Exploiting patterns and templates for technical documentation

Alessandro Caponi
Department of Computer Science
University of Bologna
Italy
alessandro.caponi@unibo.it

Angelo Di Iorio
Department of Computer Science
University of Bologna
Italy
angelo.diiorio@unibo.it

Fabio Vitali
Department of Computer Science
University of Bologna
Italy
fabio.vitali@unibo.it

Paolo Alberti
Alstom
Bologna
Italy
paolo.alberti@alstomgroup.com

Marcello Scatá
Alstom
Bologna
Italy
marcello.scata@alstomgroup.com

ABSTRACT

There are several domains in which the documents are made of reusable pieces. Template languages have been widely studied by the document engineering community to deal with common structures and textual fragments. Though, templating mechanisms are often hidden in mainstream word-processors and even unknown by common users.

This paper presents a pattern-based language for templates, serialized in HTML and exploited in a user-friendly WYSIWYG editor for writing technical documentation. We discuss the deployment of the editor by an engineering company in the railway domain, as well as some generalized lessons learned about templates.

CCS CONCEPTS

• **Information systems** → *Collaborative and social computing systems and tools*; *Digital libraries and archives*;

KEYWORDS

Collaborative editing, HTML, Technical documentation, Templating

ACM Reference Format:
Alessandro Caponi, Angelo Di Iorio, Fabio Vitali, Paolo Alberti, and Marcello Scatá. 2018. Exploiting patterns and templates for technical documentation. In *DocEng '18: ACM Symposium on Document Engineering 2018, August 28–31, 2018, Halifax, NS, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3209280.3209537

1 INTRODUCTION

HTML has become the beginning and end of markup needs. With CSS, microformats, microdata and structural classes it handles all the things for which XML was designed at the beginning.

What is not yet completed in HTML is the support for a fourth dimension of markup, that of the authoring and processing control (that we call APC): for instance information about change tracking, reuse of fragments from other documents, authorship attribution are not yet expressed in a straightforward and widely shared way.

In this paper we focus on fragments' reuse and templates, and we investigate a pattern-based approach for handling and embedding such information in HTML. The context in which we experimented our approach is the production of industrial design documents. In that domain, the structure of the documents is pretty much established at the beginning of a project and repeated, typographical autonomy is disdained, and semantics identification is of limited usefulness. On the other hand, the control of authoring and processing events is very important as well as the control of reusable content.

This work follows our previous research on patterns [6] and HTML as language for writing specialized documents [11]: we started focusing on scientific articles and now we also work on technical documentation. We propose a HTML-based format, called ADF, able to express APC information and an editor, called SSE, that includes modules for automatic generation of content, numbering and document structures, and sophisticated mechanisms for templating and provenance characterization. We also introduce the PDX library of DOM extensions for handling automatic behaviors, on top of which SSE was built.

We applied this approach successfully in an engineering domain with the help of an important multi-national company in the railway domain. The paper also describes this case-study, starting from its specific needs and moving to possible generalizations of the overall approach.

It is then structured as follows. Section 2 discusses how HTML has been used in practice for expressing authoring information and, in particular, templating data. Section 3 describes our case-study, while our approach, format and tools are presented in Section 4. Discussions and conclusions are drawn in Section 6.

2 RELATED WORK

HTML has been increasingly gaining centrality in markup languages, up to take over XML in several cases. Related technologies and standards make it possible to do a lot of things directly in HTML: CSS let designers create high-quality documents even for printing (instead of using XSL-FO), RDFa and Microformats embed data for automatic processing and reasoning by software agents, structural classes for DIV and SPAN associate presentation and dynamic behavior to single elements, that are finally delivered though JS frameworks.

Particular relevant to our work are some efforts in using HTML for expressing data about the authoring and processing of content (APC, Authoring and Processing Control). In particular, we focus on solutions to handle change-tracking and templating. The same specifications of HTML include the elements INS and DEL to respectively indicate the insertion and the deletion of a piece of content. These elements are very generic and are meant to be applied to modifications of entire subtrees, text fragments or mixed structures. Their usage is not totally precise: for instance, the deletion of an item in a list (LI element) has to be expressed as the deletion of the content of the item instead of the item itself.

The controlled inclusion of content is a further example of APC. Very basic mechanisms are provided in HTML with the element OBJECT: the content is included dynamically in the rendered page but very little information is available about provenance and the connection with the included content is rather weak.

Much more sophisticated inclusion mechanisms have been studied by hypertext and document engineering experts. For instance, researchers proposed to allow for permanent connections between the included content and its original source. A very common scenario, implemented by hot-links and warm-links in the early days of hypermedia [17] and re-proposed by the W3C standard XLink [14] in the@show and @actuate attributes, shows that changes in the source fragment are automatically propagated to all those including it. That is very useful to include real-time information such as news and can be adapted to HTML content too. Similarly, XInclude [9] provides a general and standard solution for inclusions in XML settings that can be applied to HTML. These mechanisms are actually rooted in the early days of the hypertext research. The pioneer Xanadu [12] was meant to be a shared workspace where users could freely link, edit and reuse text fragments.

The key concept of Xanadu were the *transclusions*, i.e. permanent and live inclusions of content. The original idea of transclusions has been implemented in different forms over the years and is still an active topic for the community. For instance, Wikipedia supports basic transclusion mechanisms[1]: entire articles or sections can be included from one wiki page to another. Differently from the original idea, the link is not locked to a particular version of the transcluded content. The most recent version is always included. The adoption of these mechanisms, however, is still limited as confirmed by [1]. The authors studied the writing habits of Wikipedia editors in order to identify which transclusions are used and to which degree, digging into a collection of 20 million articles. They concluded that the editors still have difficulties in understanding and exploiting transclusions, due to the fragmented and incomplete documentation, as well as the cognitive overhead of complex and nested transclusions.

The system presented in [3] allows users to build composite e-books and is based on an interesting classification of transclusions in three different types as proposed by [2], which made a distinction between *Trans-quotations*, *Trans-references* and *Trans-annotations* depending on the rights of reproducing and distributing the transcluded content. The e-books are composed of chunks of content permanently linked to the original source: if there is an updated version, the system downloads it and updates the document appropriately. The overall system relies on a customized metadata schema that describes the relations between the transcluded fragments and their provenance.

Some forms of transclusions were also proposed to allow users to collect any content from the Web. In [10] the authors introduced the idea of *partial bookmarking* and extended a Web browser so as to let users collect portions of any web page and to combine them into new documents shown in a different tab. The environment can be used for reading multi-source content but does not allow users to re-publish content.

It is also worth mentioning here a totally different use of transclusions: in [8] the authors proposed an open-source and cross-platform technology, called Auckland Interface Model (AIM), that allows designers to build GUIs decomposing them by using the transclusion metaphor. AIM separates GUI layout, GUI content and GUI data, and enables end-users

[1]https://en.wikipedia.org/wiki/Wikipedia:Transclusion

to treat GUIs as documents. Such an approach proved to be beneficial for GUI developers and users, facilitating reuse and consistency.

The annotation of text fragments, not only for transclusions but for any automatic processing, has become reality for HTML with the diffusion of Microformats and Microdata. Microformats are small patterns of HTML to represent common things like people, events, reviews, tags and so on. Microdata extends HTML, by introducing some new general-purpose properties, and can be used together with several vocabularies like Schema.org.

Alternative proposals have also been studied by the academic community. To overcome some limitations of microformats in terms of flexibility and editing difficulties, [13], [4] presented a language for semantic embedding in HTML, called XTiger, and an editing tool working on that language that makes the authoring process easier and safer. In fact, the tool uses descriptions expressed XTiger to help authors to produce valid semantically-annotated content. Several other editors for semantic content have been proposed by researchers and professionals, see [7] for more details.

3 REUSING CONTENT FOR TECHNICAL DOCUMENTS: A CASE-STUDY

The production of semantically-annotated content in HTML, in particular that of authoring and processing control, has been widely studied in the literature but is still deployed in a limited manner in mainstream applications.

This research, in fact, started from an analysis we did on methods and tools for writing documentation for engineering projects. These projects often require stakeholders to produce a of documents. Some of these are rigidly structured and composed of well-defined items that can be assembled and delivered into the final document: this is the case, for instance, of manuals associated to products and machines. These are handled by specialized tools that are very effective, and characterized by peculiar difficulties, editing patterns and approaches [16].

These are many other documents which are equally important but difficult to be written by following the same approach and using the same tools. Some examples are: notes, documentation for tenders, business plans, quality plans, legal notices and so on. These are often made of common pieces that are simply copied&pasted across documents, due to the lack of mainstream effective tools supporting such a reuse.

This also happened in our case-study. Alstom is a multinational company operating worldwide in rail transport markets, active in the fields of passenger transportation, signalling and locomotives. The Alstom site at Bologna (Italy), center of excellence for the signalling technology, is involved in a lot of projects (national and international) producting a huge number of documents for each project. A medium size project has a documentation master list of around 1.000 documents, while the documentation master list of a complex project can reach more than 10.000 different documents.

A not negligible part of such documents are written directly using a standard word processor, starting from scratch or from the most similar document written for a different project. Copy&paste errors, typo and layout mismatch are frequent causes of low quality document delivery.

Alstom Bologna therefore considered to move to new tools for writing such documentation aiming at:

- improving the documents quality and timeliness
- giving the possibility to write and verify documents in parallel
- sharing different templates and standard chapters through out specific libraries of content
- having limited impact on final users (authors) way of working

These desiderata were fullfilled by implementing a Web-based environment, called SSE (Smart Structured Editor), that allows users to easily (i) produce structured content enriched with some metadata to easy the document tracking and exploring, and (ii) share content fragments via templating. SSE authors do not have to deal with with typographic and presentational aspects but just to focus on content, while all formatting, quality check and generation of tables and indexes is demanded to the system.

In the next section, we will introduce the core HTML-based format developed for the project, called ADF and strongly based on patterns. In the following one we will provide more details on the SSE templating mechanisms and their application to the case-study.

4 ADF: EXPLOITING HTML PATTERNS AND TEMPLATES

The main goal of this research was to study the definition and application of a pattern-based subset of HTML to the production of technical documents, with particular attention to templating mechanisms. Actually, the vocabulary is independent of specific markup expectations. It can be expressed in many different markup languages, such as XML, HTML and even RDF. The HTML serialization was used throughout the project and will be used here for all the examples.

The vocabulary is built on three key pillars:

- **Pattern-based structural and semantic specification**: each element of the language addresses and describes precisely one aspect of the document according to the pattern theory of documents developed by the authors of this document, and an analysis of the structural and semantic characteristics of the documents most frequently used in Alstom.

- **Structure reuse**: small and big text fragments can be identified in good documents of the past, extracted and reused in newer ones so as to generate a complete library of templates ready to use and reuse. Templates may be composed of a few words, a few lines, long portions of documents, and even structures and scaffoldings (more details in the following subsection)
- **Provenance**: each part of the document, from metadata to individual fragments, is clearly and unambiguously associated to the moment in time and the actor that generated it, with a full history of its modifications in time. This allows to maintain a full tracking mechanism of the content of the document and who created it.

The research is rooted in our past research on structural patterns. We have been investigating how the structure of digital documents can be segmented into recurring atomic components, which can be addressed independently and manipulated for different purposes. Instead of defining a large number of complex and diversified structures, we identified a small number of structural patterns that are sufficient to express what most users need. The two main characterizing aspects of our patterns are: (i) orthogonality (each pattern needs to have a specific goal and to fit a specific context) and (ii) assemblability (each pattern can be used only in some locations, within other patterns). Patterns allow authors to create unambiguous, manageable and well-structured documents and the regularity of pattern-based documents makes it possible to perform easily complex operations, conversions and manipulations.

The patterns were used to define RASH, a markup language that restricts the use of HTML elements to only 32 elements, and enables one to include also RDF statements. In addition, it uses the new Digital Publishing WAI-ARIA Module 1.0 [5] for specifying accessible structural semantics to the various document parts. It is designed for encoding scientific articles and already adopted by several conferences in computer science field. While the framework that has been developed for RASH includes a native editor for facilitating the creation of Web-first articles [15], there are some peculiarities of the (ALSTOM) technical documentation that suggested us to fork RASH in a new language called ADF and to build a native ADF editor.

Basically, an ADF document is a HTML page composed of three parts:

- metadata providing non-content information about the document, its context, and the default parameters used. These metadata are expressed in JSON-LD and RDF-a and stored within the document itself
- front matter including the front page, and the initial tables
- the body of the document with the actual content; this content is fully compliant to RASH, and thus to both the pattern-theory and HTML

Most details of ADF, especially those related to the ALSTOM needs, are not relevant for the purposes of this paper. What is most interesting here is the support for templating mechanisms provided by ADF.

Expressing and using templates in a ADF

A template in ADF is a fragment of content placed inside a document but originating elsewhere, and possibly shared by more than one document. The origin of the template is called the *source*, and each document that uses the template is called a *client* of the template. The template is composed of *fixed content* (received as is from the source and not designed to be modified) and *variable content* (parts that are empty or filled in with default text, and meant to be modified and customized differently by each client document). Some templates may have no variable content, and be composed only of predetermined fixed content not meant for modification by clients. These can be called *formulaic* templates.

There is no restriction on the content of templates. This means that templates can contain anything from whole documents down to individual words, as well as hierarchies, tables, lists, figures, etc. Templates can even nest, e.g., when a variable of a template contains another template. Furthermore, the vocabulary does not prevent fixed content from modification by the document's author, unless specifically prevented from changes through the use of a specific class (*docreadonly)*

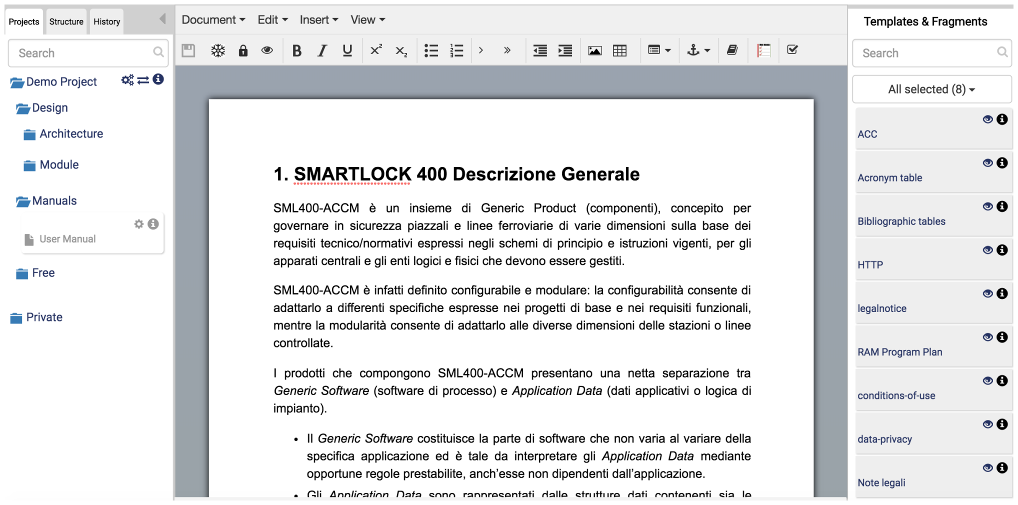
The HTML representation of a template sees the template as an instance of a special RDF-a class called, appropriately enough, "*Template"*. A template must always be associated to a subtree, i.e., a single HTML element must contain the whole template. A very small text fragment is shown below:

```
<span typeof="Template" property="tpl:source"
  content="templateEngine.php?tpl=AlstomName">
     Alstom Ferroviaria S.p.A. </span>
```

The only required property for templates is "tpl:source", whose value must point to the URI of the source of the template, be it a file somewhere, a network service or even a fragment of the document itself.

Additional attributes associated to a template and not meant to be displayed, such as the version used, or the author of the template, can be specified via empty <span> elements, as shown in the following snippet:

```
<span typeof="Template" property="tpl:source"
   content="templateEngine.php?tpl=AlstomName">
     <span property="tpl:version" content="1.0"/>
      Alstom Ferroviaria S.p.A. </span>
```



1. SMARTLOCK 400 Descrizione Generale

SML400-ACCM è un insieme di Generic Product (componenti), concepito per governare in sicurezza piazzali e linee ferroviarie di varie dimensioni sulla base dei requisiti tecnico/normativi espressi negli schemi di principio e istruzioni vigenti, per gli apparati centrali e gli enti logici e fisici che devono essere gestiti.

SML400-ACCM è infatti definito configurabile e modulare: la configurabilità consente di adattarlo a differenti specifiche espresse nei progetti di base e nei requisiti funzionali, mentre la modularità consente di adattarlo alle diverse dimensioni delle stazioni o linee controllate.

I prodotti che compongono SML400-ACCM presentano una netta separazione tra *Generic Software* (software di processo) e *Application Data* (dati applicativi o logica di impianto).

- Il *Generic Software* costituisce la parte di software che non varia al variare della specifica applicazione ed è tale da interpretare gli *Application Data* mediante opportune regole prestabilite, anch'esse non dipendenti dall'applicazione.
- Gli *Application Data* sono rappresentati dalle strutture dati contenenti sia le

Figure 1: Overall interface of SSE

Variables are properties of the template. Properties whose value is inside the content of the element can be directly modified by the author. Properties whose value is specified via the content attribute of the element may or may not be modified depending on the purpose of the variable. Since each client of the template as its own private copies of the variables, we need to introduce an identity to the template. Furthermore, most probably the content of the variable is not a plain string, but an HTML fragment. This requires explicit specification of the datatype of the property.

```
<span typeof="Template" property="tpl:source"
   content="templateEngine.php?tpl=AlstomName"
    resource="#AlstomName">
    <span property="tpl:version" content="1.0"/>
    Alstom Ferroviaria S.p.A. -
    <span property="premise" datatype="rdf:HTML">
      Sede di Bologna
    </span>
</span>
```

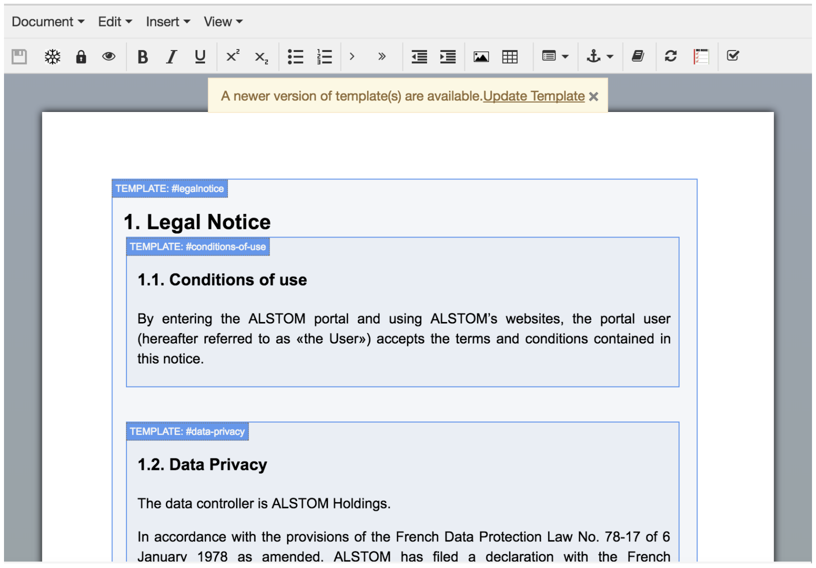
Nested templates can be used, but it may become difficult to associate a property to the correct template. For this reason, if a template contains another template, both use a *resource* attribute and all SPAN elements defining properties use an *about* value to point to the correct template.

```
<p typeof="Template" property="tpl:source"
   content="templateEngine.php?tpl=SignatureBlock"
    resource="#SignatureBlock">
    <span property="tpl:version" content="3.0"
     about="#SignatureBlock"></span>
     This document is signed in <span
        property="location"
       about="#SignatureBlock">Bologna</span> on
    <span property="date" about="#SignatureBlock">
        21 January 2017</span> by
    <span property="signee" about="#SignatureBlock">
        P. Alberti</span> on behalf of
    <span property="onBehalfOf" datatype="rdf:HTML"
       about="#SignatureBlock">
      <span typeof="Template" property="tpl:source"
       content="templateEngine.php?tpl=AlstomName"
         resource="#AlstomName">
        <span property="tpl:version" content="1.0"
          about="#AlstomName"></span>
         Alstom Ferroviaria S.p.A. -
       <span property="premise" datatype="rdf:HTML"
       about="#AlstomName"> Sede di Bologna </span>
      </span>
  </span>
</p>
```

The previous code, for instance, shows a template with identified (*id* attribute) *SignatureBlock* that has some properties (location, date, etc.) and contains another template called *AlstomName*. The *content* attribute contains the URL where the template content is. In this example the content is delivered by a php script with a given *tpl* parameter that holds the template identifier.

5 WRITING AND SHARING ADF TEMPLATES IN SSE

SSE was built to to help authors create high-quality and consistent ADF documents and to reduce their effort. Fig. 1 shows the main SSE interface. Note that SSE is a fully Web application that can be accessed directly via browser, no further installation is needed.



A newer version of template(s) are available.Update Template

TEMPLATE: #legalnotice

1. Legal Notice

TEMPLATE: #conditions-of-use

1.1. Conditions of use

By entering the ALSTOM portal and using ALSTOM's websites, the portal user (hereafter referred to as «the User») accepts the terms and conditions contained in this notice.

TEMPLATE: #data-privacy

1.2. Data Privacy

The data controller is ALSTOM Holdings.

In accordance with the provisions of the French Data Protection Law No. 78-17 of 6 January 1978 as amended, ALSTOM has filed a declaration with the French

Figure 2: Template update notification and nested templates

The interface is organized in four areas:

(1) Header: contains information about the user profile and log on buttons
(2) Project area (left column): allows users to explore projects, as well as the structure and the history of the current document.
(3) Template area (right column): allows users to include templates and fragments in the current document
(4) Document area (editor): the main component of SSE in which users can write documents

The template area is the most relevant for our purposes. It contains a list of fragments and templates that can be included inside the document that is open. The authors of templates can associate one or more categories to each template when creating it (or later on). The user can filter templates/fragments according to their category. Each project is configured to use a predefined taxonomy of categories. The user can also search templates by name. These basic filtering and search mechanisms, as we will discuss later, still have some limitations when the number of templates increases; in fact we are doing some refactoring so as to make it more scalable.

The icons on the right of each item shows provenance information about the template (author, modification date and so on) and a preview of the template content.

To include a template/fragment users can drag & drop the template in the desired position of the target document. Note that the section numbering is automatically managed by SSE after including a template.

Templates can be created or modified directly in SSE and can be edited like documents. Authorized users can access a template area that is similar to document one but with peculiar controls to manage content. The left sidebar is different as users can navigate templates. For convenience, templates re grouped into two macro-categories that are *document templates*, that contains all multiple section templates, and *fragments*, that are manual parts, legal paragraphs or technical product descriptions. The user can filter templates by categories or search by name as well.

Note that templates can be included in both documents and other templates. In fact, the right side of SSE constantly shows the list of available templates and fragments in template editor too. Nested templates can be easily created by the authors and allow for the creation of composite documents.

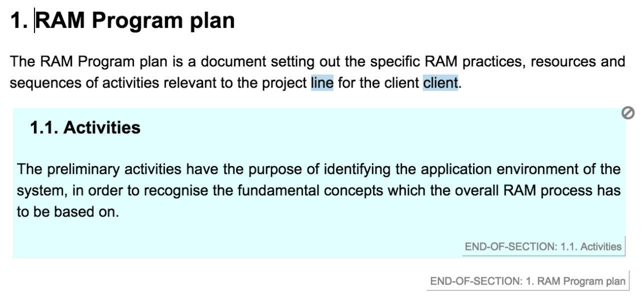
After its inclusion in a document, a template keeps being linked to the original template it comes from. SSE provides users with an opt-in mechanism: when opening a document, SSE checks all templates included in that document and shows the list of the templates that have been modified. The user can select which content to update and which not. The user has always the opportunity to update a template whenever he wants using a toolbar button.

Figure 2 shows template update notification and nested templates.

The user can also use project metadata within templates by selecting from the project properties list (e.i. name, client, ...). The template author can see the property label and they will be substituted by SSE with the values inside the document editor. This can be useful if you consider that templates are available on different projects.

Another important feature is that the user can lock sections or part of text so that the document author cannot change that parts after including the template. For example the user wants to insert legal notice that must not be changed by the document author.

Figures 3 shows two metadata and a locked section inside a template. Locked sections show a deny icon on the right top corner and have an azure background. Metadata have a blue background.


1. RAM Program plan

The RAM Program plan is a document setting out the specific RAM practices, resources and sequences of activities relevant to the project line for the client client.

1.1. Activities

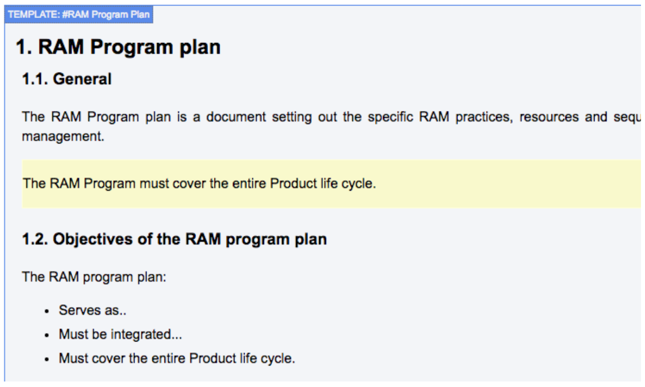
The preliminary activities have the purpose of identifying the application environment of the system, in order to recognise the fundamental concepts which the overall RAM process has to be based on.

END-OF-SECTION: 1.1. Activities

END-OF-SECTION: 1. RAM Program plan

Figure 3: Some of the template features: locked sections and project metadata

An author can change all not-fixed sections and blocks of a template, however if the user updates the template every change will be lost. That is why ADF vocabulary also contains *variables*. Block variables allow users to insert paragraphs, sections, tables, etc. Figure 4 shows a template variable inside the document editor.


TEMPLATE: #RAM Program Plan

1. RAM Program plan

1.1. General

The RAM Program plan is a document setting out the specific RAM practices, resources and sequ
management.

The RAM Program must cover the entire Product life cycle.

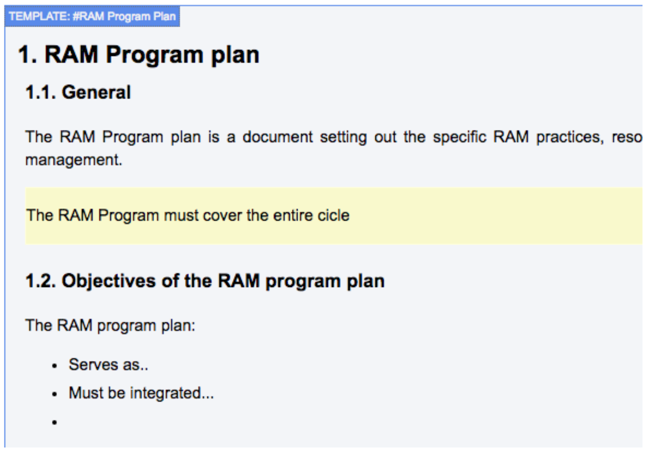
1.2. Objectives of the RAM program plan

The RAM program plan:

- Serves as..
- Must be integrated...
- Must cover the entire Product life cycle.

Figure 4: A template variable (yellow block)

When the template is updated, the instantiated content of a variable is not affected. An example is shown in Figure 5: the template is up-to-date (the last element of the list is deleted) but the variable content is the same as before.


TEMPLATE: #RAM Program Plan

1. RAM Program plan

1.1. General

The RAM Program plan is a document setting out the specific RAM practices, reso
management.

The RAM Program must cover the entire cicle

1.2. Objectives of the RAM program plan

The RAM program plan:

- Serves as..
- Must be integrated...
-

Figure 5: the template variable is not affected by template update

Implementation notes

SSE in written in Node.js and has a typical architecture based on the MVC (Model View Controller) pattern. As expected, the most important component is the JS front-end, while the backend only manages documents storage and format conversions.

The front-end is based on a customized version of TinyMCE[2] a WYSIWYG HTML-based editor that can be instantiated within an existing HTML page. TinyMCE is a large project (more than 150 contributors) and includes modules for formatting, table insertion, image editing, customizable themes. It is already integrated with frameworks such as JQuery and Angular.js and highly customizable. This is the main reason why we selected TinyMCE as core component of SSE. The customization we put in place was mainly to ensure consistency with our patterns theory.

Last but not least, SSE uses PDX[3], a DOM extension for managing the "behavior" of the documents. The term "behavior" indicates how specific aspects of a document element (style and/or content) are determined by the event happening to it or to related elements. Consider, for instance, the frequent action of renumbering a section due to the insertion/deletion of other sections, or numbering a reference to a table or an image, or updating a template: all these dynamic behaviours are described in PDX by annotating the affected HTML elements and invoking the PDX Javascript library that modifies the DOM accordingly. In particular, PDX includes two behaviours: -able elements, hat possess information enabling behaviors, and -ing elements, that act based on the information of connected elements. There are also -er elements, that provide characteristics of both. Technical

[2]https://www.tinymce.com/
[3]http://www.fabiovitali.it/pdx/

details of PDX are not discussed for the sake of brevity but they are a key component for SSE and templates functioning.

6 DISCUSSION AND CONCLUSIONS

Now ADF and SSE are currently used by a small team of Alstom users to create and share documents. We are collecting feedback in two ways. On the one hand, SSE includes a form that authors can use at anytime to signal issues and suggest improvements. On the other, we organized hands-on sessions in which we collected feedback from the final users in form of free text notes.

These sessions include 4-6 testers and comprise two testing methods:

(1) *cognitive walkthrough* on some common and complex tasks
(2) *guerilla user testing*, on some simple tasks on average documents (of around 30 pages)

This preliminary testing phase was not exhaustive but gave us valuable indications about the applicability of templates in real scenarios and about main usability issues.

We also stressed SSE application loading and working some documents with more than 500 pages. This was an important way to understand the scalability of the framework behind SSE. We have no problems working with big document but we have found that the data formats converter (for instance from ADF to .DOCX) is quite slow on that kind of documents.

In the near future we will focus on testing and stressing template and subtemplating inclusion.

We briefly list here some aspects that our testers highlighted so far, with particular attention to templates:

- the overall interface of SSE is perceived as quite usable, even if some users consider it a bit overloaded; some users suggest us to use more menus instead of showing all available options in the toolbar
- the idea of showing a list of available fragments in the right sidebar is functional; on the other hand, users are quite skeptical about the scalability of such an approach with a large amount of fragments; the presence of a search toolbar and filtering mechanism – that were not available in the first version of SSE – is rated as fundamental though
- templates are considered effective tools to force users to create documents with a given structure; in particular, the testers appreciated the possibility of creating templates with all headings blocked. Since other users can only fill these sections, but cannot remove them, consistency is easier to maintain. This is a critical point: while ADF allows for complex updating mechanisms and users can even modify an included content, most authors did not use these templates but preferred more constrained ones
- the creation of templates is still considered a difficult task: in particular, users experienced some difficulties in managing both plain documents and templates in the same workspace; our current solution is to provide two separate environments (highlighted with clear visual clues) that user can easily switch from/to
- updates of templates are useful operations but they need to be reinforced by introducing preview mechanisms: currently, SSE only shows a description of the template being updated but users cannot see in advance how the document looks like if a template is updated
- templates are too static; most users suggested that a document might include some content under given conditions but this cannot be expressed in ADF, neither is possible with SSE; this is a very interesting and promising direction, that we plan to address in the near future.

In conclusion, the preliminary experiments with ADF and SSE are promising. The pattern-based model of the language, and in particular its serialization in HTML and enrichment with semantic information, proved to be feasible and effective. On the other hand, extensive and complete usability tests are still needed.

Some extensions and refactoring are already planned for both ADF and SSE. ADF is being extended to also handle conditional templates and more sophisticated controls; also, we are investigating how to model objects outside the railway domain by exploiting the same approach. SSE is evolving too: we are working to support multiple tabs, multi-format conversions (for instance, to export in DITA format and to import content directly from common word-processors) and to provide users with a more cohesive environment to work on documents and templates.

REFERENCES

[1] Mark Anderson, Leslie Carr, and David E. Millard. 2017. There and Here: Patterns of Content Transclusion in Wikipedia. InÂăProceedings of the 28th ACM Conference on Hypertext and Social Media (HT '17). ACM, New York, NY, USA, 115-124.
[2] Ja-Ryoung Choi, Sungeun An, and Soon-Bum Lim. 2014. Spatial hypertext modeling for dynamic contents authoring system based on transclusion. In Proceedings of the 25th ACM conference on Hypertext and social media (HT '14).
[3] Ja-Ryoung Choi and Soon-Bum Lim. 2016. A digital publishing system with open resources transclusion and tracing. In Proceedings of the 4th International Workshop on Document Changes: Modeling, Detection, Storage and Visualization (DChanges '16). ACM, New York, NY, USA, , Article 3 , 4 pages.
[4] Francesc Campoy Flores, Vincent Quint, and Irène Vatton. 2006. Templates, microformats and structured editing. In Proceedings of the 2006 ACM symposium on Document engineering (DocEng '06). ACM, New

York, NY, USA, 188-197.

[5] M. Garrish, T. Siegman, M. Gylling, S. McCarron, Digital Publishing WAI-ARIA Module 1.0, W3C Recommendation, https://www.w3.org/TR/dpub-aria-1.0/, 2017.

[6] A. Di Iorio, S. Peroni, F. Poggi, F. Vitali "Dealing with structural patterns of XML documents". Journal of the American Society for Information Science and Technology, JASIST, vol. 65, no. 9, pp. 1884–1900.

[7] Ali Khalili and SÃűren Auer. 2013. Review article: User interfaces for semantic authoring of textual content: A systematic literature review. Web Semant. 22 (October 2013), 1-18.

[8] Lung-Chen Lee, Christof Lutteroth, and Gerald Weber. 2010. Improving end-user GUI customization with transclusion. In Proceedings of the Thirty-Third Australasian Conference on Computer Science - Volume 102 (ACSC '10), Bernard Mans and Mark Reynolds (Eds.), Vol. 102. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 163-172.

[9] J. Marsh, D. Orchard, and D. Veillard. XML Inclusions (XInclude) Version 1.0. http://www.w3.org/TR/xinclude/, 2006.

[10] Takehiro Nagatomo, Takahiro Tachibana, Keizo Sato, and Makoto Nakashima. 2016. Partial Bookmarking: A Structure-independent Mechanism of Transclusion for a Portion of any Web Page. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16 Adjunct). ACM, New York, NY, USA, 185-186.

[11] Peroni, S., Osborne, F., Di Iorio, A., Nuzzolese, A.G., Poggi, F., Vitali, F. and Motta, E., 2017. Research Articles in Simplified HTML: a Web-first format for HTML-based scholarly articles. PeerJ Computer Science, 3, p.e132.

[12] T. H. Nelson. Literary Machines. Mindful Press, Sausalito, CA, USA, 1990.

[13] Vincent Quint and Irène Vatton. 2007. Structured templates for authoring semantically rich documents. In Proceedings of the 2007 international workshop on Semantically aware document processing and indexing (SADPI '07). ACM, New York, NY, USA, 41-48.

[14] S. De Rose, E. Maler, D. Orchard, and N. Walsh. XML Linking Language (XLink) Version 1.1. http://www.w3.org/TR/xlink11/, 2001.

[15] Gianmarco Spinaci, Silvio Peroni, Angelo Di Iorio, Francesco Poggi, and Fabio Vitali. 2017. The RASH JavaScript Editor (RAJE): A Wordprocessor for Writing Web-first Scholarly Articles. In Proceedings of the 2017 ACM Symposium on Document Engineering (DocEng '17). ACM, New York, NY, USA, 85-94.

[16] Ingo Stock and Michael Weber. 2006. Authoring technical documentation using a generic document model. In Proceedings of the 24th annual ACM international conference on Design of communication (SIGDOC '06). ACM, New York, NY, USA, 172-179.

[17] N. Meyrowitz. Hypertext|does it reduce cholesterol, too? Pages 287-318, 1991.