A Software Architecture for Centralized Management of Structured Documents in a Cooperative Editing Workflow

Milliam Maxime Zekeng Ndadji(✉) and Maurice Tchoupé Tchendji(✉)

Department of Mathematics and Computer Science, University of Dschang, PO Box 67, Dschang, Cameroon
{ndadji.maxime,maurice.tchoupe}@univ-dschang.org, ndadjimaxime@yahoo.fr,ttchoupe@yahoo.fr

Abstract. Nowadays, electronic documents are widely used as media of exchange between actors involved in a given business process. Generally, their contents provide information on both what has already been done on this procedure and what remains to be done and by whom it should be done. Badouel and Tchoupé proposed a modelling of the life cycle of such documents through an administrative workflow in which, each of the participants manipulates a partial replica of the document representing at some point, the current state of the process execution; the overall state of the process being obtained by merging different partial replicas. This paper presents a modular software architecture for the implementation and management of such workflow systems.

Keywords: Software architecture · Structured documents
Cooperative editing workflow · Workflows management system
Cross fertilization

1 Introduction

Workflow technology, emerged in the 80s, knows a quickly growing in the industry of computer-aided production. This interest is due to the ability of this one, to allow companies to reduce costs of their productions, to quickly and easily develop new products and services, and therefore, to be competitive [4]. Workflow technology provides methods and tools (notations, management systems, ...) for the specification, automation, optimization and monitoring of business processes [4].

A complex workflow system may be composed of several subsystems (sites) distributed across a network; these coordinating themselves through a workflow management system that can use for this purpose, documents (artifacts) issued from the various subsystems. Badouel and Tchoupé [5] have theorized a business process running approach in which, the state of the process at some point is represented by a document, and stakeholders from subsystems work by editing

C. M. F. Kebe et al. (Eds.): InterSol 2017/CNRIA 2017, LNICST 204, pp. 279–291, 2018.
https://doi.org/10.1007/978-3-319-72965-7_26

and exchanging (partial) replicas of documents representing their perceptions of the workflow at any given time. Therefore, each subsystem (actor) has a partial view of the overall state of the workflow at any given time, and the current (global) workflow state is given by the merging of different artifacts (documents) from the various subsystems. In their model, collaborations between actors can be divided into three sequential phases (Fig. 1): the *distribution phase* where global artifact (a structured document) is replicated (partially) to each subsystem (site), the *edition phase* in which local processes of subsystems are executed, inducing an update of the local replica of the global artifact, and finally, the *synchronization phase* in which the various local artifacts updated are merged into a global artifact. Thus presented, the execution of such a workflow is similar to the cooperative editing of a structured document. That is why in the rest of this manuscript, in order to alleviate the speech and use an easily accessible jargon, we will only use expressions from the field of structured editing.

We present in this paper, a modular software architecture for the implementation of a centralized management of cooperative editing workflows where, stakeholders operate on partial replicas of the document edited collectively. Some business components of this architecture have already been developed in previous works [5,12–14]. We describe here primarily, the other technical components and their orchestration for the effective implementation of a cooperative editing workflow management system as described in [5].

The rest of this manuscript is organized as follows: we present a literature review on cooperative editing workflows as well as software architectures used for their implementation (Sect. 2). We then present (Sect. 3), a software architecture for designing cooperative editing workflow management systems as described in [5], and a sample implementation via an editor prototype we developed. Section 4 is devoted to the conclusion.

2 Basic Concepts and Approaches to Manage Cooperative Editing Workflows

2.1 Workflow and Cooperative Editing

A workflow can be defined as a collection of tasks organized and performed either by software systems, by humans or both, in order to accomplish some business process [4]. The aim of the latter is to streamline, coordinate and control business processes in an organized, distributed and computerized environment. [4,17] informally define a business process as an ordered sequence of tasks (made by men or by programs, even both sometimes) that meets a specific scheme and leads to a specific result (*tracking a medical record* [15] is a frequent example of business process). It is obvious that a business process can be (considered as) a Computer-Supported Cooperative Work (CSCW). Therefore, workflow management[1] needs to facilitate trade, coordination, collaboration and co-decision

[1] Informally, we design by workflow management, the modeling and computerized management of all tasks and various actors involved in a business process.

between the participants to the underlying business process. To do this, we can use electronic documents [4,5,10] as media (carriers): it is said in this case, that the workflow is document-centric [3]. In such workflows, documents move from one site to another, and are edited progressively[2]: They therefore contain, at any time, informations that are indispensable to the cooperation. Since these documents are edited by different actors, it is a cooperative editing.

Cooperative editing is a work of hierarchically organized groups that operates according to a schedule involving delays and a division of labor (coordination). Like any CSCW, cooperative editing is subject to spatial and temporal constraints. Thus, we distinguish distributed or not, and asynchronous or synchronous cooperative editing. When distributed, the various editing sites are geographically dispersed and each of them has a local copy of the document to be edited; systems that support such an edition should offer algorithms for data replication [23] and for the fusion of updates. When asynchronous, various co-authors get involved at different times to bring their different contributions.

A cooperative editing workflow goes generally, from the creation of the document to edit, to the production of the final document through the alternation and repetition of distribution, editing and synchronization phases. The literature is full of several cooperative editing workflows and of their management systems. We present a few in the next section.

2.2 Cooperative Editing Workflows and Management Approaches

Real-Time Cooperative Editing Workflows. In these generally centralized systems (Etherpad[3] [8], Google Docs[4], Framapad[5], Fidus Writer[6] [11], ...), the original document is created by a co-author on the central server. The latter then invites his colleagues to join him for the editing; they therefore connect to the editing session usually identified by a URL (distribution phase, although the document is generally not really duplicated). During an editing session (synchronous editing phase), all connected co-authors work on a single copy of the document but in different contexts. When the integration is automatic, changes performed by one of them are immediately (automatically) propagated to be incorporated into the basic document (synchronization phase), and the latter is then redistributed to others. The changes are saved progressively and the server usually keeps multiple versions of the document.

The majority of real-time editors uses the model of operational transformations [6,16]. Their architectures are therefore based on the one defined by this model. Meaning that, they distinguish two main components: an *integration*

[2] Actors do not necessarily publish documents through specialized software (texts editors); they perform their tasks related to their respective areas of expertise and documents are amended accordingly.
[3] http://www.etherpad.org/.
[4] https://www.docs.google.com/.
[5] http://www.framasoft.org/.
[6] https://www.fiduswriter.org/.

algorithm, responsible for the receipt, dissemination and execution of operations and a *set of processing functions* that are responsible of "merging" updates by serializing two concurrent operations. This type of editing workflow works great for small groups.

Asynchronous Cooperative Editing Workflows: This edit mode is distinguished by real distribution phases in which, the document to edit is replicated on different sites, using appropriate algorithms [23]. A co-author may then contribute at any time by editing his local copy of the document. Here we focus on a few asynchronous cooperative editors operating in client-server mode.

Wikiwikiweb: Wikis [19,22] are a family of collaborative editors for editing web pages from a browser. To edit a page on a Wiki, you must duplicate it and contribute. After editing, you just have to save to publish a new version of that page. In the case of a competing editing, it is the last published version which will be visible. Even though it is still possible to access the previously published versions, there is no guarantee that a new version of the page preserves intentions (incorporates aspects) of previous versions. For this aspect, a Wiki can be seen much more as a web page version manager.

CVS (Concurrent Versions System): Under CVS [2], versions of a document are managed in a space called repository and each user has a personal workspace. To edit a document, the user must create a replica in his workspace. He will amend this replica, then will release a new version of the document in the repository. In case the document is edited by several people concurrently and at least one update has already been published, the author wishing to publish a new update will be forced to consult and integrate all previous updates through dedicated tools integrated in CVS.

SVN (Subversion): SVN[7] [21] was created to replace CVS. Its main goal was to propose a better implementation of CVS. So as CVS, SVN relies on an optimistic protocol of concurrent access management: the copy-edit-merge paradigm. SVN provides many technical changes like: a new commit algorithm, the management of metadata versions, new user commands and many others features.

Git: The main purpose of Git[8] [20] is the management of various files in a content tree considered as a deposit (all files of a source code for example). To edit a deposit, a given user connects to it and clones (forks). He obtains a copy of that deposit, modifies it locally through a set of commands provided by Git. Then he offers his contribution to primary maintainer who can validate it and thus, merges it with the original deposit. During this operation, new versions of modified files are created in the main repository. It is therefore possible under Git, to access any revision of a given file.

[7] http://www.subversion.apache.org/.
[8] https://www.git-scm.com/.

Badouel and Tchoupé Cooperative Editing Workflow. Badouel and Tchoupé [5] proposed a workflow for cooperative editing of structured documents (those with regular structures defined by grammatical models such as DTD, XML schema [18], ...), based on the concept of "view". The authors use context-free grammars as documents templates. A document is thus, a derivation tree for a given grammar.

The life cycle of a document in their workflow can be sketched as follows: initially, the document to edit (t) is in a specific state (initial state); various co-authors who are potentially located in distant geographical sites, get a copy of t that they edit locally. For several reasons (confidentiality, security, efficiency, ... [12]), a given co-author "i" has not necessarily access to all the grammatical symbols that appear in the tree; only a subset of them can be considered relevant for him: that is his *view* ($\mathcal{V}_i$). The locally edited document, is therefore a *partial replica* (denoted $t_{\mathcal{V}_i}$) of the original document. This one is obtained by *projection* (π) of the original document with regard to the view of the considered co-author ($t_{\mathcal{V}_i} = \pi_{\mathcal{V}_i}(t)$). The edition is asynchronous and local documents obtained are called updated partial replicas denoted by $t^{maj}_{\mathcal{V}_i}$.

The authors of [5] focus only on the positive edition; edited documents are only increasing, and the co-authors cannot remove portions of the document when a synchronization was already performed. For both ensure that property and to be able to tell a co-author where he shall contribute, the documents being edited are represented by trees with *buds* that indicate the only places where editions are possible. Buds are typed; a *bud of sort* X is a leaf node labeled X_ω: it can only be edited (extended in a sub-tree) by using a X*-production* (production with X as left hand side).

When a synchronization point[9] is reached, all contributions $t^{maj}_{\mathcal{V}_i}$ of different co-authors are merged in a single global document t_f[10]. To ensure that the merging is always possible (convergence), the authors of [5] assume that on each site, the editions are controlled by a local grammar. These local grammars are obtained from the global one, by projection along the corresponding views [12]. It can still happen that updates being not compatible[11] and therefore the merging being impossible. To overcome this, we have proposed in [13], an algorithm to reconcile partial replicas in conflict.

Figure 1 give an overview with a BPMN (Business Process Modeling Notation) orchestration diagram, of the workflow of cooperative editing of a structured document according to Badouel and Tchoupé [5] proposal; at site 1,

[9] A synchronization point can be defined statically or triggered by a co-author as soon as certain properties are satisfied.

[10] Sometimes the edition should be continued after the merging (it is so, when there still buds in the merged document): the document must be redistributed to each of the n co-authors, a (partial) replica $t_{\mathcal{V}_i}$ of t_f, ensuring that $t_{\mathcal{V}_i} = \pi_{\mathcal{V}_i}(t_f)$, for the continuation of the editing process.

[11] This is particularly the case if there is at least one node of the global document accessible by more than a co-author and edited by at least two of them using different productions.

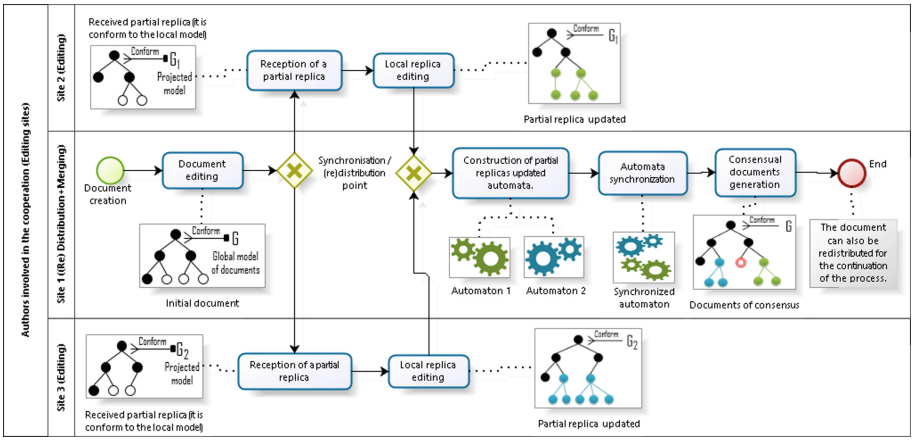


Fig. 1. A BPMN orchestration diagram sketching a cooperative editing workflow of a structured document according to Badouel and Tchoupé [5].

operations of (re)distribution and merging of the document in accordance with a (global) model G are realized; at sites 2 and 3, edition of partial replicas in accordance with (local) models G_1 and G_2 derived by projecting the global documents template G are done.

In summary, the workflow of Badouel and Tchoupé is different from the others because of the concept of view and by the fact that it manipulates exclusively (partial) structured documents. The other workflows globally differ in their approaches and objectives.

We propose in the next section, a flexible software architecture for automating business processes that could be modelled as workflows of Badouel and Tchoupé [5].

3 A Software Architecture for Centralized Management of Cooperative Editing Workflows

3.1 Motivations

The contribution we make to the implementation of workflow management systems is based on the type of workflow specified in [5]. The choice focused on this type of workflow is motivated by the fact that:

1. *It applies to structured documents*: this leads to the fact that we can locally perform validations in accordance with a local model derived from the global one;
2. *It is particularly suited for administrative workflows*: concepts of view and partial replica introduced by Badouel and Tchoupé, make that the type of workflow they offer is particularly adapted for the specification of many

administrative processes. Consider, for example, the process "tracking a medical record in a health center with the reception and consultation services": the aforesaid record can be modelled as a structured document in which the members of the host service (reception) cannot view and/or modify certain information contained therein; those information, requesting the expertise of the consulting staff for example. Therefore we can associate views to each of these services. It remains only to specify the medical record's circuit and a workflow of the type described in [5] is obtained;

3. *It does not exist a generic architectural model describing precisely an approach for the implementation of this type of workflow*: the only prototype [14] which was designed around the concepts handled (view, partial replica, merging, ...) for this type of workflow, was more a graphic tool (editor) for the experimentation of concepts and algorithms presented in [5]; workflow management is not addressed in it: this tool cannot be used to specify an editing workflow, it does not support routing or storage of artifacts, nothing is done concerning monitoring, ... yet these concerns are among the most important to be taken care of by a workflow management infrastructure [10].

3.2 Proposed Architecture

Overall Operations. The architecture that we propose is composed of three tiers: *clients*, a *central server* and *administration tools*. We consider that, each participant in a given workflow has a client. Initially, the workflow owner (comparable to a deposit owner in Git (see footnote 8) [20]) connects to the server from his client. He creates his workflow by specifying all necessary informations (the workflow name, the overall grammar, different participants, their rights and their views, the basic document and the workflow's circuit), then triggers the process. Next, participants concerned by the new created workflow receive an alert message from the system, inviting them to participate. Each participant must therefore connect himself to the server to obtain a partial replica of the workflow model (encoded in a specification file written in a dedicated DSL (Domain Specific Language [1])) and state (his local document model, a partial replica of the initial document, ...) according to his rights and his view on the given workflow. A given participant performs its duties and submits its local (partial) replica to the central server which performs synchronizations as soon as possible and the process continues (see Fig. 1) until the end. For specific needs (authentication, access to corporate data, ...), clients and server may require the intervention of an administration tool (database, paperwork and many others). These three tiers are interconnected around a middleware as presented in Fig. 2.

Server Architecture. The server is responsible for the storage, restoration, execution and monitoring of workflows. Its architecture is based on three basic elements as shown in Fig. 2(a): its *model*, *storage module* and its *runtime engine*.

1. *The model*: it is the one orchestrating all the tasks supported by the server. It consists of a workflow engine, a set of parsers and three communication

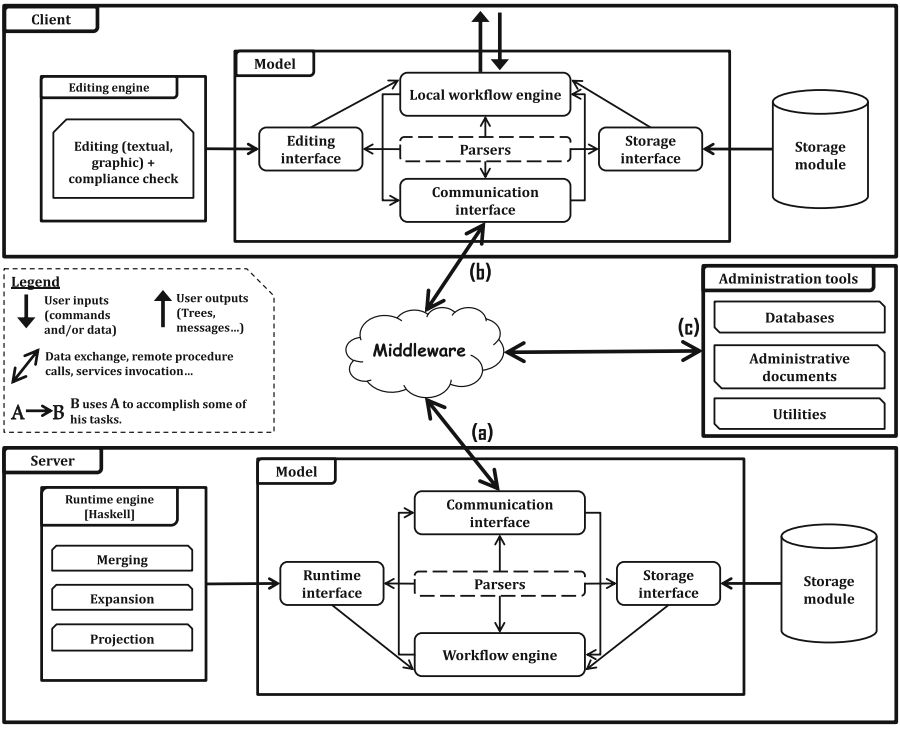


Fig. 2. A software architecture (3-tiers) for centralized management of workflows of cooperative editing of structured documents.

interfaces (the interface with the middleware, that with the storage module and the one with the runtime engine).

2. *The storage module*: it is responsible for the storage of workflows. Like CVS, it maintains a main repository for each workflow. The repository space of a given workflow includes its specification file written in a DSL [1] which is the subject of a work in progress[12]. There are also (global) document versions showing the state of the workflow at given times. These versions of the underlying documents, facilitate the control and monitoring of workflows.
3. *The runtime engine*: it consists of implementations of projection, expansion [5] and consensual merging [13] algorithms. These implementations are used by the workflow engine in the realization of these tasks. A runtime engine written entirely in Haskell, was proposed in [14]. However, it is quite rigid and almost impossible to adapt to the architecture presented here. To this end, we present in Sect. 3.3, a more flexible version of the latter.

[12] Indeed, workflows such as we manipulate, can not be easily specified in their entirety with the help of current notations (BPMN, statescharts [5], Petri Network with Objects [15], ...).

Client Architecture. The client (Fig. 2(b)) is also based on three entities: a *model*, an *editing engine* and a *storage module*. The model is responsible for organizing and controlling the execution of tasks and user commands. For each new local workflow, the model generates an editing environment which is used by the editing engine to provide conventional facilities of structured document editors (compliance check, syntax highlighting, graphical editing of documents presentations, ...). Each workflow is locally represented by a specification file and by one structured document (an update of a partial replication of a global one) representing the current perception of the overall workflow from the current local site. When reaching synchronization phases, the local structured document is forwarded to the server site, where it is merged with others in one structured document representing the current state of the overall workflow: they are therefore, coordination supports between workflow engines of the client and of the server.

The Middleware. The middleware is responsible for the interaction between different tiers of our architecture. It must be designed so that the coupling between these tiers is as weak as possible. One can for this purpose, consider a Service-Oriented Architecture (SOA) in which:

- Our clients are service clients;
- The server is a service provider for clients and a client of services offered by the administration tools;
- The administration tools are service providers.

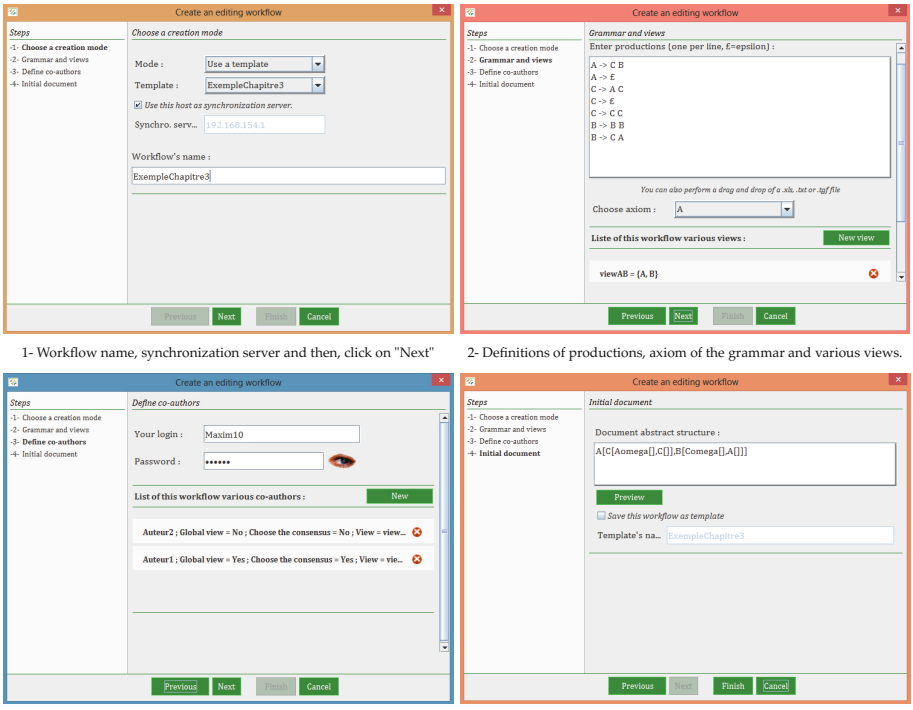
With such an architecture, we can guarantee the independence of each tier and thus, an easier maintenance.

3.3 TinyCE v2

Due to its technical nature and to the number of technologies it needs for its instantiation, the architecture presented in this paper has not yet been fully implemented. However, many of its components have already been implemented and tested in a test project called TinyCE v2[13] (a Tiny Cooperative Editor version 2).

TinyCE v2 is an editor prototype providing graphic and cooperative editing of the abstract structure of structured documents. It is used following a networked client-server model. Its user interface offers to the user, facilities for the creation of workflows (documents, grammars, actors and views (Fig. 3)), editing and validation of partial replicas (Fig. 4). Moreover, this interface also offers to him, functionality to experiment the concepts of projection, expansion and consensual merging (Fig. 5). TinyCE v2 is designed using Java and Haskell languages. It offers several implementations of our architecture concepts namely: parsers, storage modules, server's runtime engine, workflow engines and communication interfaces.

[13] TinyCE v2 is a more advanced version of TinyCE [14].



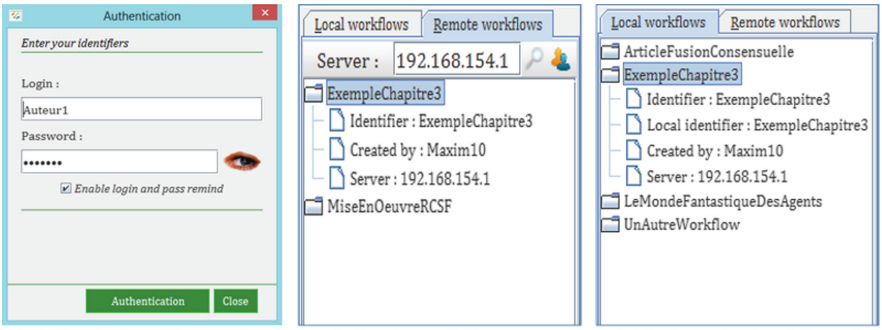
1- Workflow name, synchronization server and then, click on "Next"

2- Definitions of productions, axiom of the grammar and various views.

3- Informations on workflow owner and on different co-authors

4- Creation of the global initial document

Fig. 3. Some screenshots showing process of creation of a workflow of cooperative editing in TinyCE v2.



1- Authentication of the co-author Auteur1

2- Showing available remote workflows for Auteur1

3- Displaying local workflows of Auteur1

Fig. 4. Some screenshots of TinyCE v2 showing the authentication window of a co-author (Auteur1) as well as those displaying the various local and remote workflow in which he is implicated.

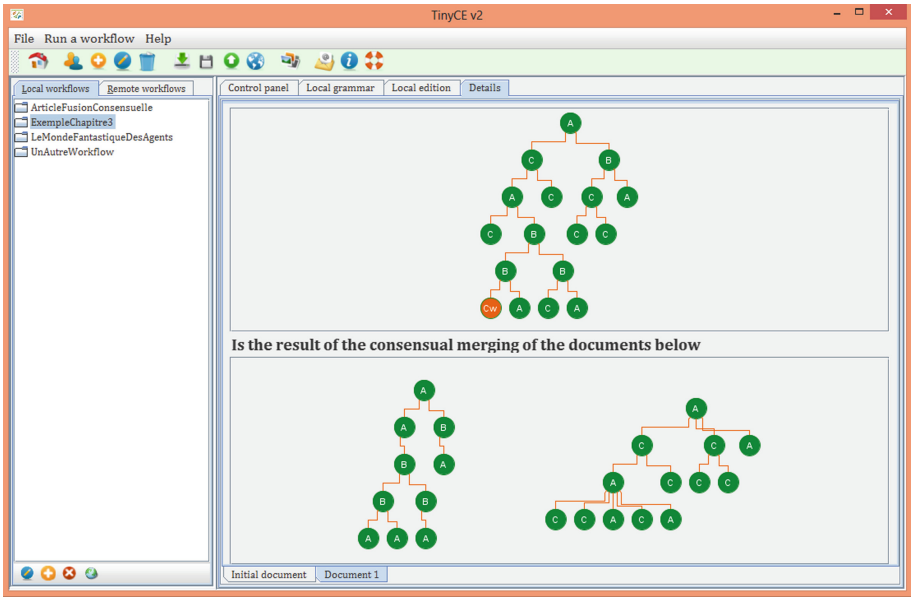


Fig. 5. An illustration of consensual merging in TinyCE v2.

As in [14], the runtime engine of TinyCE v2 exploits the possibility offered by Java, to run an external program. Indeed, we designed an interface of TinyCE v2 (runtime interface) capable of launching a Haskell interpreter (GHCi - Glasgow Haskell Compiler interactive - [7] in this case) and make it execute various commands. When creating a workflow, TinyCE v2 generates a Haskell program file (.hs) [9], containing data types and functions necessary to achieve the operations of projection, expansion and consensual merging on the structured document representing the state of that workflow. In this way, we considerably reduce the use frequency of parsers presented in [14]. The functions are more open to changes as they are contained in a text file and not in a compiled program as in [14]. In fact, the main differences between our Java-Haskell cross-fertilization approach and the one of [14] are almost the same that drive the debates on interpreted and compiled languages; our approach is likened to interpreted languages and that of [14], to compiled languages. So, even though our approach can present security risks (that can be addressed using PKI (Public Key Infrastructure) and standard encryption systems like AES (Advanced Encryption Standard), RSA (Rivest Shamir Adleman) ...), it has the advantage of being portable and easier to maintain.

4 Conclusion

We presented in this paper, a 3-tiers software architecture for the production of a system for centralized management of administrative workflows that can be modelled as cooperative editing of structured documents. This type of workflow has been subject of previous works. The particularity here is that, according to their views, actors only have partial perception of the overall document. Previous works has been focused on the definition of basic concepts, mathematical models and punctual functions of (models and documents) replication and fusion (some of them are written in the lazy functional language named Haskell). Here, we have proposed a modular architecture for such a system by presenting a coherent arrangement of independent modules. Our architecture is flexible because it leaves to the designer the choice of certain design methods and to the developer, the one of the main languages and tools to use. Inspired by the TinyCE system [14], we proposed an implementation of our architecture. In short, the architecture presented here, its different components and their implementation are the fruit of this work.

Some of our subsequent work focus on the design of a DSL for specifying workflows theorized in [5]. An avenue to explore is rather in creating a DSEL (Domain Specific Embedded Language) of BPMN and enrich it by the concepts of grammatical model, view, editing, compliance and validation. We are also investing to complete the implementation of the architecture presented in this paper.

References

1. Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. ACM SIGPLAN Not. **35**(6), 36 (2000)
2. Berliner, B.: CVS II: parallelizing software development. In: Proceedings of the USENIX Winter 1990 Technical Conference, Berkeley, Californie, Etats-Unis, pp. 341–352. USENIX Association (1990)
3. Frye, C.: Move to workflow provokes business process scrutiny. Softw. Mag. **14**(4), 77–85 (1994)
4. Georgakopoulos, D., Hornick, M., Sheth, A.: An overview of workflow management: from process modeling to workflow automation infrastructure. Distrib. Parallel Databases **3**, 119–153 (1995)
5. Badouel, E., Tchoupé, M.: Merging hierarchically structured documents in workflow systems. Electron. Notes Theor. Comput. Sci. **203**(5), 3–24 (2008). Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008), Budapest
6. Oster, G.: Réplication optimiste et cohérence des données dans les environnements collaboratifs répartis. Autre [cs.OH]. Thèse de Doctorat/Ph.D., Université Henri Poincaré - Nancy I (2005)
7. GHC: The Glasgow Haskell Compiler: GHC. http://www.haskell.org/ghc/
8. Giannetti, J., Lord, M.-A.: Une plateforme Web pour soutenir la réécriture collaborative: EtherPad, Première partie. Formation et profession **23**(1), 71–73 (2015). https://doi.org/10.18162/fp.2015.a51

9. Haskell: A purely functional language. http://www.haskell.org
10. Institute of Management Accountants: Implementing automated workflow management. Business Performance Management, IMA Publication Number 00354 (2002). ISBN 0-86641-290-5
11. Wilm, J., Frebel, D.: Real-world challenges to collaborative text creation. ACM, 14 September 2014. ISBN 978-1-4503-2964-4
12. Tchoupé, M.T., Atemkeng, M.T., Djeumen, R.: Un modèle de documents stable par projections pour l'édition coopérative asynchrone. In: CARI 2014 Proceedings, vol. 1, pp. 325–332 (2014)
13. Tchoupé, M.T., Ndadji, M.M.Z.: Réconciliation par consensus des mises à jour des répliques partielles d'un document structuré. In: CARI 2016 Proceedings, vol. 1, pp. 84–96 (2016)
14. Tchoupé, M.T.: Fertilisation croisée d'un langage fonctionnel et d'un langage objet: application à la mise en oeuvre d'un prototype d'éditeur coopératif asynchrone. In: CARI 2010 - Yamoussoukro, pp. 541–549 (2010)
15. Chaâbane, M.A., Bouzguenda, L., Bouaziz, R., Gargouri, F.: Spécification des processus workflows évolutifs versionnés. Schedae, prépublication numéro 11, fascicule numéro 2, pp. 21–29 (2007)
16. Tlili, M.: Infrastructure P2P pour la Réplication et la Réconciliation des Données. Base de données [cs.DB]. Thèse de Doctorat/Ph.D., Université de Nantes (2011)
17. Curcin, V., Ghanem, M.: Scientific workflow systems - can one size fit all? In: Proceedings of the 2008 IEEE, CIBEC 2008 (2008)
18. W3C. extensible markup language (xml), W3C Recommendation 1.0 (second edition), octobre 2000
19. Cunningham, W.: Wikiwikiweb history (2005). http://c2.com/cgi/wiki?WikiHistory
20. Wikipédia: git - Wikipédia. https://fr.wikipedia.org/wiki/git
21. Some of the authors of Subversion: Version Control with Subversion. http://svnbook.red-bean.com/
22. Wikimedia: Wikipedia: the free encyclopedia that anyone can edit (2005)
23. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. **V**(3), 1–44 (2005)