XMCDA v3
XMCDA history, benefits of XMCDA 3, implementations
(Java, R, Python)

Sébastien Bigaret, Patrick Meyer
Télécom Bretagne
UMR CNRS 6285 Lab-STICC
Structure of the presentation

• General purpose of XMCDA
• XMCDA v2 : motivations for an evolution
• What’s new in XMCDA v3 ?
• Consequences
• Tools for developers
• Illustration
• Roadmap
GENERAL PURPOSE OF XMCDA
An MCDA “method” as an algorithmic sequence
Solutions by Decision Deck

• To make algorithms **interoperable**: XMCDA, a data standard for MCDA “data”
• To make algorithms **easily available**: XMCDA web-services
• To create complex algorithmic **workflows** of algorithms: diviz
XMCDA

• A unique communication language with and between MCDA algorithms
• Standardization and unification of multiple schools of thought
• Representation of MCDA data elements in XML according to a grammar (the XMCDA XML schema)
Calculation resources

• MCDA algorithms which are made available for anybody over the Internet: XMCDA web-services
• Hosted at Télécom Bretagne, Centrale Supélec, OVH
• Maintained by the TB diviz team
• Many different contributors
A tool for composing the XMCDA web-services and local resources
XMCDA v2 : MOTIVATIONS FOR AN EVOLUTION
XMCDA v2

• Lack of **coherence**:
  A discrimination threshold is stored in `<criterion>` whereas a weight can be stored outside

• Non-unicity of the **representation**:
  A weight can either be stored in `<criterion>`, or outside of it in a `<criterionValue>`

• Potentially uncontrollable **increase** of the number of tags in case of evolutions:
  `<rankedLabel>, <fuzzyLabel>, <rationalLabel>, <labelledLabel> ...`

• Multiple “ad-hoc” **implementations**
  Java, R, Python, C++, ...
XMCDA v2

• Some consequences:
  – A danger for the interoperability of the programs/web-services
  – An extensive implementation of XMCDA is very difficult, and the resulting API would be very complex
  – Failure of many developments and student projects

• => Necessity for an evolution
XMCDA v3

Approved by the consortium at the 11th Decision Deck workshop in Paris (23-24 / 09 / 2013)
WHAT’S NEW IN XMCDA v3
Declaration of main MCDA “objects”

• Alternatives, criteria and categories are declared without any supplementary data / preferential information
  – no weight, scale or thresholds in criterion, no rank in category, ...

• Also for sets of alternatives, criteria and categories
  – set of learning alternatives, ...

• These are the only “objects” that can be referenced in other XMCDA tags
Data and preferential information

• No choices, i.e., no duplicate places to store a certain information
  – Values are now stored once and for all in a `<values>` compound tag, even if it is a single value
  – Relations are one and for all represented by matrices

• The `mcdaConcept` attribute becomes central: it specifies what information is contained in a general tag
  – In XMCDA 2, categories had ranks. In XMCDA 3, ranks are now stored in the general `<categoriesValues>` tag
Outputs

<programExecutionResult>

- Messages for the user about the execution: informations at different levels (debug, info., warnings & errors)

- A formal status:
  OK, WARNING, ERROR, TERMINATED
Outputs: prg. execution status

<programExecutionResult>
  <status>error</status>
  <messages>
    <message level="error">
      <text>
        Parameter nb_iter_max: invalid value 7.3 (real)
        The value must be a positive integer
      </text>
    </message>
  </messages>
</programExecutionResult>
CONSEQUENCES
XMCDA v3 consequences

• Real **interoperability** between programs / web-services
  – No need for human intervention to guarantee this interoperability

• Simplification of the **xsl + css** (representation of the data in browsers / diviz)

• Easier generation of **man/machine interfaces** for data input / output

• Facilitated **integration** into existing software
TOOLS FOR DEVELOPERS
Observations ... and difficulties

- Writing or extending partial XMCDA parsers
- Writing or adapting algorithms to XMCDA web-services infrastructure
1. Standard implementation of XMCDA v3

• In Java
• Can be called from other programming languages
  – Currently experimented from Python and R
• Reading / writing of XMCDA v3 / v2 files
• Conversion of v2 <-> v3
• Manipulation of XMCDA objects (adding, deleting, ...)
1. Standard implementation of XMCDA v3

- Source code on gitlab: https://gitlab.com/XMCDA-library/XMCDA-java
- Examples of use in Java, Jython & R: https://gitlab.com/groups/XMCDA-library
2. Program skeleton generator

- Inputs:
  - programming language
  - standardized description of the program

- Output: files hierarchy with identified todo spots

- => Simplified coding & integration
3. Program description

- XML description of the program
- Integration into diviz & GUI
- Documentation (web & diviz)
- Program skeleton (Java, Python, R)
ILLUSTRATION
Illustration

weightedSum-DDWS13: a simple weightedSum
- Two inputs:
  - A performance table
  - The criteria’ weights
- One parameter: the aggregation operator, either “weightedSum” or “normalizedWeightedSum”
- Pre-requisites: the set of criteria is the same in the performance table and in the criteria’ weights
weightedSum description (1/3)

inputs

```xml
<input id="performanceTable" name="performanceTable"
   displayName="performanceTable"
   isoptional="0">
   <documentation>...</documentation>
   <xmcda tag="performanceTable"/>
</input>

<input id="criteriaWeights" name="criteriaWeights"
   displayName="weights" isoptional="0">
   <documentation>...</documentation>
   <xmcda tag="criteriaValues"/>
</input>
```
weightedSum description (2/3)

parameter

```xml
<input id="parameters" name="parameters" displayName="operator" isoptional="0">
  <documentation>..</documentation>
  <xmcda tag="methodParameters"><![CDATA[
    <methodParameters>
      <parameter name="operator">
        <value>
          <label>%1</label>
        </value>
      </parameter>
    </methodParameters>
  ]]>></xmcda>
  <gui status="preferGUI">
    <entry id="%1" type="enum" displayName="operator">
      <documentation>
        <description>Aggregation operator</description>
      </documentation>
      <items>
        <item id="weightedSum">
          <description>weighted sum of the evaluations of alternatives on criteria</description>
          <value>sum</value>
        </item>
        <item id="Normalized weighted sum">
          <description>Computes the normalized weighted sum ..</description>
          <value>normalizedWeightedSum</value>
        </item>
      </items>
    </entry>
  </gui>
</input>
```
weightedSum description (2/3)

parameter

- XMCDA v3 tag: methodParameters

- One parameter id: operator of type enum

- Valid values:
  - weightedSum
  - normalizedWeightedSum
<output id="alternativesValues" name="alternativesValues"
    displayName="alternativesValues">
    <documentation>…</documentation>
    <xmcda tag="alternativesValues"/>
</output>

<output id="msg" name="messages" displayName="messages">
    <documentation>…</documentation>
    <xmcda tag="methodMessages"/>
</output>
Appearance in diviz
weightedSum: code generation (Python)

TODO markers in comments
weightedSum:
Step #1: write the algorithm

```python
def calculate_weighted_sum(inputs):
    """
    Calculates the weighted sum.
    Parameter performance_table: a dictionary (alternative id,criterion id)->value
    Parameter weights: a dictionary { criterion id -> weight }
    Returns: a dictionary { alternative id -> double }
    """
    performance_table = inputs.performance_table
    weights = inputs.weights

    # iterate on (alternative_id, criterion_id):
    # add the evaluation of each criterion on each alternative,
    # multiplied by the criterion's weight, to the alternative's weighted sum
    return alternatives_values
```
```
weightedSum – Step #2
XMCDA → data to feed the algorithm

class Inputs:
    """
    Class Inputs gathers all the information retrieved from the XMCDA inputs
    """
    def __init__(self):
        pass # TODO -- related to check_and_extract_inputs()

def check_and_extract_inputs(xmcd, xmcd_exec_results):
    inputs = Inputs()
    check_inputs(inputs, xmcd, xmcd_exec_results)

    if not (xmcd_exec_results.isOk() or xmcd_exec_results.isWarning()):
        return None

    return extract_inputs(inputs, xmcd, xmcd_exec_results)

def check_inputs(inputs, xmcd, xmcd_exec_results): # TODO
    # ... check XMCDA inputs

    # Check as much things as possible before signalling an error
    # so that the user gets the maximum nb of errors we can detect
    if not (xmcd_exec_results.isOk() or xmcd_exec_results.isWarning()):
        return None

    return inputs

def extract_inputs(inputs, xmcd, xmcd_execution_results): # TODO
    """Transform XMCDA inputs into what we need for the algorithm"
    # ...  

    # we may encounter errors here as well, in which case, deal as above
    # and try to collect more errors before exiting.

    return inputs
weightedSum

Step #2a: prepare data structure

- Performance table
- Weights
- Aggregation operator

```
WEIGHTED_SUM = "weightedSum"
NORMALIZED_WEIGHTED_SUM = "normalizedWeightedSum"
AGGREGATION_OPERATORS = (WEIGHTED_SUM, NORMALIZED_WEIGHTED_SUM)

class Inputs:
    ""
    Class Inputs gathers all the information retrieved from the XMCDA inputs
    ""
    def __init__(self):
        self.performance_table={} # { (alternative id, criterion id) -> value }
        self.weights={} # { criterion -> weights }
        self.operator=None
```
Step #2b: check XMCDA inputs

```python
def check_inputs(inputs, xmcda, xmcda_exec_results):
    # Check the performance table object:
    # - we have one and only one performance table
    # - it has no missing values
    # - all values are numeric

    # Check weights
    # - we have one and only one performance table
    # - all values are numeric

    # Check that perf.table and weights have the same
    # set of criteria

    # Check parameters
    # - one and only one <programParameter>
    # - it has a single parameter whose id is 'operator'
    # - the value of the operator is known

    return inputs
```
weightedSum

Step #2c: extract XMCDA inputs

def extract_inputs(inputs, xmcda, xmcda_execution_results):
    "Transform XMCDA inputs into what we need for the algorithm"
    # get the operator
    inputs.operator = ...

    xmcda_perf_table = xmcda.performanceTablesList.get(0)

    # Build input.performance_table by extracting
    # each criterion's id and alternative's id and their value
    inputs.performance_table = ...

    # Build inputs.weights by extracting the weight associated
    # the each criterion's id
    inputs.weights = ...

    # Remember: the algorithm only takes a perf.table and a
    # vector of weights! → normalize the weights if needed
    if inputs.operator == NORMALIZED_WEIGHTED_SUM:
        # normalize inputs.weights
        ...

    return inputs
weightedSum – Step #3
Transform algorithm data to XMCDA

```python
XMCDA_v3_TAG_FOR_FILENAME = {
    # output name -> XMCDA v3 tag
    'alternativesValues': 'alternativesValues',
    'messages': 'programExecutionResult',
}

[...]

def convert(results, xmcda_execution_results):
    ""
    Converts the outputs of the computation to XMCDA objects
    Returns: a map with keys being the names of the outputs,
    and their corresponding XMCDA v3 values
    (NOT including 'messages.xml')
    ""
    # for XMCDA tag: alternativesValues
    xmcda_alternatives_values = XMCDA()
    # ...

    return {
        'alternativesValues': xmcda_alternatives_values,
    }
```
Step #4: write XMCDA input files

Example: parameters.xml (XMCDA v3)

```xml
<?xml version="1.0" ?>
<xmcda:XMCDA ... >
<programParameters>
  <parameter id="operator">
    <values>
      <value>
        <label>weightedSum</label>
      </value>
    </values>
  </parameter>
</programParameters>
</xmcda:XMCDA>
```
Run it!

```
./weightedSum.sh --v3 \  
  -i tests/in1.v3 \  
  -o tests/out1.v3

./weightedSum.sh --v2 \  
  -i tests/in1.v2 \  
  -o tests/out1.v2
```
weightedSum tests

./tests.sh --v2

./tests.sh --v3
Properties / comments

• Your coding is independent of v2 and v3
• Tests allow validation of the deployed web-service

The full source code for this example can be found at:
https://gitlab.com/XMCDA-library/example-simpleWeightedSum
ROADMAP
Roadmap

Conversion v2 -> v3 (diviz & web-services)

diviz and web-services v3

web-service proposal

1 year

web-service proposal

diviz and web-services v2

freezed diviz and web-services v2