a limited problem as the usually have been licensed as *GPL2 or later*.

Backwards compatibility management with 3rd party code

Decision

No action required. Some effort will be made to allow J3 extensions to work with J4.

Reason

Decision can’t be made in advance. Ultimately we could bump to next major version to cater for bc break caused.

Multilingual support

Decision

Multilingual support is provided. With the implementation of the J3 solution, a compound id (id+langcode) has to be used.

Reason

The orthogonal structure allows to implement different solutions - e.g., one working like JoomFish/Falang, another like it is done in J3.
If the results from the PHPCR performance test are satisfying, translated items could be just nodes in the graph, selected by the ACL, language setting etc.
The compound id allows us to keep the id-number for different languages. Thus we get rid of complicated associations, and additionally get a fallback mechanism.

Routing

Decision

The routers along the lines suggested by Niels and Hannes (with possible additions from the Framework's router) are used as a base. The menu system is used to generate custom route patterns for specific sites. The route must still run through JComponentHelper. For REST we can support "best practice" URLs using the same router.

Reason

The approach provides ability for auto-generating route patterns, but allow for customization. Best practice RESTful URLs come out of the box. A router.php will no longer be needed in components.

As there are going to be multiple MVC systems used within 3rd party components the routing must be independent of the MVC layer. JComponentHelper is therefore used to "dispatch" components in the CMS and therefore all routing should call the component helper.

References

https://github.com/nibralab/joomla-architecture/blob/master/poc/router.php
https://github.com/Hackwar/joomla-cms/tree/jrouter
https://github.com/joomla-framework/router

MVC Implementation

Decision

The proposed command structure will be implemented. (see references; naming according to GOF Design Patterns)
A command decides which model(s) to use. It is responsible for making input available to the model and to add the output to the visitable output object graph. Thus, the model can be literally any object with public members - in theory even a J1.5 MVC triad or a non-Joomla solution.
The output graph is transformed into a streamable format (according to content negotiation), and - if appropriate - wrapped into a PSR-7 response object.

Reason

The command/controller approach gives most possible flexibility for the implementation of models, so it is possible to integrate existing software. it allows to do proper CQRS, if wanted, by letting read ('Query') and write ('Command') commands use their own model.
The renderer approach allows to serve any output channel (JSON, XML, HTML, PDF, ePub, …) without the model or controller even knowing about that.

References

Command structure http://nibralab.github.io/joomla-architecture/command-structure.html
Content negotiation
https://github.com/nibralab/joomla-architecture/blob/master/poc/renderer-factory.php

Renderer https://github.com/nibralab/joomla-architecture/blob/master/poc/dynamic-renderer.php

Adding a custom renderer

Decision

Examples will be drawn up for

1. CLI
2. HTML
3. Download
4. JSON-LD (i.e. something that isn't included in core by default)

Reason

Obvious.

Custom fields

Decision

Custom fields will get implemented as a horizontal component.

Reason

As a horizontal component, custom fields are provided for any existing and future vertical component.

Data Context Interaction

Decision

Interesting, but needs more exploration and concrete examples. Also: what are specific use-cases in the CMS.

Reason

It is also an orthogonal division of responsibilities: adding behaviour (roles) to objects only within the context of use-cases.

References

DCI: http://fulloo.info/Documents/
http://www.infoq.com/presentations/Reflection-OOP-Social
http://www.slideshare.net/HermanPeeren/dci-dddbe

Composition vs Extension

Decision

We prefer composition over inheritance. When tempted to use traits, consider composition.

Reason

Loose coupling increases maintainability and supports change.

Dependency Management

Decision

Use Dependency Injection (Containers) wherever possible. Avoid Service Locators, if possible. Global services are requested by raising an event (e.g., ‘requestLogger’), which returns a logger.

Reason

With Dependency Injection, the provided services and objects are controlled by the calling instance. Dependency Injection Containers may help to keep the signature of constructor short, if several dependencies are required.
Service Locators are appealing on first sight, but tend to turn into an anti-pattern, as they provide global references and cannot easily be tailored for special use cases (HMVC, for example, may need different services for subsequent requests).
Requesting services using events decreases coupling. The event dispatcher is responsible for caching the result of those events, so there is no performance impact.

FIG Standards

Decision

We’ll follow PSR-3, PSR-4, and PSR-7. PSR-1 has to be reviewed; if feasible, we’ll follow it, too. We don’t adopt PSR-2. We will keep an eye on PSR-6 (Caching) and will adopt it if it is agreed before we reach beta 1.

Reason

As a voting member of the PHP-FIG we want to live up to our responsibility.
PSR-2 is not an option for Joomla, because its own code style (which actually is older than PSR-2) is widely adopted in the community.

BDD, TDD

Decision

Although testing is strictly not within the scope of architecture, we want to encourage test-first development. Leading by example we will write Behat behavioral tests, integration tests and unit tests for the structures we propose.

Reason

Red-Green-Refactor will yield better and less buggy code. Good tests will give developer documentation, too.

Migration

Decision

Supporting migration shall be an integral part of every deliverable in the context of Joomla! architecture, implementation and new features.

Reason

We want to deliver a non frustrating migration experience: migration without frustration.

Inform and educate 3rd party developers

Decision

Provide documentation and guidance for 3rd party developers to migrate their extensions. This is both a marketing and technical challenge.

Reason

We want to deliver a non frustrating migration experience

DDD

Decision

Using DDD is at the discretion of the implementers. Working group is to be informed upfront to discuss. Core components should be refactored to follow DDD with minimum impact on the outside view.

Reason

Affects the components (model) and not so much the system as a whole.
It should be possible to use the same user interface as in J3 (the paradigm change under the hood), so current users do not get distracted.

BC Policy

Decision

No change required.
In development care has to be taken, defaulting to private, public and using protected only if needed.
Also see next section (APIs).
Where API's are going to be replaced in future Joomla versions, we should ensure that the classes and methods are marked as deprecated.

Reason

Existing BC definition should be ok

APIs

Decision

Postpone the decision until the formalisation of PSR5 and discussion on acceptance
If PSR-5 is finished and accepted by Joomla, API relevant members (properties and methods) are annotated with '@api'. Public members, which are not part of the API, should be annotated as '@internal'. Changing members not marked '@api' are subject to change without notice, not causing a BC.

Reason

PSR-5 defines an '@api' and an '@internal' tag for this purpose. Using these tags makes clear, which elements are covered by the BC promise.

Reference

https://github.com/phpDocumentor/fig-standards/blob/master/proposed/phpdoc.md

JForms

Decision

JForm will be refactored to support not only form generation, but also list and detail views. The input is provided by a form definition class, which is populated through an XML file, or - if feasible - through JSON or Yaml files (optionally).

Reason

JForm provides a template independent gateway to data structures. Supporting list and detail views in addition to the forms will reveal the real power of the concept. Allowing other file formats for the definition input will enhance flexibility.

Define custom URLs using alias field

Decision

It is intended to extend the use of the alias field. If an alias starts with a ‘/’, its URL is at root level.

Example: /my-article becomes example.com/my-article

If the alias does not start with ‘/’, it gets prepended with its category. This is continued, until a leading slash is found or root is reached.

There has to be a way to suppress the id in the URL.

The current behavior has to be retained, however, as the default behavior.
A final decision is taken after an elaborated proposal is provided and reviewed.

Reason

Joomla users complain about not being able to define SEF URLs for articles. This approach provides an intuitive way to provide that feature without breaking current behavior.

Framework Reconciliation

Decision

Stuff gets merged both ways. CMS changes seep up into the framework

profiler, session and language.

Reason

It makes sense to keep CMS and Framework as close to each other as possible.