TOPPERS 活用アイデア・アプリケーション開発コンテスト

部門　：　活用アイデア部門
作品のタイトル　：　TOPPERS 統合仕様書のリッチテキスト化
作成者　：　名古屋大学　（代表：高田光隆）
共同作業者　：
対象者　：　TOPPERS ソフトウェア開発者（アプリケーション開発者）及び TOPPERS カーネル開発者
使用する開発成果物　：　TOPPERS 統合仕様書,TOPPERS 新世代カーネルガイドライン

目的・狙い

TOPPERS のカーネル開発、ソフトウェア開発において TOPPERS 統合仕様書を参照することが必須となっているが、仕様書がプレインテキストで記述されているため仕様書を読む際に目次ジャンプや関連情報へのリンクなど機能が不足しており、特に初学者が利用する際に開発効率が上がらない。そのため以前よりリッチテキスト化が試みられコントリビュートソフトウェアとして登録もされているが、仕様書を記述するためのベースフォーマットではないため、二重メンテナンスが発生し、仕様書メンテナンスや移行のための作業コストが高い。実際に最新の統合仕様書に追従できていない。
そこで現状のプレインテキストによる記述をなるべく変更せずに、リンク機能や文字修飾が可能なリッチテキスト化にするため、軽量マークアップ言語の一種である AsciiDoc という記述形式を使うことで、テキストベースでの仕様書からコストをあまりかけずに、リッチテキスト化することが可能とした。また作業ツールに関してはオープンソースで入手可能であり環境構築もホスト環境に依存せずに作ることができることを確認した。
また現状の統合仕様書は PDF フォーマットだけしか用意されていないが、AsciiDoc では html や Eclipse ヘルプなど複数のファイルフォーマットへの変換が可能であるため、開発時に有効な環境で利用できることも可能になる。

アイデア/アプリケーションの概要

- 統合仕様書からテキスト部分を抜き出す
- テキストを AsciiDoc 形式に書き直す（書き足す）
- AsciiDoc のファイルをリッチテキスト化（PDF、html ファイル）するツールとして、AsciiDoctor を使用する（http://asciidoctor.org/）
- ツールを実行する環境として github と Travis-CI を使う。クラウドサービスを利用することで複数メンバーが仕様書を作成できる環境を構築する
- 仕様書の AsciiDoc ファイルを github にコミットすると、Travis-CI で AsciiDoctor を自動実行し、結果を github Pages へ自動的に出力する
- TOPPERS 統合仕様書のページ例
  - https://mitsut.github.io/toppers_kernel_spec/
- TOPEPRS 新世代カーネルへのマイグレーションガイド例
  - https://mitsut.github.io/toppers_migration/

1 TOPPERS 統合仕様書の問題点

TOPPERS 統合仕様書は TOPEPRS 新世代カーネルと呼ばれる TOPPERS/ASP 以降のカーネルに適用される仕様書であり、現在 ASP カーネル、FMP カーネル、HRP2 カーネルや第 3 世代カーネルの ASP3 カーネル、HRP3 カーネルの実装に適用されている。

様々なカーネルで仕様書が利用されているが、ITRON 仕様からは拡張している箇所が多数あるため、ITRON 仕様書をベースにすることなく新規に作成している。仕様書のメンテナンスのし易さやツールの関係上、テキストファイルでのメンテナンス管理を行っている。そのため利用する側からすると以下の点での機能不足や不便な点がある。

- 目次からのジャンプ機能
- API リファレンスからの該当する API 情報へのジャンプ機能
- 文字修飾による表示の見易さ
- PDF 以外のファイルフォーマット（html など）

これらを解決するために、仕様書のリッチテキスト化の検討を行った。

2 リッチテキスト化の検討

2.1 MS-WORD を使った取り組み

仕様書のリッチテキスト化については何度か取り組みや検討が行われている。MS−WORD での仕様書のリッチテキスト化については、杉本氏が統合仕様書の Version 1.4.0 及び 1.5.0 を作成し、TOPPERS プロジェクトの Contributed Software としてコミットしている。
http://dev.toppers.jp/svn/contrib/ngki_spec_rtf/branches/

MS-WORD で作成しているために目次ジャンプ機能や文字修飾による表現の自由度は実現できている。しかし、テキストファイル形式ではないため、プレインテキストからの移行コストは高く、バージョン管理ツールを使った場合にデータの差分情報がわからない、WORD もしくは PDF ファイル以外のファイル形式への変換が難しいという問題は解決できていない。

2.2 DITA による検討

テキストファイル形式によるリッチテキスト表現としては、昔から TeX や DocBook による形式があり、T-Kernel の仕様書や他の仕様書などでも DocBook 形式で書かれている場合がある。
DocBook では XML（タグ）によるマークアップで記述するのだが、DocBook ではタグの自由度が高いため、テクニカルライティング業界では技術文書を DITA(Darwin Information Typing Architecture という形式で書く傾向にある。
DITA ではトピックという単位で文章のまとまりを派生させて記述する手法で、DocBook に比べると利用タグを限定でき、定型文書などを書くのに向いている。
DITA では現状の問題点で上がっている機能不足なども解決できるのだが、(1) テキストファイルから DITA 形式へ変更するためのタグの追加/修正コストが大きい。(2) DITA の書式は XML ファイルの一部なので、XML エディタなどの何らかのツールを用いないと仕様書のコンテキストを理解するのが難しい。(3) DITA 形式から PDF への出力を行うには DITA−OT というオープンソース実装のツールを使用することができるが出力表現が配布できるレベルではないため、有償の DITA ツールを使う必要があるため、出力の実行環境が限られてしまう。
などの問題点がある。

2.3 軽量マークアップ言語による検討

DocBook や DITA のようなマークアップ言語ではなく、wiki や Markdown などの軽量マークアップ言語での検討を行うことにした。軽量マークアップ言語はブログ作成ツールのアシスタントツールや wiki システムの記述用のマークアップとして Markdown を始め使われている。ただ Markdown は軽量マークアップ言語としては普及されているが、記法拡張のための言語仕様が定められていないため、

方言が乱立されており、システムごとに互換性がないものが多い。
その点 AsciiDoc に関しては Markdown に比べると普及度合は低いが、同等の可読性と表現力があり、言語拡張のための言語仕様が決められている。また実績においても、O'reilly でも採用されるなど技術文書を AsciiDoc で記述される例がある。
また今回の検討で Asciidoctor という ruby 実装の AsciiDoc のツールがあり、オープンソースで公開されている。TOPPERS 第 3 世代カーネルではホスト環境として ruby が必須になっているので、ツールのインストールに関する敷居も低いと思われる。そこでプレインテキストを asciidoc のマークアップへ変更し、Asciidoctor を使用して出力する方法を検討した。

3　リッチテキスト化の作業内容

3.1 テキスト情報の抜き出し

TOPPERS 統合仕様書のドキュメントは PDF ファイル形式になっているが、文章の文字列に関してはコピーが可能であるため、ファイルから文字列を全コピーし、プレインテキストの形式で保存する。PDF ファイル形式ではドキュメントの行番号も情報として含まれているので、保存したプレインテキストには行番号情報も含まれている。そのため、エディタのマクロ機能や矩型選択もしくは awk などのテキスト処理を用いて行番号情報を削除する必要がある。
またプレインテキストのファイル行が大きいので作業にあたって統合仕様書の章ごとにファイルを分割した。

3.2 AsciiDoc 形式への変更

リッチコンテキスト化するにあたり、AsciiDoc では豊富な表現のマークアップがあるが、プレインテキストからの変換コストをなるべく抑えるために、以下の必要な機能が実現出来るための変更にとどめることにした。

3.2.1 目次、セクション

目次についてはセクション（章、節）をマークアップすることでツールが自動的に必要なレベルのセクションまでを目次にまとめてくれる。

図 1 . Asciidoc での目次及びセクションのマークアップ例

セクションの記述は行の先頭が「= 」で始まるマークアップになっており、「=」円記号の数が増えるとセクションのレベルが下がるようになっている。目次のセクションレベルは統合仕様書の 3 レベルと同じに合わせた。図 1 の例では「:toclevels:」で目次に出力するセクションレベルを設定することができ、レベル 4 になっている。これはレベル 1 が表題になっているからである。
目次については「:toc:」を追加することでツールが目次を作成し、ファイル形式に応じた目次フォーマットで出力する。目次フォーマットについてはも設定が可能ではあるが、本検討・実装ではツールのデフォルトを採用し、目次からのリンクなど所定の目標を達成することができている。

3.2.2 リンク

目次からのジャンプとしてのリンクだけでなく、仕様書内で章または節についての参照を記述している箇所がある。Asciidoc ではリンク元にリンク情報(id)をマークアップしておくことで、リンク先でリンク元 id を示すだけで、リンクを作成することができる。
リンク情報は 2 重の中カッコ「[[ ]]」でリンク id とリンク情報をくくる記述をするだけで可能である。参照先ではリンク id を 2 重の不等号マーク「<< >>」で括ることでリンクを作成することができる。

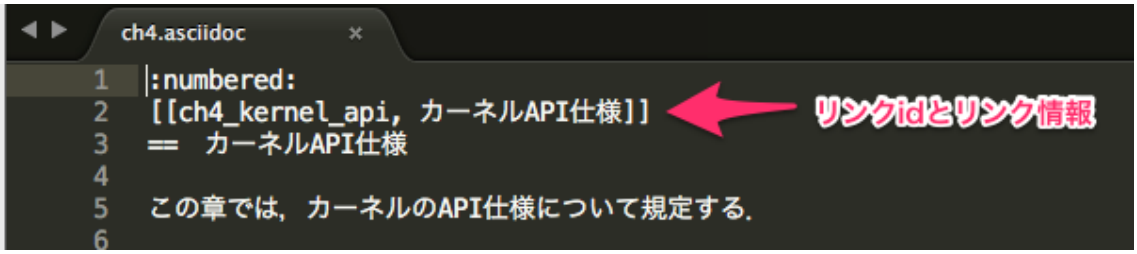

図 2. リンク元のマークアップ例

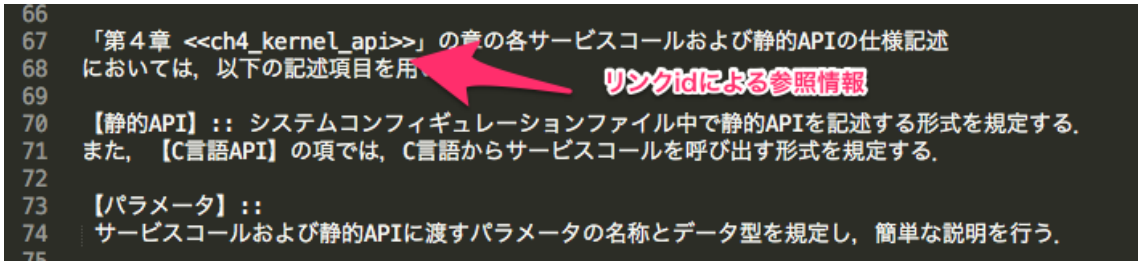

図 3. リンク先でのリンク id を使ったリンク情報のマークアップ例

3.2.3 表

Asciidoc では表の記述は、パイプライン「|」やカンマ区切り「,」を用いて表の行や列の区切りを使用したり、csv 形式の外部ファイルをインクルードしたりすることも可能である。

統合仕様書では表の目的に応じて、「処理単位の開始・終了とシステム状態」の表に関してはパイプラインを使い、「機能コード表」と言った表形式で管理をしておいた方が良いものに関しては、csv ファイルの別形式で保存し外部ファイルからの読み込みで表出力を行うこととした。

3.2.4 その他機能

Asciidoc では文章修飾も豊富に揃っているが変更コストを考えて、(1) なかぐろ「・」や数字を使ったリスト表現。(2) 中間コードや API などを表現するためのコードブロック (3) 改行に対してマークアップの追加を行った。文字修飾（色、文字種）に関しては行っていない。

3.3 実施した作業コスト

Asciidoc の書式を知らない状態でテキストの抜き出しから、マークアップを埋め込み、希望している html ファイルや pdf ファイルの出力に約 2、3 日程度で可能となった。

また作業のワークフローがわかった後に別のファイルとして「TOPPERS 新世代マイグレーションガイド」についてもトライしてみたが数時間で作業が完了した。プレインテキストからの移行のための学習コストも低く見積もることができると思われる。

4 クラウドサービスを使用したデプロイ環境

前節ではローカルマシンで作業する手順を示したが、デプロイ時に作業環境依存にならないようにするために、クラウドサービスを使用してドキュメント作成のデプロイ環境を構築した。これらの環境はすべて無償アカウントで利用できるため、環境構築のためのメンテナンスコストも抑えることができる。ここでは環境で使用したツールとそのワークフローについて説明する。

4.1 Asciidoctor

AsciiDoc から HTML5 や DocBook 出力を行う高速テキストプロセッサである。

ruby で実装されており、RubyGems からインストールが可能になっている。

MIT License で Github からコードが公開されているので、今後仕様タグのトレースなど機能拡張をしたい場合にでも改造が可能になっている。

http://asciidoctor.org/

4.2 GitHub 及び GitHub Pages

リポジトリサーバの一つで仕様書の asciidoc ファイルをコミットしている。
GitHub pages はリポジトリで公開できるウェブページとなっており、本デプロイ環境では生成した html ファイルや pdf ファイルを公開するために使用している。

4.3 Travis-CI

継続的インテグレーションツールの 1 つで GitHub と連携して、設定したリポジトリから自動でソースコードをチェックアウトして、あらかじめ指定したビルドやテスト処理を実行することができる。
また処理結果をメールや各種 API を使って、開発者へ通知ができる。
さらに処理が正常に完了した場合に、あらかじめ指定したホスティングサービスへソフトウェアをデプロイすることができるようになっている。
本環境では Asciidoctor を実行する環境とファイル生成後デプロイに使用している。

4.4 Docker

Docker はクラウドのサービスを動かすためのコンテナ型の仮想環境として最近使用されている。コンテナ型の仮想環境ではベース環境からの差分パッケージ（コンテナ）だけの用意で良いため、他の仮想環境に比べると使用リソースが少なくて良いなどのメリットがある。また仮想環境を使用することによりツールの使用バージョンを固定化することができる、Docker 環境設定をテキストベースで用意することができるなどメンテナンス性が良いため採用した。

4.5 作業フロー

作成からデプロイまでの作業フローを下図に示す。

① ローカル環境にて作成した仕様書の asciidoc 形式のファイルを GitHub のリポジトリへコミットする
② Travis-CI 側ではリポジトリがコミットされた場合に、リポジトリをチェックアウトし、チェックアウトしたファイルにある.travis.yaml を実行する
③ .travis.yaml では asciidocter をインストールし、docker を使用可能にしたのちに、asciidoctor を実行し、html と pdf ファイルを作成する
④ 作成が成功した場合は GitHub Pages

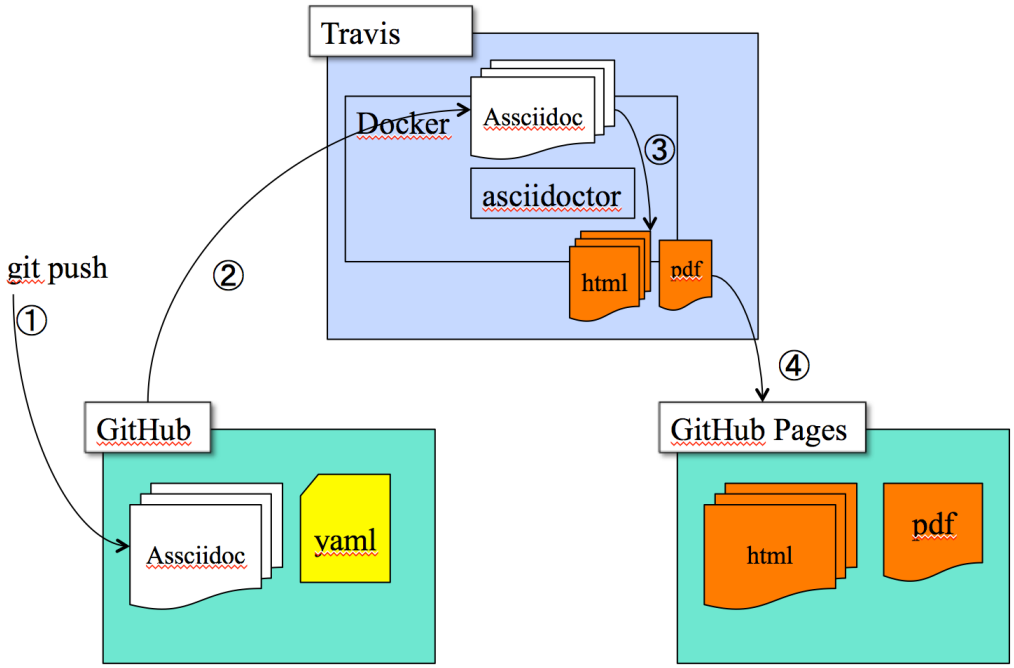

図 4. 作業ワークフロー

5　作業結果と今後の課題

プレインテキストをリッチテキスト化する手段として軽量マークアップ言語である Asciidoc を採用することで、変更コストをあまりかけずに実現することができた。

また当初の目的とした目次ジャンプや API リファレンス一覧からのリンク機能、外部参照ファイルを用いた表の作成なども実現することができた。

今回のクラウドの実現環境として以下のリポジトリやリンクにに全ての情報がある。

- GitHub のリポジトリ
- https://github.com/mitsut/toppers_kernel_spec
- https://github.com/mitsut/toppers_migration

- Travis の URL
- https://travis-ci.org/mitsut/toppers_kernel_spec
- https://travis-ci.org/mitsut/toppers_migration

- 出力先の GitHub Pages
- https://mitsut.github.io/toppers_kernel_spec/
- https://mitsut.github.io/toppers_kernel_spec/book.pdf
- https://mitsut.github.io/toppers_migration/
- https://mitsut.github.io/toppers_migration/migration.pdf

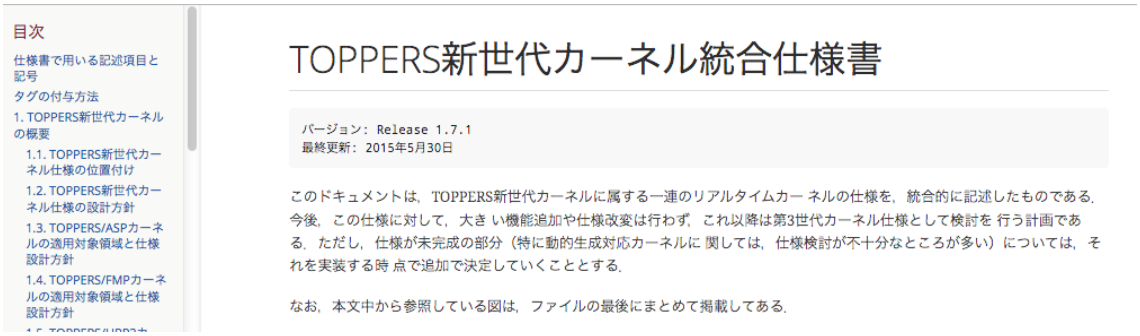

図 5. html 形式による出力例（左ペインが目次でリンク可能になっている）

今後は仕様書の図の情報が入手できなかったので、図の埋め込みや細かな表現変更を行うことで配布ドキュメントとして完成させつつ、仕様書の更新に合わせてリッチテキスト化も追従し、仕様書の管理フォーマットを移行できる筋道を示したい。
また仕様書の仕様タグについてもリンク id の拡張を行いタグのトレースやリンクの記述についての検討を行う。