PROJECT REPORT

on

DiTaBBu - Digital Talking Books Builder

created at

Faculdade de Ciências da Universidade de Lisboa

by

Rui Miguel do Nascimento Dias Lopes

Summer 2005
Abstract

This report presents a framework for the production of rich Digital Talking Books (DTB), Digital Talking Books Builder (DiTaBBu). DTBs are defined as a set of navigation structures over multimedia contents with multimodal user interaction. Such complexity brought the need of definition of an architecture for processing these structures, the Architecture for Pipelined Processing (APP). With APP, it becomes easier to define all processing components that help defining the production of DTBs. The work done focuses on the implementation of APP and processing components for DiTaBBu, according to several requisites, as well as providing good flexibility in the book production configuration steps. A high-level DTB specification has been also created – High-Level Digital Talking Book (HLDTB) –, as a way to configure easily DTB production (by non-expert users), coupled with a graphical user interface to further simplify the DTB creation batch processing control. An evaluation process was defined and executed, as a way to prove DiTaBBu’s flexibility to create rich and coherent multimodal and multidevice DTBs.

(Projecto de 36 ECTS)
## Contents

1 Introduction 15

2 Requirements 17
   2.1 Digital Talking Books and beyond ............................... 17
      2.1.1 Users ..................................................... 18
      2.1.2 Features ............................................... 18
      2.1.3 User interface ......................................... 19
      2.1.4 Playback platforms .................................... 20
         2.1.4.1 DTBook ............................................ 20
         2.1.4.2 SMIL ............................................... 21
         2.1.4.3 HTML+TIME/XHTML+SMIL ............................. 21
      2.1.5 Annotations .......................................... 22
         2.1.5.1 Annotea ............................................ 23
         2.1.5.2 Ambulant Annotator ................................. 23
      2.2 Requirements for DTB Production Frameworks ................. 24
         2.2.1 Vision ............................................... 24
         2.2.2 Source material .......................... 24
         2.2.3 Technology ........................................ 25
         2.2.4 Users ............................................... 26
      2.3 High Level DTB Specifications ................................. 27
      2.4 Summary ............................................... 27

3 Related work 29
   3.1 Pipeline Processing Architectures ............................ 29
      3.1.1 Cocoon ............................................. 29
      3.1.2 SXPipe ............................................... 30
      3.1.3 XPL ................................................ 30
      3.1.4 NetKernel ........................................... 31
      3.1.5 DPML ............................................... 31
   3.2 Human-Computer Interaction Patterns .......................... 32
      3.2.1 Patterns and languages ............................... 32

5
## List of Figures

4.1 APP architecture .................................................. 35
4.2 APP stages sequence ................................................ 36
4.3 APP stage definition ............................................... 38
4.4 APP example pipeline decomposition .............................. 39
4.5 APP processing component ....................................... 40
4.6 APP Ant implementation .......................................... 42
4.7 APP component development cycle ............................... 42

5.1 DiTaBBu stage decomposition and execution ...................... 46
5.2 DiTaBBu input content structures ................................ 47
5.3 DiTaBBu presentation profiles .................................... 50
5.4 Synchronization visual guides .................................... 51
5.5 Components implementation ..................................... 52

6.1 HLDTB architecture .................................................. 55
6.2 HLDTB implementation mechanism ............................... 59
6.3 HLDTB graphical user interface mockup .......................... 59
6.4 HLDTB content location task ...................................... 60
6.5 HLDTB input devices selection task .............................. 61
6.6 HLDTB output devices selection task ............................ 62
6.7 HLDTB navigation structures task ............................... 62
6.8 HLDTB synchronization presentation task ........................ 63
6.9 HLDTB extra features selection task ............................ 64
6.10 HLDTB output format selection task ............................. 65
6.11 GUI implementation UML packages .............................. 65
6.12 GUI implementation UML classes ............................... 66

7.1 DTB playback on Internet Explorer (as specified by configuration #1) .... 72
7.2 DTB playback on SMIL player (as specified by configuration #3) ......... 73

9
List of Tables

7.1 Evaluation configuration ........................................ 69
Listings

4.1 APP application definition XML ........................................... 37
4.2 APP stage definition XML .................................................. 39
4.3 APP registry XML ............................................................. 41
5.1 DiTaBBu audio timestamp content structure ............................ 47
5.2 DiTaBBu content navigation structure .................................... 48
5.3 DiTaBBu content synchronization arcs definition ...................... 48
5.4 DiTaBBu presentation stage profile ...................................... 50
5.5 DiTaBBu presentation stage profiles registry ......................... 51
6.1 HLDTB configuration snippet ............................................... 56
6.2 HLDTB input devices configuration ...................................... 56
6.3 HLDTB output devices configuration .................................... 56
6.4 HLDTB navigation configuration .......................................... 57
6.5 HLDTB synchronization presentation configuration .................. 57
6.6 HLDTB extra features configuration ..................................... 58
6.7 HLDTB output format configuration ..................................... 58
6.8 Input device selection snippet ......................................... 61
6.9 Output device selection snippet ....................................... 61
6.10 Navigation structures snippet ....................................... 63
6.11 Synchronization presentation snippet ................................. 63
6.12 Extra features selection snippet ..................................... 64
6.13 Output format selection snippet ..................................... 64
7.1 HLDTB specification for a common user with PC ..................... 70
7.2 HLDTB specification for a blind user with low computation power device 70
7.3 HLDTB specification for a partially-sighted user with PDA ......... 71
Chapter 1

Introduction

This report describes the work developed for the Post-Graduation Course on Informatics Engineering (CEPEI) at Faculdade de Ciências da Universidade de Lisboa, centred around the design and implementation of a specialized framework for the creation of DTBs, DiTaBBu. This work is a part of the IPSOM project, a project with participation of research teams from INESC, Biblioteca Nacional and HCIM/LaSIGE. IPSOM stands for “Indexing, Integration and Sound Retrieval in Multimedia Documents”, and its main goal is geared towards increasing the DTBs adoption and dissemination. Several subprojects have been defined along IPSOM’s research goals: DTB players, DTB usability guidelines, DTB adaptative interfaces, and rich content DTBs. DiTaBBu’s goal is to unite all these subprojects in a DTB production framework.

DTBs are the digital counterpart of analog talking books (widely used on blind communities), as a way to extend the book reading experience. As such, several tasks that can be performed on a paper book, are replicated in their digital counterpart (e.g., navigating over the table of contents, skipping secondary contents). However, the IPSOM project tries to go further, by creating new ways of reading books, through content enrichment and widen the target user base (e.g., guided navigation for educational purposes).

Currently, to create a DTB, a set of time-consuming tasks have to be performed manually by specialized users (usually designers) in DTB creation software similar to desktop digital publishing applications. The lack of automation on the DTB creation process delays the acceptance from potential readers of such books. Also, rules have not been defined yet on usability guidelines towards DTB interaction, specially regarding combining several devices (both input and output). As such, there is a need to ease the integration of interaction prototypes, and to ease the creation of DTB usability tests.

DiTaBBu fills this gap, being a fully-automated DTB creation framework, geared towards content creators and interaction producers. This automation potentiates batch processing of large sets of books, enabling DTBs’ quick availability for target users. DiTaBBu’s features reflect several requirements regarding DTB formats, groups of users, and target runtime platforms, without sacrificing content processing modularity, as well as leveraging its configuration tasks.
To achieve these goals, and to ease the development of the building blocks that compose DiTaBBu, a software framework has been created to support the chained processing of books’ structures and contents. This framework, APP [51], is based on the concept of component sequences that process complex input data through an easy (yet powerful) configuration mechanism.

With the ever-growing complexity and customization options of DiTaBBu, a simpler DTB configuration specification was created, High-Level DTB (hereinafter referred as HLDTB), allowing a non-expert user to create configurations easily. This configuration is tied to DiTaBBu, enabling an 100% transparent conversion from HLDTB. A Graphical User Interface (GUI) was also developed to support HLDTB specifications.

The purposed work plan is centred around several activities: study of related work, technical recommendations and standards regarding DTBs and User Interface (UI) generation tools; definition and implementation of navigation structure, presentation, synchronization, and main content interaction specification and generation modules, with emphasis on HTML Timed Interactive Multimedia Extensions (HTML+TIME) output; support structures (indexes, annotations, bookmarks, etc.) specification and generation modules, and its integration with the main content; definition and implementation of several modules for different output formats – the Synchronized Multimedia Integration Language (SMIL) and the DTBook XML element set (DTBook); and, at last, evaluation towards flexibility and modularity of the generated DTBs.

Several contributions were made with the creation of APP, DiTaBBu, and HLDTB. With APP, any kind of Extensible Markup Language (XML) based processing architecture can be designed, implemented, and deployed. DiTaBBu builds upon APP’s features to leverage rich DTB processing, both on production and feature development levels. HLDTB brings an user-friendly GUI to the DTB production process, targeted to non-technical users, increasing the availability of DTBs.

This report is structured in the following way: chapter 2 discusses all requirements for DTB production frameworks; chapter 3 presents related work, with comparisons to other approaches, focusing on their flaws; chapter 4 provides an in-depth view of APP; chapter 5 presents the DiTaBBu framework; chapter 6 content discusses HLDTB; chapter 7 provides some evaluation results, regarding all aspects related to DiTaBBu; at last, chapter 8 presents conclusions and future work.
Chapter 2

Requirements

This chapter describes requirements gathered for all DiTaBBu concerns. First, section 2.1 presents the requirements for DTBs regarding several playback issues (users, features, user interface, platforms). Built upon these, a set of requirements is presented regarding DTB production frameworks. Lastly, another layer is built upon all the previous requirements, towards high-level specifications of DTBs that aim to ease the configuration of DTB creation frameworks.

2.1 Digital Talking Books and beyond

As the world switches information management from analog to digital media, blind or print disabled persons had a chance to have richer talking books, through a bigger feature set and better sound quality, in comparison to analog talking books (usually analog cassettes). The Digital Audio-based Information SYstem (DAISY) Consortium was founded with the goal of developing the standards for the next generation of information technology for persons who are blind or print disabled. Together with the National Information Standards Organization (NISO), the DAISY Consortium developed a specification [31, 26] towards DTBs, based on well-known World Wide Web Consortium (W3C) standards – subsets of Extensible HyperText Markup Language (XHTML) and SMIL. As such, the conversion of print publications began to flourish, enabling print disabled audiences to have better talking books functionality. Along with the specifications, several DTB players were created, as well as some basic supporting tools for content creation.

DTBs can also be a base for production of rich DTBs, where new multimedia content is incorporated into the main content, as well as new navigation methods (e.g., guided tutorials on educational books). Some work towards these issues has been done in the IPSOM project [34, 21, 22, 32, 35, 33].

This section presents several requirements for DTBs: user profiles, content and navigation features, user interface concerns, playback platforms, and, lastly, annotation mechanisms.
2.1.1 Users

DTBs are targeted to several groups of users, where each group has its own special needs, requirements and expectations when using this type of books. The predominant target group is composed by visually-impaired and print-disabled users. Different organizations have been involved with these groups (European Blind Union, DAISY Consortium, USA National Library Service, NISO, amongst others), in order to gather requirements expected from each group. Also, with rich DTBs, the target user spectrum widens. As such, DTBs and rich DTBs can be targeted to several types of users and their respective usage patterns:

- Blind users: this group can benefit from audio-only books, through easy content navigation features;
- Partially sighted users: Enabling enhanced visual components will increase user experience (e.g., increased font sizes, high-contrast color schemes);
- Children: Rich DTBs can help to teach children how to read;
- Students: powerful navigation features can dismiss already-known teaching materials;
- Average user in constraint environments: audio-only books can be used in situations where visual focusing tasks become more difficult (e.g., while traveling by train).

2.1.2 Features

With the differences between each group of target users (as described previously), different editions of books must be adapted to a group’s specific requirements, through the exploration of mixing different types of available media and textual content. As such, DTB content should be compliant with one of the following features:

- Full audio with title element only (allows sequential playback but no random access);
- Full audio with navigation control (adds structured access to the content);
- Full audio with navigation control and partial text (adds limited textual searching);
- Full audio and full text (complete set of features);
- Full text and some audio (provides pronunciation of textual components);
- Text and no audio (full structured text, allowing, i.e., Braille reproduction).

As navigation through the content is a crucial feature that must be available to users, a specific requirement set must relate to content navigation features, as reported on the NISO DTB standard [31]:

18
• Support basic navigation (advancing one character, word, line, sentence, paragraph or page at a time, and jumping to specific segments);

• Fast forward and reverse, reading at variable speeds;

• Navigation through table of contents or through a navigation control file (allowing the user to obtain an overview of the material in the book);

• Cross-reference access, index navigation, and searching, as well as navigating through any secondary content;

• Skipping auxiliary content while reading a book’s main content.

2.1.3 User interface

Such diversity of users and DTB features has a deep impact on the UI. With further exploration of user disabilities, DTBs should provide different combinations of media for presentation (with strong emphasis on the audio component), combined with different interaction modes (e.g., speech interaction). As such, some UI issues must be taken into account (along the lines of voice browsers and multimodal systems):

• The audio format is a temporal medium. The composition of audio and presentation (visual or audio mediums) requires accurate synchronization, so it does not collide with human perception expectations. This issue is critical in audio-only environments, as audio is an one-dimension medium. Also, on the visual medium, the richness of spatial presentation must not interfere with human perceptions [23, 36];

• Voice commands and audio output processing consume the short-term and working memories, conflicting with planning and problem solving tasks. In contrast, visual information is processed by separate cognitive systems, thus being a better medium for content and information delivery;

• Current voice recognition technology is not at production-quality levels, having a lot of imperfections in speech recognition (leading to unavoidable errors), so voice interaction must be coupled with other interaction methods (e.g., keyboard).

As such, the combination of content and several navigation structures, coupled with multimodal interaction, poses cognitive problems that must be dealt with in UI design time [23, 36, 56]. These issues become more important with an increasing diversity of the user spectrum, specially regarding particular disabilities and usage purposes. However, as these issues have not been deeply studied yet, there is a need to test each modality and its impact when other modalities have also been enabled and combined.
2.1.4 Playback platforms

Coupled with the diversity of users, DTB features and UI requirements, there should exist several platforms for DTB content playback. As the DTBs usage varies in different situations, target playback devices must be considered. Three types of usage and playback devices have been characterized:

- Basic DTB player – portable, with simple playing digital audio capabilities (no access to full-text, aiming primarily to play continuous audio);
- Advanced DTB player – also portable, but should allow to access documents randomly, with navigation possibilities, bookmark setting, etc.;
- Computer-based DTB player – complete and sophisticated features for DTBs.

Also, DTB content distribution has a significant role on the dissemination and acceptance of produced books, thus having a strong impact on the supported playback platforms:

- Mechanisms to ease distributed DTB usage should be provided (e.g., access to digital libraries through remote reading places), based on multiple presentation facets and interaction modes;
- Ability to deliver Web browser-based solutions, as well as multi-device solutions, enabling a wider acceptance and dissemination of produced books.

From these requirements and with the limitations imposed by different types of target devices – e.g., braille devices, Personal Digital Assistants (PDA), amongst many others –, coupled with more or less constrained feature sets, there is a need to deliver the DTB content in several platforms, such as DTBook, SMIL, and HTML+TIME. Next, each one of these platforms is described in detail.

2.1.4.1 DTBook

The DTBook Document Type Definition (DTD) [14] defines a specification of the content and structure of books and other publications presented in digital talking book format. Its element set features a subset of XHTML, coupled with specific DTB structural elements (used in book content navigation mechanisms). As DTBs are usually composed by two mediums (audio and text), time synchronization between both is a relevant issue.

A DTBook-compliant book is composed by several modules: textual content, audio, synchronization, navigation, presentation styles, amongst several others. A package file is defined to incorporate all content modules, for final DTB delivery.

Two navigation categories have been defined, as a base for the structural elements definition: global and local navigation. Global navigation is tied to high level book content
navigation (chapters, sections, sub-sections, etc.), as well as skipping of secondary text elements (e.g., sidebars, annotations, footnotes). Local navigation compromises movement within single text elements (lists, tables, etc.), or within a range of text elements (e.g., groups of words, sentences, paragraphs), allowing thinner granularity for navigation (i.e., skipping to the next word, sentence, paragraph, etc.)

However, this complex specification leaves out, wisely, solutions for interaction requirements. Nevertheless, being the most disseminated standard on DTBs, DTBook is an obvious choice for content delivery.

2.1.4.2 SMIL

SMIL [8] is an XML-based language designed for the authoring of interactive multimedia presentations. SMIL allows an author to describe temporal behaviors, associate hyperlinks with media objects, as well as describing layout presentations (both for aural and visual modalities). Although SMIL is a self-contained language, it was designed with modularity concerns, allowing reuse of its syntax and/or semantics in other XML-based languages, such as XHTML or Scalable Vector Graphics (SVG) [38], particularly the modules related to timing and synchronization issues.

SMIL is a feature-rich language as it supports several types of media (e.g., audio, images, text, video) to be synchronized and presented in an easy way. It has powerful synchronization modes (parallel, sequence, exclusive, as well as any composition of these), in the lines of multimedia-supported hypermedia models [42, 43]. Other SMIL modules relate to animation, transition effects, and content control.

However, SMIL has some problems, specially regarding the layout module. This module is responsible for the positioning of media elements on visual interfaces (either in fixed or relative dimensions units). When dealing with well-sized media (e.g., videos and images), layout elements are rich enough to cover all positioning needs, but when text has to be presented, its undetermined size nature difficults the adoption of SMIL as a target platform for text based DTBs. The solution to this issue lays on the incorporation of SMIL with other markup languages (either as host or module), in tune with compound document requirements [7], as available in [60, 55]. As for DTBs not centred in textual contents (e.g., audio-only or image and audio), SMIL layout module features are enough for DTB delivery.

However, as SMIL players are available in many platforms and devices (both in free or commercial versions), SMIL should be considered a target language for DTB content delivery.

2.1.4.3 HTML+TIME/XHTML+SMIL

HTML+TIME [63], adds the SMIL 1.0 feature set to HyperText Markup Language (HTML), bringing the best of the two worlds, in an unique platform. This enables eas-
ier adoption by Web clients. XHTML with SMIL extensions (XHTML+SMIL) provides additional features, as it is a derivative from XHTML and SMIL 2.0 recommendations, following the compound document requirements presented on the previous sub-section, as well as directives from the XHTML Host Language Document Type Conformance [2]. These specifications include features from SMIL modules providing support for animation, content control, media objects, timing and synchronization, and transition effects. These features are directly integrated with XHTML and Cascading Style Sheets (CSS) [16].

Some modules of SMIL were not included in XHTML+SMIL, following the philosophy of using XHTML modules where appropriate. The layout module (the most controversial module, regarding DTB target platforms) is surpassed by the XHTML/CSS layout model. The linking module is replaced by XHTML linking functionality. The structure module was discarded because XHTML was defined as the host language. At last, XHTML metadata elements replace the SMIL metainformation module.

All the features described allow easy adoption of XHTML+SMIL by Web browser developers, leading to a bigger dissemination of working implementations, thus being a good target platform for DTB content delivery.

2.1.5 Annotations

The DTBook standard specifies a portable bookmark and highlight module, allowing users to set bookmarks and to highlight passages in a document, and optionally labeling the marked sections with text or audio notes. The modularity of this mechanism allows its export and import on compliant playback devices. The bookmarks and highlights can be set at any point in a DTB, whether it is based on audio or textual content.

There is, however, a need for richer bookmarking and highlight features, such as richer media (or even multimedia) content support. As the DTBook specification does not fully integrate multimedia in these features, other approaches were studied.

Another issue relates to navigation, presentation and interactions mechanisms for bookmark and annotation features. These can be seen as DTB content (similar to sidebars and footnotes), with the exception that they are created by the final user, not by the content producer. As such, regarding usability concerns, there must be coherence in the UI mechanisms between the DTB production content and the user bookmark and annotation mechanisms.

In order to provide a good annotation mechanism for DTBs, a set of requirements must be fulfilled [18]. As DTBs are used in time-based environments, annotations should be attached to a DTB timeline (either through absolute time, or indirectly through the DTB structure). Annotations should also relate to the content in a whole, not to any particular media. The set of requirements are:

- The document should be digital (aiding search, sharing, and editing);
- The document structure must be exposed;
• Annotations should be allowed to encapsulate other annotations (annotations on annotations);
• The document structure must support interactive annotation;
• Annotations should be attached in the document context;
• The content of annotations should be editable;
• Continuous media annotations should be played with the main content paused, as well as allowing to be played in parallel;
• Conditional viewing of annotations should be allowed (including the support for selectivity across different sets of annotations);
• Annotation creation should be an user-centred task.

Some efforts are being made to support annotation mechanisms, in the lines of the requirements above presented. These efforts are reflected in two technologies: Annotea, and the Ambulant Annotator. Next, brief descriptions of both technologies are presented.

2.1.5.1 Annotea

Annotea project [4] was created to develop collaborative support of annotations, bookmarks, and their combination, for the Web. The annotation concept is defined as notes, explanations, or other types of external remarks that can be attached to any Web document, or to a selected part of the document, without actually needing to modify the document. Annotations are stored in specialized annotation servers, thus creating the notion of portability, as well as expanding annotation collaborative environments. Bookmarking features were also introduced in [46], similar to content-less annotations, but with its own schema, capturing basic concepts also described in [39].

However, as Annotea is geared towards current Web standards, time-based concepts were not an issue to take in account. As DTBs are deeply attached to time concepts (such as synchronization), the need for a more flexible annotation specification is a crucial issue. Nevertheless, annotation distribution and collaborative edition is a strong issue for further research.

2.1.5.2 Ambulant Annotator

Ambulant [3] is a SMIL 2.0 compliant player, available in multiple platforms. On top of Ambulant, work is being developed [18] to provide SMIL-based multimedia annotations, an annotation engine named Ambulant Annotator. This work defines annotations as an expansion and enrichment of the original content, being implemented as a coordinated collection of media items. The annotations are attached to the composite presentation, not to any of the individual media objects (it is a time-based annotation scheme).
A problem arises with this concept: content contextualization. With strict time annotations, navigation through a DTB content becomes difficult. There is a need for content contextualization, helping the reader to better understand the annotation, when navigating through annotations (instead of navigating in the book’s main content). This mechanism has also to be abstracted, so it can be available in all target platforms where reading the DTB.

2.2 Requirements for DTB Production Frameworks

From all previous listed requirements, it becomes unfeasible to create DTBs by hand. As such, there is a need for an automated process for DTB content creation, configuration, and delivery. This process can be leveraged with a DTB production framework. With this type of frameworks, books are delivered in a coherent form, whether on navigation, user interface, or interaction issues.

Thinner requirement sets have been also gathered, regarding source material handling, and content repurposing tasks, on the construction of DTB production frameworks. As DTB source material and target output platforms are XML languages, DTB production frameworks must reflect general-purpose XML digital publishing frameworks requirements, as well as XML processing models requirements. Next, all requirements are presented.

2.2.1 Vision

Multimodal interaction frameworks open paths for information repurposing and creative combination of elementary media. DTBs can be the baseline for new ways of telling stories (e.g., teaching materials), through new or enriched communication scenarios, promoting content reuse from other authors, with document transclusion mechanisms [57, 58]. As such, requirements were gathered, so production architectures can manage these tasks:

- Identification and classification of multimedia units, reused in authoring processes of new DTBs, documents or documentaries;
- Provide clear modularity of book contents (identified with meta-information and content classifiers);
- Enable automatic, semi-automatic and manual content analysis techniques for content repurposing.

2.2.2 Source material

Currently, audio books for visually-impaired people are available in analog audio tapes, usually recorded by volunteers. With the progressive introduction of digital technologies in libraries (e.g., the Portuguese National Library), by providing digital content (usually
in XML/HTML envelopes), there is a need for integration of both mediums, coupled with DTB specific functionalities. These issues leverage source material handling requirements:

- Promoting a clear separation between the books’ contents (including the logical and semantic structure – e.g., media correspondence) and the books’ user interface. This will reinforce coherence amongst different usage settings of the same book;

- Automation of medium integration processes (as there is a huge amount of audio tapes and digital books already available on libraries);

- Identification of speech excerpts, within audio components (correspondence with textual content units);

- Simple and reusable specification of user interfaces (enabling the creation of DTB collections).

The INESC speech recognition research team has been working on speech alignment technologies [64], along the IPSOM project context, as a way to automate audio book indexing of its textual counterpart. This work delivers digital content in special purpose file formats, thus being used as an initial source of content for the production of DTBs.

2.2.3 Technology

When we think on the source and target material, XML digital publishing architectures and applications can be very complex. They have to manage complex input sets, handle several transformation steps and different output sets, be able to support several configuration possibilities, support batch publishing control, etc. To further explore the rich features this type of architecture has, without sacrificing scalability and maintenance, an n-tier approach has to be defined towards content processing. This kind of flexibility leverages the following requirements towards the definition of a robust and extensible digital publishing architecture:

- The ability to handle complex inputs;

- Deliver complex processing outputs;

- Ability to define logical stages of processing;

- Support different processing configurations;

- Focus on content processing reuse and maintenance;

- Ease batch processing of sets of contents;
XML digital publishing architectures are based on generic XML processing models. These models have a special set of requirements [49], concerning conceptual models of XML process interactions and the language for the description of these interactions. The processes of interest in this requirements set can be summarized by the following operations over XML information sets [25]: creation, inspection, augmentation, extraction and transformation. With focus on this set of operations, the most relevant requirements are presented:

- The language should be as small and simple as possible;
- The language must allow the inputs, outputs, and other parameters of a components to be specified;
- Given a set of components and a set of documents, the language must allow the order of processing to be specified;
- It should be relatively easy to implement a conformant implementation of the language, but it should also be possible to build a sophisticated implementation that can perform parallel operations, lazy or greedy processing, and other optimizations;
- The model should be extensible enough so that applications can define new processes and make them a component in a pipeline;
- The model should allow multiple inputs and multiple outputs for a component;
- The language should be expressed in XML. It should be possible to author and manipulate documents expressed in the pipeline language using standard XML tools;
- The pipeline language should be declarative, not based on Application Programming Interfaces (API);
- The model should be neutral with respect to implementation language.

2.2.4 Users

DTB production frameworks should be highly modular, to support different user profiles, as different framework concerns should be handled by different users. A clear separation of user tasks has to be reflected on the architecture model. In typical digital publishing architectures, top level users (whom have little technology expertise) specify which content is going to be processed, and control the batch processing of book collections. Other users should be able to specify which processing steps have to be applied on a given content, through some configuration steps. On the other hand, there is the developer. The developer creates processing tasks and integrates these into the main architecture.
2.3 High Level DTB Specifications

As DTB production complexity grows in many areas (content, processing, features, user requirements), the levels of technology expertise needed to create DTBs and rich DTBs rise. There is a need to lower these levels, opening the way for non-technical experts to produce DTBs. This way, batch production of DTBs becomes feasible, allowing the delivery of book collections with coherent user interfaces, and with seamless interaction as well. To achieve this goal, a language must be defined to describe a DTB production framework configuration, with the following set of requirements in mind:

- The language must be easy to understand (as small and simple as possible);
- The language has to reflect all DTB-related requisites;
- The mapping between the language and the framework components should reach as much components as possible (as well as their parameters).

2.4 Summary

To create a flexible framework for DTB and rich DTB production, many requirements must be fulfilled, in different aspects. DTBs must be available to different user profiles (e.g., visually-impaired, partially-sighted, children, etc.), with a rich set of features (e.g., flexible navigation structures). A coherent UI must be made available in every DTB, regarding the target user profile whom the DTB will be available. Also, different playback platforms must be taken into account when producing DTBs. Special requirements have been gathered also, regarding DTB annotation mechanisms (and similar end-user interaction, such as highlighting and bookmarking). All these DTB requirements, being so diverse, need a flexible and solid base for DTB production (in the form of an XML digital publishing architecture). Amongst its requirements, the emphasis goes to content analysis and reauthoring, content normalization, technological issues (regarding DTB processing components), as well as to the different user tasks on DTB production processes. At last, requirements for high level constructs on DTB production tasks must be defined, as a way to ease the specification of DTB user interfaces, promoting the adoption of DTB production frameworks.
Chapter 3

Related work

This chapter presents related work regarding pipeline processing architectures (section 3.1), discussing current architectures’ flaws and the need for the implementation of a new architecture, to accomplish all requirements gathered earlier. Also, related work on human-computer interaction patterns (a way to reuse well-known UI features in a coherent way) is presented on section 3.2.

3.1 Pipeline Processing Architectures

As XML [71] technologies become a de facto standard on digital publishing, the need for powerful (yet developer-friendly) supporting frameworks rises. These frameworks have to deliver content in different formats, through content processing and transformation steps.

In the following sections, several XML processing architectures and related technologies are presented: Cocoon [6, 47], Simple XML Pipelines (SXPipe) [69, 70], XML Pipeline Language (XPL) [17], NetKernel [59] and Declarative Processing Markup Language (DPML) [30].

3.1.1 Cocoon

Starting as an XML publishing framework, Cocoon soon evolved to a full-featured XML Web development framework. Its concepts focus around separation of concerns [29] (enabling interaction and collaboration of developers in a project), and component-based Web development. It is a very flexible framework, being able to work with multiple kinds of data sources (e.g., plain XML files, relational databases, and others), delivering content in multiple formats (e.g., XML, HTML [62], and Portable Document Format (PDF) [61]).

An application created with Cocoon is specified in a sitemap, a group of processing pipelines. Cocoon uses the notion of pipelines as a way to compose Web applications without programming. A pipeline is defined as a group of matchers, where each matcher is responsible for the delivery of a single unit of content. A matcher performs three consecutive tasks on a content unit: generate, transform and serialize. Each matcher can
depend on other matchers from the same pipeline, thus creating a dependency-based chain of matchers, inside a single pipeline.

This dependency-based chaining approach suits well on Cocoon purposes (i.e., Web development tasks). However, this Web orientation is single document based, as web users only see one document at a time. This feature can be seen as a limitation, when developing complex digital publishing applications, as these tend to lean towards offline content processing and generation, where multiple content generation and processing is a common practice. The dependency-based approach is not able to handle secondary outputs, as each matcher only specifies one output. Secondary outputs generated in a matcher are “left in the wild”, being unable to reach them from other matchers (thus breaking the dependency-based approach). Also, the separation of concerns achieved in Cocoon relates to each pipeline matcher, not for content processing steps (as these are blackboxed by matchers).

3.1.2 SXPipe

SXPipe is a language for describing simple XML pipelines, towards lightweight processing of XML information sets [25]. It was created as a substitute for general-purposed build tools (such as make or Ant), towards simple XML transformations. The pipelines are defined by simple components which perform actions over a given input (document inclusion, validation, transformation and serialization). Implementations of SXPipe are given an input document, process it with the pipeline specification, resulting in an output document.

As simplicity is the centre of SXPipe, several issues are left behind, regarding XML processing architectures requirements. When multiple documents are needed as the source of a pipeline, a pre-processing step of inclusion is performed (e.g., XInclude [54]). This feature is acceptable in SXPipe, but not for complex XML processing architectures, a better clarification of input sources is needed. Another issues relates to multiple output documents. As it is left to implementations how to handle pipeline outputs, it is implicit that each component hides its multiple output serialization issues from SXPipe.

3.1.3 XPL

XPL was created to fulfill all requirements of XML processing models (as described in section 2.2.3). This language defines an XML vocabulary for describing a processing model for XML components, specially regarding to XML information sets. The operations are composed in a pipeline, where infosets are created, processed and serialized. Each pipeline component describes which inputs and outputs it will be connected to, allowing the direct manipulation of several inputs and outputs infosets. Coupled with operations, some business logic can be applied (iterations and choices), to further enable flexibility of pipeline composition. Specifying parameters to each component is made by adding an
input infoset with its description.

XPL is a very powerful specification, geared towards both web application architectures, as well as complex offline XML processing. However, some issues arise in this specification. As parameters are treated as infosets (making it impossible to distinguish parameters and other inputs in a pipeline definition), it is up to each component to perform verification if a parameter is mandatory or optional. This feature should be performed by XML processing architectures, as a pre-processing step. Other issue relates to the limitation of working only with XML infosets. Non-XML sources are discarded, disabling access to all non-XML input sources (e.g., legacy systems, Relational DataBase Management Systems (RDBMS), etc.) Another problem with XPL relates to the composition of pipelines. As this feature is not introduced in XPL, it becomes impossible to modularize multiple pipelines with separation of concerns without using external composition tools.

3.1.4 NetKernel

NetKernel is an XML composition framework based on the notion of using Uniform Resource Identifiers (URI) [13] as declarative expressions of XML processing executions – in a Common Gateway Interface (CGI) -like fashion –, through a dynamic URI resolver. NetKernel defines a new URI scheme, called Active URI, consisting of a base part followed by any number of named arguments. Active URIs are defined and mapped to processing tasks, such as an XML Stylesheet Language (XSL) transformation [45], WebService invocation, amongst many other. This URI approach enables easy access to caching mechanisms already available (e.g., proxy servers). All computed resources can be combined in dependency chains, a factor that potentates caching.

While performance boosts are achieved through the combination of caching and dependency chains, the tradeoff comes with its counterintuitive nature. It is easier to create processing tasks in linear declarative expressions. Another issue relates to the URI scheme approach. This model is difficult to understand by newcomers, being too much error prone when complex tasks have to be created (i.e., real world applications). As NetKernel handles only a set of execution tasks, the combination of several tasks has to be specified outside NetKernel. For pipeline processing architectures, it should be possible to specify all tasks, as well as how each one relates to the others. The active URI approach also disables access to multiple outputs created on each processing task.

3.1.5 DPML

DPML is a declarative layer built on top of NetKernel. DPML was developed to ease the creation of Active URIs, because anything above a first order active URI is too difficult to work with directly. The language constructs of DPML (based on XML) are mapped one-to-one to an active URI.

Being a layer on top of NetKernel, DPML inherits all its benefits, as it has a strong
architecture base. Solving some of the NetKernel’s flaws is a goal achieved by DPML (especially simplifying the creation of active URIs). But as complexity grows (e.g., in digital publishing applications), the need for higher level abstractions and separation of concerns issues arise, areas where DPML does not fulfill. DPML is not able also to support the creation of multiple outputs, like its low-level counterpart.

3.2 Human-Computer Interaction Patterns

A first description of patterns was introduced in the field of architecture [1], stating that “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”. This concept was embraced by computer science especially in object-oriented programming [40], but also on the Human-Computer Interaction (HCI) field [15].

Next, HCI patterns and languages are presented, followed by related work on HCI patterns supporting tools.

3.2.1 Patterns and languages

HCI patterns describe proved UIs for tasks, either in high-level constructs (such as informal languages), or in model-based pattern repositories. These patterns are structured and organized in groups (by function or similarity), to ease the selection of individual patterns.

Welie and Veer [68] suggest that pattern languages for interaction design can help to classify the growing number of HCI patterns. While pattern languages are created for specific domains, efforts are being made to create complete languages to be used across target domains. Patterns classification is structured in a way that it becomes easy to filter unwanted patterns, given a context (e.g., if a task execution time has to be reduced, patterns are filtered towards the ones that may increase entry speed or have impact on the error rate).

Tidwell [67, 66] defines a repository of well-known and proved UI patterns (usually these are empiric proofs, tested through time). This repository defines when, why, and how to use UI patterns, whether they are used on Web sites, or desktop applications. By abstracting target platform of usage, this repository enables a better interchange of target-specific patterns (e.g., applying a Web-style top-level navigation in desktop applications).

Current work on UI patterns focuses on well-known navigation patterns, and composite layout patterns, mainly targeted to desktop based UIs. Some work is done in the creation of multi-device UI patterns with relative success, by applying different patterns to the same problem, each one for a specific device [50]. However, UI patterns lack interaction support, and time-aware issues.
3.2.2 Tools

Some UI prototyping tools are already available with built-in pattern-based support. This particular feature enables a better acceptance of HCI patterns usage, widening better usability results across UIs.

Damask [50] is a tool designed for better support of multi-device UI prototype design. With Damask, a designer creates a UI for one device, by sketching the design, paired with specifications of which design patterns the interface uses. These patterns are available for all devices, in a pattern library, helping Damask generate specific DTBs for each target device.

CanonSketch [20] is a Unified Modeling Language (UML) -based tool for model-based user interface design, with emphasis on Canonical Abstract Prototypes [24]. This tool fills the gap between expected the user behavior and the implemented UI, by specifying UIs in a designer-centred environment. CanonSketch supports UI patterns (e.g., wizards, previews, etc.), as well as the creation of new patterns, in an abstract definition. This abstraction allows further conversion to working implementations in any environment (such as an HTML page).

However, both Damask and CanonSketch are tools created for general user-centred UI prototyping, having no support for specific DTB UI patterns features.

3.3 Summary

To ease the management of DTB production tasks, a flexible processing architecture must be used. However, current pipeline processing architectures do not fulfill the requirements for a flexible production of DTBs. Most of them, do not take into account issues regarding delivering multiple outputs, per processing component, or separation of concerns in the development, configuration, and execution tasks.

Another issue relates to human-computer interaction patterns, as these define a crucial path to the high level specification of DTB UIs. However, current pattern languages and tools do not take into account any particular DTB concern, thus being not applicable to the high level specification of DTBs patterns.
Chapter 4

APP

This chapter describes APP [51], Architecture for Pipelined Processing, an XML framework designed to support the creation of transformations over complex input sets, as seen on figure 4.1.

![APP Architecture Diagram](image)

Figure 4.1: APP architecture

The following section describes APP’s architecture in depth. After, in section 4.2 implementation issues are described.

4.1 Architecture

APP was developed to ease the development of complex digital publishing applications, by giving the ability to compose different processing components seamlessly, especially regarding to separation of concerns issues for creation, management and maintenance of applications and its respective contents, as well as a way to solve dependency-based processing issues.

APP defines an application as a forward processing chain of stages, where each stage has a set of inputs and a set of outputs well defined. The first stage’s input is the appli-
cation input and the last stage’s output is the final output resulting from the processing chain (as seen on figure 4.2). All other stages use its previous one’s set of outputs as the input for processing. This way, applications can create logic separation of concerns between the processing of inputs. The need to split the processing in stages arises from the nature of digital publishing applications (and, more generally, any kind of hypertext application) where several layers over contents are well-defined, as stated in [41]. As such, the processing of each layer has to be split in the same conceptual layers.

![Figure 4.2: APP stages sequence](image)

Each stage has a set of independent processing pipelines that process independent subsets of the stage’s input, resulting also in independent subsets of the stage’s output. Subsequently, each pipeline is defined by a sequence of processing components that will handle all transformations and/or generations of the current pipeline input. All APP processing components defined by an application are registered in a central registry, and then referred throughout all pipeline definitions.

A forward processing chain approach adequates better to the creation of digital publishing applications, mainly by it is intrinsic nature to handle a set of inputs and apply a sequence of transformations (specified in a rule-based fashion - APP’s pipelines), resulting on a set of outputs. This way, the modelation of which transformation steps are to be executed, and in which order, is clearer to the user who configures the pipelines (usually it is not a developer who performs these tasks).

Another benefit from using this approach is the way complex inputs and outputs are handled. Often, on a complex digital publishing application, processing components handle a subset of the input, having as output result the modification of its input and/or the creation of new content. As a consequence of this architecture definition, managing all this complexity is easier with the use of APP.

An APP application is specified in an XML document based on Resource Description Framework (RDF) syntax [10], describing metadata associated with the application, in Dublin Core format [27, 11], and also the sequence of processing stages. These two descriptions are identified within the document with well-known URIs in `rdf:Description` blocks: `urn:app:project:metadata` for application metadata-related information, whereas `urn:app:project:stages` is used for the stage sequence definition. Splitting stages in dif-
different files, separation of concerns is further reached because each developer can centre his/her work independently on each stage, as well as stage modularization and reuse becomes an easy task.

As an use scenario, for better illustration on how to use APP, a simple application for production of cooking recipes will be used through the chapter. The goal of this particular digital publishing application is to repurpose a given cooking recipe (or a set of these), following several criterias, as well as the ability to use different types of output formats (e.g., printing, web-based, rich multimedia, etc.) As input, several contents are to be used: the recipe (split in several files, describing all steps), a list of ingredients and where to buy them, similarities between ingredients, etc. The repurposing of a recipe is defined through several steps in the first stage (repurpose.xml), while output format steps are defined in the second stage (output.xml), as presented in Listing 4.1.

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="urn:app:project:metadata">
    <dc:title>Cooking recipes</dc:title>
    <dc:author>Rui Lopes</dc:author>
  </rdf:Description>

  <rdf:Description rdf:about="urn:app:project:stages">
    <rdf:Seq>
      <rdf:li rdf:resource="repurpose.xml"/>
      <rdf:li rdf:resource="output.xml"/>
    </rdf:Seq>
  </rdf:Description>
</rdf:RDF>
```

Listing 4.1: APP application definition XML

Next, in the following subsections, APP will be further discussed through a thinner decomposition of the architecture’s structure (stages, pipelines and components), and an analysis is made on the relationship between all APP concepts, in the process of creation of digital publishing applications.

4.1.1 Stages

A stage is defined as a conceptual subdivision of a digital publishing application defined in APP. It is a step of all processing, with well-defined set of inputs and outputs. This way it can be inserted in the global processing pipeline, as long as the previous stage output contains all the input of the current stage. This rule is applied to the next stage of the application, and so on.

Each stage can be decomposed in a set of processing pipelines (as seen on figure 4.3). The stage feeds a different subset of its input to each pipeline, and executes the transformations specified in each one. The resulting outputs from the pipelines are aggregated, creating the stage final output. Each pipeline is independent from other pipelines of the
same stage, so that every output from the pipelines does not collide when the stage aggregates all outputs. As long as this rule is not broke, there is no restriction on the number of pipelines a stage can manage.

A stage specification is defined on an XML document, with a special purposed dialect created for this task, under the XML namespaces [48] urn:app:stage:config and urn:app:component:registry. The document root tag config encapsulates all pipeline definitions for the stage (each one defined with pipeline tags, enclosing all processing tasks to be executed sequentially on that pipeline). A sample stage specification is presented on listing 4.2.

4.1.2 Pipelines

A pipeline is a sequence of logical processing steps (defined with components) applied to a set of inputs, resulting on a set of outputs. On these steps, not only the set of inputs can be modified accordingly to the processing rules defined in each processing component, but also new content can be generated by them. The aggregation of all results that comes from these steps will be the pipeline’s final set of outputs.

The composition of components in APP is defined in a way that becomes feasible to handle the processing of multiple inputs and outputs in a transparent way, when specifying a pipeline definition. This way, the inter-component conjugation becomes more flexible, as seen on figure 4.4. A simple example to further illustrate this flexibility, based on this figure, can be seen in the execution step of component C6. This component has C4 as the only component with outputs that will be used as inputs in C6. In the same way, C4 uses as its input, the outputs from C1 and C3, and so on. While this approach is similar to dependency-based ones, C6 does not have to use all of C4’s outputs; it can use a subset of this.

Although the complexity level for this kind of inter-component links is high (inherited
from the complex nature of digital publishing applications), the specification of the sequence of processing steps for each pipeline is much simpler (as seen of listing 4.2). All inter-component links are automatically established by APP’s pipeline execution, through the linkage of each component’s inputs and outputs definitions, as specified on the registry.

With this approach, APP allows to explicitly define each processing pipeline, starting from the input, finishing on the output, instead of defining the pipelines through dependency chains, as discussed previously.

Listing 4.2 shows two pipelines defined in the repurposing stage of our cooking recipes sample application. The first has components to modify a recipe sauce (first, by removing potentially expensive ingredients, and then by adding some salt). The second pipeline describes modifications done to a salad (the addition of toppings). Both pipelines handle different subsets of their respective stage input. Creating this type of specifications becomes easier with APP, comparing to its counterparts (as described in chapter 3).

```xml
<config xmlns="urn:app:stage:config" xmlns:reg="urn:app:component:registry">
  <pipeline>
    <component reg:idref="cook:sauce:cheaper">
      <param name="max-price" value="5.00" />
    </component>
    <component reg:idref="cook:sauce:saltier" />
  </pipeline>
  <pipeline>
    <component reg:idref="cook:salad:topping">
      <param name="type" value="olive_oil" />
    </component>
    <component reg:idref="cook:salad:topping">
      <param name="type" value="vinegar" />
    </component>
  </pipeline>
</config>
```

Listing 4.2: APP stage definition XML
4.1.3 Components

A processing component is defined by APP as a single unit of processing of a set of inputs, resulting in a set of outputs after its execution. To increase the configuration possibilities for each component, two types of descriptors are associated: links and parameters. Links define which set of input sources are fed to the component, and also which set of outputs are produced by it. This creates a high-coupling interface for a component. However, through the identification of each link with an URI, it is possible to define a low-coupling interface to the component, so it can be reused with different sets of input sources, on any pipeline and any stage. On the other hand, parameters are defined to allow a higher configurability of the component (see picture 4.5).

![Component diagram](image)

Figure 4.5: APP processing component

Each component is registered in the central registry in an RDF resource list with an unique identifier, through the attribute `reg:id` (so it can be referenced in a pipeline definition), and the component location. Implicitly, all processing components are XML Stylesheet Language Transformations (XSLT) stylesheets (for APP prototype implementation purposes).

A special purpose component was created, under the unique `reg:id core:passthrough`, to allow input sources tunneling between stages. If an input source does not need to be processed in a stage, but is needed in a subsequent stage, the `core:passthrough` component can be used for this task without breaking separation of concerns of stages.

Associated with the component register, a small set of metadata is also defined. Identified with the URI `urn:app:component:metadata`, it is presented information about the component (defined with Dublin Core specifications). With the URI `urn:app:component:plugs`, APP knows which links are defined to the current component, to allow inter-component linkage in the pipeline execution steps, either with high-coupling or low-coupling interfaces. The creation of an URI in the `plug:urn` attribute of a link, enables a low-coupling interface for the component. At last, with the URI `urn:app:component:params`, APP allows parameters for the component to be specified. On the other hand, on the pipeline definition, components are referred as stated before, but parameters can be added, as well as low-coupling interface links. Parameters are defined with `param` tags and the attributes `name` and `value`. Links are defined with `plug` tags and the attributes `urn` and `href`. 
In the cooking recipes example, several components have been registered (see listing 4.3 for a component registry entry), creating the links between the registry identification, and the stylesheets to repurpose the recipes description and output creation.

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:reg="urn:app:component:registry">
  <rdf:Description rdf:about="urn:app:registry">
    <rdf:Bag>
      <rdf:li reg:id="cook:sauce:cheaper"
        rdf:resource="recipe/cook/sauce/cheaper.xsl">
        <!— processing component metadata —>
        <rdf:Description rdf:about="urn:app:component:metadata">
          <dc:title>Cheaper sauce repurposing</dc:title>
          <dc:creator>Rui Lopes</dc:creator>
          <dc:description>
            Recipe repurposing, by modifying sauce to use cheaper ingredients
          </dc:description>
          <dc:date>2005-06-06</dc:date>
        </rdf:Description>
      </rdf:li>
      <!— processing component plugs —>
      <rdf:Description rdf:about="urn:app:component:plugs">
        <plug:in>
          <rdf:Bag>
            <rdf:li rdf:resource="recipe.xml" plug:default="yes" />
            <rdf:li rdf:resource="ingredients.xml" />
            <rdf:li rdf:resource="prices.xml" />
          </rdf:Bag>
        </plug:in>
        <plug:out>
          <rdf:Bag>
            <rdf:li rdf:resource="recipe.xml" plug:default="yes" />
          </rdf:Bag>
        </plug:out>
      </rdf:Description>
      <!— processing component parameters —>
      <rdf:Description rdf:about="urn:app:component:params">
        <rdf:Bag>
          <plug:param name="locdoc" use="required" />
          <plug:param name="max-price" use="required" />
        </rdf:Bag>
      </rdf:Description>
    </rdf:Bag>
  </rdf:Description>
</rdf:RDF>
```

Listing 4.3: APP registry XML

4.2 Implementation

Current APP implementation prototype was developed with Ant [5, 44]. This Java-based build tool performs tasks similar to well-known build tools, such as `make`, `gnumake`,
Ant build files are specified in XML (in opposite to `make`'s shell-based commands), describing task execution trees. These tasks are implemented in Java, thus being cross-platform. By default, Ant build files are named `build.xml`.

![Figure 4.6: APP Ant implementation](image)

The implementation reads an APP application XML specification and, with two XSLT stylesheet helpers (`genmain.xsl` and `genstage.xsl`), transforms it into several Ant scripts (see figure 4.6). `genmain.xsl` accepts as its input an APP application definition file, transforming it into an equivalent Ant build script. This script performs the preparation actions for each stage (cleaning temporary values, creating directories, etc.), and executes each stage specification. `genstage.xsl` reads a stage definition file, performing a transformation to its Ant equivalent commands (e.g., applying a certain processing component to a subset of a given content). This higher level expression between Ant and APP application definitions is similar to the abstraction layers built around the `make` tool, such as the `makemake` [9], `automake` [53], and `autoconf` [52] tools.

The component-based approach of APP eases the development of complex digital publishing applications. The need of mapping requirements to the application is done by creating processing prototypes, tested over sample contents. Then, these components can be integrated into APP’s registry, and integration tests can be performed. This approach can be seen on figure 4.7. APP includes a simple mechanism for component integration tests, based on XPath [12] expressions.

![Figure 4.7: APP component development cycle](image)
4.3 Summary

In this chapter APP has been presented. Being a flexible architecture for pipelined processing of contents, APP leverages the creation, maintenance, configuration, and execution of any production task. With the promotion of configuration reuse, APP is a good tool for content sets batch processing. To prove APP’s architecture, a compliant prototype has been implemented and tested.
Chapter 5
DiTaBBu

Manual DTB production is a time-consuming task, being an easy target for error prone DTB creation (regarding all requirements gathered), potentiating broken UIs, thus delivering lower-quality books with poor usability concerns. As such, there is a need to leverage all this work towards fully automated DTB production, ready for content delivery, with coherent UIs and well-tested usability guidelines conformance. Therefore, DiTaBBu has been created as a framework to ease the modular production of DTBs, in a fully automated process.

The following sections present in more detail DiTaBBu’s architecture (section 5.1) and implementation (section 5.2).

5.1 Architecture

DiTaBBu, being defined as a framework for production of DTBs, was developed as a direct application of APP. This way, DiTaBBu inherits all benefits of APP (e.g., modularity, separation of concerns, etc.), easing the flexibility on exploratory and adjustable UIs generation of DTBs.

DiTaBBu is composed by four conceptual processes towards DTB production: structure repurpose, output format definition, interaction, and presentation. This division is mapped directly onto APP’s stage concepts, using APP as DiTaBBu’s execution platform (as seen on figure 5.1). Each stage has a set of specific processing components, used to perform transformations towards book final output.

The structure repurpose stage centres its activities around content manipulation and standardization (as well as its main structure and auxiliar navigation structures). The following stage, output format, converts the previous stage’s output to a target execution platform format. Following, interaction mechanisms are coupled with the book structures. At last, presentational design and behavior are attached, resulting in the final book. An early design option focused the output format as the second stage in DiTaBBu because interaction and presentation components implementation is specific to each target output platform.
Configuration reuse is strongly advised in DiTaBBu, as the way to create coherent UIs amongst book collections. This reuse leverages the batch processing of DTBs. Reuse can be performed at three different levels: single stage configuration, multiple stage configuration, and total configuration. With a single stage configuration, DiTaBBu maintains a single concern coherent amongst several configurations. This is useful especially in book content repurposing tasks (e.g., cleaning up book structures, and normalization of input formats). With multiple stage configuration reuse, DiTaBBu is able to deliver a DTB in coherent UIs targeted to different platforms – e.g., PDA and Personal Computer (PC). At last, total configuration reuse is advised on book collections creation processes.

Next, book content is defined (in its normalized structures), and each DiTaBBu stage is presented in more detail (structure repurpose, output format, interaction, and presentation), especially regarding implemented components and the relationship with previously gathered requirements.

5.1.1 Normalized content definition

DiTaBBu defines its input content in two different sets (see figure 5.2): main content, and navigation content. The first is composed by the main book structure (coupled with textual content and anchor points), audio timestamps (together with respective anchor points), and synchronization arcs to bridge anchor points. The second content set is defined by generic navigation structures (e.g., table of contents, sidenotes, etc.) and their anchor points, and the corresponding synchronization arcs (linking between navigation and main book structures anchor points). This generic content normalization was defined to enable flexibility towards content repurposing.

The main book structure is DTBook compliant (similar to HTML document structure), as this specification covers all document structure aspects (e.g., chapters, sections, paragraphs). However, special purpose languages have been defined for every other content structures, as the DTBook specification is not flexible enough for content repurposing tasks. Audio timestamps and navigation structures are defined in an special XML language
Figure 5.2: DiTaBBu input content structures

based on RDF and XLink [28], as seen on listings 5.1 and 5.2. Inside rdf:Description block, metadata and XLink anchors are defined, under the URIs urn:ipsom:component:metadata and urn:ipsom:component:locators, respectively. In the locators block, each anchor is defined in a loc:item element, where loc:id identifies the anchor itself, and xlink:href identifies the content location. Audio content locations are defined in audio timestamps, in a special URI formatted as time:start_timestamp,end_timestamp (timestamps are in seconds). As textual representation of spoken text can differ from its corresponding book element (e.g., three vs. 3), doctor vs. Dr.), the xlink:title attribute is used to identify the audio segment textual representation (as seen on listing 5.1). On the other hand, navigation structures define a special purpose URI locator, with the format content:#loc (where #loc locates a corresponding rdf:Description block inside the navigation structure, with the URI urn:ipsom:component:content#loc). An example is provided in listing 5.2.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:xlink="http://www.w3.org/1999/xlink"
         xmlns:loc="urn:ipsom:component:locators">
  <rdf:Description rdf:about="urn:ipsom:component:metadata">
    <dc:title>O Senhor Ventura audio</dc:title>
    <dc:creator>Rui Lopes</dc:creator>
    <dc:description>...</dc:description>
    <loc:plug type="audio/mp3" rdf:resource="audio.mp3" />
  </rdf:Description>
  <rdf:Description rdf:about="urn:ipsom:component:locators">
    <loc:loc xlink:type="extended">
      <loc:item loc:id="t1" xlink:type="locator"
               xlink:href="time:0.48,0.89" xlink:title="primeira" />
      <loc:item loc:id="t2" xlink:type="locator"
               xlink:href="time:0.89,1.63" xlink:title="parte" />
    </loc:loc>
  </rdf:Description>
</rdf:RDF>
```

Listing 5.1: DiTaBBu audio timestamp content structure
Listing 5.2: DiTaBBu content navigation structure

Regarding synchronization arcs between the main content and other content (both audio timestamps and navigation structures), these are also defined in an RDF and XLink-based language, linking from main content anchors to the other content’s anchors (see listing 5.3, showing synchronization arcs between main content and audio timestamps).

Listing 5.3: DiTaBBu content synchronization arcs definition

5.1.2 Structure repurpose stage

This stage is responsible for the normalized content structures delivery, either with enriched or original content. The normalization step is needed, as a way to support different input sources (leveraging conversions of already produced books in other formats), whether in XML-based formats (e.g., XHTML, DTBook) or non-XML formats (e.g., HTML, PDF, or speech alignment generated content).

Content extraction features are implemented in this stage, enabling the access to auxiliary navigation structures not explicitly decoupled from the normalized content (e.g., indexes, table of contents, sidenotes). This decoupling matches navigation requirements...
for skipping book’s auxiliar contents. Special purpose processing components infer over the normalized content, creating new navigation structures.

DTB structures can be huge and very complex (e.g., dictionary, encyclopædias, etc.) As defined by requisites, low computational devices are targets for DTB content delivery. As such, there is a need to simplify these structures, allowing the playback of produced DTB content on those devices. This task has been defined as granularity control, and has been implemented in the structure repurposing stage. The grain is the deepest content structure element to keep after this repurposing task. All deeper elements are discarded (but their contents are kept on the grain elements). This way, less synchronization steps are performed by the target runtime platform.

Navigation structures are deeply linked to the main content. As an user must be able to navigate throughout a book with these structures, there is a need for contextualization when jumping around the content (e.g., returning to a bookmark point). This task can be leveraged if a navigation structure does not point to arbitrary content anchors (such as specific words or phrases), but to higher level structure elements (e.g., paragraphs, sub-sections, etc.) These structures have to be in tune with post-granularity control tasks performed earlier.

5.1.3 Output format stage

This stage is responsible for the normalized content structures transformation to different output formats: HTML+TIME, SMIL, and DTBook. Currently, only these three formats are defined, but any other target format can be plugged into this stage. Each output format has its specific issues (e.g., the format language itself, multiple output content spanning, etc.) handled by specific components developed for this purpose.

Integration of anchoring points is an important task performed on this stage, both on book content and navigation structures, aided by synchronization arcs. User-customized navigation structures mechanisms (i.e., bookmarks, highlights, and annotations) are also integrated in this stage, if applicable. These structures are defined in the form of skeleton structures (as their contents are not available).

The format translation performed in this stage is agnostic on interaction methods and target devices. As such, book contents and navigation structures are mapped to raw output formats (which by themselves define a book’s presentation structures).

Each output format has specific features (presented in section 2.1.4): HTML+TIME presents the biggest feature-set (e.g., powerful scripting, multiple input devices support, flexible structures, and highly-customizable presentation features), followed by SMIL (no scripting, nor input devices support), and DTBook (limited layout features support).
5.1.4 Interaction stage

On the interaction stage, input devices are linked to the book structures (both main content, and navigation structures). The supported devices are: mouse, keyboard, speech recognition, and touch (i.e., through touchscreen technologies). Two interaction types have been defined: direct content navigation, and content navigation through auxiliar structures. The first enables the user to jump towards specific points in a book content, whereas the second is based on predefined navigation patterns (e.g., table of contents).

The previous processing stage defined which output format was to be used thereafter. As such, each input device integration is dependable on the chosen output format. This limitation was overcome by creating specific processing components for each input device, per output format. Even when an output format does not specify how to interact with a DTB, some device-dependent features can be inserted into the DTB (e.g., if a hardware player input device is chosen – featuring only play/stop, rewind, and fast forward buttons – deep navigation structures should be discarded).

5.1.5 Presentation stage

DiTaBBu’s last processing stage defines how a book is presented to the user. The approach taken is based on the presentation profiles concept. These profiles are defined as a set of presentation components, in an output format-agnostic language (see listing 5.4 for an example). The profile presentation components are applied to each book structure (both main content and navigation structures), as seen on figure 5.3.

![Figure 5.3: DiTaBBu presentation profiles](image)

```
<profile xmlns="urn:ipsom:presentation:profiles">
  <section name="layout" effect="contrast" />
  <section name="sidenotes" effect="classAnim_contrast" />
  <section name="toc" effect="classAnim_contrast" />
  <section name="bodymatter" effect="sideGuide_contrast" />
</profile>
```

Listing 5.4: DiTaBBu presentation stage profile
A central registry was created to enable access to presentation profiles by a presentation stage definition (see figure 5.3). This registry is defined in an RDF syntax, where `rdf:li` elements map between profile identifiers (`prof:id` attributes) and their locations (`rdf:resource` attributes), as presented in listing 5.5.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:prof="urn:ipsom:presentation:profiles">
  <rdf:Description>
    <rdf:Bag>
      <rdf:li prof:id="profiles:contrast" rdf:resource="profiles/contrast.xml"/>
      <rdf:li prof:id="profiles:simple" rdf:resource="profiles/simple.xml"/>
      <rdf:li prof:id="profiles:sideguide" rdf:resource="profiles/sideguide.xml"/>
    </rdf:Bag>
  </rdf:Description>
</rdf:RDF>
```

Listing 5.5: DiTaBBu presentation stage profiles registry

Presentation components implement a rich set of presentation features, based on target user requirements, opening the way for automated creation of DTBs with coherent UIs.

Each presentation component feature specifies a pattern for one of five settings: dimensioning, colouring, synchronization guidance, sound volume, and sound items. Different dimensioning components allow the delivery of UIs on multiple devices (e.g., PDA, PC), and also the positioning of content and navigation structures (e.g., table of contents on the screen’s left side). Colouring components further enhance user experience on reading a DTB, through the use of different colour schemes (e.g., colour schemes matching the printed versions of book collections), as well as delivering high-contrast colour schemes (for partially sighted or colour-blind persons). Regarding synchronization guidance, two different methods for its presentation can be used: highlighting, and side-margin marker (see figure 5.4 for an example). At last, sound volume and sound items presentation components define which sounds will be triggered when reaching specific structure points (both on book content and navigation structures), as well as define how these are mixed between the book audio content and the other auxiliar sounds.

```
highlighting
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

side-margin marker
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```

Figure 5.4: Synchronization visual guides

After presentation profiles are applied, a processing step is performed over the book structures, regarding selecting output devices. As each device has its capabilities (e.g.,
screen size, colour depth, audio features, etc.), the book presentation might need some fine-tuning, for better final content delivery. Processing components have been defined for this task.

A last processing task is performed over all DTB generated structures and contents, readying the DTB for distribution (e.g., adding DTD headers, packing files, changing file extensions, etc.)

Similarly to the interaction stage, all presentation stage components are specialized on each output format. As such, presentation profiles and output devices processing components were defined for HTML+TIME, SMIL, and DTBook.

5.2 Implementation

The DiTaBBu architecture described previously has been paired with the implementation of processing components in XSLT stylesheets, on all four processing stages (structure repurpose, output format, interaction, and presentation). XSLT was chosen as the processing language, as it is the best transformation language for XML-based languages (such as DiTaBBu’s normalized content structures), because of its declarative nature. A high-level view of implemented components is shown on figure 5.5.

Figure 5.5: Components implementation
On the structure repurpose stage, a content normalization component was developed to convert DTBook-based navigation control files to DiTaBBu navigation structure (\texttt{nxc2navcomp}). A content extraction component was developed to parse a DiTaBBu content structure and infer a table of contents structure from it (\texttt{toc-content}). Regarding granularity control, components were created to process normalized content (\texttt{gran-content}, \texttt{gran-audio}, \texttt{gran-navigation}, and \texttt{gran-arcs}).

The output format stage is responsible for the normalized content transformation to a specific output format platform. Several processing components were defined, divided in three categories: main content, navigation, and user-customized. On main content category, the \texttt{core} component creates the general structures, whereas \texttt{content}, \texttt{audio}, and \texttt{arcs} processing components, process all content structures. Regarding navigation structures, these are embedded through the \texttt{navigation} component. At last, user-customized structures mechanisms are applied by the \texttt{bookmarks}, \texttt{highlights}, and \texttt{annotations} processing components. As explained earlier, these components only introduce their own mechanisms if their features are available on the correspondent output format platform.

Input device mechanisms are introduced in the interaction stage. Each input device has defined two processing components (\texttt{jump}, and \texttt{navigation}), applied on the stage’s current DTB structure (based on the output format platform chosen in the previous stage). As some platforms do not specify how interaction is performed, these processing components do not modify DTB structures.

At last, presentation stage processing is performed. Presentation profiles are applied with the \texttt{profile} component. Output devices presentation structures fine-tuning is performed by \texttt{screen}, \texttt{audio}, \texttt{force-feedback}, and \texttt{handheld} processing components. DTB structures are then wrapt (with the \texttt{last} processing component). Like output format, and interaction stages, all presentation stage’s components are specialized towards the target output format platform chosen.

5.3 Summary

This chapter presented the DiTaBBu framework. This framework was created for DTB production tasks, in a modular way. DiTaBBu is divided into four processing stages (each one focusing on different production concerns): content structures manipulation, target output transformation, interaction specification, and presentation issues. On structure manipulation, the main tasks relate to content normalization, extraction, and creation. Target output transformation builds the DTB in specific playback languages (i.e., HTML+TIME, SMIL, and DTBook). Interaction specification introduces mechanisms for user input devices (such as mouse, voice, etc.). At last, the presentation stage is related on the DTB final looks (both visual and aural), creating DTB coherent UIs. DiTaBBu has been developed as an application of APP.
Chapter 6

HLDTB

As DiTaBBu configurations become more complex (reflecting the fulfillment of more requirements), there is a need to further simplify the configuration steps for DTB production tasks, leveraging the technical skills required to perform them. To achieve this goal, HLDTB has been created. HLDTB is a high-level DTB configuration specification, fully tied to DiTaBBu configurations, but with a user-friendly configuration language. A GUI for HLDTB was also created, reflecting all of its configuration possibilities.

The following sections present in detail the HLDTB architecture (section 6.1), its implementation (section 6.2), and the GUI (section 6.3).

6.1 Architecture

HLDTB architecture defines a high-level specification for DTB production configurations. A high-level configuration conformant with the specification is transformed into an equivalent DiTaBBu configuration, through the HLDTB engine (see figure 6.1). Each high-level configuration item is coupled with a different set of DiTaBBu processing components, defining a DTB UI pattern, in the lines of related work (see chapter 3).

![HLDTB Architecture Diagram]

Figure 6.1: HLDTB architecture
However, the simplicity revealed by the HLDTB configuration language (as required to leverage the DTB production configuration) is not able to cover all configuration possibilities of DiTaBBu. Nevertheless, HLDTB covers all main aspects of DiTaBBu needed to create rich DTBs. An user-friendly language was defined to specify HLDTB configurations, by having recognizable element names and avoiding the usage of XML namespaces, DTD technologies, and even XML preambles (i.e., the `<?xml version="1.0"?>` processing instruction), lowering the learning curve of DTB production configuration specifications. HLDTB specification language defines the `application` element as the enclosing element for all configuration item types (see listing 6.1 for an example snippet).

```
<application>
  <device-list type="input">
    <device type="keyboard"/>
    <device type="voice"/>
  </device-list>
  <navigation unit="word">
    ...
  </navigation>
</application>
```

Listing 6.1: HLDTB configuration snippet

HLDTB defines five different types of configuration items, each one focused on a different DTB production concern: input and output devices, navigation, synchronization, presentation, extra features, and output format. Next, all details of HLDTB configuration types are presented.

### 6.1.1 Devices

Devices selection (both input and output) is a priority configuration task, as it defines the way how an user interacts with a DTB. As such, HLDTB defines two sections regarding this issue. This feature is implemented with `device-list` elements (see listing 6.2 and 6.3), having distinct `type` attribute values. Each `device-list` contains a set of `device` elements, where each one defines which device is to be used.

```
<device-list type="input">
  <device type="keyboard"/>
  <device type="voice"/>
</device-list>
```

Listing 6.2: HLDTB input devices configuration

```
<device-list type="output">
  <device type="audio"/>
  <device type="handheld"/>
</device-list>
```

Listing 6.3: HLDTB output devices configuration
HLDTB input device selection features all devices from the DiTaBBu interaction stage: mouse, keyboard, voice, and touch (touchscreen). All input devices can be used at the same time, creating a multimodal input interface.

Regarding output device selection, HLDTB features all devices from DiTaBBu presentation stage: screen, audio, force-feedback, and handheld. Like on the input device selection, all output devices can be used at the same time, creating a multimodal output interface.

6.1.2 Navigation

The navigation element of HLDTB configuration specification defines which navigation structures should be available on a DTB (see listing 6.4). Its unit attribute defines the book granularity unit (as specified in the structure repurpose stage of DiTaBBu). A list of navigation structures is defined with nav-item elements (specified in their type attribute). Each item also features a navigable attribute, specifying whether the structures are navigable, or just presented to the user. If different granularity units are required on each item, its unit attribute overrides the navigation element unit attribute.

```xml
<navigation unit="word">
  <nav-item type="toc" navigable="yes" />
  <nav-item type="index" navigable="yes" unit="paragraph" />
  <nav-item type="side−notes" navigable="yes" unit="sentence" />
</navigation>
```

Listing 6.4: HLDTB navigation configuration

6.1.3 Synchronization

Synchronization presentation is also specified in HLDTB, with the synchronization element and its presentation sub-element (see listing 6.5). The presentation mode (highlight vs. side guide) is specified with the mode attribute. A temporal delay can be introduced with the value attribute (in specific time units – e.g., seconds, milliseconds). At last, the type attribute specifies which synchronization unit is to be presented (similar to granularity control).

```xml
<synchronization>
  <presentation type="paragraph" value="2s" mode="highlight" />
</synchronization>
```

Listing 6.5: HLDTB synchronization presentation configuration

6.1.4 Extra features

HLDTB’s “extra features” configuration relates to user-customized navigation structures (as defined in DiTaBBu’s output format stage). As such, the extra element lists
feature elements defining which structures will be available to the final user (see listing 6.6).

```xml
<extra>
  <feature type="highlights" />
  <feature type="annotations" />
</extra>
```

Listing 6.6: HLDTB extra features configuration

### 6.1.5 Output format

Finally, the output format is specified in HLDTB with an output-format element (see listing 6.7). This element includes a type attribute specifying which output format platform will be targeted (i.e., HTML+TIME, SMIL, or DTBook).

Also, using the layout and color-scheme sub-elements, all layouting features will be linked to DiTaBBu’s presentation stage. These elements’ names are joint, forming the final layout specification name and then are used as DiTaBBu’s presentation profile name. Layouts can be both visual or aural (audio-only).

```xml
<output-format type="html+time">
  <layout type="rich" />
  <color-scheme type="blue" />
</output-format>
```

Listing 6.7: HLDTB output format configuration

### 6.2 Implementation

As HLDTB and DiTaBBu specifications are XML-based languages, a XSLT stylesheet was developed to bridge both languages, coupled with an Ant script perform the execution of this transformation (see figure 6.2).

The mapping between each HLDTB configuration item and DiTaBBu processing components defines a reusable DTB UI pattern. In this perspective, the HLDTB specification is a set of directives pointing at which DTB UI patterns are to be used on the production of rich DTBs. As these patterns relate only to UI concerns (navigation and presentation structures), the HLDTB specification does not cover DiTaBBu’s structure repurpose stage. As such, it is assumed that only certain types of inputs are currently allowed as input to HLDTB processing mechanisms. No inference mechanisms have been implemented yet (to select proper DiTaBBu structure repurpose processing components).
6.3 Graphical User Interface

To further improve HLDTB usability, a GUI was developed to help on the DTB production tasks. The GUI approach taken was to create a wizard-style UI. A wizard is an interactive computer program acting as an interface to lead a user through a complex set of tasks using dialog steps. As such, this type of UIs suits better on controlling HLDTB configuration tasks, where each one of HLDTB’s configuration tasks is mapped into a dialog step. Using a GUI to control the production of DTBs smooths its learning curve, discarding the need to learn HLDTB configuration language, and launching HLDTB in a command shell.

The GUI is divided in four logical containers (see the mockup presented in figure 6.3): task description, task pane, navigation and execution commands, and table of contents secondary navigation.

![Figure 6.3: HLDTB graphical user interface mockup](image)

The task description container provides identification of the current selected task (with concise title and description labels). The task pane container presents the current task configuration specialized UI. The navigation and execution commands container presents a
set of command buttons for sequential task navigation (back and next), as well as HLDTB execution control buttons (generate and exit). At last, the table of contents secondary navigation container enables a full view of all tasks that can be performed in the wizard (identifying which task is currently selected), and also enables direct navigation to a specific task. After configuration tasks have been performed, the generate command button launches the HLDTB engine process, delivering the final DTB formatted as specified.

The existence of two navigation types in the wizard mockup (direct and sequential navigations) eases the navigation in a set of tasks, as the user requires less time to find and complete a task. Also, the table of contents approach taken is presented as a “road map”, allowing a better understanding of the overall content and structure of a wizard. All these results (coupled with usability methodologies and evaluation) are further described in [19].

Next, HLDTB GUI task panels are presented, as well as several GUI implementation issues.

6.3.1 Tasks
6.3.1.1 Content location

The content location task (see figure 6.4) is used to select which input content is going to be processed accordingly to the chosen HLDTB configuration. As the input content can be spawned in several files, the user only has to specify which directory contains all of them. This task is the only one which does not matches any subset of HLDTB specification.

![Figure 6.4: HLDTB content location task](image_url)
6.3.1.2 Input devices selection

This task is responsible for the input devices selection for HLDTB. A simple multiple selection list was chosen to represent visually the devices selection. Figure 6.5 presents a sample selection, and listing 6.8 its correspondent HLDTB specification snippet.

![HLDTB input devices selection task](image)

Figure 6.5: HLDTB input devices selection task

```xml
<device-list type="input">
  <device type="keyboard" />
</device-list>
```

Listing 6.8: Input device selection snippet

6.3.1.3 Output devices selection

Regarding output devices selection, the approach taken is similar to the inputs devices selection tasks. A simple multiple selection list control represents visually which output devices should be selected to be coupled in the corresponding HLDTB specification configuration. Figure 6.6 presents a sample selection, coupled with its XML representation in listing 6.9.

```xml
<device-list type="output">
  <device type="screen" />
  <device type="audio" />
</device-list>
```

Listing 6.9: Output device selection snippet
6.3.1.4 Navigation structures

The navigation structures task defines a layout for the HLDTB navigation structures subset specification. This layout is composed by a dropdown list containing all granularity units (e.g., word, sentence, paragraph), and a custom-built list, featuring all navigation structures available in HLDTB. This list is divided in three columns: available items, granularity, and navigable. When an item’s granularity is defined (through an in-place dropdown list), it overrides the default granularity unit. Figure 6.7 presents an example of use, coupled with its respective HLDTB specification snippet, in listing 6.10.
6.3.1.5 Synchronization presentation

This GUI task defines the HLDTB subset regarding synchronization presentation issues. A dropdown list is used to select the synchronization granularity unit. Presentation time delay is selected with a textbox coupled with the time unit selection (e.g., seconds, milliseconds). At last, the synchronization presentation mode (visual guidance) is selected in a dropdown list. An example use is presented in figure 6.8, and listing 6.11 its HLDTB counterpart snippet.

![Synchronization GUI](image.png)

Figure 6.8: HLDTB synchronization presentation task

```
Listing 6.11: Synchronization presentation snippet

<synchronization>
  <presentation type="word" value="0s" mode="highlight" />
</synchronization>

Listing 6.11: Synchronization presentation snippet
```
6.3.1.6 Extra features selection

The selection of extra features (i.e., bookmarks, highlights, and annotations) in the GUI has been defined with a simple multiple selection list control. Figure 6.9 presents a sample usage, and listing 6.12 the corresponding HLDTB specification snippet.

![Figure 6.9: HLDTB extra features selection task](image)

```xml
<extra>
  <feature type="bookmarks" />
</extra>
```

Listing 6.12: Extra features selection snippet

6.3.1.7 Output format selection

Finally, the last GUI task defines which output format will be used in the DTB production. This task’s interface is defined simply by three dropdown lists, defining the target output format language, the DTB layout, and the colour scheme to be used. As a layout can be audio-only (i.e., aural), the colour scheme configuration item choice is optional. Like in other tasks, an example is presented (see figure 6.10), coupled with its HLDTB counterpart snippet (see listing 6.13).

```xml
<output-format type="html+time">
  <layout type="minimal" />
  <color-scheme type="high-contrast" />
</output-format>
```

Listing 6.13: Output format selection snippet
6.3.2 Implementation

The HLDTB GUI was developed with Java technologies. This choice was based on the ability to embed the Ant engine in the GUI base code (as Ant itself is also developed in Java), allowing a tighter control over the HLDTB engine execution.

The Standard Widget Toolkit (SWT) [65] was the chosen platform for GUI development, as it is a portable lightweight GUI toolkit (in comparison with Java own’s AWT and Swing toolkits). The portability and power of all chosen development platforms (Java, Ant, and SWT) eases the creation of a robust multiplatform GUI for HLDTB.

To ease the implementation of HLDTB GUI, its high-level design was defined with a UML package diagram, as presented on figure 6.11, representing the GUI main concerns.

Figure 6.10: HLDTB output format selection task

Figure 6.11: GUI implementation UML packages

Being the outermost package, `hcim.ditabbu.hldtb` defines the base naming for all
classes in the implementation. It is responsible for program launching, as well as HLDTB execution. Ant-specific concerns are grouped in the hcim.ditabbu.hldtb.ant package. The hcim.ditabbu.hldtb.ui package is designated to all UI tasks, whereas its child package, hcim.ditabbu.hldtb.ui.panes, groups all task panes classes.

The next step in designing the HLDTB GUI was to create a UML class diagram, to clarify every interrelationship between all GUI concerns, as presented in figure 6.12.

The root package, hcim.ditabbu.hldtb, encloses the Main, Exec, ExecException, and Util classes. The Main class is responsible solely for the GUI startup. Exec and ExecException classes manage the HLDTB Ant engine setup and execution. The Util class contains utility functions to help on the configuration serialization to HLDTB XML specification language.

To capture all Ant messages (especially the ones directly related to the HLDTB implementation), the HLDTBListener class was created, as a part of hcim.ditabbu.hldtb.ant package. The modularity design of Ant enables an easy implementation of such features.

On the UI level, two classes were developed to manage all basic UI tasks (featured on the hcim.ditabbu.hldtb.ui package): BaseWindow, and BasePane. The first class is responsible for all generic GUI features described earlier (see the mockup in figure 6.3), acting also as a container for all task panes. Regarding the second class, BasePane, it features the base code for all specific HLDTB task panes. The main methods implemented by this class relate to the serialization of their respective task’s configuration, with the helper functions provided by the Util class.
At last, each task pane is implemented in the \texttt{hcim.ditabbu.hldtb.ui.panes} package. Each package features a specific UI (as presented earlier on this chapter), and is responsible for the specific serialization of its current configuration state. The only exception in the serialization process relates to the \texttt{ContentPane} class, as it is not related to the HLDTB specification, but to the HLDTB execution engine. Its serialization method returns the selected content location (used later in the \texttt{Exec} class).

### 6.4 Summary

To further enable the DTB production tasks, a high level pattern language has been developed for the specification of DTB production configuration. This language, HLDTB, defines all interaction concerns in a simple way (input and output devices, content navigation, layout features, etc.). To bridge the gap between HLDTB and DiTaBBu, a conversion task has been defined, mapping each HLDTB concern into a set of DiTaBBu processing components. A GUI was created to further ease the creation of DTBs, presenting all HLDTB features. This tool leverages the batch production of DTBs.
Chapter 7

Evaluation

This chapter describes an evaluation process to test the flexibility of APP, DiTaBBu, and HLDTB. By defining several HLDTB configurations and performing evaluations over them, the DiTaBBu framework is automatically evaluated, as HLDTB specification maps directly into a DiTaBBu specification. APP is also automatically evaluated, as DiTaBBu is a concrete application of APP. By using this approach, it is leveraged all flexibility of each tool.

Three case studies were defined, to cover several configuration possibilities: common user with PC, blind user with low computational power device, and partially-sighted user with PDA. Each configuration’s details are presented in table 7.1, in tune with target user requirements (see section 2.1.1) and content playback requirements (see section 2.1.4):

<table>
<thead>
<tr>
<th>input devices</th>
<th>config #1</th>
<th>config #2</th>
<th>config #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouse</td>
<td></td>
<td>voice</td>
<td>touch</td>
</tr>
<tr>
<td>output devices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>screen</td>
<td></td>
<td>audio</td>
<td>handheld</td>
</tr>
<tr>
<td>audio</td>
<td></td>
<td></td>
<td>audio</td>
</tr>
<tr>
<td>output format</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>HTML+TIME</td>
<td>SMIL</td>
<td>SMIL</td>
</tr>
<tr>
<td>layout</td>
<td>rich</td>
<td>aural</td>
<td>minimal</td>
</tr>
<tr>
<td></td>
<td>blue</td>
<td></td>
<td>high-contrast</td>
</tr>
<tr>
<td>navigation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>word</td>
<td>paragraph</td>
<td>paragraph</td>
</tr>
<tr>
<td>items</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(unit, navigable)</td>
<td>toc</td>
<td>back-fwd</td>
<td>toc</td>
</tr>
<tr>
<td></td>
<td>(default, yes)</td>
<td>(default, yes)</td>
<td>(default, no)</td>
</tr>
<tr>
<td></td>
<td>side-notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(default, no)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>synchronization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>paragraph</td>
<td>paragraph</td>
<td>paragraph</td>
</tr>
<tr>
<td>delay</td>
<td>2 seconds</td>
<td>2 seconds</td>
<td>0 seconds</td>
</tr>
<tr>
<td>mode</td>
<td>side-guide</td>
<td>beep</td>
<td>highlight</td>
</tr>
<tr>
<td>extra features</td>
<td>bookmarks</td>
<td>highlights</td>
<td>bookmarks</td>
</tr>
<tr>
<td></td>
<td>annotations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Evaluation configuration

Each configuration presented in table 7.1 was defined in the HLDTB pattern language,
as presented in listings 7.1, 7.2, and 7.3.

```
<application>
  <device-list type="input">
    <device type="mouse" />
  </device-list>

  <device-list type="output">
    <device type="screen" />
    <device type="audio" />
  </device-list>

  <output-format type="html+time">
    <layout type="rich" />
    <color-scheme type="blue" />
  </output-format>

  <navigation unit="word">
    <nav-item type="toc" />
    <nav-item type="side−notes" navigable="no" />
  </navigation>

  <synchronization>
    <presentation type="paragraph" value="2s" mode="highlight" />
  </synchronization>

  <extra>
    <feature type="bookmarks" />
    <feature type="highlights" />
    <feature type="annotations" />
  </extra>
</application>
```

Listing 7.1: HLDTB specification for a common user with PC

```
<application>
  <device-list type="input">
    <device type="voice" />
  </device-list>

  <device-list type="output">
    <device type="audio" />
  </device-list>

  <output-format type="smil">
    <layout type="aural" />
  </output-format>

  <navigation unit="paragraph">
    <nav-item type="back−fwd" />
  </navigation>

  <synchronization>
    <presentation type="paragraph" value="2s" mode="beep" />
  </synchronization>
</application>
```

Listing 7.2: HLDTB specification for a blind user with low computation power device
Listing 7.3: HLDTB specification for a partially-sighted user with PDA

All execution steps (either defined in the HLDTB language or with its correspondence in the HLDTB GUI) created coherent DTBs, all using the same initial content (an excerpt from “O Senhor Ventura” by Miguel Torga). The execution of HLDTB with each configuration proves the language’s flexibility, as several tasks are used: different input devices, different output devices, flexible output format specification, different navigation purposes, multiple synchronization presentation, and a mix of extra features to be available on each configuration. As HLDTB maps directly to DiTaBBu and as final DTBs have been produced, we prove the flexibility of DiTaBBu through the creation of a DTB in each configuration. APP’s own flexibility is also proved through the execution of DiTaBBu’s generated configuration.

However, as each output language has its own limitations (especially regarding interaction and presentation issues), the created DTBs do not represent an exact mapping of each configuration definition. For example, configurations #2 and #3 are targeted to SMIL output format. As described earlier (in section 2.1.4), this format does not specify interaction issues, such as voice and touch interaction, (as this task is left to SMIL player implementations), neither specifies playback-time user-defined content creation (such as bookmarks). A back/forward navigation is also impossible to implement in the SMIL language, as sequential navigation is a task left also to SMIL players.

Figures 7.1 and 7.2 present, respectively, the DTB playback outputs for configurations #1 and #3. The first figure presents a DTB composed by a table of contents (left column),
the book’s main content with a side-guide (center column), and a set of margin notes (right column). The second picture presents a portable DTB composed by the book’s main content (presenting each paragraph at a time, on top row), and its respective table of contents (presenting each item at a time, on bottom row). Regarding configuration #2, no output is presented, as the specified input and output devices related to audio-only features.

Figure 7.1: DTB playback on Internet Explorer (as specified by configuration #1)
Figure 7.2: DTB playback on SMIL player (as specified by configuration #3)
Chapter 8

Conclusions and Future Work

This report presented the work developed in the Post-Graduation Course on Informatics Engineering (CEPEI), focused on the creation of DiTaBBu, a flexible framework for automated creation of DTBs and rich DTBs. DTBs bring a new era for book reading on blind communities, as rich navigation features enable a better user experience. However, the adoption of DTBs has been sparse, as the DTB production is manually-crafted. As such, there was a need for an automated process of books, enabling the batch creation of DTBs. Also, Web browsers are available everywhere, thus producing DTBs for this platform further enabled the acceptance rates for DTBs.

When producing DTBs several issues were taken into account: user profiles, different content repurposing tasks, multiple output format formats, content navigation and synchronization, amongst many other issues. To ease the production tasks, a flexible and generic framework for composition of processing tasks has been defined, as an addendum to the purposed work, as current processing frameworks did not fulfill all the requirements for digital publishing architectures. This framework, APP citeapp, enabled an easier integration of DiTaBBu’s requirements, either for feature integration tests (related to prototype features or usability tests), or to final DTB production.

On top of DiTaBBu, a high level DTB processing configuration language was defined, along the lines of human-computer interaction patterns, named HLDTB. This language maps directly into DiTaBBu processing components, but with a simplified language, which can be used by non-technical users. A GUI was also developed for HLDTB and used to evaluate the flexibility of HLDTB, DiTaBBu, and APP.

Although APP is defined by a clean approach to complex content processing, there is still room for a lot of improvements. Currently only two types of processing components have been used: XSLT and passthrough. There is a need to support any type of components in APP’s registry, through a generic extension mechanism, either with a special purpose API, or by using other frameworks’ processing component engines (such as Cocoon’s). Also, the current prototype implementation has performance issues, thus the development of an optimized version of APP is being designed (with caching mechanisms, performance optimizers, etc.) Lastly, APP’s integration into the Eclipse platform [37] is a must have,
through a GUI that enables a better developing and configuration platform, on all APP’s concerns (e.g., components, pipelines, stages, projects, registry, etc.)

DiTaBBu’s current implementation features a set of skeleton processing components only suitable for DTB prototyping time. As these prototypes’ features become richer, their integration into DiTaBBu will be performed. With APP’s future generic extension mechanism, richer processing components will become available (such as integrated voice alignment, DTB packaging, richer feature extraction, automated structure repurpose, etc.) Dynamic production of DTBs is another feature currently being designed, allowing for more flexibility in DTBs, especially related to adaptability concerns, distributed content sources (i.e., remote book databases), etc. It is planned the support of several input and output devices, as target platforms feature a better support for DTB playback.

At last, HLDTB will support more DiTaBBu components, enabling a richer configuration of DTB production tasks. With the availability of DiTaBBu automated structure repurposing components, HLDTB will support more type of input sources. On the GUI level, HLDTB will support batch processing tasks, as well as HLDTB presets loading/saving (enabling configuration swapping between HLDTB GUI users).
Glossary

API – Application Programming Interface
    Set of definitions of the ways one piece of computer software communicates with another 26, 75

APP – Architecture for Pipelined Processing
    XML processing architecture 3, 3, 16, 35–43, 45, 53, 69, 71, 75, 76

CGI – Common Gateway Interface
    World Wide Web technology that enables a client web browser to request data from a program executed on the Web server 31

CSS – Cascading Style Sheets
    Stylesheet language used to describe the presentation of a structured document written in markup languages like HTML and XHTML 22, 22

DAISY – Digital Audio-based Information SYstem
    Consortium developing standards for DTBs 17, 17, 18

DiTaBBu – Digital Talking Books Builder
    Framework for the construction of rich DTBs 3, 3, 15–17, 45, 46, 50, 52, 53, 55–58, 67, 69, 71, 75, 76

DPML – Declarative Processing Markup Language
    XML assembly language which can script primitive XML operations into processing pipelines 29, 31, 32

DTB – Digital Talking Book
    Audio-enabled digital book 3, 3, 15–27, 33, 45, 46, 49–53, 55–60, 64, 67, 71, 72, 75, 76

DTBook – NISO DTBook XML element set
    XML element set representing content and structure of books 16, 20–22, 46, 48, 49, 52, 53, 58
DTD – Document Type Definition
XML document structure definition language 20, 52, 56

GUI – Graphical User Interface
User interface based on graphics (icons, pictures, and menus) 16, 16, 55, 59, 60, 63–67, 71, 75, 76

HCI – Human-Computer Interaction
Computer science area focused on the design and implementation of systems that people interact with 32, 32, 33

HLDTB – High-Level Digital Talking Book
Simple definition of a DiTaBBu configuration 3, 16, 55–67, 69, 71, 75, 76

HTML – HyperText Markup Language
A markup language used to structure text and multimedia documents and to set up hypertext links between documents, used extensively on the World Wide Web 21, 25, 29, 33, 46, 48

HTML+TIME – HTML Timed Interactive Multimedia Extensions
Proposal for adding timing and synchronization support to HTML 16, 20, 21, 49, 52, 53, 58

NISO – National Information Standards Organization
United States non-profit standards organization that develops, maintains and publishes technical standards related to bibliographic and library applications 17, 18

PC – Personal Computer
Any laptop or desktop computer such as a Windows machine or a Macintosh 46, 51, 69

PDA – Personal Digital Assistant
A handheld computer that serves as an organizer for personal information 20, 46, 51, 69

PDF – Portable Document Format
Adobe's Acrobat document exchange technology, de facto standard for document exchange as well as for publishing documents on the Web 29, 48

RDBMS – Relational DataBase Management System
Database management system based on the relational model 31
**RDF – Resource Description Framework**

A recommendation from the W3C for creating meta-data structures that define data on the Web 36, 40, 47, 48, 51

**SMIL – Synchronized Multimedia Integration Language**

XML language for rich media presentations 16, 17, 20–23, 49, 52, 53, 58, 71

**SVG – Scalable Vector Graphics**

A vector graphics format from the W3C for the Web that is expressed in XML 21

**SWT – Standard Widget Toolkit**

A widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented 65, 65

**SXPipe – Simple XML Pipelines**

Language for describing simple XML pipelines 29, 30

**UI – User Interface**

Aggregate of means by which people (the users) interact with a particular machine, device, computer program or other complex tool (the system) 16, 19, 20, 22, 27, 32, 33, 45, 46, 51, 53, 55, 58, 59, 66, 67

**UML – Unified Modeling Language**

Object-oriented analysis and design language from the Object Management Group 33, 65, 66

**URI – Uniform Resource Identifier**

Addressing technology for identifying resources on the Internet or a private intranet 31, 31, 32, 36, 40, 47

**W3C – World Wide Web Consortium**

International industry consortium founded in 1994 by Tim Berners-Lee to develop standards for the Web 17

**XHTML – Extensible HyperText Markup Language**

Markup language that has the same expressive possibilities as HTML, but with a stricter syntax derived as an application of XML 17, 20–22, 48

**XHTML+SMIL – XHTML with SMIL extensions**

Set of XHTML abstract modules that support a subset of the SMIL 2.0 specification 22, 22
XML – Extensible Markup Language
   General-purpose markup language for creating special-purpose markup languages
   16, 21, 24–27, 29–31, 35, 36, 38, 42, 46, 52, 56, 58, 61, 66

XPL – XML Pipeline Language
   XML vocabulary for describing the processing relationships between XML resources
   29, 30, 31

XSL – XML Stylesheet Language
   Family of languages which allows one to describe how files encoded in the XML
   standard are to be formatted or transformed 31

XSLT – XML Stylesheet Language Transformations
   XML-based language used for transforming XML documents 40, 42, 52, 58
References


