KRAD - Validation based on State

Purpose

KRAD should be able to support multiple states and to vary the validation content based on the document state (beyond the 2 states of incomplete/draft versus move-to-next-state). This would enable applications that want to validate the form's content based on multiple states, to provide the user with information both on what is still needed for the immediate submission as well as what will additionally be required at a later stage in the document's workflow process.

Today's work-life is full of interruptions and multi-tasking, and software should support the pressures users experience in moving back and forth between tasks (starting back up where they left off, re-finding their place, not losing any data they've previously entered, and so on). Users should be able to temporarily stop a task and come back to it without re-entering data and fields they've already completed, even in a web-form that is much longer. For this reason, KRAD should support temporarily saving an incomplete form so that the user can come back to the task later to complete it.

Detailed Description

(In technical analysis phase will consider the alternatives for how/what to achieve, e.g., store in temp/local cache with no change to form in database, or store user-entered values in a plug-in, to re-apply the values when user brings back up the form? work tbd)

- The state can be based on BO states (e.g. Kuali Student tracks this for objects), or it could also apply to the document's status in the workflow framework (i.e. KEW).
- Should be capable of defining these states in multiple ways (e.g., in xml, in data dictionary, etc.).

See KAI review 4-12-2012.

Usage Scenarios

Usage Scenario One

A faculty member is creating a new course and enters all the information he knows are required in order to submit the course for department approval. But he also wants to check the form before submitting it to make sure he's got all that is required for that next step as well as find out if there are other requirements his department admin will have to make sure are met before submitting it for college approval. He can select to check the form, in which case the validation checks for all future states. The validation provides error messages for items that are required for this next stage, as well as warning messages for items that will be required for other future stages in the workflow process.

Usage Scenario Two

A few years later the same faculty member is modifying the course. He selects the course to modify and enters the changed description and other required information. He can do all the steps described in usage scenario one (check form, etc.).

Usage Scenario Three

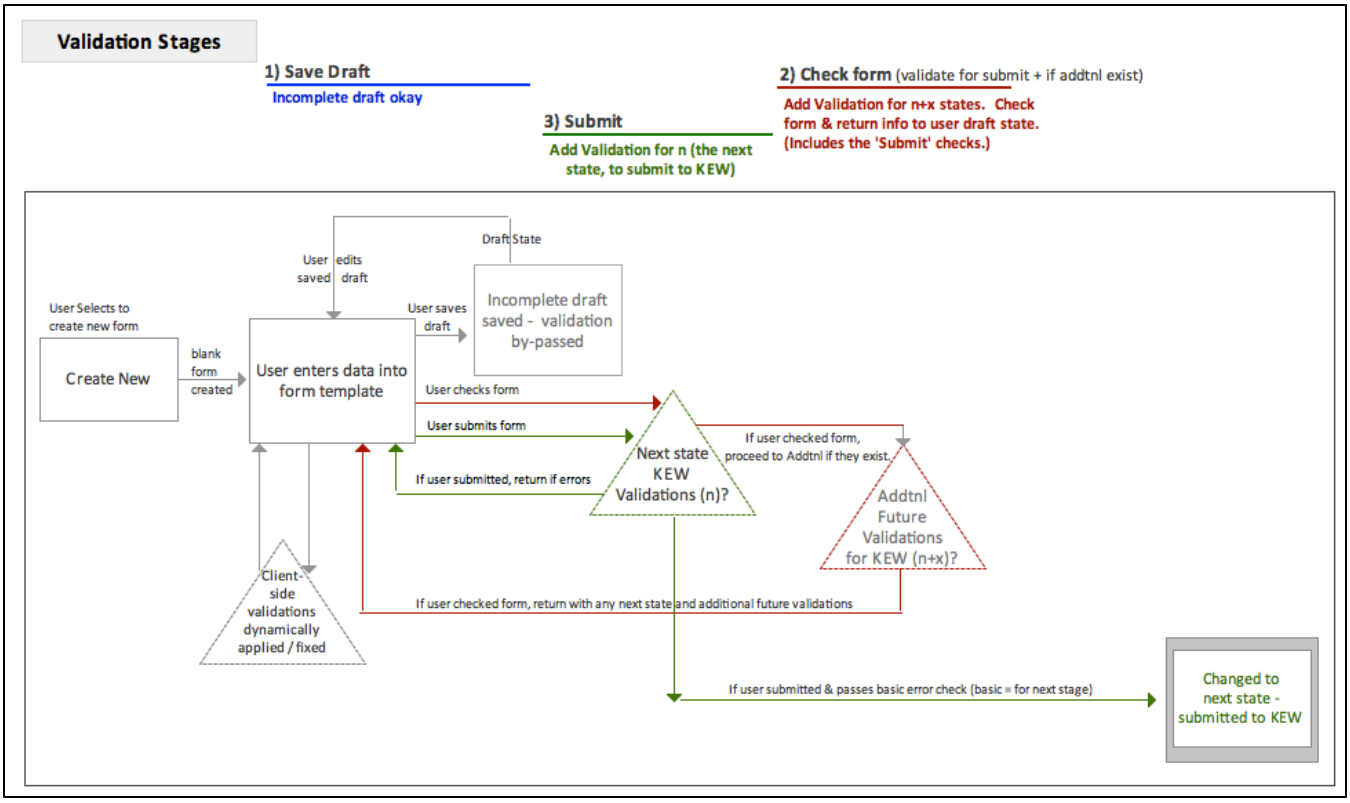
A few years later the department administrator needs to retire the course (they will no longer offer it). He selects the course to retire it, enters the required retirement reasons and dates requested. He can do all the steps described in usage scenario one (check form, etc.).

Usage Scenario Four

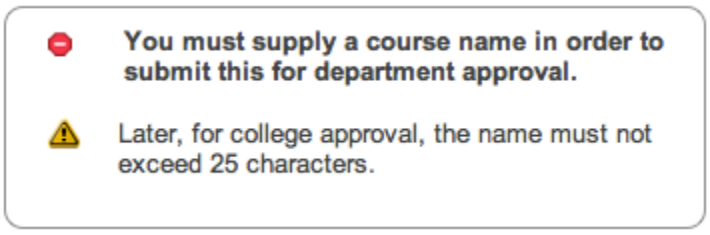
A user has spent 10 minutes filling out a web form and gets interrupted to attend a meeting that wasn't on her calendar. She has completed 20 fields out of 40 total fields. She wants to temporarily save the form "as-is" because she has to go to a meeting and doesn't have time to complete the form now, but doesn't want to lose what she's already done. She can save this to draft, which suspends all validation of incomplete/blank fields and enables temporarily saving even if there are errors on some fields (to fix later, when re-editing the draft).

Mocks and Diagrams

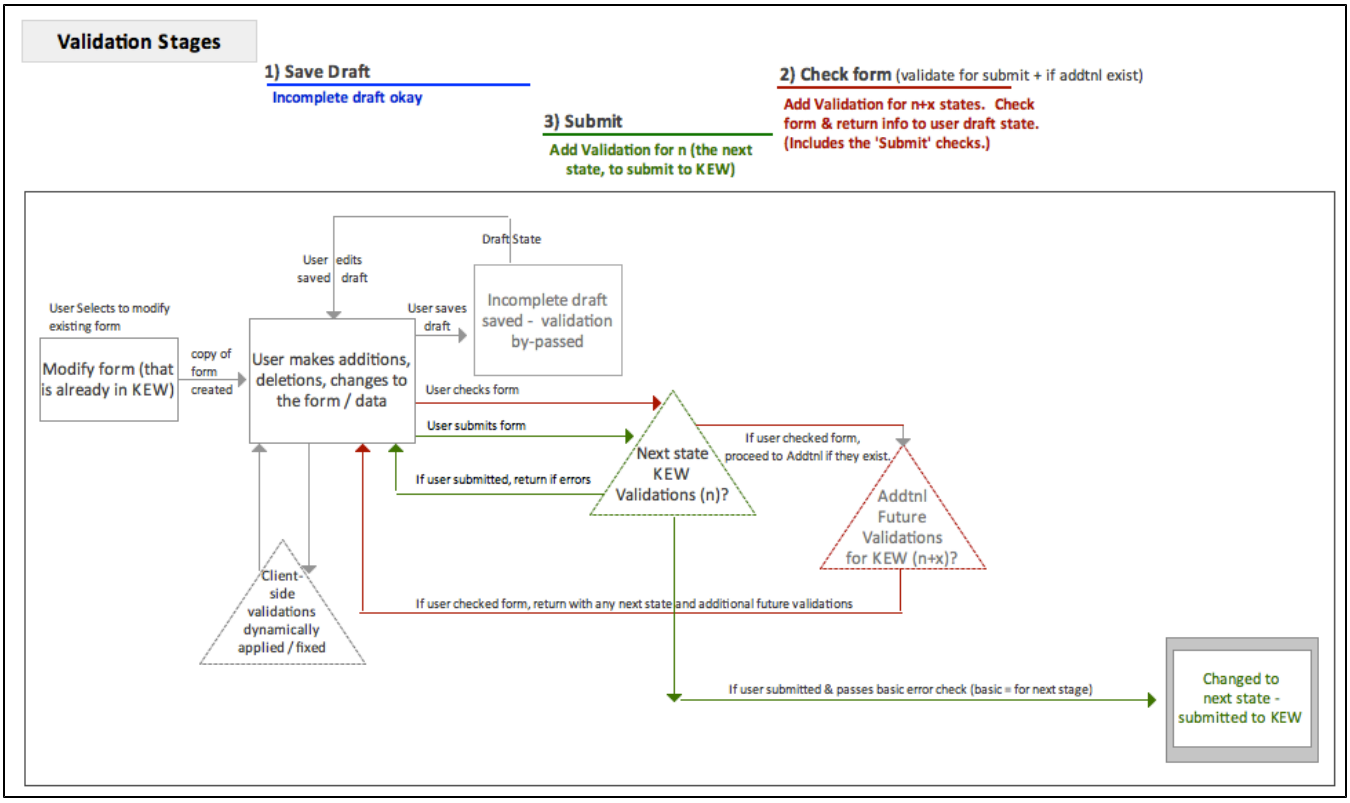
Usage scenario two flow diagram:



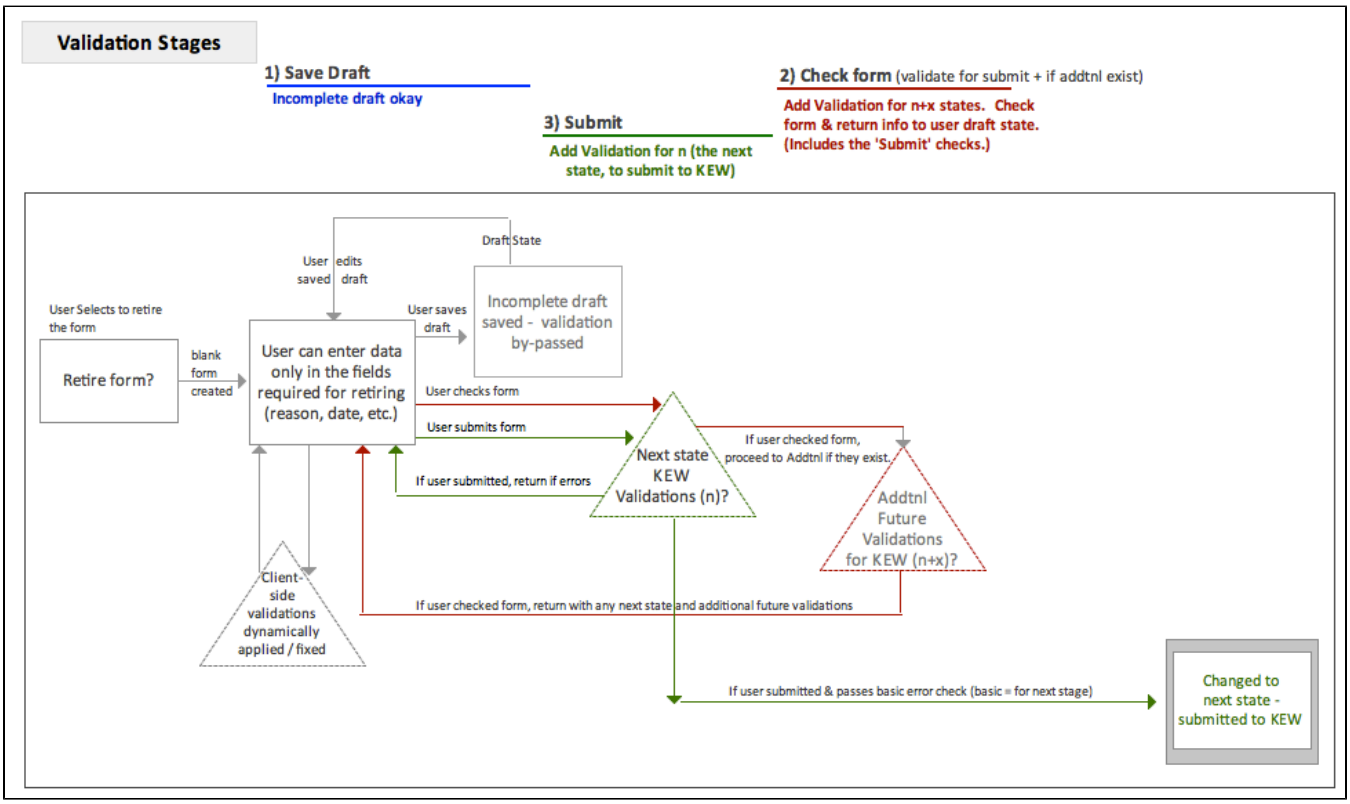
Example validation messaging returned to the user on checking the form:



Usage scenario three: Below is the same task flow for modifying an existing course (one that is already offered).



Usage scenario four: Below is the task flow for retiring a course.



Performance

If applicable, list expectations for performance (optimal and worst cases would be fine, give time in seconds):

References

- Kuali Student's State-based validation summary
- KFS requirements come-back to the KAI.

Requirements Listing

List all requirements (individual verifiable statements) that indicate whether the work for this item has been complete. If there are requirements that are not essential to the functionality but would be nice to have if time allows, enter those under 'Non-Essential':

Primary:

1. KRAD should enable applications to define multiple validation states associated with a business object, in multiple ways. Aspects discussed by the applications included:
    a. based on the business object's state, defined in an attribute of the object
    b. defined in the data dictionary, defined in xml
    c. defined by the object's location in a workflow process (e.g., in KEW)
    For example, there could be a new/created state, a submitted state, a department approved state, a college approved state, a university approved state, a retired/archived state.
2. KRAD should enable application logic to optionally check a form for the requirements for multiple validation states rather than a single/next state (if the validations for future states exist, defined in the multiple ways discussed above). (This check would not be the default validation but would be invoked if/when the user selects to check against future states -- see item #4 below.) For example, the validations required for the immediate next step (approval routing to the next step), as well as future validation states (for example, what is needed for proposal/review state, for approved-by-department state, for approved-by-college state, for approved-by-university state, etc.) - could be done in one UI step. (This is the underlying architecture to support #4).
3. KRAD's validation framework, by default, should define the n+ messages as warnings and prefix the messages with the state information, for example," For <State_name> or <KEW_state>: "Name cannot be more than 25 characters".
    a. The validation code logic will check for uniqueness of messages between states and only display the next state ones that are unique (so no repetition in the display).
4. KRAD should enable users to optionally choose to check a form for requirements for future (n+ states, defined by the application in the multiple ways discussed above), assuming the application has supplied the validations for multiple states (#2). For example, applications can provide a "check form" action in addition to a "submit" action, with the underlying architecture applying just the next 'happy' state's validations on submit, while looking for and applying future states' validation as well, if found, on check form.

If the secondary requirements below are implemented, then following are additional primary requirements:

1. KRAD should be able to display two different requiredness indication legends (definitions) in the UI to convey the requiredness for save differs from the requiredness for the next state (for example, for submit). And KRAD should be able to display the appropriate visual designator with the field, if it is required, to convey its requiredness applies to which state. *Note: it is assumed that the application will define fewer required-for-save constraints than for the next state (e.g., for submit). Users want to be able to save a form even when it has errors, so that they can come back to fix them later and not lose what they've already entered.*
    a. Visual details tbd, but default for now is "**=required for Save, ***=required for <next state>", where "next state" is a variable defined by the application, for example "Submit".
2. Client-side validation will include for both save and next state (i.e., submit), but will prefix the messages with the state for which they are needed. Example: Field 2: For Save, <error message>. Field 2: For <next state>, <error message>. It will include only these 2 states (no other future states). *The save errors are expected to be fewer/less restrictive than the submit errors (this will be good application design, up to the application), and users can save even if there are submit errors, as long as the client-side save errors are corrected.*
3. The validation code logic will check for uniqueness of messages between the Save and the next state and only display the next state ones that are unique (so no repetition in the display).

Secondary:

1. Users should be able to save an incomplete form (a form with required fields that are blank), to edit later and complete. (One technical design aspect suggested by the applications but not yet assessed is to suspend client-side and server-side validation when users choose to save this type of incomplete form to a draft state.)
2. Users should be able to save an incomplete form (a form that contains validation errors, warnings, or informational messages on fields already filled in), to edit later and complete. (One technical design aspect that has been suggested by the applications but not yet assessed is to store the user choices/entries in a plug-in, to reapply those choices the next time the user opens this form -- which the server still sees as a new blank form, under the covers).

Dependencies

List any functional or technical work that must be completed before work on this item can begin:

1. Technical analysis / feasibility assessment of the above. Dev/UX discussion/transition work on the alternatives.

Issues

List any issues that need to be resolved before work on this item can begin:

Functional:

1. item
2. item

Technical:

1. item
2. item

QA or Regression Testing Plan

List steps needed to test the basic functionality of this update, enhancement, bug fix

1. test/steps
2. test/steps

Checkoff

Functional Analysis Complete? **No (completed by SME)**

Needs Review by KAI? **No (completed by SME)**

Technical Analysis Complete? **No (completed by DM)**

Needs Review by KTI? **No (completed by DM)**

Estimate: **30 hours (completed by DM)**

Technical Design: Link Here **(completed by DM)**

Jira: KULRICE-6678

Final Documentation: Link Here **(completed by DM),**