i18n in Magnolia 5

Rationale

Magnolia 5.0 went slightly backwards - we lost some translations. The mechanisms are still in place, but we "forgot" to implement them; many dialogs are configured with "hardcoded" labels. Some code, too.

More importantly, the way we apply i18n through the UI is currently inconsistent or inexistent. Most new constructs (apps, action bars, ...) have no notion of i18n. Blindly applying the same slightly dated concept we've been using up to 4.5 would blow the configuration out of proportion. Below are a few suggestions to explore and make this simpler, smaller, or more consistent.

Analysis of current situation

getLabel() vs getI18nBasename() vs usages

Each translatable item currently has a `getLabel` (or `getTitle` or `getWhatever`) method, some have several (`getDescription`). They also have a `getI18nBasename()` method. This combination makes it unclear whose responsibility it is to actually translate (i.e call MessagesManager) the piece of text. Should it the the FieldDefinition itself ? The FieldFactory ? The FormBuilder ? Something we hide in Vaadin (we could wrap `com.vaadin.ui.Component#setCaption`)

Would it help if we actually removed that stuff from our APIs and use an annotation to indicate which configured Strings need to be translated ? (e.g add an annotation on `info.magnolia.ui.form.field.definition.FieldDefinition#getLabel`the title, t

Different translation "scopes"

We need to differentiate between in-code translatable text, in-template translatable text and in-config translatable text.

In-code translations

Some UI elements are not configured. Their texts can be considered "hardcoded". IntelliJ (Eclipse too, probably) provides analysis tools to find out where i18n strings have been hardcoded. Attached is a sample report executed on a couple of modules: i18-analysis.zip (main, ui, activation, cache, imaging, workflow). Note that the html-exported report is not very useable, but the in-app report is much more practical: Screen Shot 2013-07-31 at 14.21.14.png

My initial thought was that for these, we could benefit from a tool like Localizer or GWT's i18n generator tool. Unfortunately, these tools generate code (interfaces and impl) based on the keys found in a message bundle file. They are well thought out, in that they provide the correct methods depending on parameters found in the keys, for example. (generate a `String getFileCount(int count)` based on a `file.count=There are {0} files` message, for example). This is great for code completion and type-safety. However, their generated code isn't ioc-friendly (tends to rely on static /threadlocal for Locale retrieval), and tends to be laced with static dependencies. These tools also don't handle changes to the generated code well. Which means that if you need to add a message - or change its name, or change its "signature" - you need to edit the properties file, rather than the code.

If we end up having a lot of these, or if we have some extra time🍔, we could think about rolling out our own tool - or write code that behaves similarly to that sort of generated code. Typically an interface with methods like `String getSomeKey()` and `String getFileCount(int fileCount)` and the tool would generate the implementation and the message bundle file.

In-code translations also include things like the login form and some other AdminCentral templated components, where the i18n call might currently be done in the FreeMarker template. `magnolia-ui-admincentral/src/main/resources/mgnl-resources/defaultLoginForm/login.html`: it's actually hardcoded.

The login form also has this special construct for translating `LoginException` messages - to be taken into account.

In-template translations

These are translations that are meant to be "consumed" by a site visitor; i.e used in template scripts. Some template components of STK which need a translated text item are not meant to be translated by authors; "Skip" or "Read more" type of labels.

Those currently suffer from the fact they are in the same message bundle (i18nBasename) as their respective template definition; as such, if one wants to change those text items for a given project, they have to either copy the complete STK message bundles, or customize the components. Neither is ideal. Currently, at least for FreeMarker templates, we pass `definition.getI18nBasename()` to `info.magnolia.freemarker.FreemarkerHelper#addDefaultData`, which adds an `i18n` object to the FM context with the definition's message bundle.

It should be possible for templates to use a different message bundle than the one used to translate the definitions' labels and descriptions.

Ultimately, we probably want authors to be able to translate these items; this is where a JCR-based translation tool (and/or MessageManager implementation) would make the most sense.

In-config translations

Translations in-configuration will require very little change to existing code, but will require some work on update tasks and bootstrap files. It is however where the bulk of this concept is focused - we're changing the way translation are applied quite drastically.

Status in 5.0

~~A complete code review is required to identify all places with a direct String output (such as button labels, column headers, etc.). Such places have to be replaced with a proper i18n mechanism (`MessagesManager.getWithDefault(key, defaultMsg)`, where the original String will be used as `defaultMsg`).~~

A complete manual code review is not necessary to discover where to apply i18n. A search for usages of the following (to be completed) should cover 95% of our bases.

- `info.magnolia.ui.form.field.definition.FieldDefinition#getLabel`
- `info.magnolia.ui.form.field.definition.FieldDefinition#getDescription`
- `info.magnolia.ui.form.definition.TabDefinition#getLabel`
- `com.vaadin.ui.Component#setCaption` (39 usages in the codebase I have)
- `info.magnolia.ui.dialog.FormDialogPresenterImpl#buildView`
- `...`

MessagesManager implementation

`info.magnolia.cms.i18n.MessagesManager` is the component that has been used to handled i18n up until Magnolia 4.5. It has its flaws, and could be rewritten.

- it tries to cover too many use-cases (either enforce passing a Locale, or never pass one)
- its responsibilities are not clear - it observes and holds some i18n config (but not all?) and at the same time provides translation support

- DefaultMessagesManager is still very tied to using property files.
- DefaultMessagesManager isn't cleanly decoupled from the system - it's still using content2bean "manually", etc; it's not a real "component".
- rething package name and/or class name (package mixes i18n for UI and for content, to start with...)
- It's not easy to test
- Some logic is burried in the `MessagesChain` class - which is where, for instance, we've been appending the ??? when a key wasn't found so far. Except that we've seen workarounds popup here and there to remove those, etc..
    - `grep -r '???' ./magnolia-core/src/main/java/info/magnolia/cms/i18n/`
- `info.magnolia.cms.i18n.MessagesUtil` proposes too many methods; as a result, we're completely inconsistently use those in many places in our code. Get rid of this.
- http://jira.magnolia-cms.com/issues/?jql=text%20~%20%22messagesmanager%22%20and%20resolution%3DUnresolved

Existing uses and inconsistencies

See #status.

Proposal

This concept and proposal focuses on in-configuration translations. Hopefully, the concepts can be applied to in-code translations. In-template translations will be taken into account (i.e facilitate the maintenance of the corresponding message files), but changing the mechanism they use might be considered a "next step".

Convention over configuration

We want to introduce a naming convention, both for basenames (i.e the location of the message files) and the key themselves. That convention already exists in a informal way; most items in STK for example have a fairly consistent key naming scheme.

To avoid a lot of redundant and verbose configuration, we could take this one step further and use the conventional name - if none is configured - to lookup a particular text. This will also help for cases where the dialog configuration is (very!) redundant (99% of dialog actions are "save" and "cancel" - these are currently configured for each and every dialog, making maintenance a nightmare)

With some clever fallback mechanisms, this could make translations easier and simpler.

Suggestions for a fallback chain are given below. Given a bundle, the keys are "tried" in order. The first value to be found is used.

If fields or other elements need to be empty, like today, the label has to be configured as "empty", or the corresponding key.

The "generated" key should reflect the real "location" of an element, not where it's inherited from. e.g the "save" action label of dialog foobar should first be looked up at `<dialog-name>.actions.save` before `actions.save`. Chains for inherited elements should however (ideally) also lookup the parent's key.
If performance becomes a problem (with all the fallbacks and the chained message bundles), we could introduce a simple caching mechanism.
As a counter example, here's what we've been doing so far, taken from STK: `dialogs.pages.faq.stkFAQHeader.tabMain.title.label:`

- The `dialogs` prefix is not relevant and noisy. It was historically introduced to separate those labels from the page templates and page components names and description. Indeed, we're likely to have a `stkFAQHeader.name` somewhere. Currently leaning towards using separate message bundles. Or have an non-mandatory prefix/suffix (i.e there's a chance the component's title/name needs the same label as its first tab ?)
- The `pages.faq` statement is arbitrary and is derived from the fact that the dialog in question happens to be configured under an arbitrary folder structure (`pages/faq/`)

Message bundles

`i18nBasename` is the property we use to look up a message bundle. This tells the system *where* the translation files are. It is called "basename" because i18n mechanisms typically append the locale to this and use the result as a path to a file (eg lookup `<mybasename>_de_CH.properties`, then `<mybasename>_de.properties`, then `<mybasename>.properties` - some even go as far as going up the path hierarchy (parent folders) of the basename until they find what they're looking for)

Up to Magnolia 4.5, the `i18nBasename` property was defined in a dialog definition (or further down). With 5.0, this exists in `DialogDefinition`, but also in one level below (in `FormDefinition`), and still exists in all elements below (`TabDefinition`, `FieldDefinition`, ...). The property is also present in `ActionDefinition` (members of `DialogDefinition`).

In 99% of the cases, the `i18nBasename` property set at dialog level should be enough. It is useful to keep the possibility to redefine it in actions, forms, tabs, and fields, but it should not be necessary. Defining `i18nBasename` at module level would be ideal - in terms of minimalizing redundancy anyway - but I'm not sure we'd have support for that right now. It'd be interesting to have i18nBasename in a module descriptor though. It would still be possible for individual components to override it if needed. We could also create a naming convention for i18nBasename as well (see below).

However, with the proposed key naming scheme (as well as with the existing informal one), message keys are distinct enough to consider dropping the need for specifying a message bundle. Proposal:

- We don't need to specifiy `i18nBasename` anymore for translatable items. (but we can, at the very least to maintain backwards compatibility)
- Every module will still have their own message bundle file(s); the system will chain and look for messages in all of these
    - We could imagine having a check that would warn, or even fail, when several bundles contains the same key(s).
    - However we still need to be able to override messages (for projects).
- Global chains of message bundles - look into all known bundles
- Basename helps grouping translation work - "I am now translating module X" - but that doesn't mean the basename has to be specified necessarily
- Order of message bundles chain would need to be consistent and predictable

Message-bundles-naming-convention

Instead of having a huge fat bundle within ui-admincentral, every app should have its own bundle. The naming of an app-specific bundle should have the following pattern:

app-<app-name>-messages_<locale>.properties (e.g. *app-security-messages_en.propertie*s, *app-contacts-messages_en.properties*, etc.).

It should be located directly under *<module>/src/main/resources/mgnl-i18n/* (e.g. *security-app/src/main/resources/mgnl-i18n/app-security-messages_en.properties*)

Date formats and other localized items

We should make sure things like ColumnFormatter not only use the current user's locale, but also that this is indeed an "enabled" locale. A UI entirely in english but with a date formatted in french would be silly.

Exceptions

Exceptions are exceptions. They should not be translated. The `message` of an exception is targeted at developers and admins, and it is expected that they understand english. (if only to understand the meaning of the exception class name!)

If/when exceptions are currently reported to the user, we need to actually **treat** the exception. The below shows how **not** to do this.

Incorrect example

```
try {
   ... something that fails ...
} catch (SomeException e) {
   uiComponent.alert(e.getMessage());
}
```

The below is what we should be aiming for (complete solution pending)

```
try {
   ... something that fails ...
} catch (SomeException e) {
   // if there is added value: log.error("sumfin happened: {}", e.getMessage(), e);
   // don't feel forced to bloat the logs, though.

   // if this is something the *user* should be informed about, and *can* do something about:
   uiComponent.alert(i18n.getMessage(e));

   // but really, is it ? Can a user do anything about a JCR Exception ?
   throw new RuntimeException(e); // did you need to catch it in the first place ?
}
```

Of course, the question is - what IS `i18n.getMessage(e)` ?

This is TBD. An idea is to have an extra method to decorate exceptions, like we suggest for non-configured texts in MAGNOLIA-5296 - Getting issue details... STATUS . A pattern we've used in the past (see login form) is to use the exception class simple name (`IOException` as opposed to `java.io.IOException`). I don't think we should try and use the `message` of the exception, because there is absolutely nothing that prevents anyone from injecting arbitrary text in there. However, some exceptions do have some sort of convention - a `PathNotFoundException`'s message is usage the path that was indeed not found. If possible, I'd favor using explicit properties of the exception rather than the message (`getPath()` if it existed, in this case).

But: this would spread the use of `I18nizer` much further than it should (remember that one of the initial goals was to completely hide from the developers and have the decoration done by node2bean or Guice). Perhaps it could be "hidden" in `uiComponent.handleException(e)`, and/or in a specific service. Need to think about the context, too (which would be lost if we use this last suggestion)

Implementation

Introducing a couple of concepts. The basic API will be in its own module in magnolia_main; it doesn't need to be in core, and will maybe not even depend on it. It could even be outside of magnolia_main if there is no dependency to core, but we currently lack a good location for such modules 🤷

Module and package

`magnolia-i18n` in `magnolia_main` contains the API and the underlying. Key generators live near their counterparts. (i.e in _ui mostly)

The main package is `info.magnolia.i18nsystem` as a package name.

API

- `@I18nable` annotation. "Internationalizable": marks any object as a candidate for translations. Used on interfaces/classes such as `FieldDefinition`. Is inherited.
- `@I18nText` annotation. Marks a String as to be translated.
- `I18nKeyGenerator<T>` interface. Implementations generate translation keys for <T>.
- `TranslationService`
- `I18nizer`. Transforms/decorate a given object and internationalizes it.
- `I18nParentable`. This is an interface that allows setting a "parent" on an object. This interface should not be used by client code.

Implementation "details"

- Tried to implement a Guice module that would do this transparently; also tried to implement this at Node2Bean level. Neither worked - didn't find a way to "swap" instances in Guice (i.e replace bean created by n2b by a proxy)
- I18nizer thus needs to be invoked explicitely. We have one implementation based on ProxyToys which creates a proxy. It intercepts methods that return other `@I18nable` objects (by returning one instance, a collection, or a map where value are of an annotated type) and decorate those. Those proxies in turn intercept method calls annotated with `@I18nText` and returns translated values by delegating to `i18nKeyGenerator` and `TranslatorService`.
- `I18nizer` also "injects" the `I18nParentable` interface and sets the parent object while intercepting calls.

Update tools and tasks

To migrate our own modules, we can write a tool which:

- Starts up a repo and a specific (or several) component managers
- Load up a translation file we want to migrate (or all its language counter parts)
- Imports a bootstrap file we want to migrate
- This should instantiate a whole bunch of forms etc
- Go through these one by one
    - Go through each `@I18nText` property of the object
        - Does it correspond to a key currently existing in the translation file ?
            - yes: replace key in translation file by deduced key, remove property from jcr
            - no: add key in translation file
        - If it's a key but it doesn't have an existing translation
            - add deduced key to translation file (instead of configured key), remove property from jcr
        - If it's not a key, i.e current configuration has an "hardcoded" text
            - warn, blow up, panic, ...?
            - add deduced key to translation file (instead of configured key), remove property from jcr
    - Track unused keys in translation files
    - Track possible duplicates
- Re-export file to replace bootstrap file

- Re-export translation file(s)

The same tool could maybe be used to generate update tasks, or generate some sort of config/mapping file passed to a specific update task: it will only need a list of properties to be removed from nodes, with their original value so that we don't remove a property that's been modified by a user.

Message bundles for non-english texts

Bundle languages in separate jars

- english/master language still bundled with each module, other languages bundled in 1 jar (1 jar per language, containing translations for many modules)
  - on git, this would be, for example `languages/french.git` - which would contain N files.
  - These language bundles could be a module. They could thus be versioned, and have dependencies. Dependencies to module they translate would be marked as optional, but we would be able to maintain some sort of version-compatilbility between modules and translations - granted, that wouldn't be super simple to manage - where, for example, we'd update a dependency when keys have been added/removed from that dependency.
- maintenance is somewhat easier
- but at the same time we might get "dependency" issues when modules add/remove keys
- if we have tools for migration/validation of existing translations, the same tools could be used, perhaps as a maven plugin or sthg.
  - such a tool could potentially help enforcing compatibility between versions (i.e keep a key that was removed in version X+1 of module M)
- Need some sort of version handling - keep keys for older versions, add keys for newer ones ...
  - Chain (overlay) current translation file with older ones ?

Translation processes

- Enable inline translations within a dialog
  - I'm not sure the current proposal would work to enable inline translations. It'd be nice for translators to have at least a hint of what the key used for a specific item is. And if we somehow have elements explicitly use the KeyGenerator and other API methods for this, we might as well get rid of the proxy magic and use consistently...
- Review process for in-house maintained translations as well as for contributed ones (currently relying on Google spreadsheet)
  - Have a Magnolia-hosted (or SaaS) tool to replace the google spreadsheet
    - some rules like "a translation needs to be validated by 2 other persons to be applied", "once applied it can't be changed directly - only via a 'request'", ...
    - could have a "MessagesManager" impl that fetches translations from this service
    - candidates so far: Transifex (am in contact with CEO Dimitris Glezos, who seems eager to help), Crowdin, WebTranslateIt, ... probably others.

Roadmap

- 5.1 : system in place, translations migrated for AdminCentral and most modules (DAM, STK, ...)
  - Module updates can be postponed
- ? : extract languages other than english into language-based bundles - needs a separate concept.
- ? : review processes for translating Magnolia, both internally and externally. Get contributions. With language files extracted from their modules, it might be easier. - needs a separate concept.

Status

Main issues:

- MAGNOLIA-5268 - Getting issue details... STATUS
- MGNLUI-1826 - Getting issue details... STATUS

Validate Concepts

Validate Roadmap

API

Finalize module name, location, and package name.

Implementation details (proxy, Guice module, ...)

✅ Parental relationships

❌ How does this work with Multi/composite fields

❌ Check things like `info.magnolia.ui.form.field.definition.DateFieldDefinition#getDateFormat` - might actually work with `@I18nText` or another annotation

❌ Is there a conflict with merged objects ? (i.e template definitions merged with site def prototype - but also look at other merge cases!!)

❌ An i18Basename could also be defined in site definitions (for in-template translations) - does this work ?

❌ Clarify what to be with i18nBasename properties - the current tendency is to get rid of them. Adapt update tools as needed.

❌ Replace MessagesManager with a cleaner impl

❌ Migration and updates

❌ While this implementation has focused on definition objects, (some) "live" objects already know their parent (see for example `info.magnolia.ui.form.FormItem`) - validate and justify the choice to decorate definition objects.

Known bugs or problems:

- MAGNOLIA-5315 - Getting issue details... STATUS
- MAGNOLIA-5317 - Getting issue details... STATUS
- we have no proper way, yet to i18n-ize GWT- / Vaadin-Client-Classes; neither info.magnolia.cms.i18n.MessagesUtil nore the i18n-toolz can be applied on those client-classes

Topics to validate or research

- A form can be used in different contexts. It should be translatable according to context.
  - TBD: details, examples

Existing uses and inconsistencies to fix for 5.1

- ❌ `info.magnolia.ui.form.AbstractFormItem#getMessages` - should not be public
  - Is broken: doesn't use the user locale afaict.
- ❌ `info.magnolia.ui.form.AbstractFormItem#getMessage` - should not be public
- ❌ `AbstractFormItem` defines a semi-arbitrary message bundles chain (see `AbstractFormItem#UI_BASENAMES`)
- ❌ Definition objects(`TabDefinition`, etc) have an `i18nBasename` property, which is very redundant with that of the "runtimes" objects (`FormTab`, ...). Usage seems consistent (`return definition.getI18nBasename();`) but I don't know why this isn't implemented in `info.magnolia.ui.form.AbstractFormItem`.
- ❌ Definition objects don't have a common interface. If they did, we could move i18nBasename and label in there. OTOH, some of these objects have more than 1 item to translate (label and description, for example).
- ❌ `view.addFormSection(tab.getMessage(tabDefinition.getLabel()), tab.getContainer());` ...translates the message from the tab and passes it translated before it's actually "displayed". (while the method argument is called `tabName` not `tabLabel` - but that passed object becomes an argument called `caption` later down the stack) - the below would make this sort of code much more explicit. You pass an object meant to be a label. You translate it explicitly - most likely at the last possible moment. Or we even extend Vaadin component so that they know about `I18nItem`.
- ❌ `getI18nBasename` is defined in too many places. It's inconsistent and unintuitive. Why the redundancy between `info.magnolia.ui.dialog.Dialog#getI18nBasename` and `info.magnolia.ui.dialog.definition.DialogDefinition#getI18nBasename` for example ?
- ❌ `info.magnolia.ui.form.FormBuilder#buildForm` still does "StringUtils.isNotBlank(description)", but since this is now decorated, it will never be blank or null. See MAGNOLIA-5315 - Getting issue details... STATUS though.

Suggested key patterns

The below is a rough outline. The 3 goals below are somewhat hard to reach all at once.

- avoid redundancy or noise (no `dialog.` prefix, ...)
- avoid conflicts (but allow them - on purpose - for labels that are actually meant to be the same in most situations)
- be consistent (this part is hard - sometimes we prefix with module name, sometimes with app, sometimes with nothing)

Note: as the dialog names usually follow the pattern `moduleName:dialogNameWithinTheModule`, the `<dialog-name>` part of the keys mentioned bellow is in fact `<module-name>.<dialog-name-within-the-module>` (as the ':' character cannot be part of a key).

Field labels: – optional fallback to a key without a `.label` suffix to make things less verbose

`<dialog-name>.<tab-name>.<field-name>.label`

`<dialog-name>.<tab-name>.<field-name>`

`<dialog-name>.<field-name>.label`

`<dialog-name>.<field-name>`

Field descriptions - here we can't fallback to a key without the `.desc` suffix

`<dialog-name>.<tab-name>.<field-name>.desc`

`<dialog-name>.<field-name>.desc`

Tab labels:

`<dialog-name>.<tab-name>.label (or .tablabel for explicitness?)`

Dialog (Form) labels:

`<dialog-name>.label`

`<dialog-name>`

Action labels:

⚠️ 99% of our dialogs have the same save/cancel actions. Those should be defaults. Labels should still be overridable on a dialog-per-dialog basis.

`<dialog-name>.actions.<action-name>.label`

`<dialog-name>.actions.<action-name>`

I introduced the `.actions` portion here to avoid confusion with fields; consistency would dictate having a `.fields` or `.tabs` portion for field and tabs labels too, but that would downplay the conciseness.

for all dialog-related items, we could also use `<dialog-id>` and fallback to `dialog-name`, for further specializing. ⚠️ Dialog ID is possibly currently not available; if it is, it's a single string concatenating module name and dialog name, which isn't ideal. It'd be sweet to be able to get back to module id (and app id) from a dialog.

Apps:

`<app-id>.label`

`<app-id>.icon # because the icon might have to be localized`

`<app-id>.description # because a mouse-over title of the app might be interesting ?`

`<app-id>` could be `<module-id>.<app-name>` or just `app-name` (same as for dialogs)

App launcher groups:

```
app-launcher.<app-group-name>.label # we use the app-launcher prefix, as if app-launcher was an app (which we should consider considering, I suppose)
```

Templates:

```
<module-name>.<template-name>.title # I think "title" is what we've been using in 4.x - we could use label for consistency, or simply name
```

`<module-name>.<template-name> # same as above`

`<module-name>.<template-name>.desc # Useful in template selector`

`<module-name>.<template-name>` is essentially the component ID.

Page components:

`<module-name>.<component-name>.title # see remark above`

`<module-name>.<component-name>`

`<module-name>.<component-name>.desc`

`<module-name>.<component-name>` is essentially the component ID.

Workbench columns:

`<app-id>.<sub-app-name>.views.<view-name>.<column-name>`

Example: `configuration.browser.views.list.name` for `/modules/ui-admincentral/apps/configuration/subApps/browser/workbench/contentViews/list/columns/name/label`

Actionbar in subapps:

`<app-name>.<sub-app-name>.actions.<action-name>.<name of getter or field annotated with`info.magnolia.i18n.I18nText`>`
Example: `contacts.browser.actions.addContact[.label]` for `/modules/contacts/apps/contacts/subApps/browser/actions/addContact/label`

Actionbar in choose dialogs:

`<app-name>.chooseDialog.actions.<action-name>.<name of getter or field annotated with`info.magnolia.i18n.I18nText`>`
Example: `assets.chooseDialog.actions.upload[.label]` for `/modules/dam/apps/assets/chooseDialog/actions/upload/label`

Other elements tbd ?

- actions in dialog
- ~~actionbar in subapps~~
- workbench/<view>/columns in content apps: column names
- in workbench/view/columns, we also have `formatterClass` which should be locale-sensitive
- app (label in app launcher, tab)
- page templates
- page components

Suggested i18nBasename patterns

- Defined in module descriptor
- `/mgnl-i18n/<module-name>/messages`

Dismissed proposals

Here are a couple of notes about things I tried and rejected

Change `info.magnolia.ui.form.field.definition.FieldDefinition#getLabel`. Node2bean should be made aware of `I18nItem` class (by registering a transformer)

change this

```
String getLabel();
```

into this

into this

```
 I18nItem getLabel();
```

- N2B would instanciate these
- it should be made aware of its "parent" (FieldDefinition etc)
- via injection, the impl would know what locale to use and delegate to MessagesManager (or its replacement)
- we'd have a default null-pattern type of implementation (which returns the key or an empty string, or sthg else - this could even be swapped depending on dev mode etc).
- Would this impact performance/memory usage ?
- Quick prototype of this attached - doesn't do anything, and doesn't know about Locales or message bundles - just showing the "structural" change on classes like FieldDefinition (patch file - disregard class names and packages !)

Relying on N2B will not work

- can't really get the parent into the object (children objects are instanciated first) - not without modifying usage code (ie in setLabel() { label.setParent(this) }, which would suck)
- can't really transform single properties other than with beanutils, which has even less notion of context, and whose usage is currently hardcoded in core

Duplicate keys

Ideas of supporting things like `some.key=${some.other.key}` - we have a lot of such redundant messages in english.

- mention of the CZ issue where in english a translation would be the same in 10 place but needs to be adapted in czech.
    - somewhat moderate issue if the mechanism is only supported with the same file - but why would it be
- sorta conflict with the deduction of keys (you would use the most "low" common key for all those same translations)
    - makes the CZ problem perhaps more difficult, since the translator then can't rely on keys being defined in the english file