Branching

Branching Activity

This is technical documentation for Branching. Click here for User Documentation.

This page describes how branching works, including describing how tool outputs interact with branching to do tool based branching. If you are a tool developer interested in a how-to guide for creating tool outputs then see Tool Outputs For Tool Developers.

Introduction - Background to Branching

Branching is an workflow system activity that allows a workflow designer to create different path ways (branches) for users.

Type of "split"

From more theoretical terms, a branching can be "AND", "OR" or "XOR" split. LAMS Parallel Activity is an example of "AND" split. Selected branches (two or more) MUST be followed at the same time by the execution/progress of the same user. "OR" split is often used when a user can choose how many branches to follow, when, how and in what order. E.g. the user may follow 2 branches out of 5 at will at the same time. Or, the selected branches may not be followed at the same time. The definition of "OR" branching is not clear in many systems. "XOR" branching is much simpler since there is only one clear execution transition to follow and it is more commonly used. "XOR" branching can be of user-selected but it is often associated with certain automatic conditional calculation when deciding which branch the user is allowed to proceed.

In short, the LAMS "Branching" discussed here should be more appropriately called as **Conditional Branching** (or Conditional XOR Branching).

Merge-point (or "join")

"AND" branching is often associated with a synchronizing merge-point later, which is necessary to make the split execution whole again. In case of "XOR" split, usually there is no need for synchronizing merge-point since there is only one execution/transition point moving along as the user proceeds. A simple merge-point is satisfactory. "OR" split may or may not employ a synchronizing merge-point later.

Strictly speaking, LAMS Activity has not been intended to accept multiple incoming arrows. This is particularly true in case of the Flash GUI. Therefore, probably it requires a merge-point which can accept multiple arrows coming in or a modification is required for any Activity to accept multiple arrows (maybe only if they are coming from Branches). From Java side point of view, however, nothing is explicitly stopping an Activity from accepting multiple incoming arrows.

For clarity, it seems it is more fitting if multiple arrows are entering only into a merge-point, i.e. clearly separating merge-points and Activities. It may be favored particularly because it makes it clear when and where the merge occurs and how, i.e. it occurs after and before Activities and the "state of transition" at the time of merging is not on an Activity but at the merge-point. This is also more suitable if we are to have synchronized merge-point later or we convert the current synchronization Gate Activity to a merge-point later.

As discussed later, if we implement the Branching Activity as a Complex Activity (e.g. as a "box"), the merge-point is not required for it. If we implement branching or loop as boxes, the boxes represent those functionalities and there will not be multiple arrows coming in or going out to/from boxes. Hence, there is not visible split or merge points depicted on the design diagram.

From mathematical point of view, YAWL (Yet Another Workflow Language) emphasizes that the validity of a workflow design cannot be verified if the design does not have a single "end" terminal. For this reason, even if branches in some workflow system may look like having separate "ends", it may be assumed that all the branches are merged and connected to a single "end" terminal implicitly.

Parallel and Options Activities

In LAMS, there already exist implicit split and join Activities, namely Options Activity and Parallel Activity. They are implemented as boxes.

The Parallel Activity is an AND branching (split and join) implicitly expressed as a box. There is only one each of in and out arrows to and from the Activity. In Options Activity, it has an implicit OR split at the beginning of the box. At the end of the box, there are merge and conditional branching which decides whether the user should loop back or go out.

Loop

Loop is another concept which is closely related with branching. Where there is branching and merge is allowed, there is a possibility of loop (with merge).

The Options Activity is so-called Repeat-Until Loop Box in more general terms. If the condition must be judged before the loop begins, we can think of While-Do Loop box. Some language also has For Loop box. In Loop boxes, the complete condition is unclear for LAMS where each

individual may be regarded as "completed" of the Loop in some cases or the whole "group" may have to complete to satisfy the "complete" condition of the Loop in other examples. If this is a concern, we may have to allow the Loop boxes to decide which views will take effect when judging the "complete" conditions.

Branching Types

Branching provides the flexibility Authors need to create different branches according to different criteria (results or groups), learning styles, user preferences, etc.

LAMS will introduce branching based on four criteria:

- User choice or
- Teacher choice or
- Groups (based on a previous grouping activity) or
- Output from a previous Tool activity

Learner choice

Author may provide branches for each Learner to choose depending on user preferences or instructions are given.

Teacher choice

If the Teacher wants to manually allocate the Learners to branches, then when the Learners reach the branching activity the Teacher (from Monitor) can direct the Learners to separate branches. This requires a notification mechanism and a user-stop gate (this allows each Learner to pass once a Branch for the Learner is assigned), or a synchronization gate for the whole class before the Branching activity.

Group-based branching

If a branching activity is based on a previous grouping activity, then the Author can determine which groups should go in which branch. So when learners from a particular group reach this branching activity they get sent to the corresponding branch according to the group they belong to.

Tool output based branching

If the branching is based on the output of a previous activity, then the Author can specify a particular output from the activity that is used for deciding the branching. The output can be (1) statistical values (e.g. the total or average mark , number of postings, etc.) for the all users of the activity, (2) individual values of the user (final score for this user, or number of postings for this user, etc.), (3) particular statistics on particular item (statistics for question 13, how many people chose answer B, for example), (4) particular value for particular user (e.g. the answer given by the user for item 15, or whether the answer was correct or not).

For example, say the teacher wants to use the content for a Multiple Choice activity to create a set of branches:

- **Question 1**: What's the capital of Australia?
    - Candidate Answer 1: Sydney
    - Candidate Answer 2: Melbourne
    - Candidate Answer 3: Wagga Wagga
    - Candidate Answer 4: Canberra *(correct answer)*
- **Question 2**: What's the capital of Argentina?
    - Candidate Answer 1: Buenos Aires *(correct answer)*
    - Candidate Answer 2: Tucuman
    - Candidate Answer 3: Mendoza
    - Candidate Answer 4: Cordoba
- **Question 3**: What's the capital of Bhutan?
    - Candidate Answer 1: Thimphu *(correct answer)*
    - Candidate Answer 2: Timbuktu
    - Candidate Answer 3: Las Vegas
    - Candidate Answer 4: Kathmandu

In theory, the Multiple Choice may provide many things. However, it may provide:

- Number of correct answers *(integer)* per user
- Average correct answers *(integer)* as a whole
- Answers (array) per user
- Answers-correct *(array)* per user

The teacher being able set the branch on statistic such as the *number of correct answers*, in which case she can create a branch for learners who

have answered the three questions correctly; another one for learners who have two questions answered correctly and finally another branch for those who have got one or less answers correctly. In this case, the teacher has used a general score to create branches, but he can also specify branches on particular questions.

LAMS 2.1 Restrictions

In LAMS 2.1, we will only allow branching based on a single activity. In future versions, we may do "combinations" of activities with branching able to do boolean conditional calculations, such as answered blah in one activity (a condition) and a quiz score of 3.

The 2.1 release will only support branching based on the multiple choice tool, for simple statistics such as number of correct answers. We will not be able to set up branches based on an answer for a particular question. In later versions we will implement a form of custom condition, which will support being up to select branches based on which questions were answered correctly.

In the future we may wish to add some simple statistics and comparisons between values in the branching logic itself, such as taking an average of values but that will be left until we have particular use cases. If we have instances of people wanting very complex conditional calculation, particularly working over the complex outputs of various activities, then a separate conditional engine (either as a separate tool or service) will be investigated.

Implementation

Branching will be implemented using system tools. It will require changes to the authoring, monitoring and learning modules (including the progress engine). This requires code in Flash and Java.

The learner choice will be implemented using the existing Optional Activity system tool. This will be extended to support putting Sequential Activities inside an Optional Activity. Each "branch" will be one sequential activity and at runtime the learner can select which sequential activity to do. If the author sets a maximum of one sequential activity that may be done then the learner is restricted to one branch. If the author allows multiple sequential activities to be done, then the learner can do multiple branches.

The other three types of branching will be implemented using a new Branching Activity. This will be a complex activity like Optional and Parallel Activities. It will contain a set of Sequential Activities, ordered by the orderID field. Each "branch" within the Branching Activity will be a Sequential Activity and will be like a mini learning design - it will have a series of activities joined by transitions. Within the series of activities within the sequence activity the author can select optional or parallel activities, gates and grouping as well as tool activities.

The "first" activity within the sequence activity may be determined (by the progress engine) in two different ways:

1. If there is a transition running from the sequence activity to an activity A in the sequence activity then this activity A is the first activity in the sequence activity. This will work well for branching.
2. If the previous case doesn't find a first activity, then the first activity is the contained activity with no input transition and the lowest order id. This will work if sequence activities are put within a transition based series of activities (and hence the sequence activity has a real output transition). This case isn't required at present but it may be required in the future.

The Branching Activity will be linked to a Grouping Activity, which is used to manage which branch a learner is using. So a learner will normally only be able to do one branch, as a learner is normally only in one group of a grouping activity. If the branching activity is teacher chosen, then it will be linked to a Chosen Grouping. If it is a grouping based branching, it could be Random or Chosen, depending on the GroupingActivity selected by the author.

The Author can create a new Branching Activity by drag & drop the Branching icon. Initially it has one (or zero?) branch. The author will then double click on the branching icon on the canvas to expand the work area. In the expanded work area, the author will be able to add more branches.

The properties area will specify the type of branching (Teacher Choice, Group-based or Tool Output-based). Depending on which type is selected, the author may set up how the branches are to be selected.

All the branches will be "contained" within the Branching or Optional activity, so no merge point is required.

The individual "branches" will need to be named. This could be the name of the sequential activity. This is required for the teacher to be able to select which group goes to which activity in monitoring.

Learner Choice:

The learner choice branching will require only minimal changes to implement - the existing Optional Activity will have to be extended to include Sequential Activities. The controls for the minimum and maximum number of branches already exist.

Teacher Choice:

The author can set up the branches (and name them). The teacher cannot set up the "who does what" in authoring as the selection is done learner by learner and the learners are unknown.

Group-based:

The author can select a Grouping Activity that is already in the design. Note: the same grouping cannot be reused within a branch as it would be futile - all the learners would be in one group!

This could be implemented as a drop down menu with the available grouping in the canvas. Another ideas was to connect a special arrow between a Grouping Activity and the Branching Activity. The number of Groups and the number of branches can be different.

The normal grouping field in the activity object can be used for recording which grouping applies to this branching activity.

If we can determine the maximum number of groups from the Grouping Activity (i.e. it is set to Chosen, or Random with max groups) then authoring should show an assignment table and ask the Author to assign groups to branches. If the author hasn't entered names for the Groups, then the author will need to enter names for the groups. Note: There doesn't have to be a 1-1 assignment between groups and branches.

For examples, if there are 5 Branches but only 3 Groups.

- Group [1] uses Branch 1
- Group [] uses Branch 2 (not used)
- Group [2] uses Branch 3
- Group [3] uses Branch 4
- Group [] uses Branch 5 (not used)

Or, for example, if there are 5 Groups but only 3 Branches.

- Group [1,2] uses Branch 1
- Group [3] uses Branch 2
- Group [4,5] uses Branch 3

If the number of Groups and the Branches are the same, the assignment table should still be shown.

- Group [1,2] uses Branch 1
- Group [3] uses Branch 2
- Group [] uses Branch 3 (not used)

Tool Output-based:

The author will need to connect one of the upstream Activities to this Branching. (It should not be involved in this Branching - e.g. from any of the downstream activities). After that the author will need to link the outputs of the activity to the branches.

Monitoring

What we need for the Monitor and the Progress Bar? Probably we need to show all Branches in the Monitor and the Teacher needs to click to go to each branch? How the users with different branching aligns? In case of the user Progress bar, we don't have to align different users but will we show only the branch the user has taken?

In Monitoring, there will be a new "To Do" screen for the branching activity. This will be used by the staff to select which learners (individually or in groups) go to which branch. While some of the set up will be available in Authoring, some must be done at Runtime. This screen will always be made available (even if the branching is configured in authoring) so that the staff can change it to suit the needs of the running lesson.

Learner Choice:

Nothing is required for this option as the learners select it for themselves.

Teacher Choice:

There will be a "to do" screen which is similar to the current chosen grouping screen. It will list the branches and the teacher will select which learner goes to which branch.

As the learners are added to a "branch", a group will be created for each branch and the learners assigned to the appropriate group. The teacher won't "see" this Group Activity in Live Edit (we will need to hide it somehow).

Do we want a teacher release to synchronise learners move on, or can they move on as soon as they are assigned a branch?

It would be nice to have a mechanism to notify the Teacher (e.g. email address or choose IM) when every Learner arrives at the Branching point but we don't have it for Chosen Grouping currently so we won't do it for branching.

Group-based:

There will be a "to do" screen which will be similar to the assignment screen in authoring. Can we piggyback on the authoring screen somehow. It will list the branches and the teacher will select which group goes to which branch. The group names will already exist as Grouping will have already been done.

It might be nice to show the names of the learners in each group (maybe via AJAX) so that if the groups names aren't meaningful the teacher can check who is in which group.

Tool Output-based:
See Handling Tool Output-based Branching below

Learning

Learner Choice:

The current Optional Activity screen will be sufficient.

Teacher Choice:
Group-based:
Tool Output-based:

We need a mechanism to stop users until the Teacher makes choice of branches for these users. Users who are not given branches yet are not to proceed.

Handling Tool Output-based Branching

Tool Outputs: Numeric Values, Conditions and Complex Values.

Tools will produce three types of "outputs":
(1) numeric values such as quiz mark (this will normally summary values)
(2) conditions (which resolve to a boolean value at runtime)
(3) complex values such as all the answers for a particular student, all the answers for all students.

Tool output-based Branching will use the numeric values and conditions as the basis for branch selection. When the selection is done in authoring, the author should always set up a default branch, which is the branch selected if no condition matches. (We may make this optional if we find any instances where it is guaranteed that all cases are covered.)

Example (1) At authoring time, the tool will notify branching that it can supply a numeric value, which happens to be called mark, which will be within a range 0 to 10 (assuming that there are 10 questions). The branching screen in authoring will then set up a series of ranges.

At runtime (lesson time), branching will get a quiz mark of 6, and based the series of ranges set up at authoring time, a branch is chosen.

We want to have some feedback mechanism in the tool somewhere so that if the teacher then adds another question they are prompted to check the branch settings. Not exactly sure where and how we do that at the moment, because the ranges are now only stored in branching. This is the drawback with supporting the numeric values directly in branching. The alternative was to force the tool to produce the ranges itself but that is requires a more complicated interface between the tool and the branching.

For LAMS 2.1, if a tool is used as the inputs to a branch, and the user opens up the tool activity (to bring up the authoring screen) then they will be warned that the tool is used for the branching and that they should check the branch mappings if they change the tool's details.

Example (2) At authoring time, the user will use a special tab to set up a condition, such as the answer to the third question contains the word Koala, or the answer to the third question is correct. The tool records this as a condition.

In branching in authoring, the conditions are also displayed and may be selected.

At runtime, the branching engine calls the tool to resolve the condition to a boolean value and it determines the branch based on the tool's response.

Complex values will not be consumed by branching as it poses two problems:

- branching has no knowledge of the semantics of the data so it is hard to do something appropriate with the data
- branching cannot easily record enough details about the complex values so that changes in authoring don't completely invalidate the branching details, so keeping the control in the tool makes it easier to adapt to data changes.

However in future versions of LAMS, complex values will still be output by tools as they may be consumed by other tools. This will need to be mediated by the core, as there is no way for one tool to call another tool at present. Either that or we need to add something like webservices to tools, which would hardly make them lightweight! This will be implemented in a later version of LAMS.

For LAMS 2.1, MCQ will have numeric outputs that will be using for branching, in particular quiz score.

Tool Output Definitions

Each tool that has outputs will define a set of output definitions. Some definitions will be "predefined" for a tool, e.g. "Learner's Mark". Others may be created for a specific tool activity, via a Conditions/Output tab in monitoring, e.g. Second answer contains the word "Mercury".

Each output definition object will be as follows:

```
org.lamsfoundation.lams.tool.ToolOutputDefinition {
String name;
String description;
OutputType type;
Object startValue;
Object endValue;
Object complexDefinition;
Boolean showConditionNameOnly;
List<BranchCondition> defaultConditions;
}
```

Name: Name must be unique within the current tool content. This will be used to identify the output. If the definition is a predefined definition then the name will always be the same (e.g. LEARNER_MARK) but if it is defined in authoring then it will need to made unique for this tool content (e.g. ANSWER_2_CONTAINS_1). At lesson time, the tool will be given back the name and will need to be able to uniquely identify the required output based on name, the tool session id and possibly the learner's user id.

If the tool contains generated definitions, then they must be copied when the tool content is copied, as the conditions may be modified via Live Edit. This must not modify the original design.

Description: Description is an internationalised text string which is displayed to the user as the output "name". It is the responsibility of the tool to internationalise the string. We suggest that the key for each predefined definition follow the convention OUTPUT_DESC_<output name>

Type: The type of the output value. This may be COMPLEX, DOUBLE, LONG, BOOLEAN or STRING. DOUBLE is for floating point numbers and LONG is for integers. This is implemented as an enumerated type. This is an enumerated type.

StartValue and **EndValue**: If the output value may be compared to a range, then this is the inclusive start values and end values for the range. This may be used to customise fixed definitions to ranges appropriate for the current data.

ShowConditionNameOnly: Turn on when we should only show the condition name on the Condition screen in authoring. This is useful when there are default conditions. Default to false.

DefaultConditions: Some tool output may have a some initial conditions that should be displayed in authoring, and when these exist authoring will not let the author and new conditions. Boolean conditions normally have two default conditions - true and false. This can also be used for Range conditions, such as the default "Learner Selected X nomination" in Voting

For example, if there are 10 questions in a Multiple Choice Quiz, then the definition of the learner's mark would be:

```
ToolOutputDefinition {
name = "learners.mark",
description = "Mark for an individual learner";
type = OUTPUT_LONG;
startValue = "0";
endValue = "10";
complexDefinition = null;
showConditionNameOnly = false;
defaultConditions = null; }
```

For 2.1, we will not deal with complex outputs, so for now we will not define how a complex definition is defined. The field is placed in the object so that we have the place for it when we do design the complex output definitions.

To get the tool output definitions, there is a method SortedMap<String, ToolOutputDefinition> getToolOutputDefinitions(Long toolContentId) in the ToolContentManager interface. Implementing this method forms part of the LAMS 2.1 Tool Contract.

Tool Output Data

When the learning design is then used during a lesson, the core may request the tool's output data from the tool. In 2.1, the only use will be for branching.

To get the tool output, there are two similar methods in the ToolSessionManager interface. Implementing these methods forms part of the LAMS 2.1 Tool Contract.

- Map<String, ToolOutput> getToolOutput(String[] names, Long toolSessionId, Long learnerId) .
- ToolOutput getToolOutput(String name, Long toolSessionId, Long learnerId) .

When the first method is called, the tool should return all the outputs that match the list of names. If names is null, then all possible output will be returned. If names is an array of length 0, then no output will be returned. The second specifies only one tool output and may return null if the name doesn't match to a valid name for this tool.

The toolSessionId and learnerId will be used to determine the data on which the outputs are to be based. If the output is for the entire group/class (e.g. average mark) then the tool can ignore the learnerId. If the output could be valid for either one learner or for the entire tool session (e.g. all postings) then it should apply the learnerId if supplied, or all entries for the toolSession of the learnerId is not supplied. If the output is nonsense for the whole tool session (e.g. individual mark) and the learner id is not supplied or that learner doesn't have any data then it should return an "empty" but valid value e.g. 0 for individual mark

Note: the learnerId may not be the userId of the current user as the current user may be a staff member.

The tool's output will be returned in a map, where the key is the "name" for each output and the ToolOutput is the output for that "name". At present, if there are multiple attempts at an activity for one learner, we assume each attempt would have a different toolSessionId, and hence getToolOutput[] would be called multiple times. This may not be a valid assumption.

The tool output object will be as follows:

```
org.lamsfoundation.lams.tool.ToolOutput {
String name;
String description;
ToolOutputValue value;
}

org.lamsfoundation.lams.tool.ToolOutputValue {
String type;
Object value;
}
```

ToolOutput.name and **ToolOutput.description**: As per the fields in ToolOutputDefinition
ToolOutput.value: The "value" is the real output, and is of type ToolOutputValue.
ToolOutputValue.type: As per the field in ToolOutputDefinition
ToolOutputValue.value: This may be a number (6.8, -2000), a String ("This is my posting"), a Boolean (true/false) or an array of ToolOutputValues (for Complex type). The Complex type allows the ToolOutputValues to be recursive and have as many layers as it wishes.

Linking Tool Output to Branching

In Authoring, the author can select the tool output branch as a type of branch. When this is selected, a drop down menu of all the activities preceeding this branching activity will be displayed. The user will select an activity and the activity will be set as the input activity for this branch. Once an activity is selected, a button will appear which will allow the user to map the outputs to specific branches.

When the mapping button is pressed, Flash will call the server to get the list of tool output definitions for the activity. The call includes the toolContentId, so that it can get any custom conditions set up for this particular activity. The call will return a list of ToolOutputDefinitionDTO objects.

This call will be made via the Authoring service for two reasons:

1. the core can query the tool and wrap up its response in wddx, alleviating the need for the tool to know about wddx
2. the core will filter the definitions to only return numeric and boolean output.

The ToolOutputDefinitionDTO object is the same a the ToolOutputDefinition object except that all numeric or object fields are converted to strings.

Flash will then display a screen allowing the user to define conditions. If there are default conditions, then the user can view but not modify the conditions. Otherwise, if the output is a range then the Flash screen will allow the user to set up an inclusive set of ranges. The startValue and endValue are the minimum and maximum values that can be used for a range. Once the conditions are set up, the user can allocate branches to conditions, which set up the "branch mappings".

Finally, the ToolBranchingActivity has one branch marked as the default branch. This is for safety as well as convenience in authoring. Trying to determine that all possible cases are covered may be difficult in the future (particularly once we have multiple activities as inputs or conditions defined by the tools) so if the progress engine finds that no conditions are met, it will go to the default branch. The default branch acts as the "catch-all" else case.

The output to branch mappings will then be saved as part of the Learning Design. The objects from Flash will look something like:

```
org.lamsfoundation.lams.learningdesign.dto.ToolOutputBranchActivityEntryDTO {
    Long entryID;
    Integer entryUIID;
    Integer groupUIID;
    Integer sequenceActivityUIID;
    Integer branchingActivityUIID;
    BranchConditionDTO condition;
}

BranchConditionDTO {
     Long conditionId;
     Integer conditionUIID;
     Integer orderID;
     String name;
     String displayName;
     String type;
     Object startValue;
     Object endValue;
     Object exactMatchValue;
     Integer toolActivityUIID;
}
```

and be converted to ToolOutputBranchActivityEntry and BranchCondition objects. When the data is returned to Flash, it will be returned in these DTOs. Some of these fields are "redundant" from the server's perspective (as they are determined from relationships) but they are important in the DTOs as they make it much easier for Flash to maintain the relationships.

When the data is stored in the database, the startValue, endValue and exactMatchValue will be stored as Strings but the ToolOutputBranchActivityEntry will convert them to their appropriate types based on the type field. The ToolOutputBranchActivityEntry objects go into the lams_branch_activity_entry table, along with the group based and teacher chosen mappings, and the condition details go into a lams_branch_condition table.

When the learning design is run as part of the lesson, the progress engine will detect when a learner starts the branch. It will then go through each branching mapping stored for this branching activity (based on the order id field) and for each "name" encountered it will call the tool to get the output value for that name and do the test. When it encounters an entry for which is passes the test, it will use that branch.

If the type is numeric, it will test startValue <= tool output <= endValue.

If the type is boolean, it will test exactMatchValue == tool output.

If the progress engine goes through all the entries and cannot find a match then it would send the user to the branch marked as the else branch (in the ToolBranchingActivity).

For 2.1, we will not address activities that can return multiple sets of data, such as SCORM.