Multi-Dimensional Context-Aware Adaptation of Service Front-Ends

Project no. FP7 – ICT – 258030

Deliverable 4.2.1

Algorithms for Advanced AL

Due date of deliverable: 30/09/2011

Actual submission to EC date: 30/09/2011

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
[PU]	**[Public]**	**Yes**

Document Information	
Lead Contractor	UCL
Editor	Vivian Genaro Motti
Revision	22.06.2011
Reviewer 1	CNR/ISTI – Fabio Paterno, Christian Sisti, Lucio Davide Spano
Reviewer 2	
Approved by	
Project Officer	Jorge Gasós

Contributors	
Partner	**Contributors**
UCL	Vivian Genaro Motti
CNR/ISTI	Fabio Paterno, Christian Sisti, Lucio Davide Spano

Changes			
Version	**Date**	**Author**	**Comments**
1	24/01/2011	UCL – Vivian Genaro Motti	Basic structure of the document: table of content, initial content based in the Description of work
2	22/06/2011	UCL – Vivian Genaro Motti	Review of the structure, more detailed content description
3	15/09/2011	UCL – Vivian Genaro Motti	Insertion of adaptation methods (content provided by CNR)
4	16/09/2011	UCL- Vivian Genaro Motti	Insertion of algorithms for adaptation techniques
5	20/09/2011	UCL- Vivian Genaro Motti	Completing adaptation techniques
6	26/09/2011	ISTI - CNR	Review Version
7	28/08/2011	UCL – Vivian Genaro Motti	Final Version reviewed

Executive Summary

The main goal of this deliverable is to present a first version of a library of adaptation algorithms. These algorithms are being iteratively built mainly according to the templates previously created to describe adaptation techniques, and also according to the Use Cases descriptions. Different granularity levels are considered, combining techniques that target specific properties of user interface elements, a specific type of user interface element itself, such as text or images, and also adaptation methods, which compose multiple techniques together.

In this first release, the algorithms are described by means of pseudo-code and dataflow diagrams, these implementations are technology-independent, allowing their application in multiple contexts, considering varied devices, and permitting further refinements. All the essential information about the adaptation process is provided, including input data, aimed output and essential steps to perform adaptation. Additional refinements and extensions are planned for the next releases of this library.

Table of Contents

1 Introduction

1.1 Objectives

This deliverable presents a first version of a set of algorithms to implement adaptation, considering different methods, techniques and strategies. The algorithms are described independently of technology by means of pseudo-code and dataflow diagrams. The descriptions are presented in a structured manner: defining the required input content, steps of processing and also expected results.

The goal of having a library of algorithms for advanced adaptation logic consists in providing multiple alternatives for implementing adaptation. Stakeholders can take better decisions by analysing the algorithms, identifying and selecting the most appropriate approach for each context of use. The techniques here described will also contribute with the implementation of machine learning algorithms.

1.2 Audience

The target audience consists of researchers and practitioners with interest in this domain (algorithms for advanced adaptation logic).

1.3 Related documents

- D2.1.2 - CADS and CARF (the techniques described in this deliverable are also available at D2.1.2)
- D2.2.1 - CARFO (R1) provides additional information about adaptation techniques, such as their compatibility level
- D3.1.1 - Reference Models (the description of the Use Cases defines already what triggers the adaptation techniques and also the pre-conditions required to perform it)
- D4.3.1 – Adaptation Engine will benefit from the algorithms described in this deliverable

1.4 Organization of this document

Chapter 1 presents the goal, audience and related documentation with this deliverable. In Chapter 2, the description of the work is presented. Chapter 3 provides background information and fundamental concepts for the understanding of the library of algorithms. In Chapter 4, the algorithms for adaptation techniques and methods are presented. Chapter 5 presents the demonstration algorithm for the adaptation method that transform desktop contents into audio modality. Chapter 6 presents final remarks and concludes this deliverable.

2 Description of Work

2.1 Motivation

In order to provide adaptation, it is necessary to consider a set of different techniques, methods and strategies. A library of algorithms allows developers to: (i) identify the possibilities of adaptation techniques, (ii) select techniques that are more relevant for their application, (iii) identify requirements and specifications of the adaptation process (input data), (iv) access information about the adaptation processing and (v) identify the expected results.

2.2 Goal

The library of algorithms aims at organizing the techniques, methods and strategies for adaptation previously gathered, as well as describing them in a detailed manner and independently of technology. This document will aid developers to implement and re-use this knowledge to implement adaptation.

In this first release the algorithms focus on content adaptation. For the next releases, the library will be extended including also adaptation for navigation and presentation. The algorithms are being built in an iterative manner, i.e. described by means of CARF templates, detailed as Use Cases, and defined in terms of dataflows, algorithms and pseudo-code, before actual implementation. This iterative approach permits important decisions to be taken along the evolution stages of Serenoa project.

2.3 Description

Task 4.2 Algorithms for AAL [led by **UCL**]

1. This task will be concerned with devising and coding algorithms for implementing AAL. We envisage different kinds of algorithms to be developed, such as, paginations, layout optimization, optimal re-distribution of components, resizing, zooming and scaling, calculating graceful degradations or upgradations, etc.

3 Background Information

The library of adaptation algorithms is based on: the adaptation techniques previously defined in the CARF, detailed with the templates (D2.1.2), and described as a Use Case (D3.1.1).

3.1.1 Adaptation Techniques

There are many different applications from both: scientific and commercial domains, that benefit from adaptation processes and that are published or that were already reported in the literature. In previous steps of the project we collected a set of such adaptation techniques, listed them in the reference framework (CARF), detailed them with a template, and defined them as use cases. Now, in order to ellaborate the adaptation algorithms, the rationale description will be transformed in an algorithm containing essential information about each technique, as described below.

The algorithm consists in a description of adaptation techniques, mainly taking into account: the required input data (data format, type, source of the information), the essential steps of processing (illustrated by dataflow diagrams), and the aimed results (in terms of outputs). The algorithm is a function defined in terms of 5 properties:

- **Pre-conditions:** describe which are the necessary resources to perform the adaptation, regarding both, software and hardware aspects, for instance in order to perform *Image crop* it is necessary to have one or more images available in the application
- **Input:** defines the type of input required to perform the adaptation. It can be specific, such as a number, or a set of resources, such as a web page composed by multiple content types.
- **Output:** defines the outcome of applying the algorithm, for instance an audio description.
- **Dataflow diagram:** illustrates the sequence of steps, as well as required flows and loops in the processing. The begin and end of the process are presented in ovals; the conditions are presented with diamonds; the data flows are presented with arrows; the processes are presented with rectangles and input and output data is represented with parallelograms.
- **Algorithm:** a detailed description of all the steps required to perform the adaptation (alternative flows can also be included).
- **Pseudo-code:** defines the algorithm in terms of programming commands, detailing the logic of the processes

The algorithms are provided in different levels of details. Depending on the domain of application, some algorithms define already specific parameters or leave the decision for the developer, there are some alternative approaches for implementation that are briefly presented as a commentary.

3.1.2 Adaptation Methods

The adaptive and adaptable applications that have been developed and reported in the literature consider a set of context information, and also a set of adaptation techniques to provide adaptation for the end users. Usually it is necessary to establish a reasoning to infer the best order and the relations between the techniques in order to provide efficient adaptation.

For instance if the goal is to improve the readability of a text content, then the properties of the font can be modified, such as color, size, spacing and alignment, and also the background color of the interface. We call this set of techniques, an Adaptation Method.

The adaptation methods will be also defined in terms of their pre-conditions, inputs and outputs, followed by a literal description of the tasks (in terms of adaptation techniques). The description may be complemented with illustrations, references, and examples.

3.1.3 Advanced Logic

The advanced logic is provided by a combination of adaptation techniques, which compose adaptation methods. Some adaptation techniques are not compatible to be performed together, such as *translate text to audio* and *re-size text font*; this reasoning will be formally defined in the CARFO (D2.2.1). The ontology will define the compatibility relationships with semantic information and constraints about the adaptation

techniques, and this knowledge will provide support to implement adaptation using machine learning techniques.

The use of machine learning techniques provides the support to implement the advanced logic; the main goal is to associate the adaptation techniques with the context of the user in an efficient manner. The adoption of machine learning allows the adaptation process to be better adjusted to the actual context of use, this occurs because the interaction of the user, as well as her evaluation of the adaptation (acceptance or rejection) is used to adjust and improve the adaptation process.

Besides, machine learning techniques allow complex context to be considered, for instance when multiple information of the actual context of use are relevant and must be taken into account to decide the most appropriate adaptation techniques, methods and strategies. The machine learning also facilitates the inference process, necessary to combine the best decisions for each case of use.

3.1.4 Strategies to Present Adaptation

The advanced logic considers also distinct strategies to present the adaptation process (such as animations), and certain principles (such as graceful degradation and progressive enhancement). The use of these concepts is closely associated with the context, because there are alternative manners to implement and provide them, and not always their application is recommended, for instance, animation aims to provide users a smooth transition between initial the interface and the adapted one, however it requires processing capabilities of the device in use that must be taken into account.

This deliverable provides an overview about these concepts, and defines some examples in which they can be applied. However, it is clear that there are different manners to implement and provide strategies and principles for adaptation.

- Animation: consists in presenting to the end user intermediary steps between the initial interface and the adapted one; it aim to prevent the end user disruption;
- Graceful Degradation: consists in allowing the application to be degraded, i.e. removing, reducing and replacing functionalities that are less relevant, cannot be performed or that disturb the interaction (reducing the performance for example)
- Progressive Enhancement: consists in providing only the basic functionalities in the main application and allowing it to be enhanced according to the context of use, i.e. optimized use of the resources of the user; it follows goals that are opposite to graceful degradation

The graceful degradation and progressive enhancement are intimately related to the capabilities of the platform in the target context of use. For instance, taking into account one single resource (such as an image) and one specific property (size), there is a set of sequential adaptation processes to achieve degradation or enhancement, i.e. the smaller the screen space available, the smaller the image is re-scaled, or vice-versa.

3.2 Final Remarks

The library provides developers with a collection of adaptation algorithms that can be analysed and selected in terms of input entries, aimed output and essential processing steps. The choice to define the algorithms regardless of technology (programming language) is justified by the fact that the compatible technologies vary according to the platform, and, at this moment of the project, it is preferable to compose the library independently of the device type.

The library here described takes into account the fact that the algorithm will vary depending of the technology supported by the actual context of the use and the constraints imposed by it. Thus, the description provided here is detailed, without being technology-specific. Clearly, there are other possible alternatives for implementing and also for refining these techniques, some of these alternative approaches are briefly discussed in the comment section of the algorithms.

4 Algorithms for AAL - Adaptation Techniques and Methods

The adaptation techniques consist in performing transformations that are specific for certain resources, such as images or text content, these transformations may be also specific to one single property of the content, such as the size or the color. This section presents the algorithms for part of the adaptation techniques listed in CARF. These techniques are organized according to the resource that they target on and to the properties of adaptation.

4.1 Adapting Content

This section presents algorithms that are applied for contents regardless of their type.

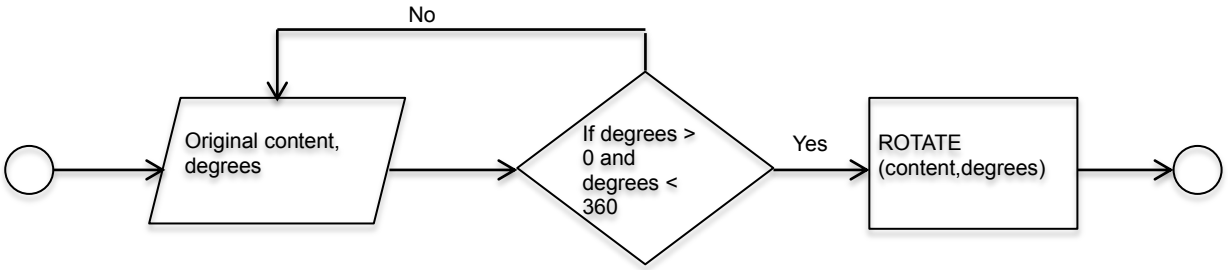
4.1.1 Change Orientation (aka Change Direction, Rotate)

Pre-conditions: there is a content that can be rotated

Input: the original content, a value defining the degrees of rotation

Output: the content is presented in a new direction

Dataflow:



Algorithm: The original content is taken as an input; the value of degrees for rotation is taken as an input;

The value is verified (must be greater than 0 and lower than 360)

The content is rotated according to the value of degrees provided

Pseudo-code:

```
READ content
READ degrees  // >0 and <360
IF degrees > 0 AND degrees < 360 THEN
      ROTATE(content,degrees)
```

Comments: depending on the shape or size of the content, the layout may be affected; in this case, complementary adaptation techniques must be performed, such as re-size; we are assuming here a clockwise orientation, however a signal indication the sense can also be considered.

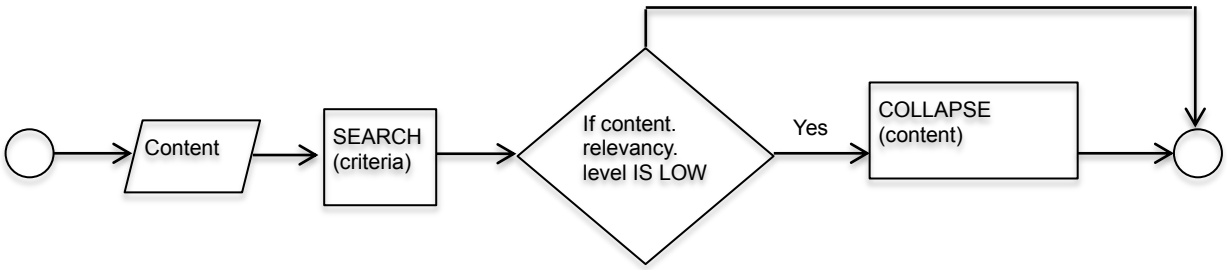
4.1.2 Collapse to Zoom (Method)

Pre-conditions: there is a set of elements in the UI available

Input: the UI elements and contents

Output: part of the content is collapsed, part is re-sized

Dataflow:



Algorithm: The contents are taken as input

According to given criteria, certain elements of the content are collapsed

The UI is presented to the end user with a new layout

Pseudo-code:

```
READ contents
SEARCH(criteria)
IF content.relevancy_level IS LOW
      COLLAPSE(content)
```

Comments: one approach to define the criteria for this algorithm is for instance the type of the elements, e.g. images are collapsed, and text content remain visible. In the approach proposed by Baudish et al. [2004], for instance, the user is responsible for defining the criteria by means of gestures.

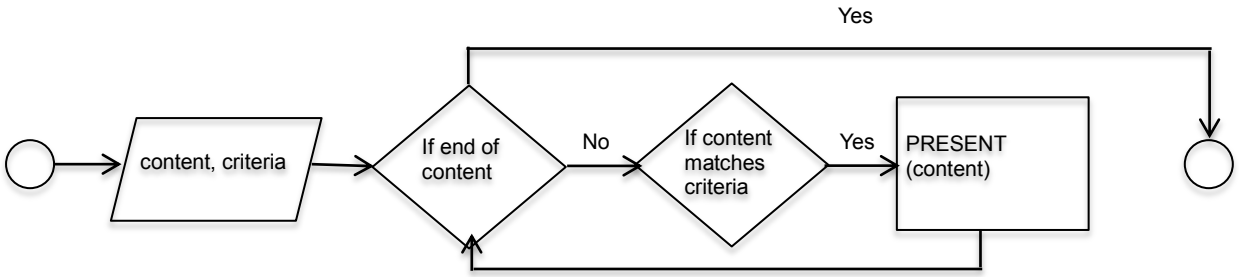
4.1.3 Conditional

Pre-conditions: there is a content organized in parts (e.g. sections) and criteria to present it

Input: the original content (organized in parts), criteria to present it conditionally

Output: the content is presented in a new sequence

Dataflow:



Algorithm: The original content is taken as an input; the criterion with conditions is taken as an input

For every part of the content, it is verified

If it matches the criteria then it is presented

Pseudo-code:

```
READ content
READ criteria
FOR EACH content_part
      IF content_part MATCHES criteria THEN
         PRESENT(Content_part)
```

Comments: the criteria can vary according to many different aspects of the content, syntactic ones, such as size or type, or semantic ones, such as subject or difficulty level; for the conditional to be performed, the layout must also be considered; again, complementary adaptation techniques, such as pagination or scrolling, may be necessary.

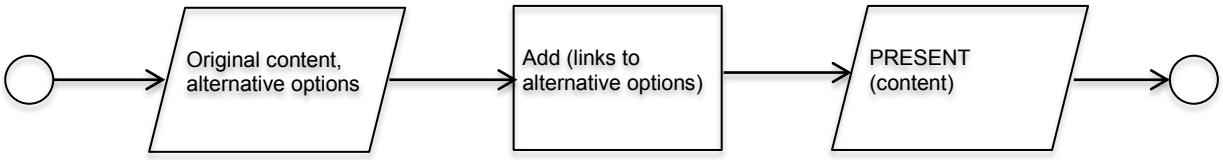
4.1.4 Frame based

Pre-conditions: there is content provided, and alternative contents associated to it

Input: the original content, and alternative options

Output: the content is presented with links to additional information

Dataflow:



Algorithm: The original content is taken as an input; the alternative options are taken as input

The contents are linked

The enhanced content with alternative information is presented to the end user

Pseudo-code:

```
READ content
READ alternative options
FOR EACH alternative option
      LINK(original_content, alternative_option)
```

Comments: the alternative options can have different formats or additional contents, the association process will then depend on each case, e.g. semantic information.

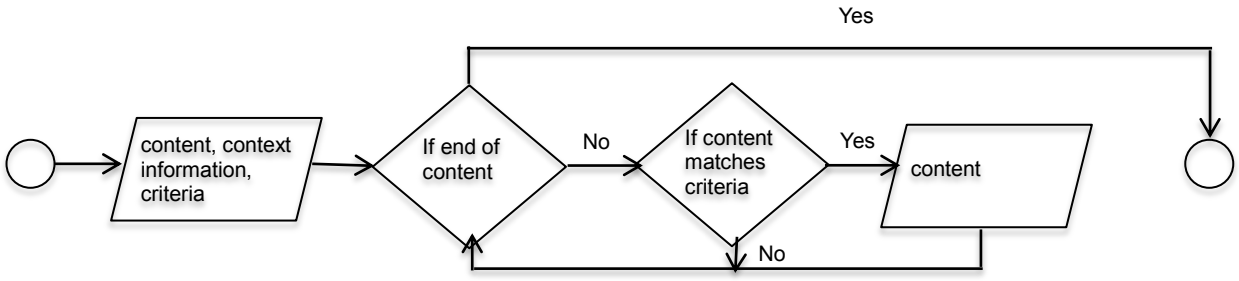
4.1.5 Filter

Pre-conditions: there is a content that can be filtered, and criteria to perform it

Input: the original content, context information, criteria to filter the content accordingly

Output: the content is filtered and then presented to the end user

Dataflow:



Algorithm: The original content is taken as an input; the context information is taken as an input; the criteria are taken as input;

The content is checked (according to the context information and criteria previously provided)

The content is filtered and presented to the end user accordingly

Pseudo-code:

```
READ content
READ context_information
READ criteria
FOR EACH content_part
      IF content_part MATCHES criteria GIVEN context_information THEN
         PRESENT(Content_part)
```

Comments: the criteria will vary according to the content, or context information; additional reasoning such as the ones provided by machine learning techniques may support this process (clearly, depending on the complexity level of the filtering)

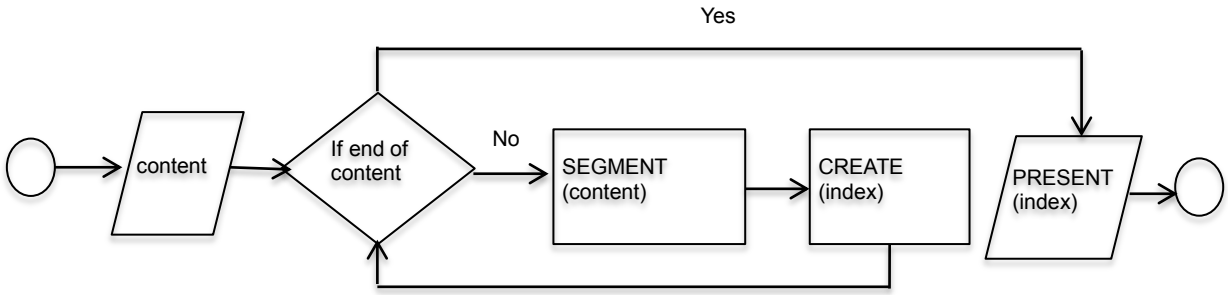
4.1.6 Indexed Segment

Pre-conditions: there is a set of contents

Input: the content

Output: the content is segmented and links to its parts are provided and presented to the end user

Dataflow:



Algorithm: The original content is taken as an input;

The content is checked and segmented in parts

The segments identified are presented to the end user with access links

Pseudo-code:

```
READ content
FOR EACH content_part
      SEGMENT(content_part)
      CREATE_INDEX(content_part)
PRESENT(index)
```

Comments: contents can be segmented according to different criteria, one possibility is to use its own type, for instance creating an index to images, texts and videos that are provided by the application; clearly other criteria can also be specified

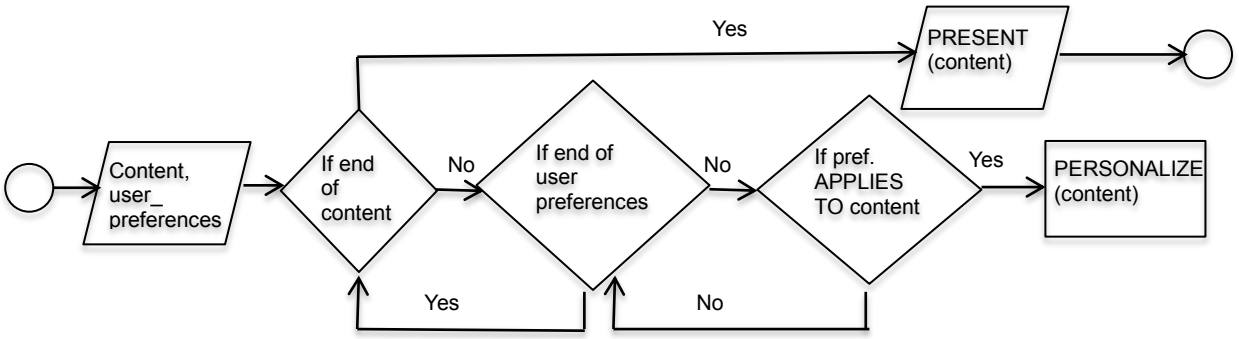
4.1.7 Personalize (Method)

Pre-conditions: there is a content that can be filtered and user preferences known

Input: the original content, the user preferences

Output: the content is personalized according to the user preferences

Dataflow:



Algorithm: The original content is taken as an input; the user preferences are taken as an input;

The content is verified and personalized

The personalized content is presented to the end user accordingly

Pseudo-code:

```
READ content
READ user_preferences
FOR EACH content_part
      FOR EACH user_preferences
            IF user_preferences APPLIES to content_part THEN
                  PERSONALIZE(content_part,user_preferences)
```

Comments: the user preferences can be manually gathered, by explicitly requesting users to fill in information, or automatically inferred according to the analysis of user interaction history; so, for each type of resource the content can be properly personalized

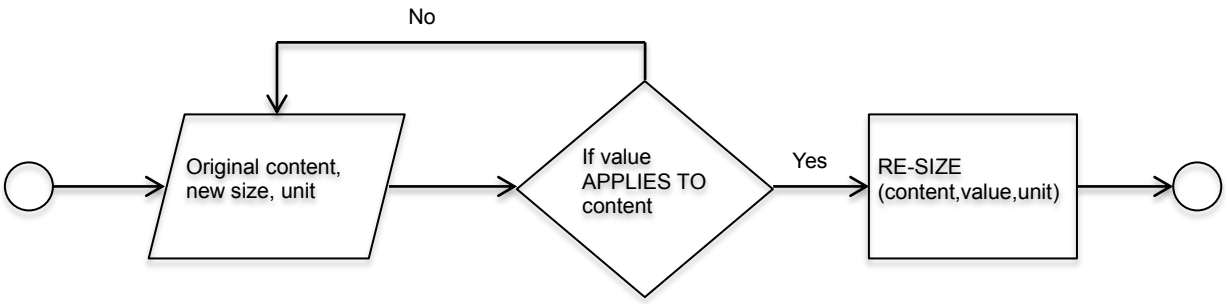
4.1.8 Re-size

Pre-conditions: there is a content to be adapted

Input: the original content, a value defining the new size for the content (e.g. 10), a unit for this value (e.g. px, cm, %)

Output: the original content is presented with a new size

Dataflow:



Algorithm: The content is taken as an input; the value and unit for re-sizing is taken as an input;

The content is checked

The value is checked, it must be different of the current size of the content

The size is changed to the new value and presented to the end user

Pseudo-code:

```
READ content
READ value  // new size, positive value, >0
READ unit  // px, cm, %
```

```
IF value != content.size THEN
      RESIZE(content, value, unit)
```

Comments: depending on the new size selected the content might be unreadable, sizes that are too small may prevent reading, and sizes too big may affect the layout and also affect the readability, in this case compatible techniques such as, fish-eye (distorted view), re-distribution, pagination or scroll bars must be considered.

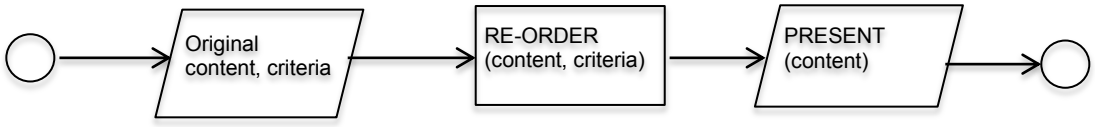
4.1.9 Re-order

Pre-conditions: there is content to be adapted and criteria to re-order it

Input: the original content, criteria defining the new order for the content

Output: the content is presented in a new order

Dataflow:



Algorithm: The content is taken as an input; the criteria for the new order are taken as input;

The content is analysed and re-ordered accordingly

The re-ordered content is presented to the end user

Pseudo-code:

```
READ content
READ criteria  // new order
RE-ORDER(content,criteria)
```

Comments: the criteria can take into account context information or properties of the resources, for instance the images can be re-ordered according to their resolution.

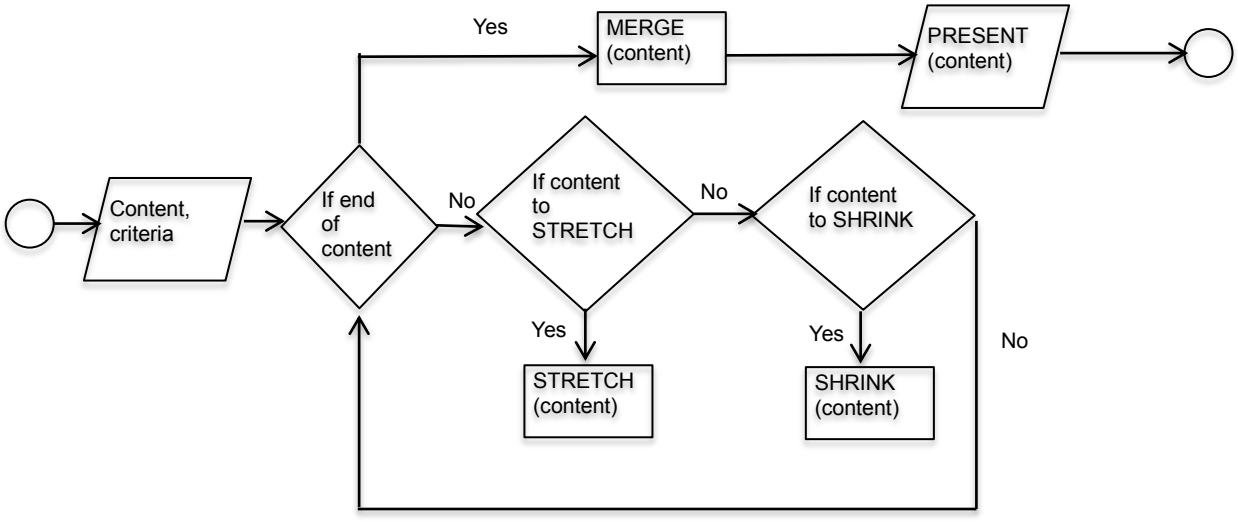
4.1.10 Stretch

Pre-conditions: there is content to be adapted and criteria defining what to stretch (expand) or shrink (collapse)

Input: the original content, criteria defining the new layout for the content

Output: the content is presented with parts of it stretched or shrunk

Dataflow:



Algorithm: The content is taken as an input; the criteria for the new layout are taken as input;

The content is analysed and processed accordingly

The adapted content is presented to the end user

Pseudo-code:

```
READ content
READ criteria
FOR EACH content_part
      IF content_part TO_STRETCH THEN
            STRETCH(content_part)
      IF content_part TO_SHRINK THEN
            SHRINK(content_part)
MERGE(content)
PRESENT(content)
```

Comments: the criteria can take into account for instance semantic aspects of the content (e.g. stretching contents related to the term searched by the user). Stretching here does not mean increasing the size, but expanding the content.

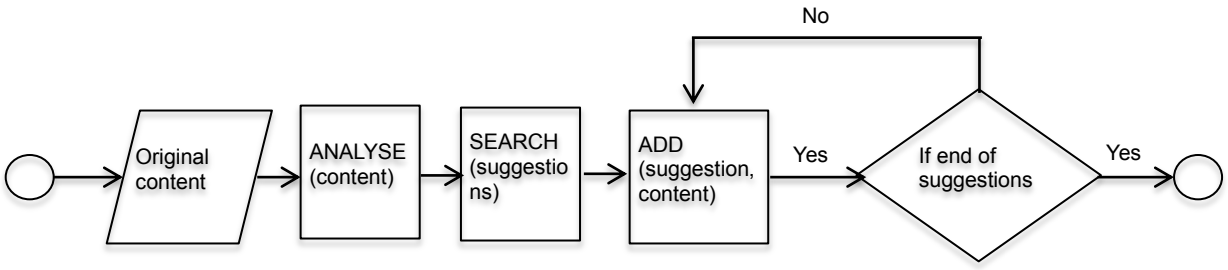
4.1.11 Suggest

Pre-conditions: there is content associated to the context of the user

Input: the context of use

Output: the content is presented with additional suggestions for the end user

Dataflow:



Algorithm: The context of use is identified

Suggestions are searched and retrieved

The suggestions found are presented to the end user

Pseudo-code:

```
ANALYSE context of use
SEARCH FOR suggestions
FOR EACH suggestion
      ADD(suggestion, content)
PRESENT(content)
```

Comments: the suggestions can be related with semantic characteristics of the content, or take into account similar activities and behaviours of other users with similar interests (e.g. e-commerce context).

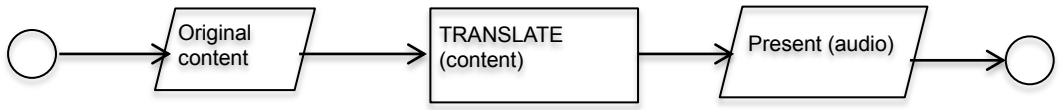
4.1.12 Translate to Audio

Pre-conditions: there is content to be translated

Input: the original content

Output: an audio version of the content

Dataflow:



Algorithm: The content is taken as an input;

The content is analysed;

An audio version is generated;

The audio content is presented to the end user;

Pseudo-code:

```
READ content
TRANSLATE content  // to the audio version
PRESENT(audio)
```

Comments: alternative implementations of this algorithm include: maintaining a repository with audio versions of contents, and then searching and presenting the right version to the end user; reading the alternative text of an image; presenting a song associated with the content.

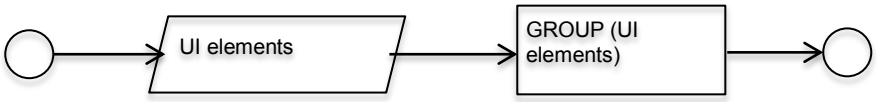
4.1.13 Smart View

Pre-conditions: the UI is composed by different elements

Input: the elements of the UI

Output: the elements grouped according to logical sections (e.g. the topic, or structure of the contents)

Dataflow:



Algorithm: The UI elements are taken as input;

The elements are processed and grouped in logical sections;

The new UI is generated and presented to the end user

Pseudo-code:

```
READ UI elements
GROUP(elements)
```

Comments: the sections can be defined according to the type of the element (e.g. images, text) or according to semantic content (e.g. weather, news).

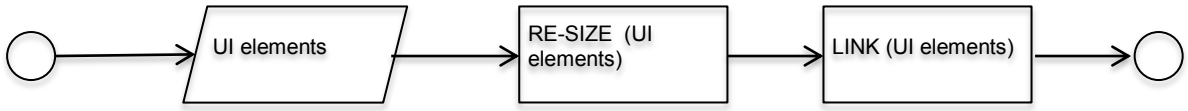
4.1.14 WebThumb (Method)

Pre-conditions: the UI composed by some elements

Input: the elements of the UI

Output: the elements of the UI as thumbs and linked to the original content

Dataflow:



Algorithm: The UI elements are identified;

The elements are processed according to their types (re-sized);

The processed elements are linked to their original content;

The final UI is generated and presented to the end user

Pseudo-code:

```
READ UI elements
PROCESS(UI elements)
LINK (UI element)
```

Comments: images and text content can be reduced to fit in a smaller UI space, and then linked to their original content.

4.2 Adapting Audio Content

This section presents some algorithms that target adaptation of audio contents. [Feiten et al., 2005] motivate the audio adaptation, considering the context of use and defend the need of standards in this domain.

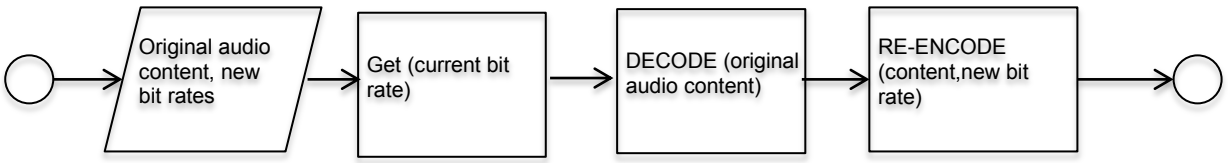
4.2.1 Change Bit Rate

Pre-conditions: there is audio content to be adapted

Input: the original audio content, a value defining the new bit rate (in bps)

Output: the content is presented in a new bit rate

Dataflow:



Algorithm: The original content is taken as an input; the new bit rate is taken as an input;

The current bit rate is identified

The content is decoded

The content is re-encoded with the new bit rate aimed

Pseudo-code:

```
READ content
READ new bit rate //validate, unit
GET current bit rate
DECODE content
REENCODE(content, new bit rate)
```

Comments: analyse the context, define units, check processing / computing capabilities.

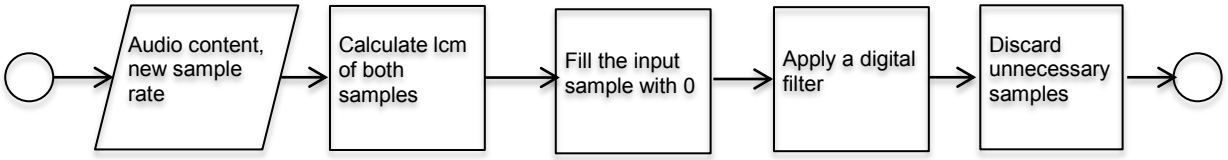
4.2.2 Change Sample Rate

Pre-conditions: there is an audio content available

Input: the original audio content, a new rate for re-sampling it

Output: the content is re-sampled at a new rate

Dataflow:



Algorithm: The original content is taken as an input; the value of the new sample rate is taken as an input;

The current sample rate is identified

The least common multiple of both samples is calculated (current and new)

The value is used to insert 0 to fill the input sample

A digital filter is applied to remove high frequencies

Samples are discarded to generate the output

Pseudo-code:

```
READ content
READ new sample rate  // validate (kHz)
GET current sample rate
CALCULATE lowest common multiple of the two sample rates (current and new)
GET new necessary samples
INTERPOLATE necessary sample values
REPLACE missing samples //with 0
APPLY digital filter //to remove high frequency content
GENERATE the output content by taking every lcm sample
```

Comments: there are different approaches to implement this algorithm, that can be chosen depending on the capabilities of the device[1].

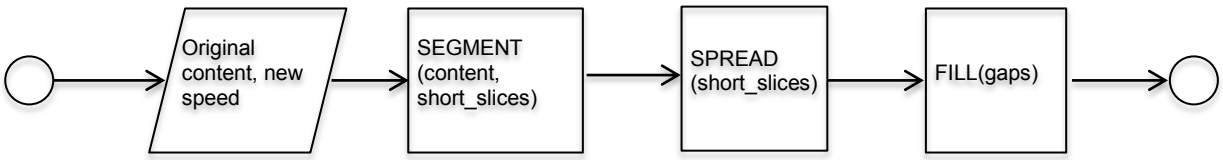
4.2.3 Change Speed

Pre-conditions: there is audio content to be played

Input: the original content, a new rate for speed

Output: the content is played in a new speed

Dataflow:



Algorithm: The original content is taken as an input; the new speed is taken as an input;

The new speed is verified (must be valid)

The content is sliced into short segments

These short segments are spread further apart in time (in case of slower speed)

The gaps generated are filled by duplicating bits of the segments

Pseudo-code:

```
READ content
READ new speed  // must be validated
```

[1] See: http://en.wikipedia.org/wiki/Sample_rate_conversion for further information

```
IF new speed IS VALID THEN
      Content == content.SEGMENT(short_slices)
      Content == content.SPREAD(short_slices)
      Content == content.FILL(gaps)
```

Comments: there is also the approach of modelling which consists in analyse the content at a high level and then reconstruct it with a different speed from its high level description[2].

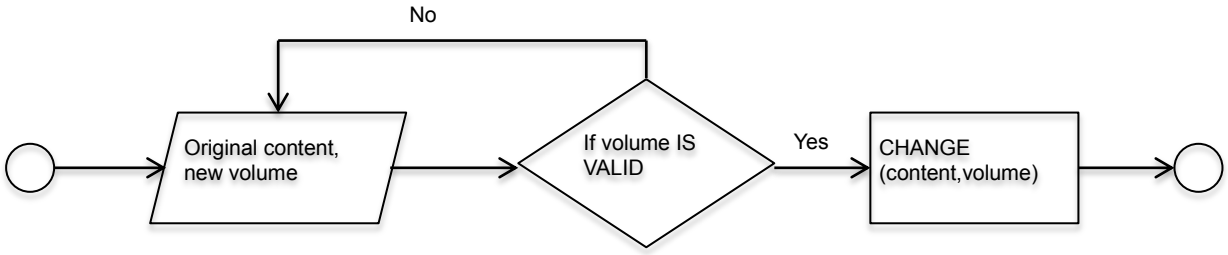
4.2.4 Change Volume

Pre-conditions: there is audio content

Input: the original content, a value defining the new volume for the sound

Output: the content is presented with a new volume

Dataflow:



Algorithm: The audio content is taken as an input; the value for the new volume is taken as an input;

The value is verified (must be valid)

The content is processed to have its volume level adapted accordingly

Pseudo-code:

```
READ content
READ volume  // must be validated
IF volume IS VALID THEN
      CHANGE(content,volume)
```

Comments: volumes too silent may prevent or difficult listening, and volumes too loud may damage the end user hearing capabilities.

4.2.5 Convert Audio Channel (Method)

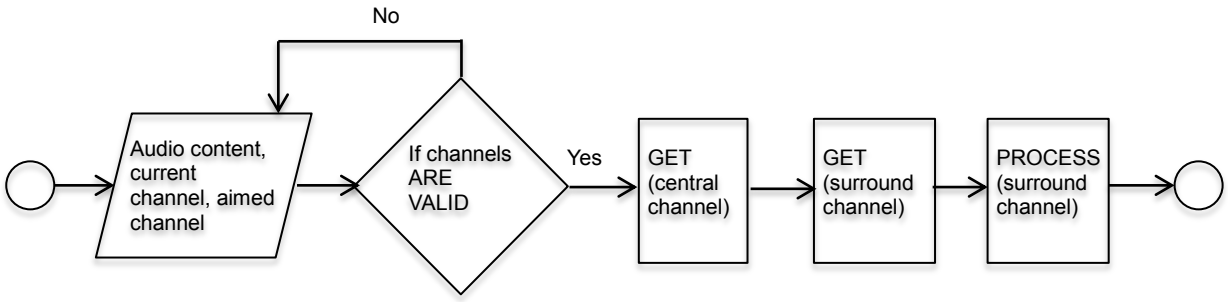
Pre-conditions: there is audio content

Input: the audio content, values defining the current channel and the aimed one

Output: the content is presented with a new channel type

Dataflow:

[2] http://www.seventhstring.com/resources/slowdown.html



Algorithm: The original content is taken as an input; the values of channels are taken as input;

The value is verified (must be valid)

The left and right channels are identified from the original content

The central and surround channels are derived

The central channel is processed with low-pass filters generating 2 channels

The surround channel is processed with time delay, low-pass filter, and phase shifter to generate 2 channels

The content is converted from stereo to 5.1 channel type

Pseudo-code:

```
READ content
READ current channel type
READ aimed channel type  // must be validated
IF channel type IS VALID THEN
      Left_channel == content.EXTRACT(left_channel)
      Right_channel == content.EXTRACT(right_channel)
      Central_channel == content.EXTRACT(left_channel,right_channel)
      Surround_channel == content.EXTRACT(left_channel,right_channel)
      Central_channel == central_channel.LOWPASSFILTER_1()
      LFE_channel == central_channel.LOWPASSFILTER_2()
      RL_channel == surround_channel.PROCESSRL()
      RR_channel == surround_channel.PROCESSRR()
```

Comments: this example targets transformations between stereo and 5.1 channels, clearly there are other possibilities not only for implementation but also channels to be considered. For applications aiming audio contents further algorithms must be also considered. Source: [Chun et al., 2009]

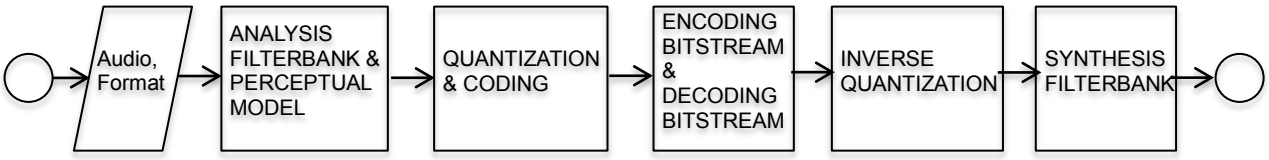
4.2.6 Convert Audio Format

Pre-conditions: there is audio content

Input: the audio content, the current format, and the aimed format for conversion

Output: the audio content with the aimed format

Datflow:



Algorithm: The original content is taken as an input; the value of degrees for rotation is taken as an input;

The value is verified (must be valid)

A Filter bank decomposes the input signal into subsampled spectral components (time/ frequency domains). Together with the corresponding filter bank in the decoder it forms and analysis / synthesis system

The perceptual model uses the time domain input signal or the output of the analysis filter bank, to estimate the actual masking threshold, which is computed using rules known from psychoacoustics and depends on time and frequency

The spectral components are quantized and coded to keep the noise introduced by quantizing but below the masking threshold[3]

A bitstream formatter assemble the bitstream, which typically consists of the quantized and coded spectral coefficients and some side information (e.g. bit allocation information)

The content is rotated according to the value of degrees provided

Pseudo-code:

```
READ audio
READ format
      ANALYSIS FILTERBANK
      PERCEPTUAL MODEL
      QUANTIZATION
      CODING
      ENCODING BITSTREAM
      DECODING BITSTREAM
      INVERSE QUANTIZATION
      SYNTHESIS FILTERBANK
```

Comments: This approach presents the conversion to a MP3 format, clearly it is one among many different possibilities of format conversion, different formats require different processes, the dataflow diagram illustrates an MP3 encoder, designed by [Brandenburg, 1999]. The algorithm is described in a high abstraction level. Algorithm in Java[4].

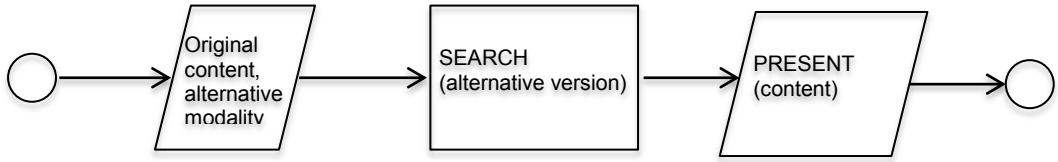
4.2.7 Translate Audio Modality

Pre-conditions: there is audio content and an alternative

Input: the audio content, the alternative modality

Output: the content is presented in a different modality

Dataflow:



Algorithm: The content is taken as an input; the alternative modality is taken as input too

The content is analysed;

An alternative version is searched;

If found the alternative content is presented to the end user;

Pseudo-code:

```
READ content, alternative modality
SEARCH content  // alternative version
PRESENT(content.alternative_version)
```

Comments: alternative implementations can be automatically generated depending on the type of modality

[3] Depending on the algorithm this step can be performed in many different ways, from simple block companding to analysis-by-synthesis systems using additional noiseless compression

[4] http://download.oracle.com/javase/tutorial/sound/converters.html

aimed.

4.2.8 Translate Audio Language

Pre-conditions: there are audio content and also additional versions of it in other languages

Input: the audio content, the aimed language

Output: the content is presented in a different language

Dataflow:

Algorithm: The content is taken as an input; the aimed language is taken as input

An alternative version of the content in the aimed language is searched;

If found the content in the aimed language is presented to the end user;

Pseudo-code:

```
READ content, aimed language
SEARCH content  // aimed language
PRESENT(content.aimed_language)
```

Comments: it requires an extensive effort to produce and maintain multiple versions of the content in multiple languages; automated tools can be considered, however the results may not be so precise.

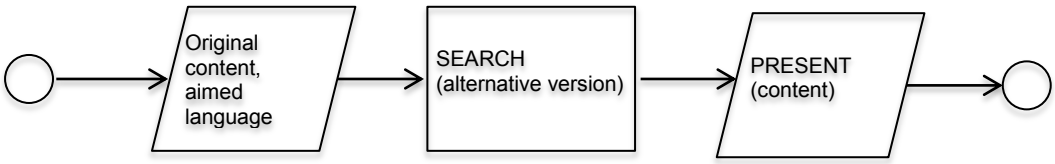
4.2.9 Simplify

Pre-conditions: there are audio content and also additional versions of it in different difficulty levels

Input: the audio content (or a reference to it)

Output: the content is presented in a simplified version

Dataflow:



Algorithm: The content is taken as an input;

An alternative version of the content in a simplified version is searched;

If found the simplified content is presented to the end user;

Pseudo-code:

```
READ content
SEARCH simplied(content)
PRESENT(content.simple_version)
```

Comments: a repository must be maintained with alternative versions of the same content in different difficulty levels; automated algorithms for simplification may be considered too.

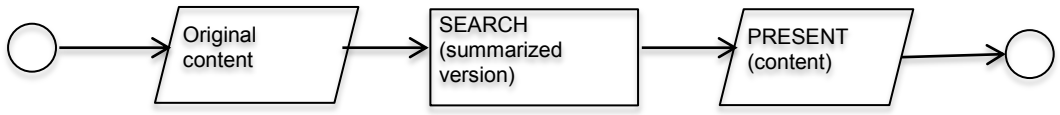
4.2.10 Summarize

Pre-conditions: there is audio content and also a summarized version of it

Input: the audio content

Output: a summary of the content is presented

Dataflow:



Algorithm: The content is taken as an input

A summarized version of the content is searched;

If found the summary of the content is presented to the end user;

Pseudo-code:

```
READ content
SEARCH summary of the content
      IF summary of the content IS FOUND THEN
            PRESENT(content.summary)
```

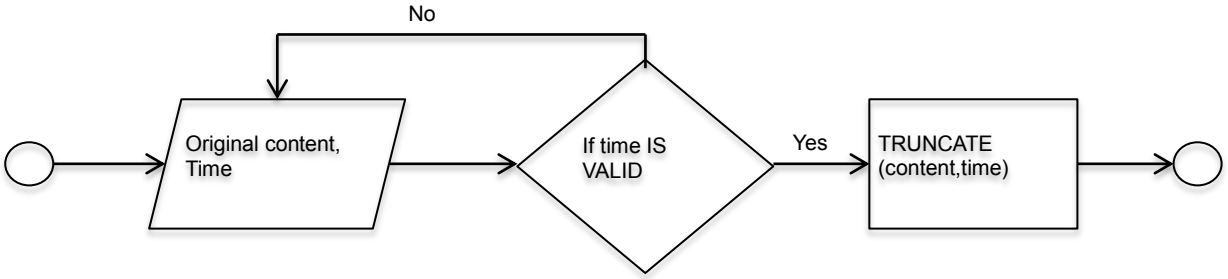
4.2.11 Truncate

Pre-conditions: there is audio content

Input: the audio content, a value defining the criteria to truncate it (e.g. time in seconds)

Output: the content is presented truncated

Dataflow:



Algorithm: The original content is taken as an input; the time to truncate it is taken as an input;

The value is verified (must be greater than 0 and lower than the end time of the content)

The content is truncated according to the time provided, and presented to the end user

Pseudo-code:

```
READ content
READ time  // must be validated, e.g. value in seconds
IF time > 0 AND time < content.duration THEN
      TRUNCATE(content,time)
PLAY(content)
```

Comments: other criteria can be also defined to truncate the content, e.g. the amount of words spoken, in this case the audio content must be analysed and properly processed.

4.3 Adapting Text

The Figure 1 presents a catalog of 7 techniques to adapt text content regarding its appearance. The text content can be replaced by alternatives, such as a video or an image, or its properties may be modified, for instance its size, color, style (font type), effects, length (truncation) or compression. This figure provides an

overview of possibilities to adapt text regarding its appearance, however there are other alternatives that may be also considered in this domain. The adaptation techniques listed regarding size and truncation, corresponds to graceful degradation (following the horizontal axis to the right direction) or progressive enhancement (following the horizontal axis to the left direction).

Besides the techniques illustrated by Figure 1, there are further possibilities to perform adaptation regarding aspects from text other than aesthetics. For instance, regarding semantic aspects of a text content, it can be summarized, described, explained or simplified.

This section provides detailed algorithms that cover some of the possible adaptation techniques for text content, these techniques are more common of use and relevant in the context of Serenoa project. Additional algorithms will be considered, included and devised too in the next phases of the project.

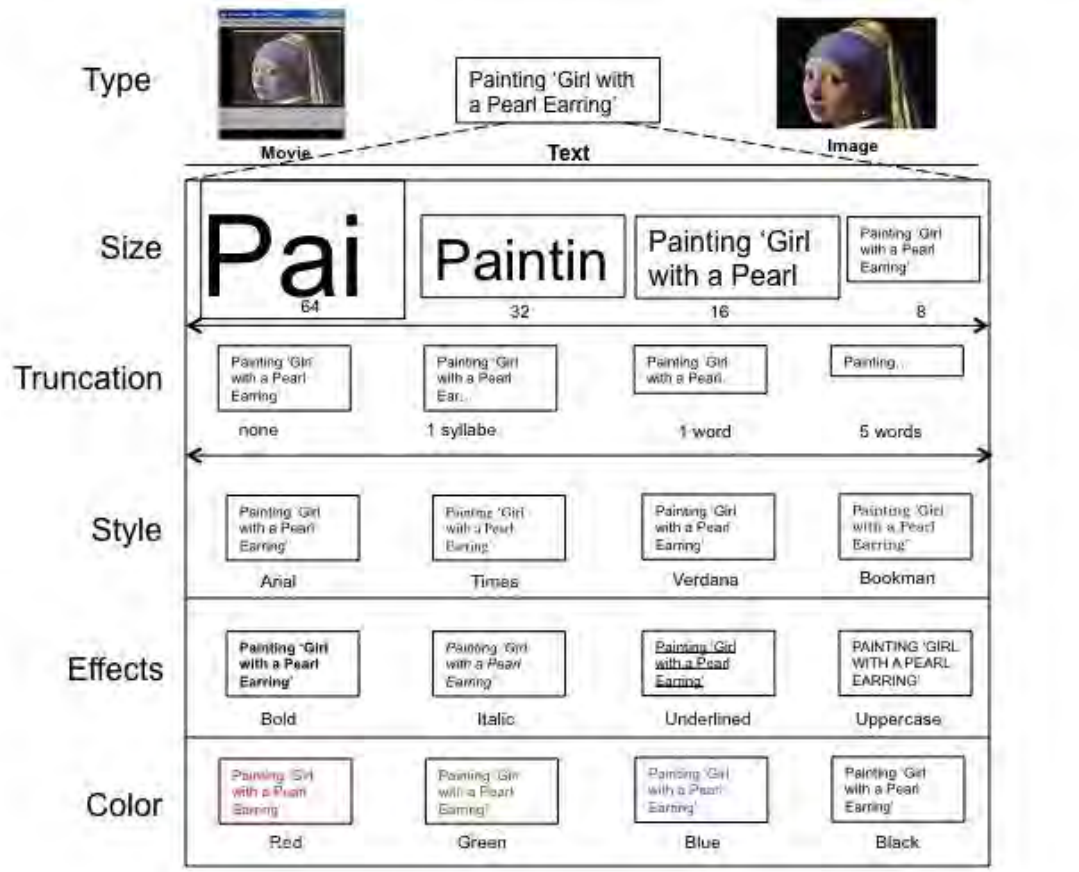


Figure 1. Set of adaptation techniques concerning different properties of a text

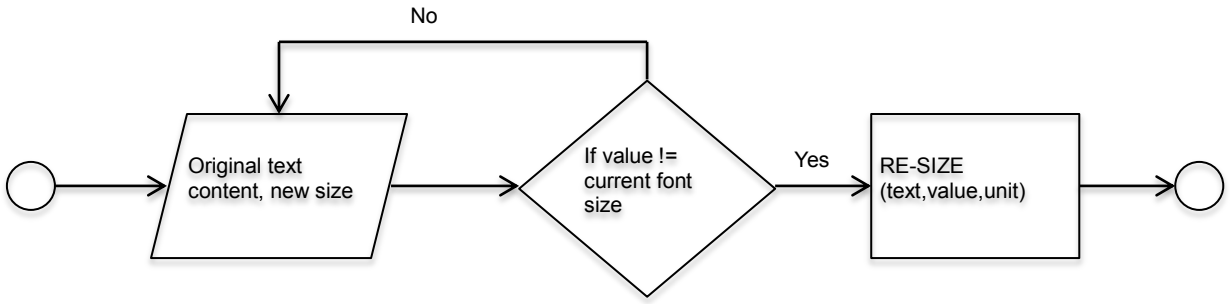
4.3.1 Re-size

Pre-conditions: there is a text content to be adapted

Input: the original text content, a value defining the new size for the font (e.g. 10), a unit for this value (e.g. px, cm, %)

Output: the text content is presented in a new font size

Dataflow:



Algorithm: The text content is taken as an input; the value and unit for re-sizing is taken as an input;

The text is checked to verify if its length has at least one character

The value is checked, it must be different from the font size of the original text content

The font size is changed to the new value and presented to the end user

Pseudo-code:

```
READ text content
READ value  // new font size, positive value, >0
READ unit  // px, cm, %
IF text_content.length > 0 AND value != text.font_size THEN
      RESIZE(text, value, unit)
```

Comments: depending on the new size selected the text might be unreadable, sizes that are too small may prevent reading, and sizes too big may affect the layout and also affect the readability, in this case compatible techniques such as, fish-eye (distorted view), re-distribution, pagination or scroll bars must be considered.

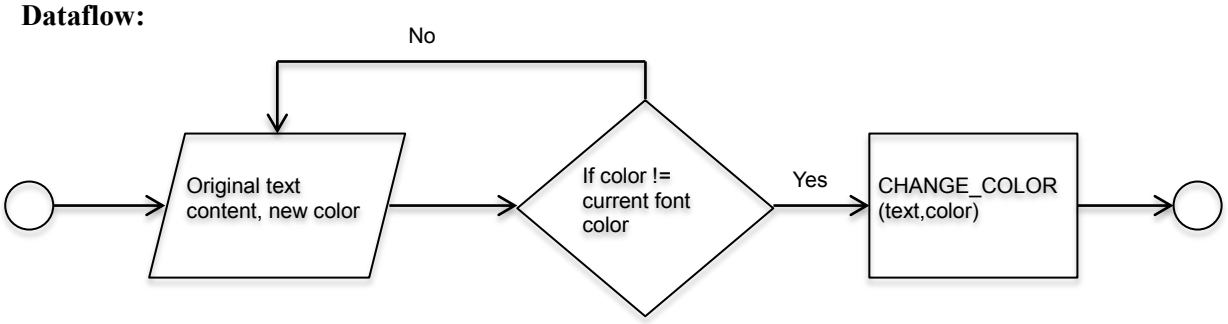
4.3.2 Change Color

Pre-conditions: there is a text content to be adapted

Input: the original text content, a value[5] defining the new color for the text (e.g. blue, #0011FF, RGB(213,111,56))

Output: the text content is presented according to the color specified as input

Dataflow:



Algorithm: The text content is taken as an input; the value for the new coloring is taken as an input;

The text is checked to verify if its length has at least one character

The current color of the text content is checked to verify if its different from the one provided

The font color is adapted according to the new color provided

The adapted text is presented to the end user

[5] There are many different notations for color systems; RGB itself can be represented by a triplet of number, percentages, digital 8-bit per channel, or digital 16-bit per channel. CMYK and HSV are other alternative options.

Pseudo-code:

```
READ text content
READ value  // new font color, must be validated
IF text_content.length > 0 AND value != text.font_color THEN
      CHANGE_COLOR(text, value)
```

Comments: depending on the contrast between the new color selected and the background, the text might be unreadable, in this case the contrast level must be checked and maintained at an appropriate level.

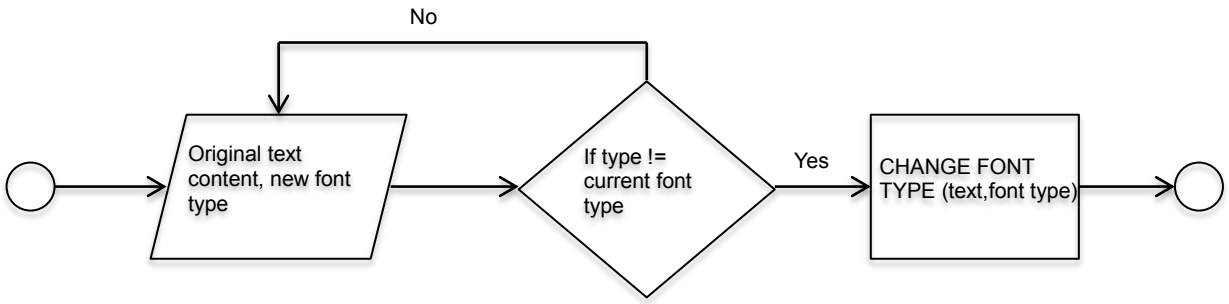
4.3.3 Change Font Type (Style)

Pre-conditions: there is a text content to be adapted, the font type is available and compatible for the context

Input: the original text content, a value defining the new type of the font (e.g. Verdana, or Sans-serif)

Output: the text content is presented in a new font type

Dataflow:



Algorithm: The text content is taken as an input; the value for the new font type is taken as an input;

The text is checked to verify if its length has at least one character

The current font type of the text content is checked to verify if its different from the one provided

The font type is adapted according to the new type provided

The adapted text is presented to the end user

Pseudo-code:

```
READ text content
READ value  // new font type, must be validated and accepted
IF text_content.length > 0 AND font_type != text.font_type THEN
      CHANGE_FONT_TYPE(text, font_type)
```

Comments: the font type must be available and compatible with the specifications of the system of the end user; certain types of font may affect readability and change the layout of the content.

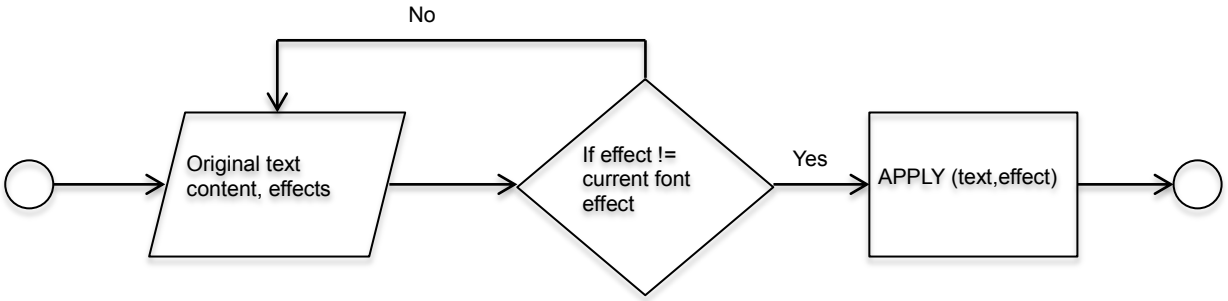
4.3.4 Change Effects

Pre-conditions: there is a text content to be adapted

Input: the original text content, one or more values defining the new effect(s) for the font (e.g. italic or bold)

Output: the text content is presented with the new effects

Dataflow:



Algorithm: The text content is taken as an input; the effect name(s) is(are) taken as input(s);

The text is checked to verify if its length has at least one character

The effect name is checked, it must be different from the effects already applied to the original text content

The effect is applied to the text content and presented to the end user

Pseudo-code:

```
READ text content
READ effect // e.g. bold, italic
IF text_content.length > 0 AND effect != text.effect THEN
      APPLY(text, effect)
```

Comments: depending on the effect type, the layout may be affected. Possible effects include: alignment (left, right, centred, justified); spacing, serif, transparency, orientation (in terms of degree) and highlight.

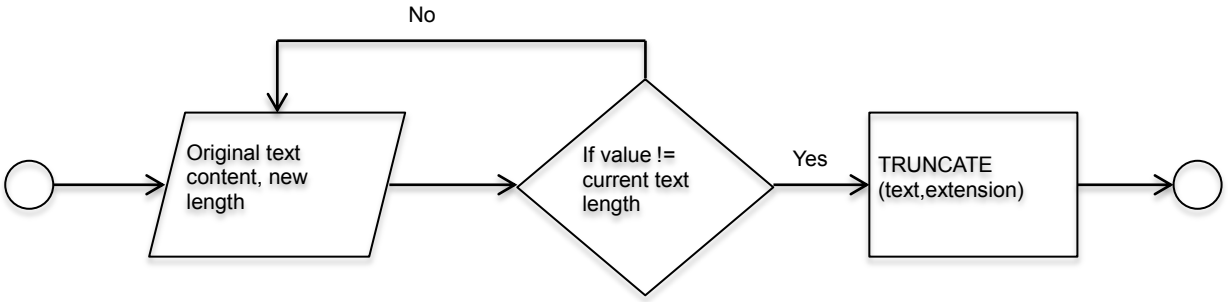
4.3.5 Truncate

Pre-conditions: there is a text content to be adapted

Input: the original text content, a value defining the extension of the text

Output: the text content is presented in a new font size

Dataflow:



Algorithm: The text content is taken as an input; the extension for truncation is taken as an input;

The text is checked to verify if its length has at least two characters

The extension is checked, it must be greater than the current length of the original text content

The length of the text is changed according to the new extension and presented to the end user

Pseudo-code:

```
READ text content
READ extension  // in terms of characters, syllables, words
IF extension != text.length THEN
      TRUNCATE(text, extension)
```

Comments: the extension may be defined in terms of characters, syllables, words, sentences, and so on. The content truncated can be completed with symbols indicating and providing access links to its continuity (such

as … or ->).

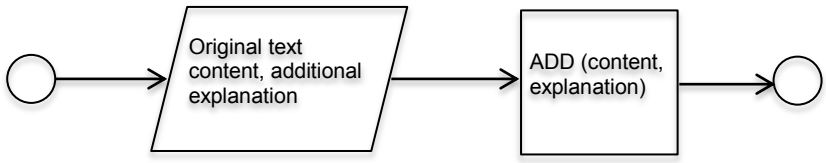
4.3.6 Add Explanation

Pre-conditions: there is additional content about a concept

Input: the original text content, an additional explanation

Output: the text content is associated with the additional explanation

Dataflow:



Algorithm: The text content is taken as an input; the additional explanation is taken as an input;

An association is created to present to the end user access to the additional content

Pseudo-code:

```
READ text content
READ additional explanation
ADD_LINK(content, explanation)
```

Comments: the additional explanation can be manually read or retrieved automatically (e.g. with webservices), the access can be provided with links, icons, and presented in additional pages or sections for content.

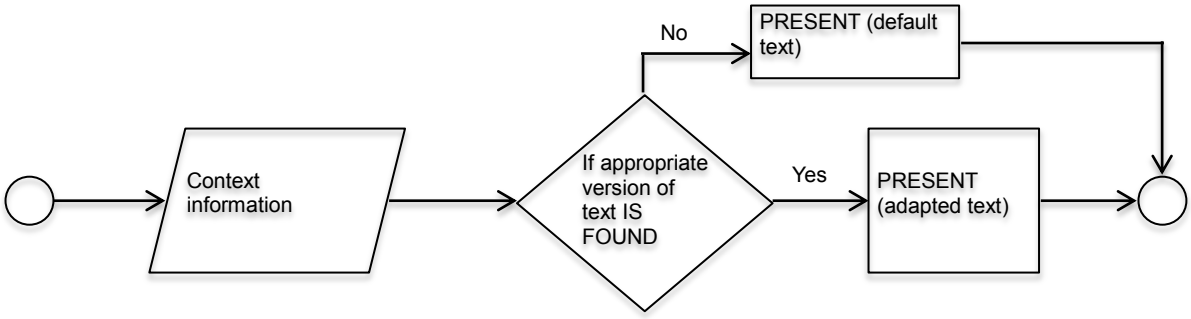
4.3.7 Altering Fragments (aka Explanation Variants)

Pre-conditions: there are alternative versions of the text content, and associations of them with the context

Input: context information

Output: the text content is presented in an appropriate version for the context of use

Dataflow:



Algorithm: The context information is taken as an input;

An appropriate version according to the context is searched //e.g. adapted to the expertise level of the user

If an appropriate version is found then it is presented to the user, else a default version is presented

Pseudo-code:

```
READ context information
SEARCH(text, context)
IF appropriate_version IS FOUND THEN
      PRESENT(appropriate_text)
ELSE
```

```
      PRESENT(default_text)
```

Comments: there are different context information that can be associated with text contents, for instance the level of expertise of the user may be taken into account to select an adequate version of text content for her.

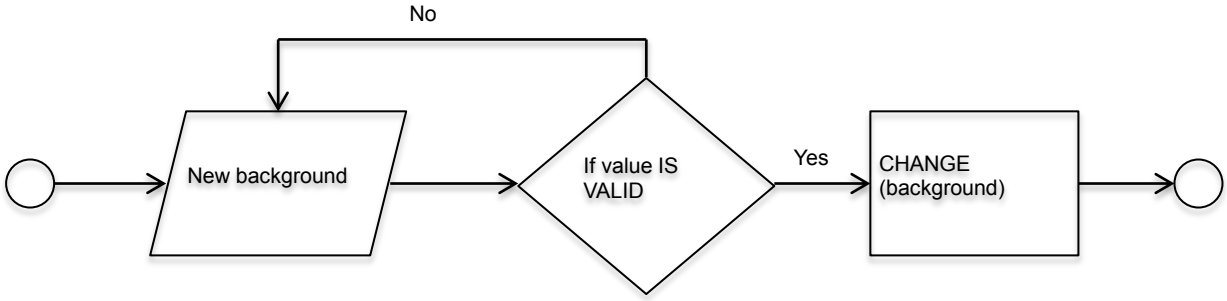
4.3.8 Background

Pre-conditions: there is a background to be adapted

Input: the new background value

Output: the background is presented adapted

Dataflow:



Algorithm: The new background value is taken as an input;

The value is validated

The new background value is applied to the interface and it is presented to the user

Pseudo-code:

```
READ background
IF background IS VALID THEN
      CHANGE(background)
```

Comments: the background can be set in terms of color, texture, images, brightness level, etc according to the context of use or user preferences.

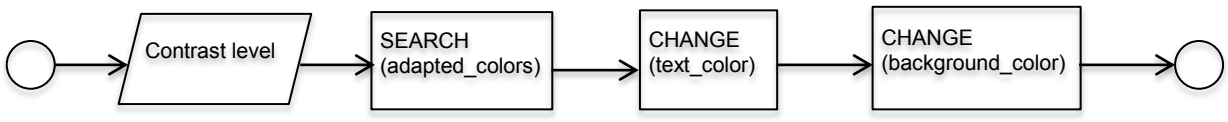
4.3.9 Contrast (Method)

Pre-conditions: there is a definition of specific colors (or brightness levels) for contrast

Input: the contrast level

Output: the contrast between the text content and the background is adapted

Dataflow:



Algorithm: The contrast level is taken as an input;

Specific colors (values or brightness levels) are searched for the contrast level defined;

The text color is changed;

The background color is changed too;

Pseudo-code:

```
READ contrast level
SEARCH(contrast, text_color, background_color)
```

```
IF text_color IS FOUND AND background_color IS FOUND THEN
      CHANGE(text_color);
      CHANGE(background_color);
```

Comments: there are alternative versions to implement this method, for instance changing the tones of the current colors instead of modifying the color itself.

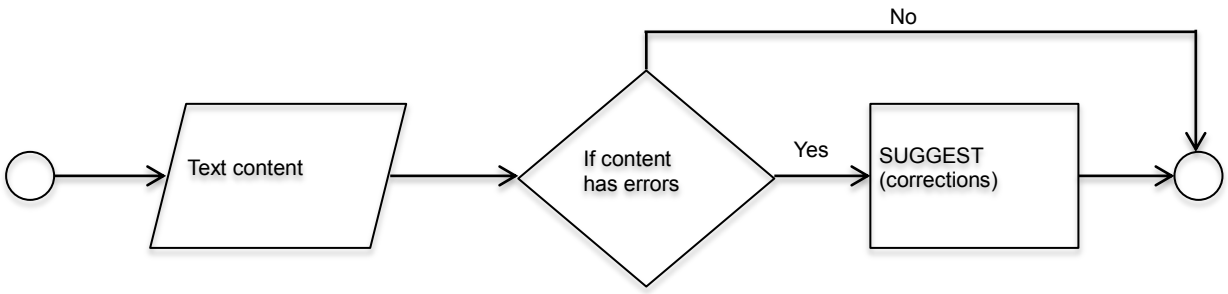
4.3.10 Correct

Pre-conditions: there is a text content and rules to validate it

Input: text content

Output: suggestion of correction

Dataflow:



Algorithm: The text content is taken as an input;

The text is verified according to grammar rules, orthographic rules, and so on

If errors were found the user is notified and suggestions (if existing) are provided

Pseudo-code:

```
READ text content
VERIFY(text,grammar)
IF text HAS ERRORS THEN
      NOTIFY(errors)
      SUGGEST(corrections)
```

Comments: there are different aspects of the text that can be validated, e.g. grammar, and orthography. The end user can be notified, or have suggestions of corrections presented, to accept or reject.

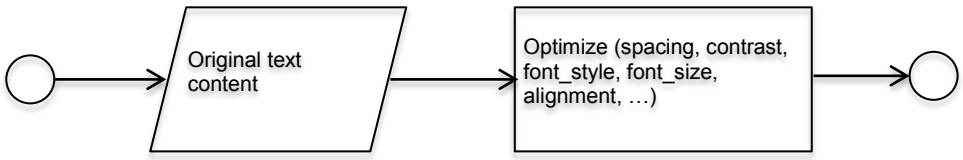
4.3.11 Readability (Method)

Pre-conditions: there is a text content to be adapted

Input: the original text content

Output: the text content is presented with better readability level

Dataflow:



Algorithm: The text content is taken as an input;

The spacing is optimized;

The contrast level is optimized;

The font style is optimized;

The font size is optimized;

The alignment is optimized; …

The content is presented to the end user;

Pseudo-code:

```
READ text content
OPTIMIZE(contrast, font_size, font_style, spacing, alignment, …)
```

Comments: by optimize we mean use the space available in the UI as efficiently as possible, in order to increase the readability level. Clearly further definitions are necessary to implement this algorithm for different contexts of use.

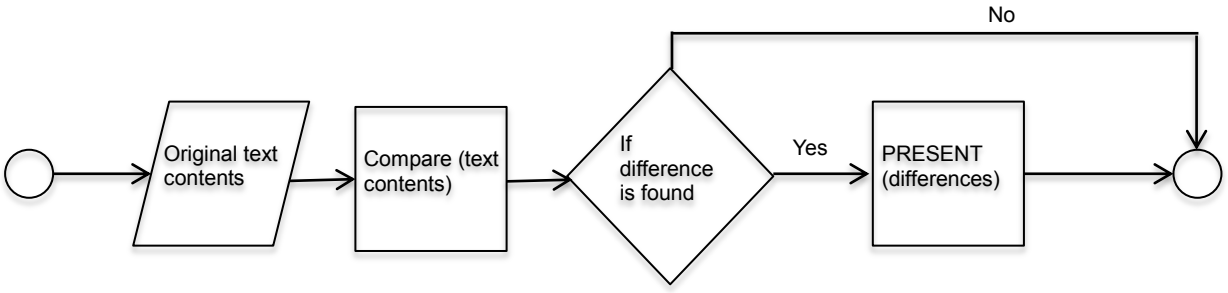
4.3.12 Comparative Explanation (aka Compare)

Pre-conditions: there are two or more text contents and given criteria to compare them

Input: the two text contents

Output: a comparison between contents

Dataflow:



Algorithm: The text contents are taken as input;

The text contents are compared

The results of the comparison process are presented to the end user

Pseudo-code:

```
READ text contents
COMPARE(text contents)
PRESENT(results)
```

Comments: the comparison can be done in terms of identical concepts presented (such as words, sentences, or sections), the differences can be highlighted to the end user. Conversely, the similarities can also be remarked. Graphics can be also used to illustrate the comparison results.

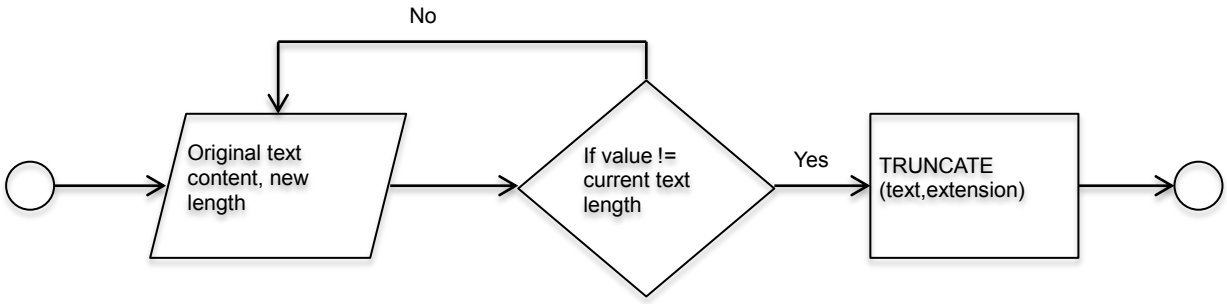
4.3.13 Describe

Pre-conditions: there are additional descriptions to the terms (or expressions) of a text

Input: part of a text content (e.g. one word)

Output: a description about the term selected

Dataflow:



Algorithm: Part of a text content is taken as an input (e.g. a word);

A description about the term is searched;

If the description is found then it is presented to the end user

Pseudo-code:

```
READ term
SEARCH (term)
IF description (term) IS FOUND THEN
      PRESENT description(term)
```

Comments: the descriptions can be presented in a new interface (page), or in an alert window (popup), these specification may take into account user preferences.

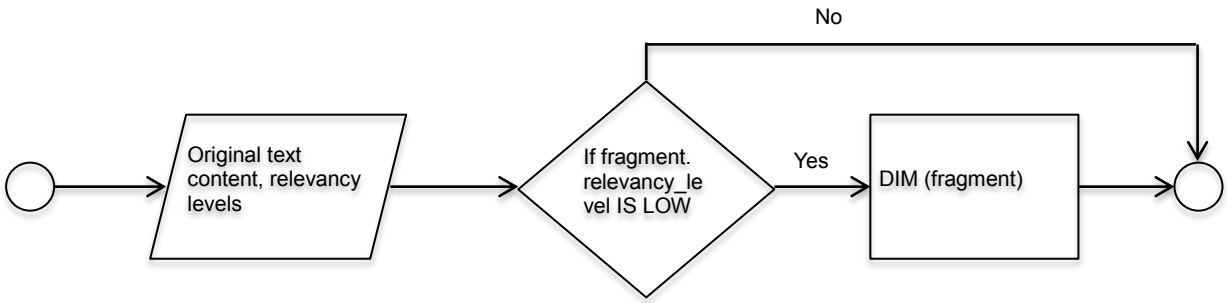
4.3.14 Dim Fragments

Pre-conditions: there is a text content to be adapted, and a definition of the relevance levels of its parts

Input: the original text content, the relevance level

Output: the text content is presented with fragments of it dimmed according to its relevance level

Dataflow:



Algorithm: The text content is taken as an input; the relevance level of its fragments is taken as an input;

The text is processed;

Less relevant fragments are dimmed and presented to the end user;

Pseudo-code:

```
READ text content
READ relevancy levels  // according to the context of use
IF fragment.relevance_level IS LOW THEN
      DIM(fragment)
```

Comments: the context of use defines the relevance level of each fragment of the text content. Multiple levels of relevance can be also defined and considered to dim the fragments proportionally.

4.3.15 Explanation Variants

See Altering Fragments

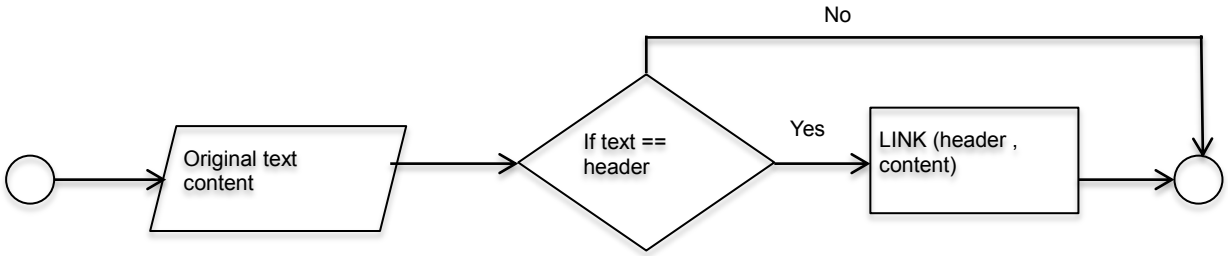
4.3.16 Outlining

Pre-conditions: there is a text content to be adapted

Input: the original text content (semantically structured)

Output: only the headers of the content are present (linked to the original text)

Dataflow:



Algorithm: The text content is taken as an input;

The headers are identified;

The header become links to the original content;

Only the headers are presented to the end user;

Pseudo-code:

```
READ text content
SEARCH(headers)
IF header IS FOUND THEN
      LINK(header)
```

Comments: the techniques reduces the space of the UI required to present the content, thus the original layout may be affected requiring the execution of additional techniques.

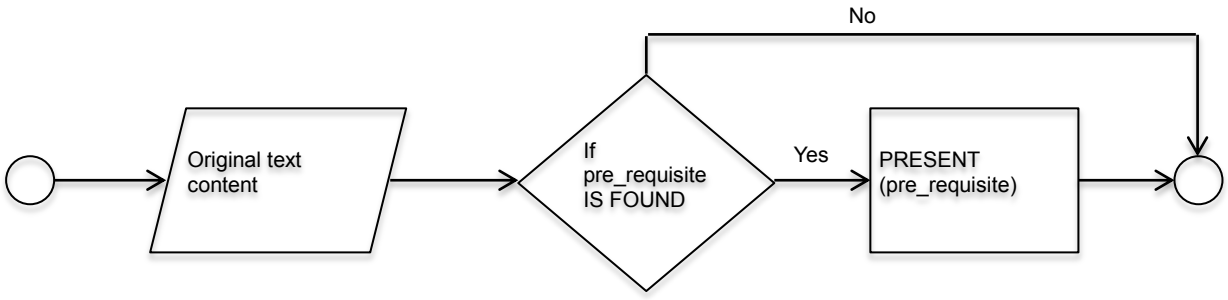
4.3.17 Pre-requisite

Pre-conditions: there is pre-requisite content available

Input: the original text content, context information

Output: the pre-requisite content to understand the text is presented to the end user

Dataflow:



Algorithm: The text content is taken as an input;

The text is checked to verify if there is pre-requisite content available

If pre-requisite content is found, then it is presented to the end user

Pseudo-code:

```
READ text content
SEARCH(pre-requisite) // in terms of context of use
IF pre-requisite IS FOUND THEN
      PRESENT(pre-requisite)
```

Comments: according to the context of use it may be interesting to have alternative versions of the pre-requisite content (e.g. different levels of details or complexity).

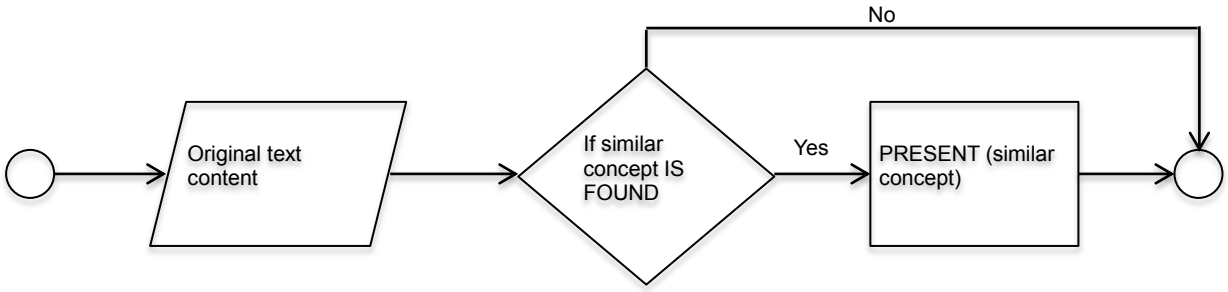
4.3.18 Similarity

Pre-conditions: there is a text content associated with similar concepts

Input: the original text content

Output: concepts that are similar to the original concept

Dataflow:



Algorithm: The text content is taken as an input;

Similar concepts are searched;

If similar concept is found, then it is presented to the end user

Pseudo-code:

```
READ text content
SEARCH (similar concept)
IF similar concept IS FOUND THEN
      Present (similar concept)
```

Comments: similar concepts may take into account semantic information about the original text, but also additional concepts, and alternative formats (e.g. news articles, pictures, music, etc).

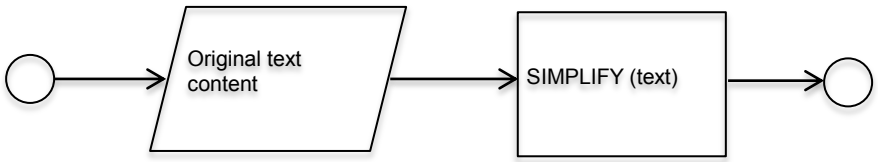
4.3.19 Simplify

Pre-conditions: there is a text content to be simplified

Input: the original text content

Output: simplified version of the text

Dataflow:



Algorithm: The text content is taken as an input;

The text is simplified;

The simplified version of the text is presented to the end user

Pseudo-code:

```
READ text content
SIMPLIFY (text content)
PRESENT(simplified_version)
```

Comments: there are algorithms to perform simplification of text already developed and reported in the literature, the levels of simplicity (or complexity) may take into account the user profile and also the context of use. Simplification can be performed by means of web services or manually. An alternative approach to implement this adaptation technique consists in having alternative versions of the same text, with different complexity levels, available in a repository, to be retrieved later

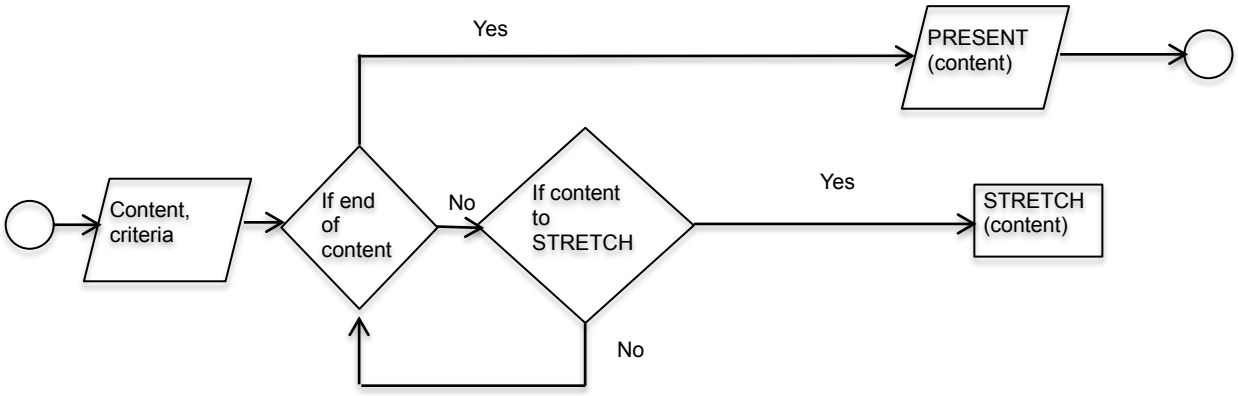
4.3.20 Stretch

Pre-conditions: there is text content to be stretched

Input: text content, criteria defining when to stretch it

Output: the content is presented with parts of it stretched

Dataflow:



Algorithm: The content is taken as an input; the criteria for the new layout are taken as input;

The content is analysed and processed accordingly

The adapted content is presented to the end user

Pseudo-code:

```
READ content
READ criteria
FOR EACH content_part
      IF content_part TO_STRETCH THEN
            STRETCH(content_part)
PRESENT(content)
```

Comments: the criteria can take into account for instance semantic aspects of the content (e.g. stretching contents related to the context of use).

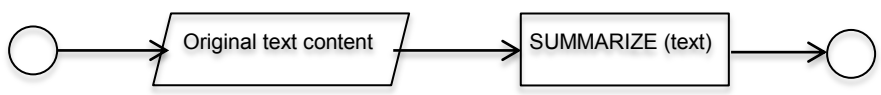
4.3.21 Summarize

Pre-conditions: there is a text content to be summarized

Input: the original text content

Output: a summary of the text

Dataflow:



Algorithm: The text content is taken as an input;

The text is summarized;

The summary of the text is presented to the end user

Pseudo-code:

```
READ text content
SUMMARIZE (text content)
PRESENT(summary)
```

Comments: there are algorithms to perform summarization of text already developed and reported in the literature, usually the text content is analysed, and according to the relevance level of the sentences and words, they may be removed or maintained. Different levels of details can be considered, always taking into account the actual context of use. Summaries can be generated by means of web services or manually. An alternative approach to implement this adaptation technique consists in having alternative versions of the same text, with different detail levels, available in a repository, to be retrieved later

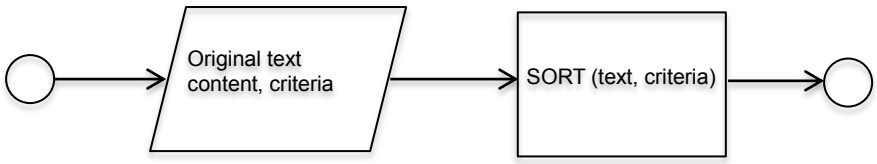
4.3.22 Sort

Pre-conditions: there is a text content to be sorted and criteria for sorting it

Input: the original text content, the criteria to sort data

Output: the text content is presented sorted

Dataflow:



Algorithm: The text content is taken as an input; the criteria to sort it are taken as input;

The text is processed to sort it according to the criteria

The sorted text is presented to the end user

Pseudo-code:

```
READ text
READ criteria  // e.g. alphabetical order, chronological, ascendant
SORT(text,criteria)
```

Comments: simple data can be sorted in alphabetical, numerical, chronological, orders. Complex data may require more than one criterion to be sorted.

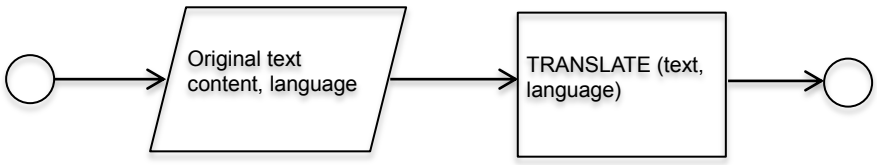
4.3.23 Translate

Pre-conditions: there is a text content to be translated

Input: the original text content, the language in which the text will be translated

Output: the text content is presented in a different language

Dataflow:



Algorithm: The text content is taken as an input; the language for translation is taken as an input;

The text is translated according to the language provided

The text is presented to the end user in a different language

Pseudo-code:

```
READ text
READ language  // must be validated
TRANSLATE(text, language)
```

Comments: alternative approaches to implement this technique include storing the content in alternative languages in the repository, searching and retrieving it according to the context of use; or using web services to provide automatic translation of content, it is worthy to notice that automatic translations not always produce good results.

4.4 Adapting Images

The Figure 2 illustrates a set of techniques to adapt images regarding its appearance. Images can be replaced by alternatives, such as a video or textual description, according to the context of use; besides an image may have its properties modified, for instance regarding its size, color, format, resolution, truncation or compression level. This figure considers alternatives for graceful degradation along its horizontal axis following the right direction, e.g. while the movie format requires more capabilities of the device in use, a text alternative is simpler to be rendered; the same applies for each of the properties illustrated below. Following the horizontal axis to the left direction provides alternatives for progressive enhancement.

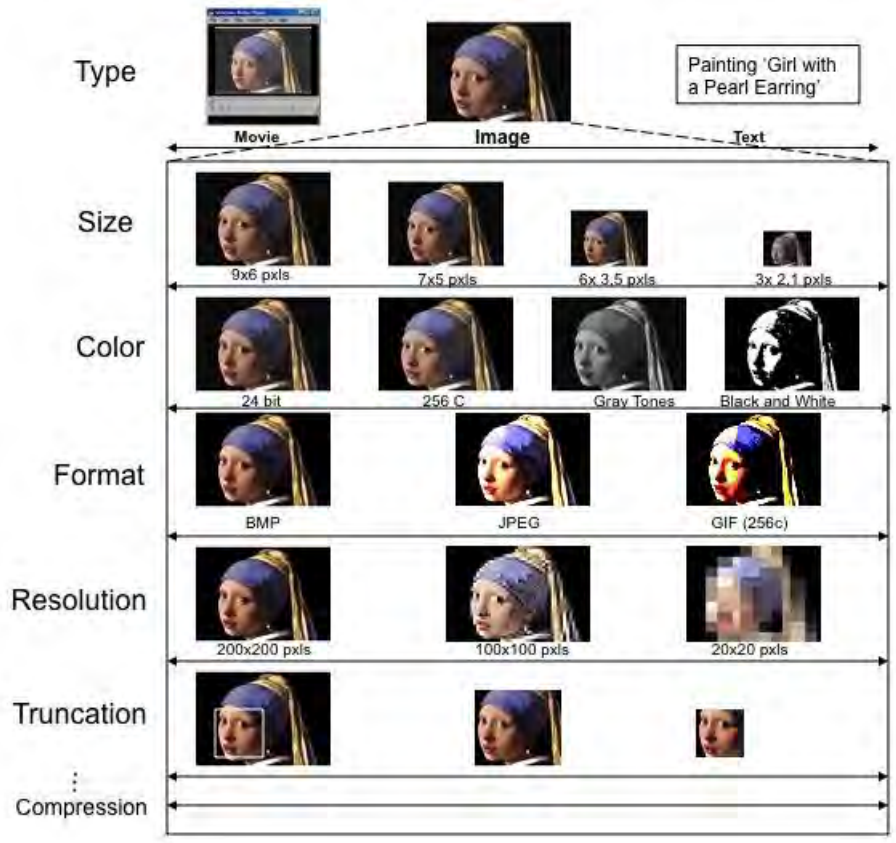


Figure 2. Set of adaptation techniques concerning different properties of an image [Based on: Ostachkov, 2001]

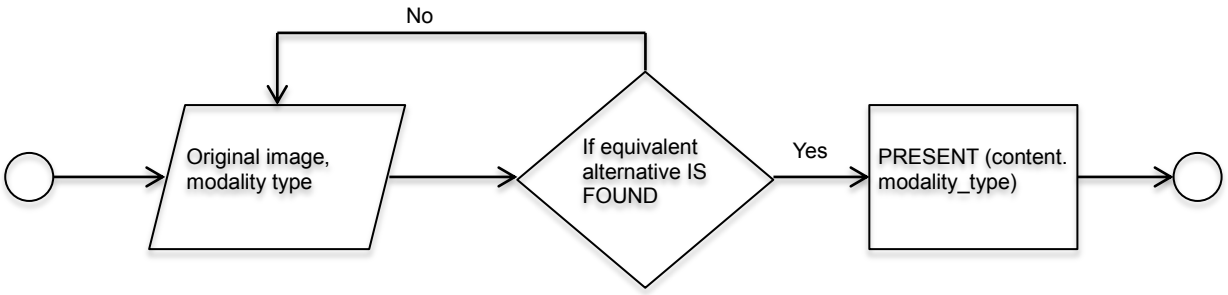
4.4.1 Change Modality Type

Pre-conditions: the application has one or more images available in its content, and a database with content of alternative modalities associated with the original image

Input: the original image, the aimed modality

Output: content in the alternative modality type specified

Dataflow:



Algorithm: The image is taken as an input; the alternative modality type is taken as an input;

The alternative modality type is checked

If an alternative equivalent to the original image is found then

The new content type is presented to the end user

Pseudo-code:

```
READ image file
READ modality_type  // video, text, audio; must be validated
SEARCH content.modality_type
IF alternative modality type for the image is found THEN
      PRESENT(content.modality_type)
```

Comments: to search for alternative modalities based on an image, it is necessary to rely on semantic descriptions of it, or analyse its properties (color, shapes); the alternative modalities provided as inputs must be validated; the final layout may be impacted; each modality may have specific requirements to be accessed (e.g. players and codecs for videos, speakers for audio, etc.).

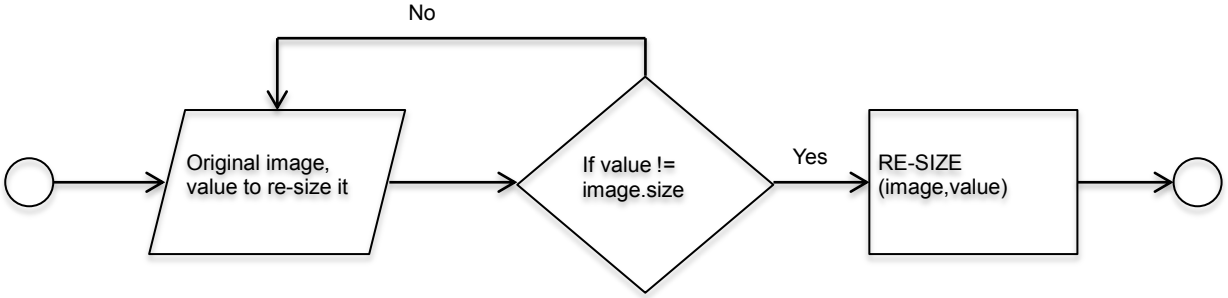
4.4.2 Change Size (aka Enlarge, Reduce)

Pre-conditions: the application has one or more images available in its content

Input: the original image, a value defining the new size (e.g. -10%, or 7x5 pixels)

Output: the image re-scaled according to the input parameters

Dataflow:



Algorithm: The image is taken as an input; the value indicating the re-size ratio is taken as an input;

The value for the re-size is checked, it must be different than the original dimension of the image

The image is read as a matrix of pixels

The pixels are processed and re-scaled, according to the input parameters (e.g. – 10 %)

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
READ value  // e.g. - 10 %
IF value != image.size THEN
   New_image == image.RESIZE(value)
```

Comments: the adapted image may affect the layout, requiring for instance re-distribution of the content, scrolling, or pagination. Images too reduced or too large may become inaccessible.

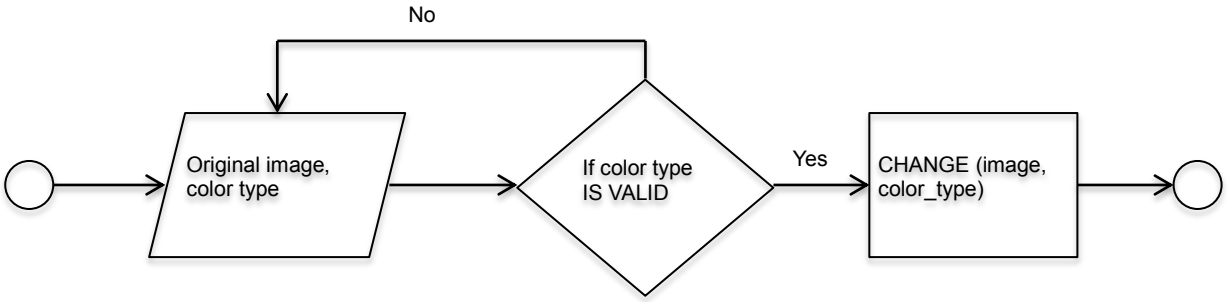
4.4.3 Change Color Type

Pre-conditions: the application has one or more images available in its content

Input: the original image, a definition of the new color type

Output: the image is presented in a new color type

Dataflow:



Algorithm: The image is taken as an input; and the new color type is taken as an input;

The new color type is checked (validated)

The image is processed accordingly

The new image is presented to the end user

Pseudo-code:

```
READ image file
READ color type  // 256, black and white, gray tones
IF color type IS VALID THEN
      CHANGE(image,color_type)
```

Comments: there are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same image but different color types, and then searching and retrieving the right one to present it to the end user.

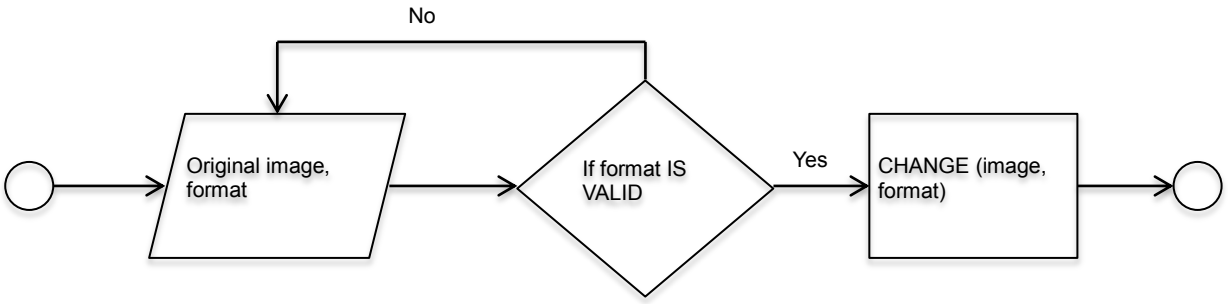
4.4.4 Change Format

Pre-conditions: the application has one or more images available in its content

Input: the original image, a definition of the new format

Output: the image is presented in a new format

Dataflow:



Algorithm: The image is taken as an input; and the new format is taken as an input;

The new format is checked (validated)

The image is processed accordingly

The image in the adapted format is presented to the end user

Pseudo-code:

```
READ image file
READ format  // JPEG, BMP, PNG
IF format IS VALID THEN
      CHANGE(image,format)
```

Comments: there are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same image but different formats, and then searching and retrieving the right one to present it to the end user.

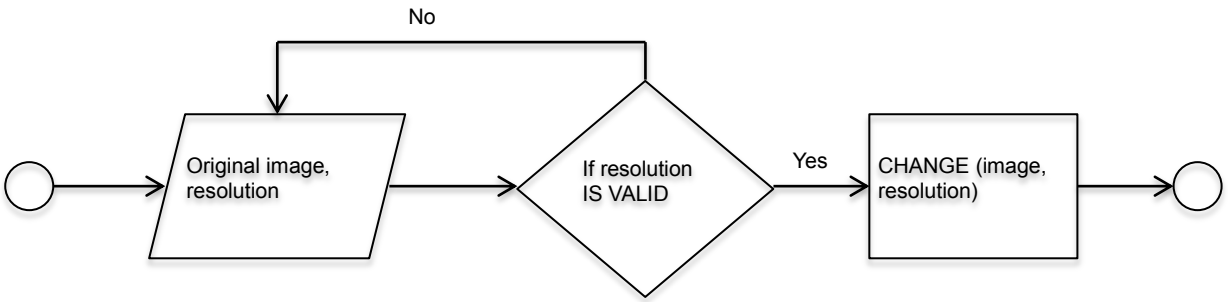
4.4.5 Change Resolution

Pre-conditions: the application has one or more images available in its content

Input: the original image, a definition of the alternative resolution

Output: the image is presented in a new resolution

Dataflow:



Algorithm: The image is taken as an input; and the new resolution is taken as an input;

The new resolution is checked (validated)

The image is processed accordingly

The image in the adapted resolution is presented to the end user

Pseudo-code:

```
READ image file
READ resolution  // must be validated
```

```
IF resolution IS VALID THEN
      CHANGE(image,resolution)
```

Comments: there are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same image but different resolutions, and then searching and retrieving the right one to present it to the end user.

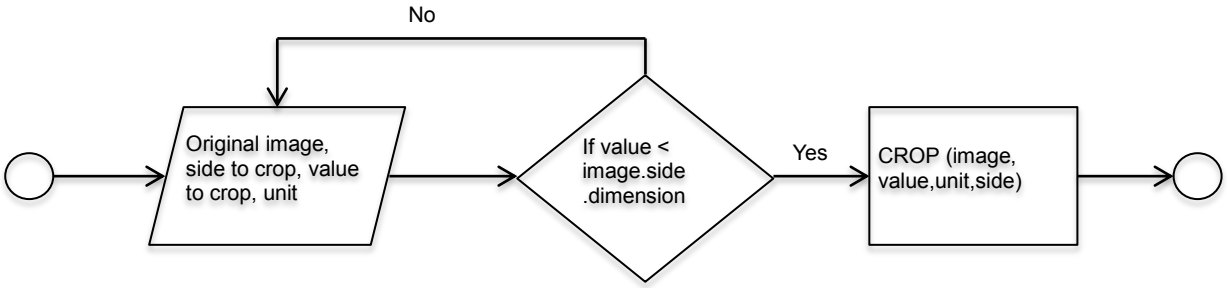
4.4.6 Crop Image (aka Truncation)

Pre-conditions: the application has one or more images available in its content

Input: the original image, a definition of the side to be cropped (top, bottom, left, right, vertical, horizontal, all), a value defining the space to be cropped (e.g. 10), a unit for this value (e.g. px, em, cm, %)

Output: the image cropped according to the input parameters

Dataflow:



Algorithm: The image is taken as an input; the side to be cropped is taken as an input; and the amount of the image to be cropped is taken as an input;

The value to be cropped is checked, it must be less than the original dimension of the image

The image is read as a matrix of pixels

The pixels located in the extremes of the image are removed, according to the input parameters (e.g. 10 px at the top part)

The new image is generated and can be used in the adapted application

Pseudo-code:

```
READ image file
READ side  // top, bottom, left, right, horizontal, vertical, all
READ value  // should be less than the dimension at the chosen side to crop
            // and greater than 0
READ unit //if other than pixel should be converted
IF value < image.side THEN
   New_image == image.CROP(image, side, value)
```

Comments: depending on the technology to be used this CROP function is already implemented and available, the unit may be converted to pixels or not depending on the case too.

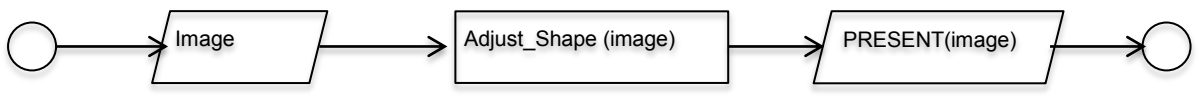
4.4.7 Adjust Shape

Pre-conditions: the context is associated with the image

Input: image

Output: the image is adjusted in shape according to the context

Dataflow:



Algorithm: The image is taken as an input;

The borders are detected and adjusted according to the context;

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
DETECT_BORDERS(image)
ADJUST_SHAPE(image)
```

Comments: examples of use of this technique include virtual reality environments, in which the user can interact virtually with the application.

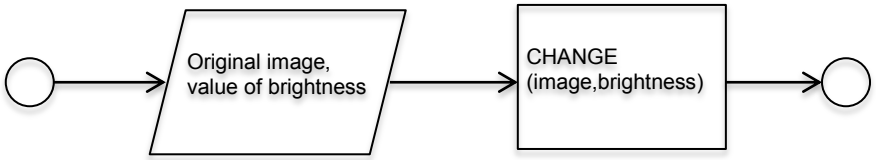
4.4.8 Change Brightness

Pre-conditions: the application has one or more images available in its content

Input: the original image, a value defining the new brightness level

Output: the image with the new brightness level

Dataflow:



Algorithm: The image is taken as an input; the value indicating the brightness is taken as an input;

The image is read as a matrix of pixels

The pixels are processed and each brightness level is changed

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
READ brightness
FOR EACH image.pixel
      CHANGE(image.pixel,brightness)
```

Comments: one approach to define the brightness level is selecting higher, or lower, levels of brightness, providing immediate feedback to the user.

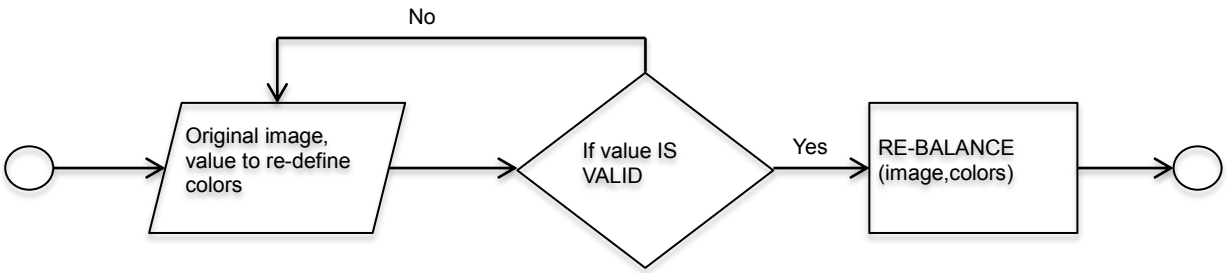
4.4.9 Change the Color Balance (aka Change Color Map)

Pre-conditions: the application has one or more images available in its content

Input: the original image, a value defining the change in color balance

Output: the image processed accordingly

Dataflow:



Algorithm: The image is taken as an input; the value indicating the new color balance is taken as input;

The image is read as a matrix of pixels

The pixels are processed and the color balance of the image modified

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
READ value  // must be validated
IF value IS VALID THEN
      FOR EACH image.pixel
            CHANGE(image.pixel.color)
```

Comments: the color balance of pictures can be improved, sub-properties of the color balance can also be used to control the adaptation (e.g. white balance).

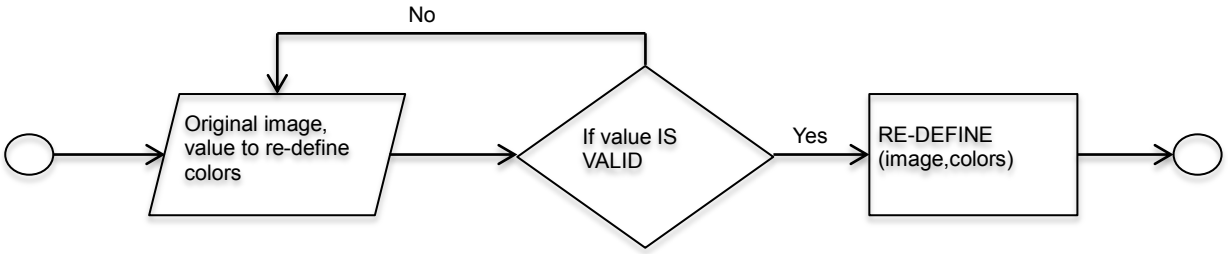
4.4.10 Color Translation

Pre-conditions: the application has one or more images available in its content

Input: the original image

Output: the image has certain colors modified

Dataflow:



Algorithm: The image is taken as an input;

The image is read as a matrix of pixels

The pixels are processed and certain colors are replaced, according to a given criteria

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
IF pixel_color == given_color THEN
      REPLACE(pixel_color,aimed_color)
```

Comments: context of use will define the specificities of the criteria to perform color translation.

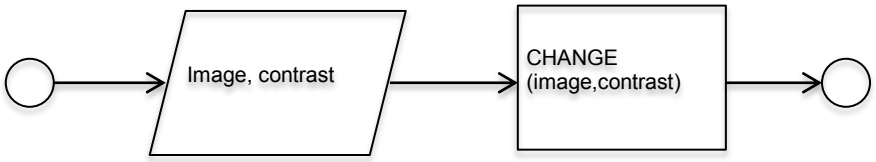
4.4.11 Change Contrast

Pre-conditions: the application has one or more images available in its content

Input: the original image, a value defining the new contrast level

Output: the image with the new contrast level

Dataflow:



Algorithm: The image is taken as an input; the value indicating the contrast is taken as an input;

The image is read as a matrix of pixels

The pixels are processed and the contrast level is changed

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
READ contrast //must be validated
FOR EACH image.pixel
   CHANGE(image.pixel,contrast)
```

Comments: one approach to define the contrasts level is selecting higher, or lower, levels of it, and provide immediate feedback to the user.

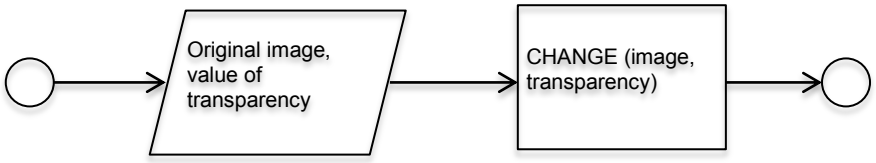
4.4.12 Change Transparency

Pre-conditions: the application has one or more images available in its content

Input: the original image, a value defining the new transparency level

Output: the image with the new transparency level

Dataflow:



Algorithm: The image is taken as an input; the value indicating the transparency is taken as an input;

The image is read as a matrix of pixels

The pixels are processed and each transparency level is changed

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
READ transparency   //must be validated
FOR EACH image.pixel
      CHANGE(image.pixel,transparency)
```

Comments: one approach to define the transparency level is selecting higher, or lower, levels of it, and provide immediate feedback to the user.

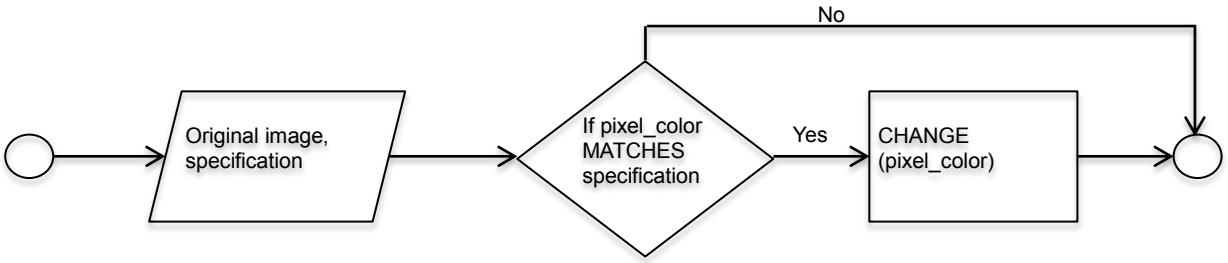
4.4.13 Quantization

Pre-conditions: the application has one or more images available in its content

Input: the original image, a value specifying the quantization type

Output: the adapted image

Dataflow:



Algorithm: The image is taken as an input; the specification for quantization is taken as an input;

The specification is validated

The image is read as a matrix of pixels

The pixels are processed and their colors changed, according to the quantization specified

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
READ specification  // must be validated
IF image.pixel.color MATCHES specification THEN
      CHANGE(image.pixel.color)
```

Comments: the specification defines which colors of the image will be replaced by other colors.

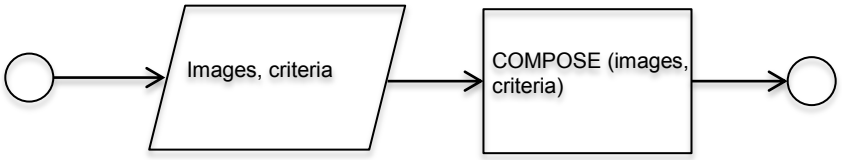
4.4.14 Digital Composition

Pre-conditions: the application has one or more images available in its content

Input: the images, criteria for organizing their layout

Output: the image re-composed

Dataflow:



Algorithm: The images are taken as an input; the criteria for organizing them are taken as an input;

The images are composed according to the given criteria

The new image is generated with the composition and presented to the end user

Pseudo-code:

```
READ images
READ criteria  // must be validated
COMPOSE(images,criteria)
```

Comments: the images can be composed according to a shape, symbol, specific alignment, and so on. For instance, for a large screen, images can be displayed side by side, and for vertical screen, they can be displayed as one single column, with a vertical scroll bar if necessary.

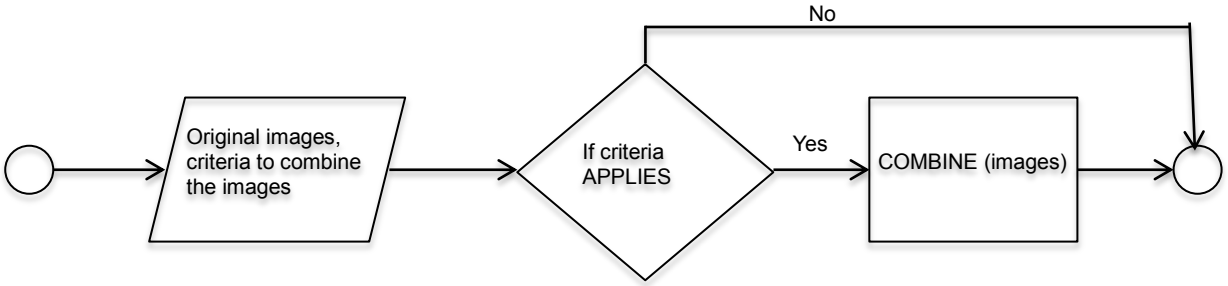
4.4.15 Matte

Pre-conditions: the application has two or more images available in its content

Input: the images, criteria defining the application of the matte

Output: the combined images

Dataflow:



Algorithm: The images are taken as input; the criteria to apply matte are taken as input too;

The images are processed and combined according to the matte criteria

The new image is generated and presented to the end user

Pseudo-code:

```
READ images
READ criteria  // must be validated
APPLY(matte_criteria,images)
```

Comments: one example of application is using one image to fulfil another only in specific parts (defined for instance by a color criterion), e.g. fill the blue region of an image with another given image.

4.4.16 Daltonize

Pre-conditions: the application has one or more images available in its content

Input: the original image

Output: the image accessible for color-blind users

Dataflow:

Algorithm: The image is taken as an input;

The image is read as a matrix of pixels;

The pixels are processed and the colors modified;

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
DALTONIZE(image)
```

Comments: another possible approach consists in taking also the type of color-blindness of the user to generate images that are accessible in specific cases of this visual impairment.

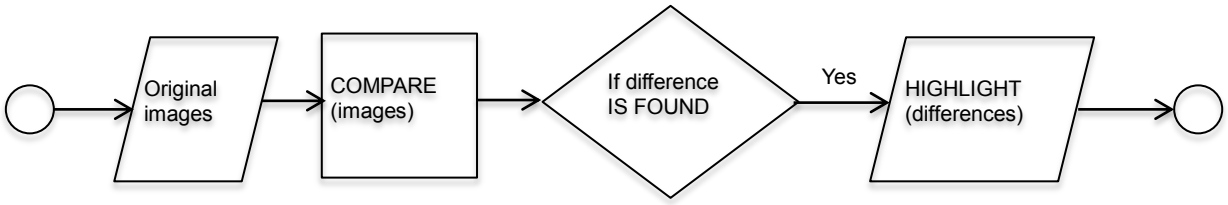
4.4.17 Differentiate

Pre-conditions: the application has two or more images available in its content

Input: the images

Output: the differences between images

Dataflow:



Algorithm: The images are taken as input;

The images are compared;

The differences between images are detected and highlighted to the end user

Pseudo-code:

```
READ images
DIFFERENTIATE (images)
IF difference IS FOUND THEN
      HIGHLIGHT(difference)
```

Comments: the images can be also compared according to a given criteria, such as color, shapes, or brightness levels. The results can be presented to the end user, highlighted in the original images, as number, or graphics

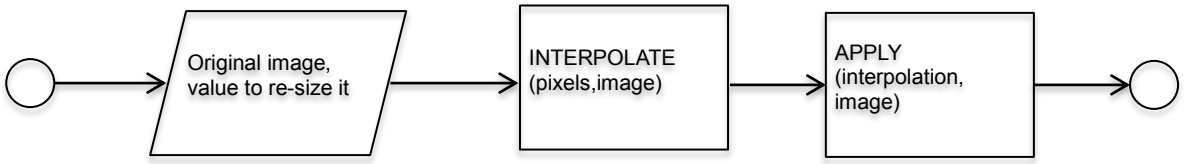
4.4.18 Interpolate

Pre-conditions: the application has one or more images available in its content

Input: the original image

Output: the image with more pixels defined

Dataflow:



Algorithm: The image is taken as an input;

The value for intermediary pixels are estimated according to the neighbour pixels

The interpolated values are added to the original image

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
CREATE(interpolation)
ADD(pixels, image)
```

Comments: the amount of pixels to be interpolated can be defined as an input too.

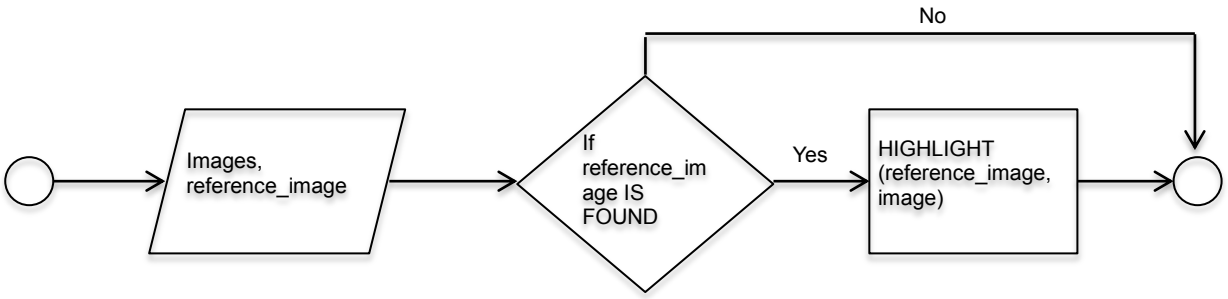
4.4.19 Geometric Hash (aka Recognize, Segment)

Pre-conditions: the application has one or more images available in its content

Input: two or more images

Output: the reference image identified in the other images

Dataflow:



Algorithm: The images are taken as input;

The reference image is searched in the other images;

If found then it is highlighted in the other images

The resulting image is generated and presented to the end user

Pseudo-code:

```
READ images
READ reference_image
SEARCH(reference_image,image)
IF reference_image IS FOUND THEN
      PRESENT(result)
```

Comments: the reference image to be identified in the other images can be for instance, in a simple case, a geometric figure, or in a more complex example, a face.

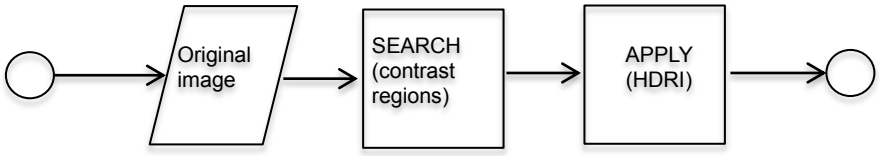
4.4.20 High Dynamic Range Imaging (Method)

Pre-conditions: the application has one or more images available in its content

Input: the original image

Output: the image tones are adapted (including contrast levels)

Dataflow:



Algorithm: The image is taken as an input;

The image is read as a matrix of pixels

The pixels are processed and the tones are changed, according to the HDRI specifications

The new image is generated and presented to the end user

Pseudo-code:

```
READ image file
DETECT(contrast levels)
APPLY(HDRI)
```

Comments: this algorithm consists in a set of techniques to balance the darkest and lightest areas of an image.

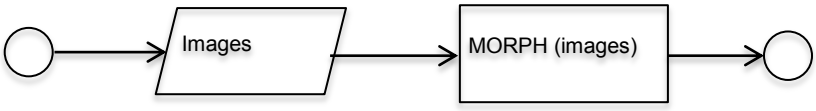
4.4.21 Morph

Pre-conditions: the application has one or more images available in its content

Input: the original images

Output: the resulting image based in the input images

Dataflow:



Algorithm: The images are taken as input;

The images are processed to mix both image contents

The new image is generated and presented to the end user

Pseudo-code:

```
READ images
MORPH(images)
```

Comments: animation can be used to present the results of this technique for the end user.

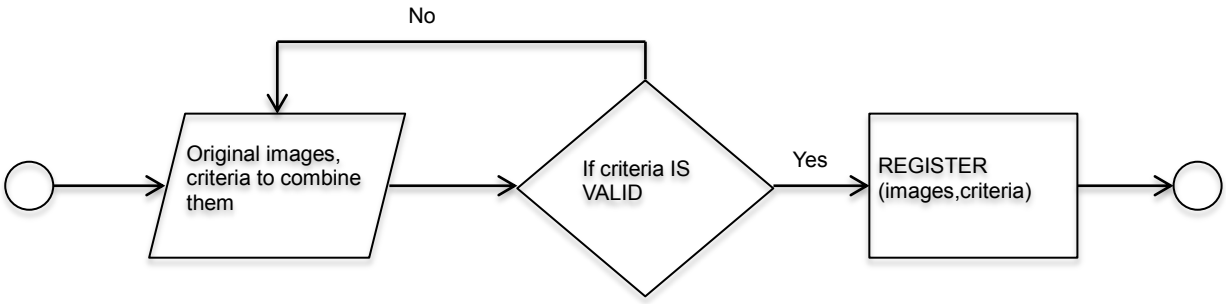
4.4.22 Register (Method)

Pre-conditions: the application has one or more images available in its content

Input: the original images

Output: one image based on the combination of the input images

Dataflow:



Algorithm: The images are taken as input; the criteria to combine them are taken as input;

The criteria are validated;

The images are combined

The new image is generated and presented to the end user

Pseudo-code:

```
READ images, criteria
IF criteria IS VALID
      REGISTER(images,criteria)
```

Comments: the images can be combined given a certain criteria, for instance aligning one element in common, or according to color properties. The validation is defined according to the context of use.

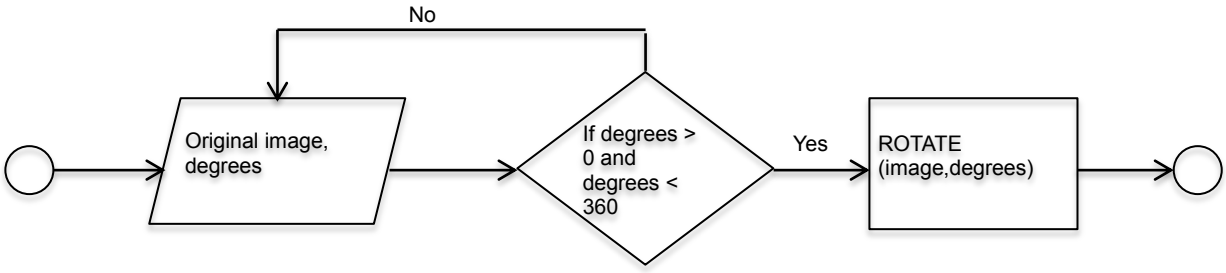
4.4.23 Rotate

Pre-conditions: there is at least one image that can be rotated

Input: the original image, a value defining the degrees of rotation

Output: the image is presented in a new orientation

Dataflow:



Algorithm: The image is taken as an input; the value of degrees for rotation is taken as an input;

The value is verified (must be greater than 0 and lower than 360)

The image is rotated according to the value of degrees provided

Pseudo-code:

```
READ image
READ degrees  // >0 and <360
IF degrees > 0 AND degrees < 360 THEN
   Image == image.ROTATE(degrees)
```

Comments: depending on the shape or size of the image, the layout may be affected; in this case, complementary adaptation techniques must be performed, such as re-size; we are assuming here a clockwise orientation, however a signal indicating the sense for rotation can also be considered.

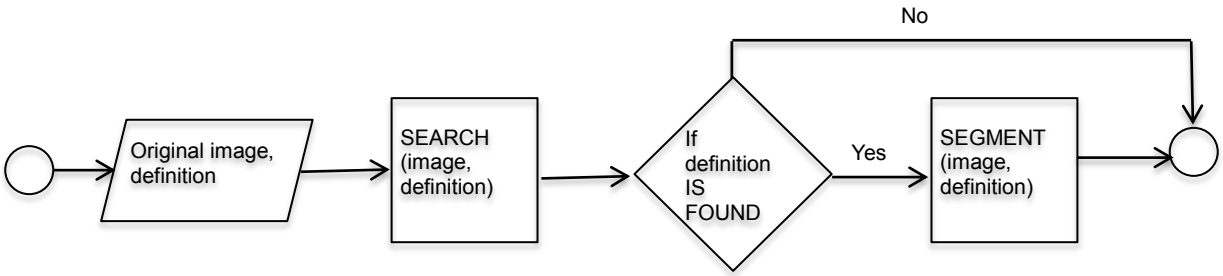
4.4.24 Segment (aka Recognize, Geometric Hashing)

Pre-conditions: the application has one or more images available in its content

Input: the original image, a definition for segmenting it

Output: the image segmented according to the input parameters

Dataflow:



Algorithm: The image is taken as an input; the definition for segmentation is taken as input;

The definition is checked to verify its validity

The image is processed

If the segments are found, then they are remarked and presented to the end user

Pseudo-code:

```
READ image file
READ definition  // e.g. color, shape,image
SEARCH(image,definition)
IF definition IS FOUND THEN
```

```
SEGMENT(image,definition)
```

Comments: the image can be segmented according to different criteria, such as geometric forms or colors.

4.5 Adapting Video

This section describes algorithms to implement adaptation in Video content, some of the algorithms have the same definition when applied to images, however the application is performed for each frame of the video file.

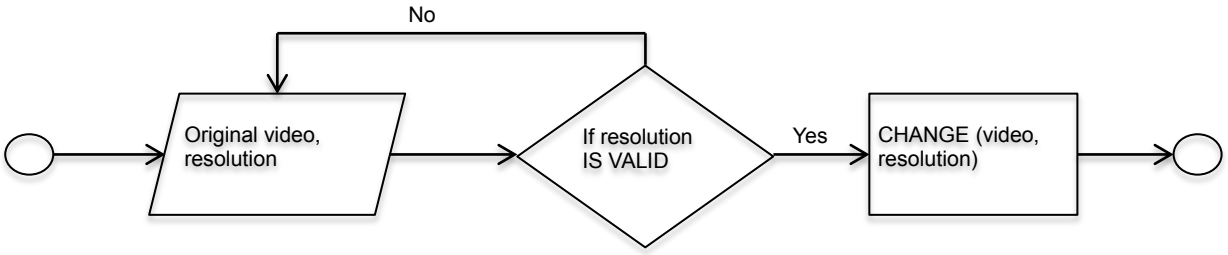
4.5.1 Change Resolution

Pre-conditions: the application has one video available in its content

Input: the original video, a definition of the alternative resolution

Output: the video is presented in a new resolution

Dataflow:



Algorithm: The video is taken as an input; and the new resolution is taken as an input;

The new resolution is checked (validated)

The video is processed accordingly

The video content in the adapted resolution is presented to the end user

Pseudo-code:

```
READ video file
READ resolution  // must be validated
IF resolution IS VALID THEN
      CHANGE(video,resolution)
```

Comments: there are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different resolutions, and then searching and retrieving the right one to present it to the end user.

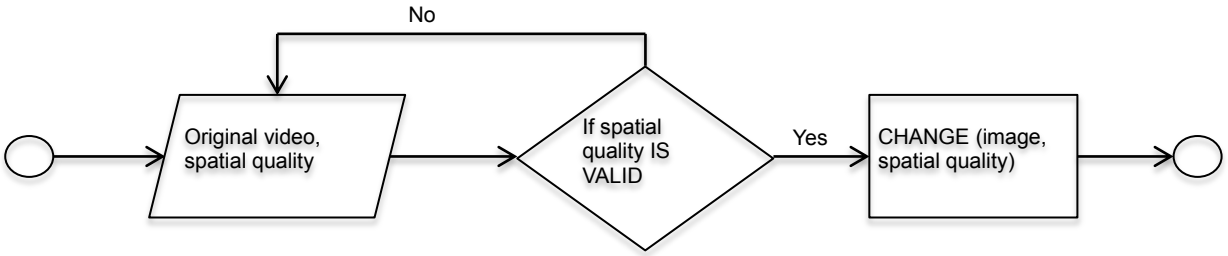
4.5.2 Change Spatial Quality

Pre-conditions: the application has one video available in its content

Input: the original video, a definition of the alternative spatial quality

Output: the video is presented in a new spatial quality

Dataflow:



Algorithm: The video is taken as an input; and the new spatial quality is taken as an input;

The new spatial quality is checked (validated)

The video is processed accordingly

The video in the adapted spatial is presented to the end user

Pseudo-code:

```
READ video file
READ spatial quality  // must be validated
IF spatial quality IS VALID THEN
      CHANGE(video, spatial quality)
```

Comments: there are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different spatial qualities, and then searching and retrieving the right one to present it to the end user. The spatial quality value must be validated.

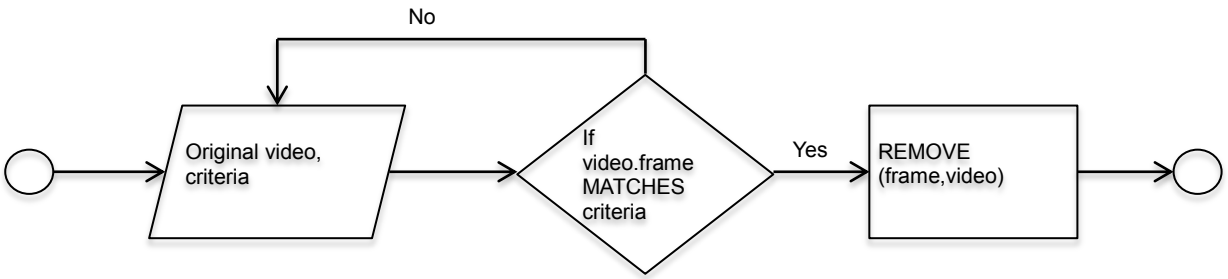
4.5.3 Skip

Pre-conditions: the application has one or more video available in its content

Input: the original video, criteria to remove frames of it

Output: the video file, with frames removed

Dataflow:



Algorithm: The video is taken as input;

The criteria to remove frames is taken as input;

The video is processed, frames that match the criteria are removed

The new video is generated and presented to the end user

Pseudo-code:

```
READ video, criteria
IF video.frame MATCHED criteria THEN
      REMOVE(frame,video)
```

Comments: criteria to remove certain frames can take into account time, content, context of use, and so on. There are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different content removed, and then searching and retrieving the right one to present it to the end user. The criteria to skip frames must be validated.

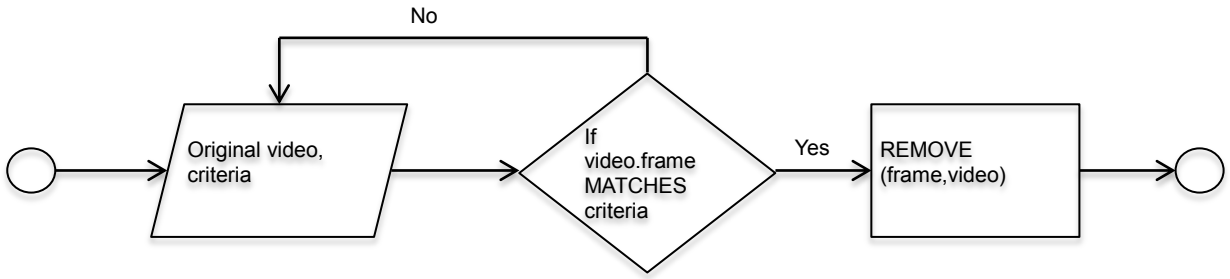
4.5.4 Reduce

Pre-conditions: the application has one or more video available in its content

Input: the original video, criteria to reduce it

Output: the reduced video

Dataflow:



Algorithm: The video is taken as input;

The criteria to reduce the video are taken as input;

The video is processed; frames that match the criteria are removed

The new video is generated and presented to the end user

Pseudo-code:

```
READ video, criteria
IF video.frame MATCHED criteria THEN
      REMOVE(frame,video)
```

Comments: criteria to reduce video content can take into account time, content, context of use, and so on. There are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different content removed, and then searching and retrieving the right one to present it to the end user. The criteria to skip frames must be validated.

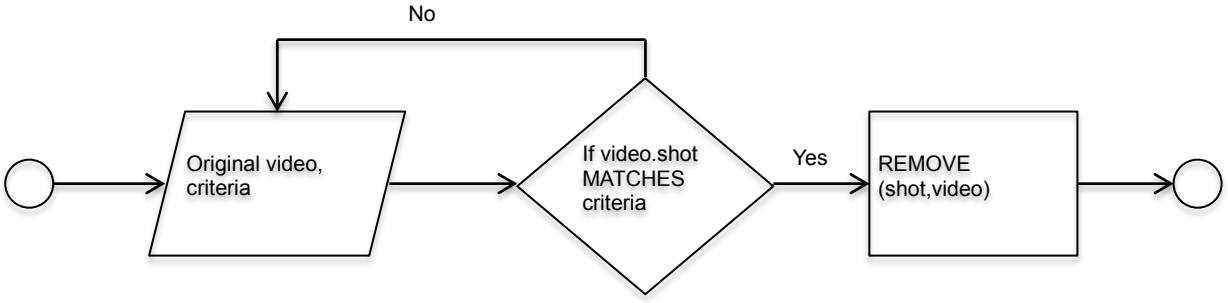
4.5.5 Remove Shot

Pre-conditions: the application has one or more video available in its content

Input: the original video, criteria to remove shots of it

Output: the video file, with shots removed

Dataflow:



Algorithm: The video is taken as input;

The criteria to remove shot is taken as input;

The video is processed; shots that match the criteria are removed

The new video is generated and presented to the end user

Pseudo-code:

```
READ video, criteria
IF video.shot MATCHED criteria THEN
      REMOVE(shot,video)
```

Comments: criteria to remove certain shots can take into account time, content, context of use, and so on.

There are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different shots removed, and then searching and retrieving the right one to present it to the end user. The criteria to remove shots must be validated.

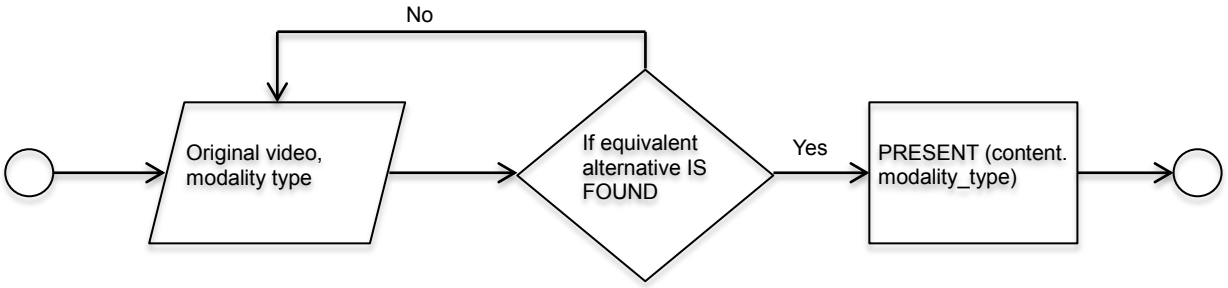
4.5.6 Replace

Pre-conditions: the application has one or more video available in its content, and a database with content of alternative modalities associated with the original video

Input: the original video, the aimed modality

Output: content in the alternative modality type specified

Dataflow:



Algorithm: The video is taken as an input; the alternative modality type is taken as an input;

The alternative modality type is checked

If an alternative equivalent to the original image is found then

The new content type is presented to the end user

Pseudo-code:

```
READ video file
READ modality_type  // image, text, audio; must be validated
SEARCH content.modality_type
IF alternative modality type for the image is found THEN
      PRESENT(content.modality_type)
```

Comments: to search for alternative modalities based on a video, it is necessary to rely on semantic descriptions of it; the alternative modalities provided as inputs must be validated; the final layout may be impacted; each modality may have specific requirements to be accessed (e.g. speakers for audio, etc.).

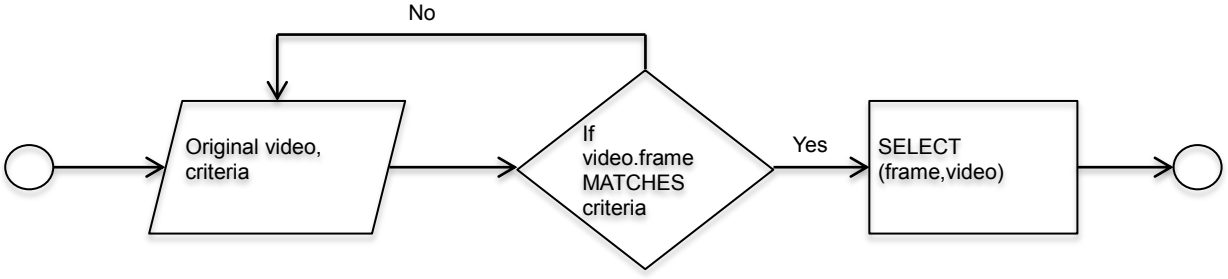
4.5.7 Select

Pre-conditions: the application has one or more video available in its content

Input: the original video, criteria to select frames of it

Output: the video file re-composed with selected frames only

Dataflow:



Algorithm: The video is taken as input;

The criteria to select frames are taken as input;

The video is processed; frames that match the criteria are selected

The new video is generated and presented to the end user

Pseudo-code:

```
READ video, criteria
IF video.frame MATCHES criteria THEN
      SELECT(frame,video)
```

Comments: criteria to select certain frames can take into account time, content, context of use, and so on. There are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different frames selected, and then searching and retrieving the right video to present it to the end user.

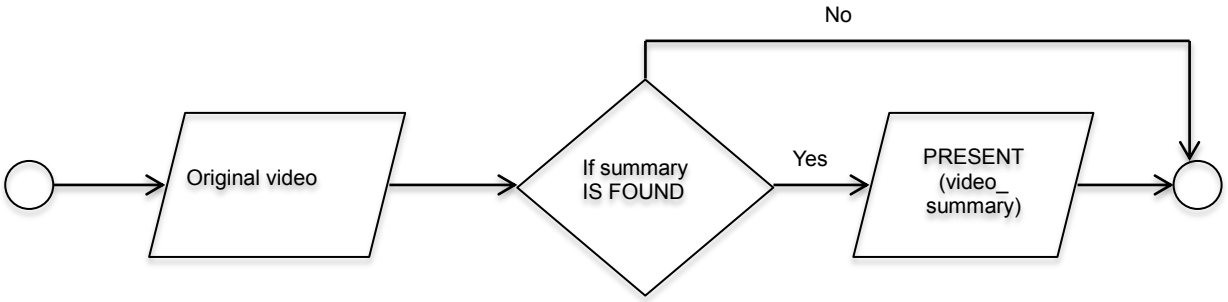
4.5.8 Summarize

Pre-conditions: the application has one a video available in its content

Input: the original video

Output: a summarized version of the video

Dataflow:



Algorithm: The video is taken as input;

A summary of the video is searched;

If found then it is presented to the end user

Pseudo-code:

```
READ video
SEARCH(video_summary)
IF video_summary IS FOUND THEN
      PRESENT(video_summary)
```

Comments: there are other approaches that can be also used to implement this technique.

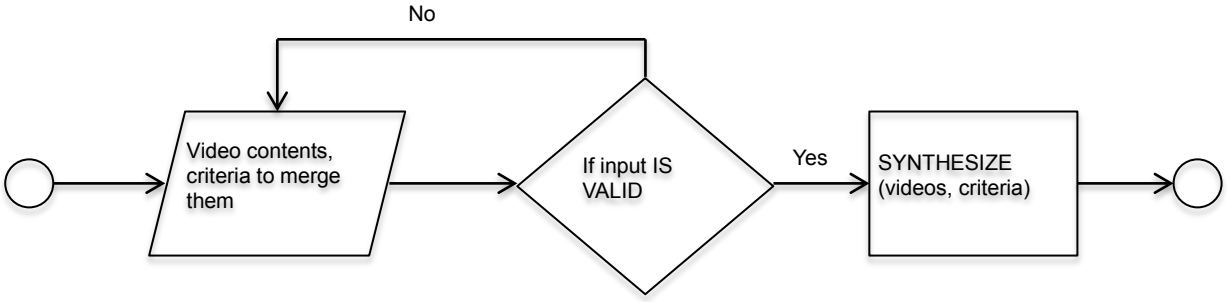
4.5.9 Synthesize

Pre-conditions: the application has one or more video available in its content

Input: video contents, criteria to merge them

Output: one video created based on the combination of the input contents

Dataflow:



Algorithm: The video contents are taken as input; Criteria to merge them are taken as input;

The videos are combined according to the given criteria;

The new video is generated and presented to the end user;

Pseudo-code:

```
READ videos, criteria
SYNTHESIZE(videos,criteria)
```

Comments: the videos can be combined given a certain criteria, for instance inserting frames in a given time of the video, alternating video contents, and so on.

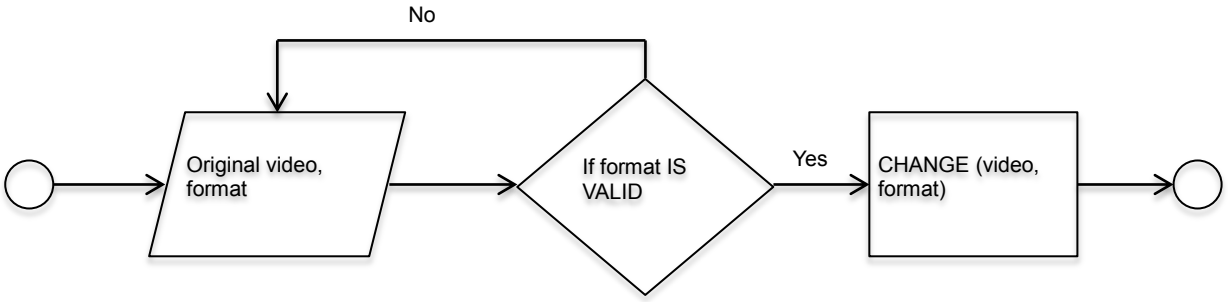
4.5.10 Transcode

Pre-conditions: the application has one or more video available in its content

Input: the original video, a definition of the new format

Output: the video is presented in a new format

Dataflow:



Algorithm: The video is taken as an input; and the new format is taken as an input;

The new format is checked (validated)

The video is processed accordingly

The video in the adapted format is presented to the end user

Pseudo-code:

```
READ video file
READ format  // DivX, MPEG-4
IF format IS VALID THEN
      CHANGE(video,format)
```

Comments: there are alternative manners to implement this algorithm, for instance providing a repository with alternative versions of the same video but different formats, and then searching and retrieving the right one to present it to the end user.

4.6 Adapting UI Elements

This section describes algorithms to implement adaptation in UI Elements.

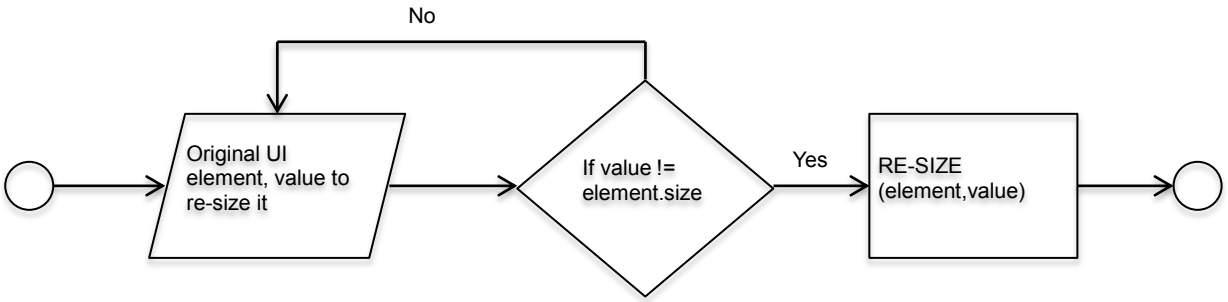
4.6.1 Change Size (aka Re-size, Re-scale, Reduce, Enlarge)

Pre-conditions: the application has one or more UI elements available in its content

Input: the original UI element, a value defining its new size (e.g. -10%, or 7x5 pixels)

Output: the UI element with a new dimension according to the input parameters

Dataflow:



Algorithm: The element is taken as an input; the value indicating the re-size ratio is taken as an input;

The value for the re-size is checked, it must be different than the original dimension of the element

The element is re-scaled, according to the input parameters (e.g. – 10 %)

The new element is generated and presented to the end user

Pseudo-code:

```
READ UI element
READ value  // e.g. - 10 %
IF value != element.size THEN
      RESIZE(element,value)
```

Comments: the adapted element may affect the layout, requiring for instance re-distribution of the content, scrolling, or pagination. Elements too reduced or too large may become inaccessible.

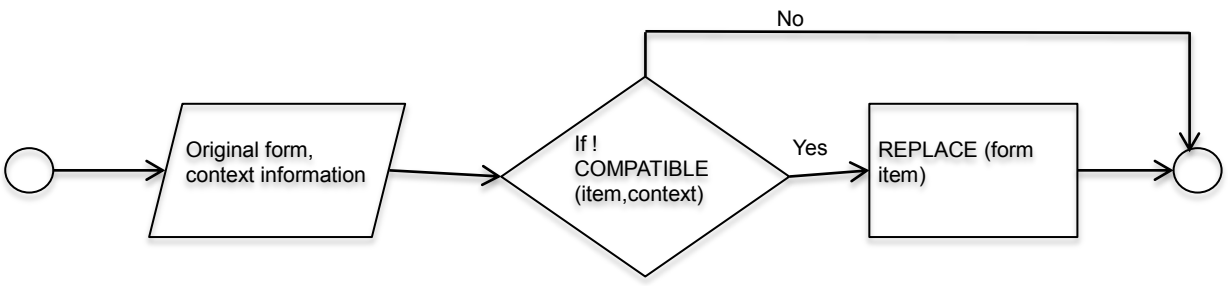
4.6.2 Replace

Pre-conditions: the application has one or more UI elements available in its content

Input: the original UI elements, the context information

Output: the UI with certain elements replaced

Dataflow:



Algorithm: The UI elements are taken as input;

The compatibility between UI elements and context if checked

If the elements are not compatible, then they are replaced by equivalent ones

The new UI is generated with the new elements and presented to the end user

Pseudo-code:

```
READ UI elements
GET context information
IF ! COMPATIBLE (element,context)
      REPLACE(element,new_element)
GENERATE(UI)
```

Comments: the layout must be taken into account, once the new elements may have different dimensions and affect the original layout. Animation can be performed to present this adaptation to the end user.

4.6.3 Adapt Form

Pre-conditions: the application has one or more forms available in its content

Input: the original form, the logic to define its filling

Output: the adapted form

Dataflow:

Algorithm: The form is taken as an input;

The logic to fill in the form is implemented;

The adapted form is generated and presented to the end user;

Pseudo-code:

```
READ form
GET(logic)
IMPLEMENT(adaptation_rules)
GENERATE(adapted form)
```

Comments: depending on the context of use different rules must be defined to fill in the form items, e.g. displaying additional fields according to the content provided by the end user.

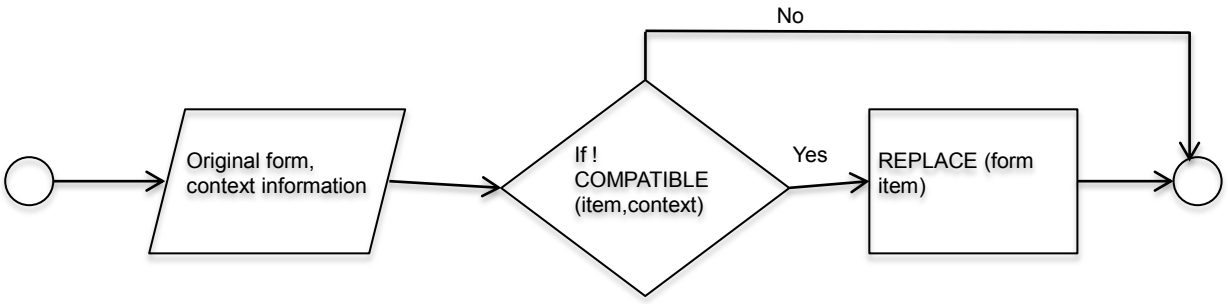
4.6.4 Adjust Form

Pre-conditions: the application has one or more forms available in its content, there is a logic defining which items are compatible or not with the context

Input: the original form, the context of use

Output: certain items of the form are replaced according to the context

Dataflow:



Algorithm: The form is taken as input; the context is taken as input;

The context and form items are checked;

If there are not compatible, then an equivalent form item replaces the original one;

The adapted form is generated and presented to the end user;

Pseudo-code:

```
READ form, context information
CHECK_COMPATIBILITY(form_item, context)
IF !COMPATIBLE(form_item,context)
      REPLACE(form_item)
```

Comments: for smaller screens, for instances, there are certain form items that are more appropriate for filling a form, the same occurs for interaction modalities.

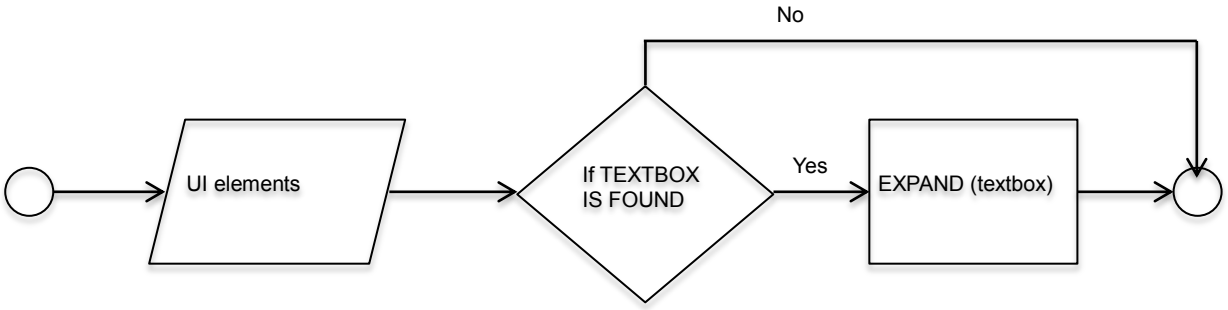
4.6.5 Expand TextBox

Pre-conditions: the application has one or more textboxes available in its content

Input: the textbox

Output: the expansible textbox

Dataflow:



Algorithm: The textbox is taken as input;

The textbox is set to expansible

The expansible textbox is presented to the end user

Pseudo-code:

```
READ UI elements
SEARCH(textbox)
IF textbox IS FOUND then
      Textbox.expansible = true
```

Comments: this function illustrates one possibility of implementation, depending on the technology adopted

other approaches of implementation should be adopted. A criterion that also defines the property of expansibility of a textbox is the length of the content of it, for long inputs it is interesting to have this property set as true.

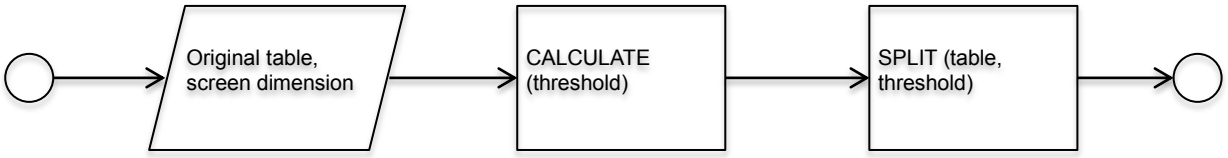
4.6.6 Split Table

Pre-conditions: the application has one or more tables available in its content

Input: the original table

Output: the table split in different pages

Dataflow:



Algorithm: The table is taken as input;

The screen dimension is identified

The amount of columns to be used as a threshold is calculated

The table is split in two or more pages according to the dimension of the screen

Pseudo-code:

```
READ table
DETECT(screen dimension)
CALCULATE(threshold)
SPLIT(table,threshold)
```

Comments: additional criteria, such as the length of each column, can also be used to calculate the amount of columns defined to split the table. Links can be added to facilitate the access to the table contents.

4.6.7 Transform Table

Pre-conditions: the application has one or more tables available in its content

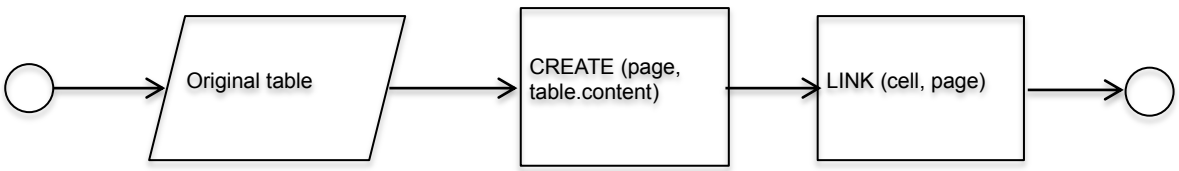
Input: the original table

Output: each cell of the table becomes a new page

Dataflow:

Algorithm: The table is taken as input;

The content of each cell is replaced by a link



Pages are created for the content of each cell

The table of links is presented to the end user

Pseudo-code:

```
READ table
CREATE(pages, content)
LINK (cell, pages)
```

Comments: one possibility is to number the cells of the table, to provide links to the original content.

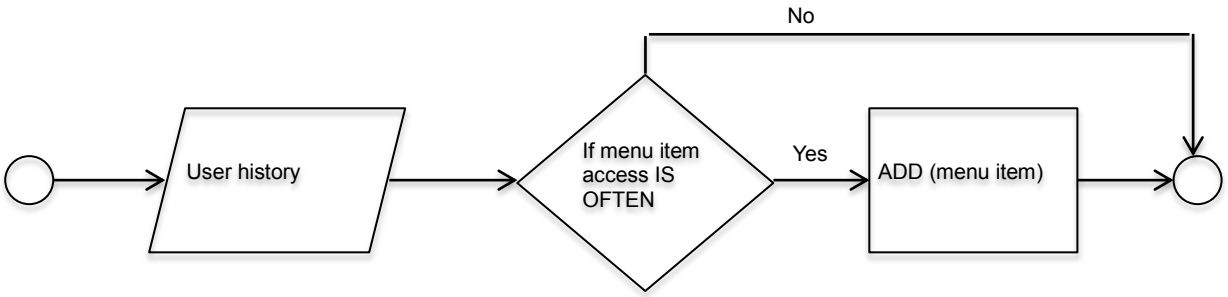
4.6.8 Split Interface

Pre-conditions: the application has menu options, and the history of the user actions is tracked

Input: the user history

Output: a new menu is created with popular items

Dataflow:



Algorithm: The history of user actions is taken as input;

The tasks accessed the most are identified

A new menu is created with the most popular items only

Pseudo-code:

```
READ user history
SEARCH (common tasks, user history)
CREATE (new menu)
```

Comments: the developer can define the number of items in the new menu.

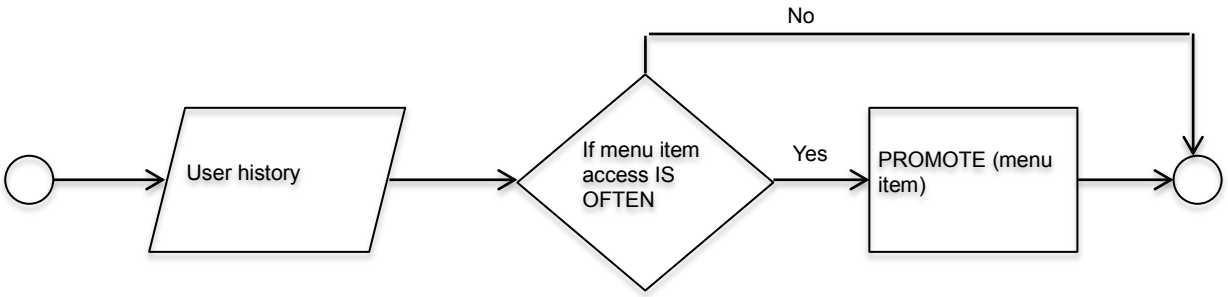
4.6.9 Moving Interface

Pre-conditions: the application has menu options, and the history of the user actions is tracked

Input: the user history

Output: the menu with promoted items

Dataflow:



Algorithm: The history of user actions is taken as input;

The tasks accessed the most are identified

The menu items are promoted according to the user history

Pseudo-code:

```
READ user history
SEARCH (common tasks, user history)
```

```
PROMOTE (menu items)
```

Comments: the menu items can be promoted by creating shortcuts to access most frequent tasks.

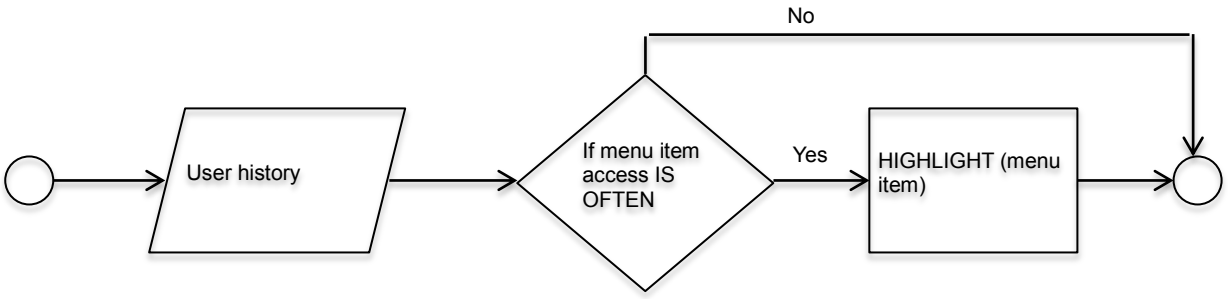
4.6.10 Visual PopOut Interface

Pre-conditions: the application has menu options, and the history of the user actions is tracked

Input: the user history

Output: the menu with highlighted items

Dataflow:



Algorithm: The history of user actions is taken as input;

The tasks accessed the most are identified

The menu items are highlighted according to the user history

Pseudo-code:

```
READ user history
SEARCH (common tasks, user history)
HIGHLIGHT (menu items)
```

Comments: the menu items can be highlighted with color, signs, or augmented, according to the history of interaction of the end user.

5 Demonstration Algorithm for AAL - The desktop-to-vocal method

In this section we show an adaptation method that exemplifies the transformation of desktop web pages in vocal interfaces. This demonstration will be the first prototype implementation from the Algorithm Library and used as a test case for future implementations.

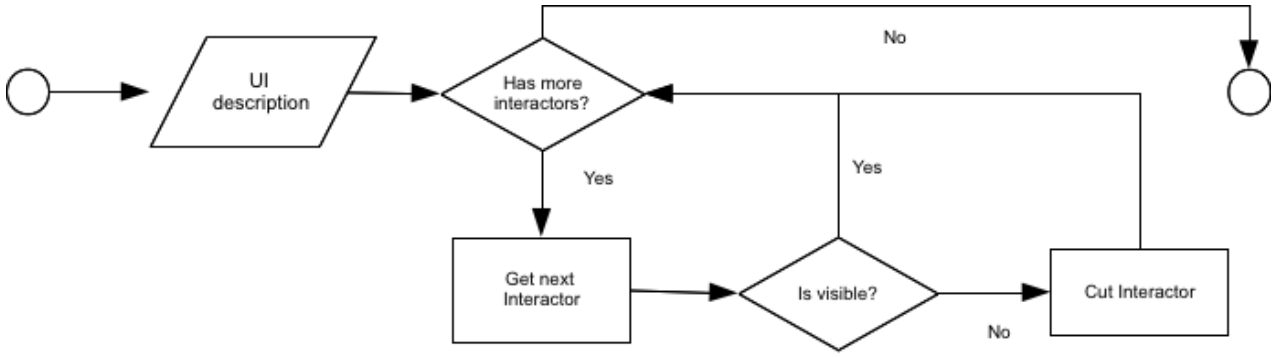
1) Remove not graphically visible interactors

Precondition: The application contains interactors that are not graphically visible

Input: A logical description of the application UI.

Output: A logical description of the application UI, without the invisible elements.

Dataflow:



Algorithm:

The UI description is taken as input. The procedure executes a loop on each UI interactor: if the interactor is visible, it is maintained otherwise it is deleted from the UI description. The resulting UI description is returned as output.

Pseudocode

```
READ Ui
FOREACH Interactor i IN Ui.getInteractors()
      IF notVisible(i) THEN remove(Ui, i)
```

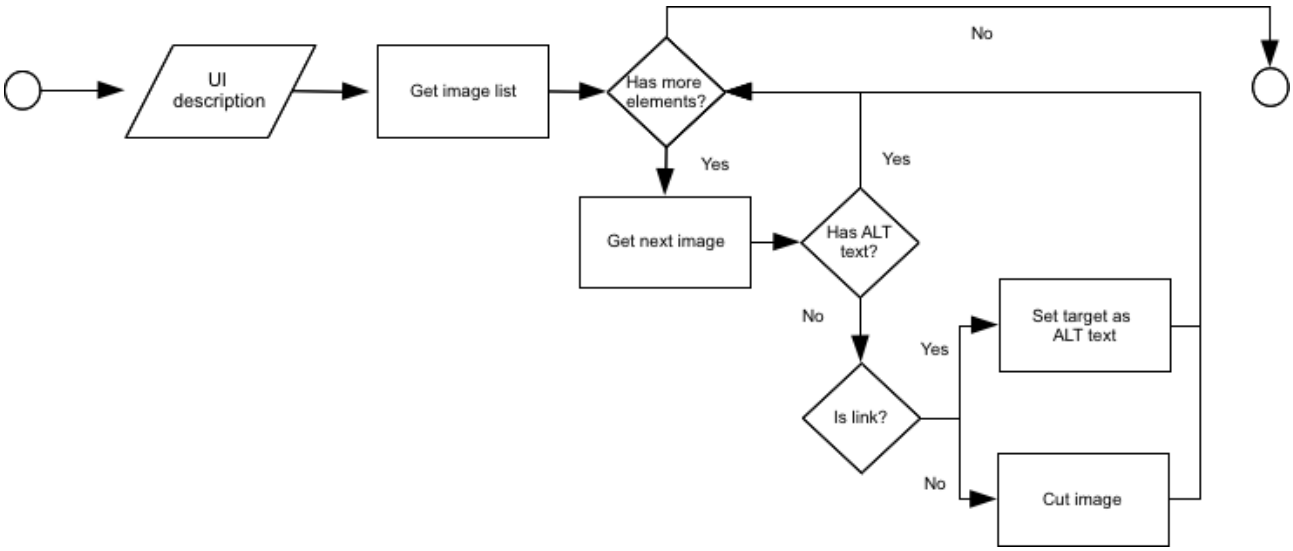
2) Remove the images that cannot be rendered vocally

Precondition: The application UI contains images without alternative descriptions.

Input: The application UI description.

Output: The application UI description, without the images missing the alternative description.

Dataflow:



Algorithm

The UI description is taken as input. The procedure executes a loop on each UI image: if the image has an associated alternative text, it is kept. If no alternative text has been provided, and the image is a link, the procedure sets the link target as the alternative text for the image, which is kept. Otherwise the image is deleted from the application UI.

Pseudo-code

```
READ Ui
FOREACH Image img IN Ui.getInteractors()
      IF NOT hasAlternativeText(img) THEN
            IF isLink(img) THEN
                  img.altText := getLinkTarget(img)
            ELSE
                  remove(Ui, img)
```

Comments: The image links are maintained in order to keep the UI navigation.

3) Normalize text

Precondition: the UI description contains text that cannot be rendered by vocal browsers (e.g. Chinese writing).

Input: the application UI description and a list of

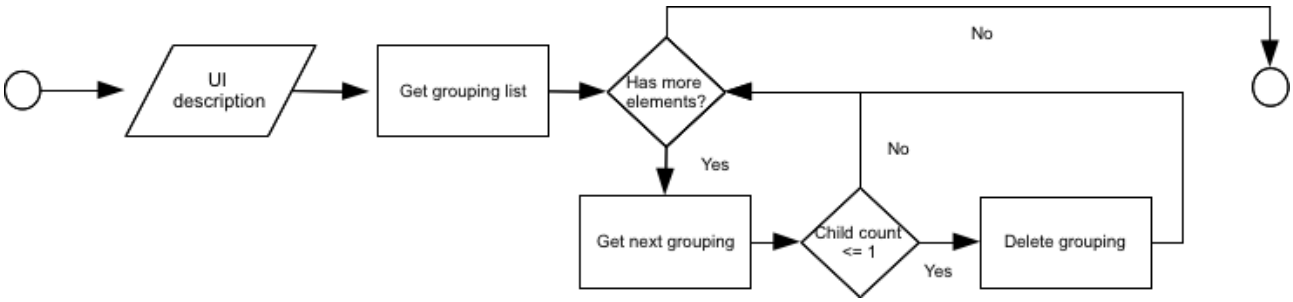
4) Correct grouping inconsistency

Precondition: the application UI contains groupings without content or with only one child.

Input: the application UI description

Output: the application UI description, without grouping without content or with only one child.

Dataflow:



Pseudocode

```
READ Ui
FOREACH Grouping g IN Ui.interactors
      IF childCount(g) <= 1 THEN
            addChildren(g.parent, g.children)
            remove(g.parent,g)
```

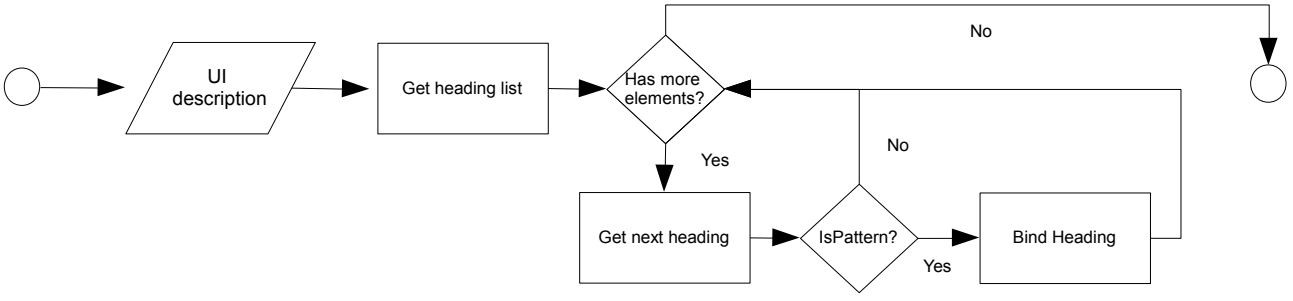
Algorithm

The UI description is taken as input. The procedure executes a loop on each UI grouping: if its children count is minor or equal to one, the grouping content are moved up in the Ui hierarchy and the grouping is deleted from the UI.

Comments: such kind of groupings can be created for layout purposes or can be the result of the application of some interactor deleting adaptation rule.

5) Title recognition

6) **Precondition**: the application UI contains headings that could be the title of more than one grouping**.**
Input: the applicationUI description.
Output: the application UI description with groupings bound with the correct title.
Dataflow:



Pseudocode

```
READ Ui
FOREACH Heading h IN Ui.interactors.attributes
      P = getPattern(h)
      IF p != null
            bind(h, p.getRelatedGrouping())
```

Algorithm

The UI description is taken as input. The procedure executes a loop on each UI interactors heading: if one of the available patterns is recognized, the heading became the title of the related grouping.

Comments: usually there is no strong semantic binding between page components and headings, connecting them is delegated to the user's perception of the actual presentation of the page.

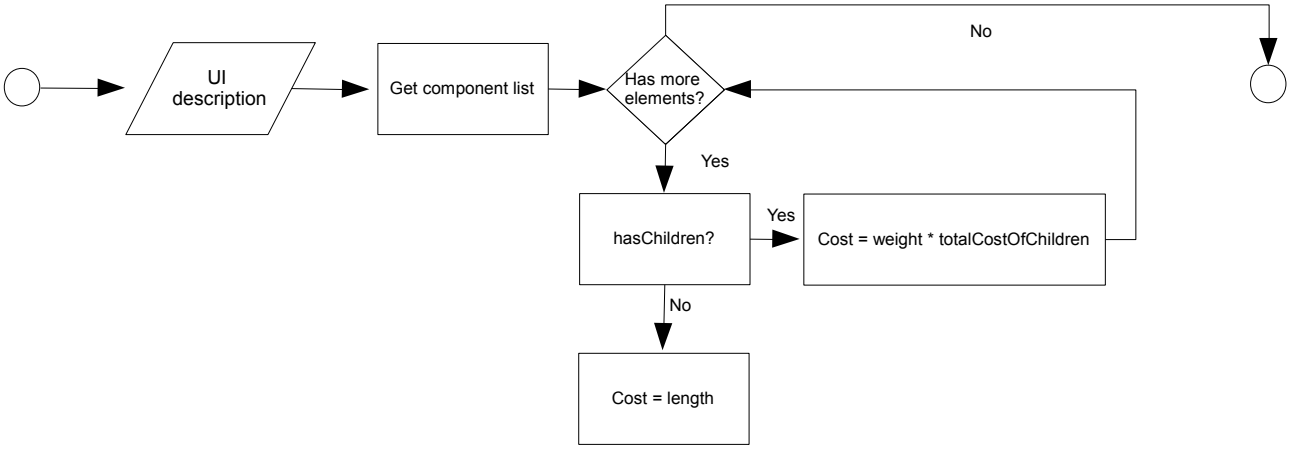
7) Cost calculation

Precondition: the application UI contains different page components (bullet-list, form, …) of different weight in terms of interaction impact**.**

Input: the applicationUI description.

Output: the application UI description with a cost associated at each page component.

Dataflow:



Pseudocode

```
READ Ui
FOREACH PageComponent pc IN Ui.pageComponents
      IF pc != null
            IF pc.hasChildren() != null
                  pc.setCost(pc.weight * calculateCost(pc.getChildren()));
            ELSE pc.setCost(pc.lenght);
```

Algorithm

The UI description is taken as input. The procedure executes a loop on each UI page component and then calculate recursively its cost.

Comments:

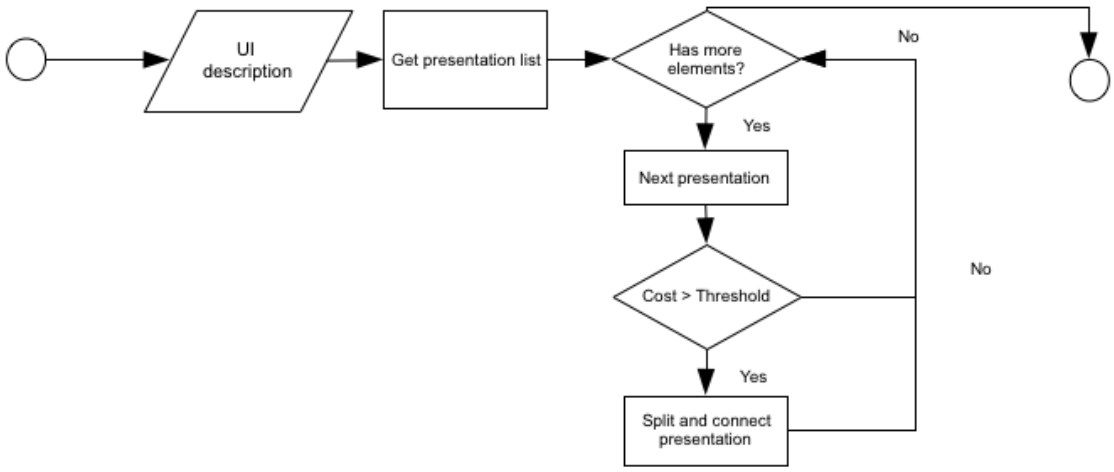
8) Page splitting

Precondition: the application UI contains different page component (bullet-list, form, …). A cost has been assigned to each of them. A splitting threshold for the cost has been fixed.

Input: the application UI description, with the associated component costs and the splitting threshold.

Output: the application UI description with presentations that contain components with a cost lower than the specified threshold.

Dataflow:



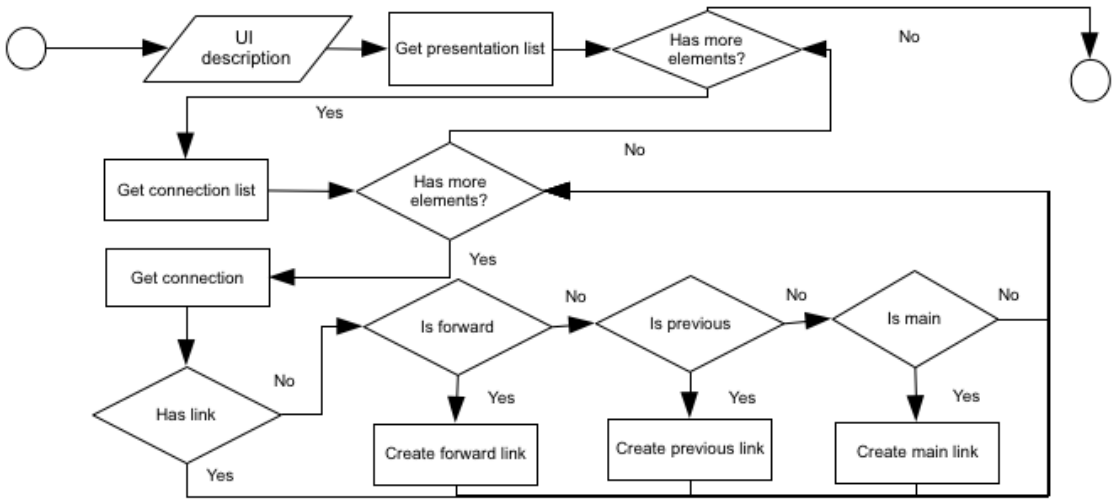
Pseudocode

```
READ Ui
FOREACH PageComponent pc IN Ui.pageComponents
   IF pc.getCost > Ui.threshold
         split(threshold, pc, Ui.pageComponents);
```

Algorithm
The UI description is taken as input. The procedure executes a loop on each UI page component and then, if the cost of the component is above a fixed threshold, the current page is splitted. The splitting procedure selects a cut point in the original page in order to maintain balanced the cost of two parts. Then creates two new presentations, which are linked with connections to the original one.
Comments: the generated connections are of three types:

1. Forward, which links the original presentation with one of its children
2. Previous, which links on child presentation with the original one
3. Main, which links the child presentation with the starting point of the application

9) Menu generation
Precondition: the application UI contains different page components (bullet-list, form, …), with page-splitting generated connections. The connections generated by the page splitting are marked as *forward, previous* or *main*.
Input: the applicationUI description, with with page-splitting generated connections..
Output: the application UI description with a navigation structure based on menus.
Dataflow:



Pseudocode

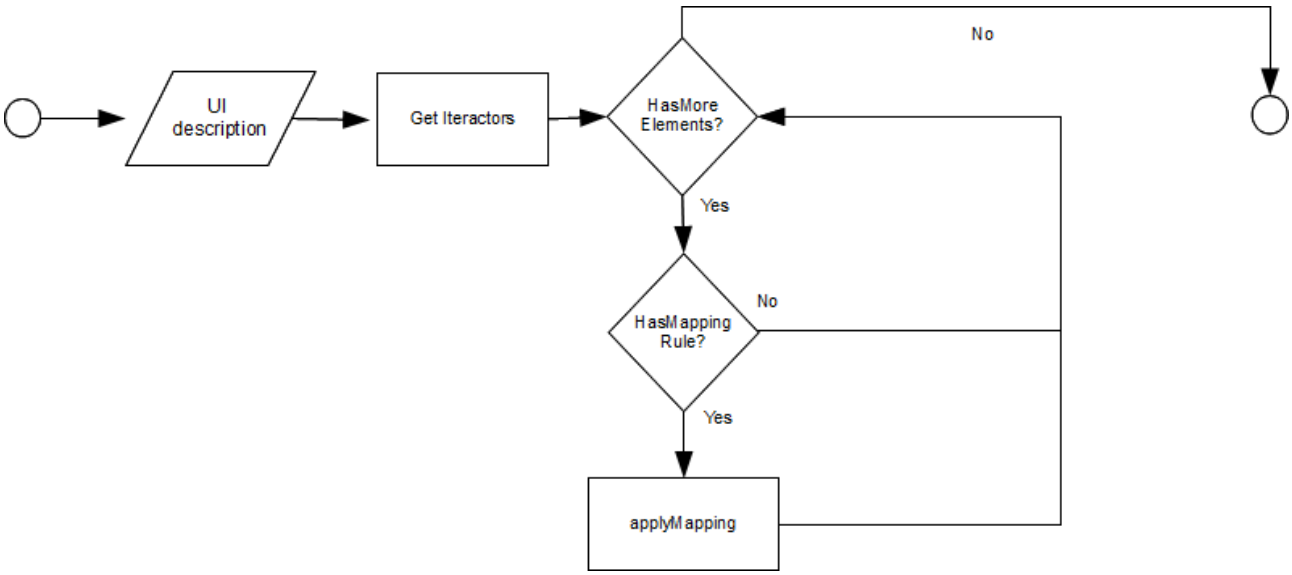
```
READ Ui
FOREACH PageComponent pc IN Ui.pageComponents
   FOREACH PageComponent pc IN Ui.pageComponents
         FOREACH Connection c IN  pc.connections
               IF NOT hasLink(c)
                     IF (isForward(c))
                           generateLink(c, title(c.getTarget())
                     ELSE IF(isPrevious(c))
                           generateLink(c, "Previous")
                     ELSE IF(isMain(c))
                           generateLink(c, "Main menu")
```

Algorithm

For each presentation the algorithm checks all the connections. If the connection is not associated with any link, then it has been generated by the page splitting, so a link for that connection is needed. If the connection is marked as forward, the link content will be set to the connected page title. If the connection is marked as previous the link content will be the word "Previous". If the connection takes to the main page, the link content will be the word "Main menu".

10) Interactor mapping

Precondition:
Input: the applicationUI description.
Output: the Vocal UI description semantically equivalent to the application UI.
Dataflow:



Pseudocode

```
READ Ui
FOREACH Interactor i IN Ui.Interactors
  IF i != null
   IF r = getMappingRule(i) != null
         Apply(i, r);
```

Algorithm
The UI description is taken as input. The procedure executes a loop on each UI elements and look for a mapping rule. If found then apply it.
Comments: a number mapping rules are available, one for almost each UI elements.

5.1.1 Desktop to Vocal

- Overall method: Pipe of the following methods
- Content Optimization (apply 1-2-3)
- Structure optimization (apply 4-5)
- Structure redesign apply(6-7-8)
- Mappings (apply 9)

6 Conclusion

This library provides to developers an extensive set of adaptation techniques, methods and strategies and permits developer to select them according to the context of use, supporting the process of development of adaptive and adaptable applications.

6.1 Final Remarks

In this first release of the deliverable, the techniques, methods and strategies were defined regardless of technology, providing developers flexibility in the implementation, and leaving room for further refinements. However, all the essential information, such as input data, main steps, aimed output and alternative flows were considered and already defined for each of the algorithms described.

These algorithms were already partially implemented during the first phase of the project, they will be concluded to compose the prototype specified for T4.2 Algorithms for AAL.

The decision to build the algorithms in an iterative manner, i.e. describing the adaptation techniques by means of CARF templates, defining Use Cases, and then detailing its dataflow, before actual implementation has many advantages, among which, we highlight:

- Permits defining functions that are common between different algorithms and that can be re-used once implemented
- Provides a set of alternative approaches of implementation, allowing best choices according to the actual context of use
- Permits specific decisions to be taken along the evolution of the project, such as specific details about the algorithms and Serenoa architecture
- Allows the analysis of the techniques in early stages of development, facilitating to abstract, generalize, or specify them according to the actual context of use

6.2 Future Work

The next efforts of this task consist in refining the adaptation algorithms (techniques and methods) and associating them with appropriate strategies for presentation.

Besides, the techniques, methods and strategies will be further investigated and defined in order to compose the advanced adaptation logic with machine learning techniques. These techniques are able to manage compositions of different adaptation rules, and to consider simultaneously multiple context information and adaptation concepts (techniques, methods and strategies).

The algorithms described in this deliverable focus on content adaptation; for the next releases, algorithms to adapt navigation and presentation will be also considered.

The implementation of the techniques will be concluded, and effectively connected to the Serenoa architecture.

References

[Baudish et al., 2004] Baudisch, P., Xie, X., Wang, C., and Ma, W.-Y. Collapse-to-Zoom: Viewing Web Pages on Small Screen Devices by Interactively Removing Irrelevant Content. In Proceedings of UIST 2004 (technote), Santa Fee, MN, Nov 2004, pp. 91-94.

[Brandenburg, 1999] Brandenburg, Karlheinz (1999). "MP3 and AAC Explained" Available at: http://www.telos-systems.com/techtalk/hosted/Brandenburg_mp3_aac.pdf

[Chun et al., 2009] Chan Jun Chun, Yong Guk Kim, Jong Yeol Yang, and Hong Kook Kim, Real-Time Conversion of Stereo Audio to 5.1 Channel Audio for Providing Realistic Sounds, Internation Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 2, No. 4, December 2009

[Feiten et al., 2005] B. Feiten, I. Wolf, E. Oh, J. Seo, and H.-K. Kim, "Audio adaptation according to usage environment and perceptual quality metrics", *IEEE Trans Multimedia*, vol. 7, no. 3, pp.446 - 453 , 2005.

[IMA] IMA Digital Audio Focus and Technical Working Groups, Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems. October, 1992. Available at: http://www.phatcode.net/res/222/files/ima_adpcm.pdf

[Paternò and Sisti a, 2011] F. Paternò, C. Sisti, Model-Based Customizable Adaptation of Web Applications for Vocal Browsing, ACM SIGDOC, Pisa, October 2011

[Paternò and Sisti b, 2011] F. Paternò, C. Sisti, Adapting Desktop Web Pages for Vocal Browsing, INTERACT 2011 Proceedings, LNCS, Volume 6948, Part III pp. 628 – 635, Springer, Lisboa, September 2011

[Ostachkov, 2001] A. Ostachkov (IAG 23M), Analyse et conception d'un Modulateur de Présentation de l'Information sur des Terminaux Mobiles, Multi-Plates-Formes (Master Thesis)

[Vetro, 2004] A. Vetro "MPEG-21 digital item adaptation: enabling universal multimedia access", *IEEE Multimedia*, vol. 11, p.84 , 2004.

Acknowledgements

- TELEFÓNICA INVESTIGACIÓN Y DESARROLLO, http://www.tid.es
- UNIVERSITE CATHOLIQUE DE LOUVAIN, http://www.uclouvain.be
- ISTI, http://giove.isti.cnr.it
- SAP AG, http://www.sap.com
- GEIE ERCIM, http://www.ercim.eu
- W4, http://w4global.com
- FUNDACION CTIC http://www.fundacionctic.org

Glossary

A SERENOA-wide glossary of terms can be found online at:

http://serenoa.morfeo-project.org/glossary-of-terms

BPS: bit per second