Computational Creativity Infrastructure for Online Software Composition: A Conceptual Blending Use Case

Martin Žnidaršič[1], Amílcar Cardoso[2], Pablo Gervás[4], Pedro Martins[2], Raquel Hervás[4], Ana Oliveira Alves[2], Hugo Gonçalo Oliveira[2], Ping Xiao[3], Simo Linkola[3], Hannu Toivonen[3], Janez Kranjc[1], Nada Lavrač[1]

[1]Jožef Stefan Institute, Ljubljana, Slovenia
[2]CISUC, DEI, University of Coimbra, Coimbra, Portugal
[3]Department of Computer Science and HIIT, University of Helsinki, Finland
[4]Universidad Complutense de Madrid, Spain

Abstract

Computational Creativity is a subfield of Artificial Intelligence research, studying how to engineer software that exhibits behaviors which would reasonably be deemed creative. This paper shows how composition of software solutions in this field can effectively be supported through a Computational Creativity (CC) infrastructure that supports user-friendly development of CC software components and workflows, their sharing, execution and reuse. The infrastructure allows CC researchers to build workflows that can be executed online and be reused by others with a single click on the workflow web address. Moreover, it allows building of procedures composed of software developed by different researchers from different laboratories, leading to novel ways of software composition for computational purposes that were not expected in advance. This capability is illustrated on a workflow that involves blending of texts from different domains, blending of corresponding images, poetry generation from texts as well as construction of narratives. The paper concludes by presenting plans for future work.

Introduction

Computational creativity (CC) systems use as their basic ingredients different types of resources, including musical, pictorial and textual, to name a few. This paper focuses on infrastructure support to CC systems that base their creativity on textual resources. Such CC systems include poetry generation, metaphor creation, generation of narratives, creation of fictional ideas and conceptual blending, which all represent CC tasks which request manipulation of text resources that are provided as inputs.

Infrastructures supporting text-based creative systems are scarce. Ideally, a text-based CC system would automatically build creative artefacts from the given text resources, which the end user would then inspect and potentially adapt to their needs. An attempt in this direction is the FloWr system for automated flowchart construction, optimisation and alteration (Charnley, Colton, and Llano 2014). While getting software to write CC code directly is a long-term research goal, that line of research is—with the exception of FloWr—still in its infancy stage. A substantially more mature area of research concerns the development of infrastructures supporting modular development, sharing and execution of code used in text mining tasks. Text mining has numerous open source algorithms and natural language processing (NLP) software libraries available (such as NLTK (Bird 2006) and scikit-learn (Pedregosa et al. 2011)). However, even text mining and NLP experiments are still difficult to reproduce, including the difficulty of systematic comparison of algorithms. To this end, a number of attempts have been made to develop easy-to-use workflow management systems, allowing users to compose complex processing pipelines in a modular visual programming manner.

Related work As regards the work related to the platform presented in this paper, we first mention myGrid[1] which is used primarily for bioinformatics research, having in mind experiment replication. It is currently probably the most advanced workflow management system, although, due to its complexity, not very easy to use. The most important part of myGrid is Taverna, which is conceived as a suite of tools used to design and execute scientific workflows. A multilingual Internet service platform Language Grid[2], which is based on a service-oriented architecture and supports a web-oriented version of the pipeline architecture typically employed by NLP tools, is open source, but it is quite complex to install and use. The ARGO platform[3] is a more recent development, which enables workflows to have interactive components, where the execution of the workflow pauses to receive input from the user, but ARGO is not open source and does not have sophisticated utilities for cataloguing the available web services or workflows, nor a system of access permissions.

Our recently developed platform ClowdFlows[4] (Kranjc, Podpečan, and Lavrač 2012) is web-based thus requiring no local installation, is simple to use and install, and available as open source under the MIT Licence. While ClowdFlows is mainly devoted to data mining, its fork TextFlows[5] is focused on text mining and NLP workflows. A fork platform for facilitation and reuse of computational creativ-

[1]http://www.mygrid.org.uk/
[2]http://langrid.org/
[3]http://argo.nactem.ac.uk/
[4]http://clowdflows.org
[5]http://textflows.org

ity software is called ConCreTeFlows[6]. It is an independent platform with a specific backend that is being continuously adapted to computational creativity tasks and tools. As forks of ClowdFlows, TextFlows and ConCreTeFlows benefit from its service-oriented architecture, which allows users to utilize web-services as workflow components. The distinguishing feature of these platforms is the ease of sharing and publicizing workflows, together with an ever growing roster of reusable workflow components and entire workflows. As completed workflows, data, and results can be made public by the author, the platform can serve as an easy-to-access integration platform for data mining, text mining or computational creativity processes. Each public workflow is assigned a unique URL that can be accessed by anyone to either replicate the experiment, or use the workflow as a template to design new similar workflows.

Contributions In this paper we present ConCreTeFlows and illustrate its use in a specific use case of conceptual blending (introduction to blending theory is provided on page 3). This example employs multiple software components that are being developed by various members of the computational creativity community. The presented composition of software aims to conduct conceptual blending conceptually, textually and visually. Given two descriptions of arbitrary concepts in natural language, the presented approach provides conceptual graph representations of both concepts and their blend, a textual description of the blended concept and even a set of possible visual blends.

The paper is structured as follows. The first section presents ConCreTeFlows as a special purpose workflow management platform aimed at supporting computational creativity tasks. In the next section is the core of this paper. It provides a description of the use case and the basics of its theoretical foundations, followed by presentation of all the important methods and software components that are applied for its purpose. Last part of this section is devoted to critical discussion and ongoing work on the presented components. The paper concludes with a brief summary and plans for further work.

Software Infrastructure

This section briefly describes the main components of the ConCreTeFlows. It is a special purpose workflow management platform, aimed at supporting (primarily text-based) computational creativity tasks.

Like ClowdFlows, ConCreTeFlows can also be used in a browser, while the processing is performed in a cloud of computing nodes. The backend of ConCreTeFlows uses Django[7], which is an open source web framework. The graphical user interface is implemented in HTML and JavaScript, using jQuery[8] and jQuery-UI[9] libraries. ConCreTeFlows is easily extensible by adding new packages and workflow components. Workflow components of several types allow graphical user interaction during run-time, and visualization of results by implementing views in any format that can be rendered in a web browser. Below we explain the concept of workflows in more detail and describe the basic concepts of ConCreTeFlows.

The workflow model is the main component of the ConCreTeFlows platform and consists of an abstract representation of workflows and workflow components. The graphical user interface for constructing workflows follows a visual programming paradigm which simplifies the representation of complex procedures into a spatial arrangement of building blocks. The basic unit component in a ConCreTeFlows workflow is a processing component, which is graphically represented as a widget. Considering its inputs and parameters every such component performs a task and stores the results on its outputs. Different processing components are linked via connections through which data is transferred from a widget's output to another's input. An alternative widget input for a widget are parameters, which the user enters into widget's text fields. The graphical user interface implements an easy-to-use way of arranging widgets on a canvas to form a graphical representation of a complex procedure. Construction of new workflows thus requires no expertise, apart from knowing (usually from widget documentation) the inputs and outputs of the widgets to ensure their compatibility. Incorporation of new software components, on the other hand, requires basic programming skills in Python or SOAP web-service development in any programming language.

ConCreTeFlows implements its own workflow execution engine. Currently there are no ways to reuse the workflows using third party software. We plan to implement special widgets that will define inputs and outputs for REST API endpoints which will allow execution of workflows on variable inputs by any third party software.

The ConCreTeFlows graphical user interface is shown in Figure 1. On the top of the graphical user interface is a toolbar where workflows can be saved, deleted, and executed. Underneath on the left is the widget repository, which is a list of available widgets grouped by their functionality. Click on a widget in the repository adds it to the workflow construction canvas on the right. A console for displaying success and error messages is located on the bottom.

Workflows in ConCreTeFlows are processed and stored on remote servers from where they can be accessed from anywhere, requiring only an internet connection. By default each workflow can only be accessed by its author, although one may also chose to make it publicly available. ConCreTeFlows generates a specific URL for each workflow that has been saved as public. The users can then simply share their workflows by publishing the URL. Whenever a public workflow is accessed by another user, a copy of the workflow is created on the fly and added to his private workflow repository. The workflow is copied together with widgets' parameter settings, as well as all the data, in order to ensure the experiments can be repeated. In this way the user is able to tailor the workflow to his needs without modifying the original workflow.

[6] http://concreteflows.ijs.si
[7] https://www.djangoproject.com
[8] http://jquery.com
[9] http://jqueryui.com

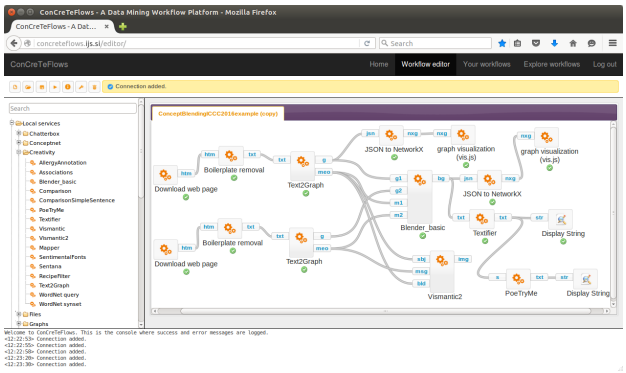


Figure 1: A screenshot of the ConCreTeFlows graphical user interface opened in the Mozilla Firefox Web browser, presenting a motivational CC use case.

Conceptual Blending Online

The elements of the conceptual blending (CB) theory (Fauconnier and Turner 2002) are an inspiration to many algorithms and methodologies in the field of computational creativity (Veale and O'Donoghue 2000; Pereira 2005; Thagard and Stewart 2010; Schorlemmer et al. 2014). A key element in the theory is the *mental space*, a partial and temporary structure of knowledge built for the purpose of local understanding and action (Fauconnier 1994). To describe the CB process, the theory makes use of a network of four mental spaces (Figure 2). Two of these correspond to the *input spaces*, i.e., the content that will be blended. The process starts by finding a partial *mapping* between elements of these two spaces that are perceived as similar or analogous in some respect. A third mental space, called *generic*, encapsulates the conceptual structure shared by the input spaces, generalising and possibly enriching them. This space provides guidance to the next step of the process, where elements from each of the input spaces are *selectively projected* into a new mental space, called the *blend space*. Further stages of the process elaborate and complete the blend.

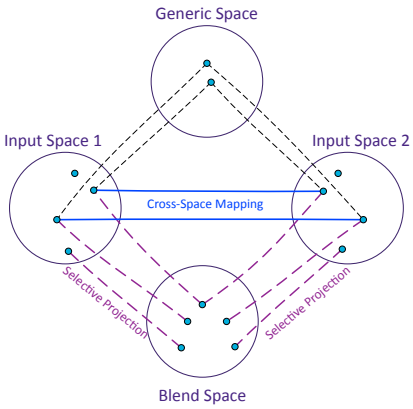


Figure 2: The original four-space conceptual blending network (Fauconnier and Turner 2002).

In most computational approaches to CB, the input and blended spaces are represented as computational versions of *Conceptual Maps* (Novak 1998), i.e., graphs where nodes are concepts and arcs are relations between them (see Figure 3).

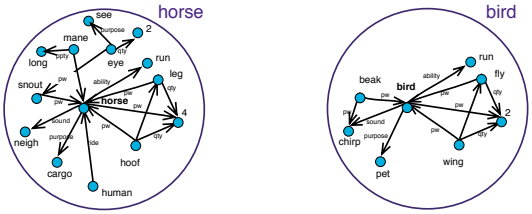


Figure 3: Concept maps of horse and bird.

Graph representations of concept blends are useful for automated analysis and further processing, but are not very suitable and appealing for human perception of the blended spaces. To improve on this aspect of conceptual blending, we have developed methodologies and algorithms for visual blending and for textual representation of concept graphs. Using these new techniques, we designed a CB process that results in conceptual blends that are described in natural language and enriched with visual representations. The process is sketched in Figure 4, where the boxes represent the main (software) components and the arrows indicate the flow of data from inputs to outputs.

Each of the main process components (that is, each box from the sketch in Figure 4) is implemented as an independent software solution and represented as a widget or a group of widgets in ConCreTeFlows.

In the following, we describe the process components and their implementations in detail, with a presentation of the whole workflow and some exemplary results at the end.

Construction of Conceptual Networks

The *Concept Network Builder* component from Figure 4 accepts a textual description of a concept in natural language and on its basis produces a conceptual graph.The set of possible concepts and relations in the resulting graph is open and not limited to a particular fixed set (such as relations in ConceptNet[10]) or linguistic characteristic. We decided to represent also relations as concepts, which allows treating a particular entity as both a concept or relation, depending on the context. For example, the concept of *eating* can be used to relate the concepts of *cows* and *grass*, but it can also be a concept related through *is a* with *animal activity*.

The software component that creates these conceptual graphs from text is implemented in ConCreTeFlows as the Text2Graph widget. This component first uses the Ollie triplet extractor (Mausam et al. 2012) to extract the triplets from the given text. The only text transformation before triplet extraction is uncapitalization of sentences. The resulting triplets are used to create a graph. In this process, the entities in the triplets can be lemmatized (this choice is

[10] http://conceptnet5.media.mit.edu/

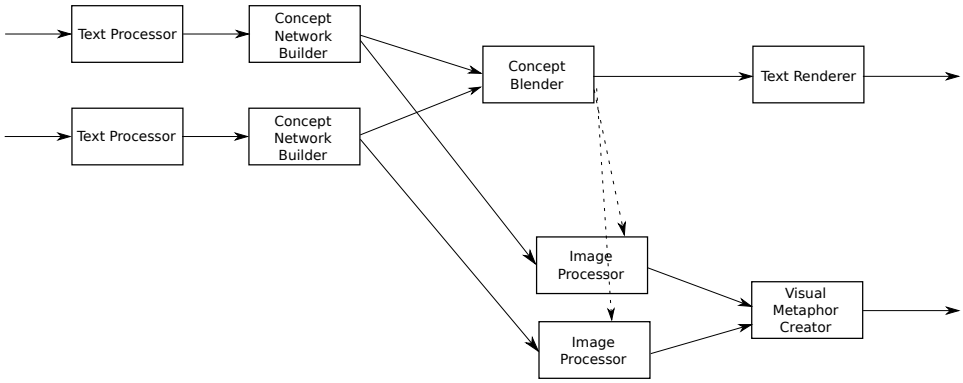


Figure 4: Sketch of the workflow for conceptual blending with visual and textual representations of blends.

left to the user). If the *Main size* parameter of the resulting graph is set, the graph is filtered to contain only a limited neighborhood around the *Main entity*, that is, the node in the graph that a user might be most interested in. The *Main entity* can be set by the user, but if it is not, it is selected automatically as the graph node with the highest out-degree.

Conceptual Blending

The *Concept Blender* component takes care of blending two elements that are represented as graphs. In our design of the process (Figure 4), the mapping between the elements from the input spaces is to be done by a human as an initial step (to select the two inputs to take part in blending) or to be done in the *Concept Blender* on all possible pairs of elements from the two input spaces. Following our representation, these would be pairs of conceptual graphs.

In the current baseline implementation, the *Concept Blender* expects only one pair of input elements, which it fully blends (merges the two conceptual graphs) without any influence of the *Generic Space*. An elaborate concept blending component, that is based on Divago framework, is in development as described in subsection on further work.

Visual Blending

In order to generate visual blends, the visual module, consisting of the *Image Processor* and the *Visual Metaphor Creator* in Figure 4, takes inputs from either the Concept Network Builder or the Concept Blender. From the first, it can take two concepts from two concept graphs (in this version, their main entities) as inputs to be visually blended. The resulting visual blend is not a representation of a blend created by Concept Blender, but an independent visual blend of the two concepts. For this purpose the visual module first finds photos tagged with the two concepts in Flickr, respectively. For ensuring the relevance and quality of the photos, we use a set of image analysis methods bundled together in QualiPy[11]. The image processor separates the subject and the background of each photo, and inpaints the background to hide any marks of the subject. The visual metaphor creator implements three visual operations: juxtaposition, replacement and fusion, as described by Xiao and Linkola (2015). In effect, it puts one object in the context of another, or gives an object the texture of another object (see Figure 5 for an example).

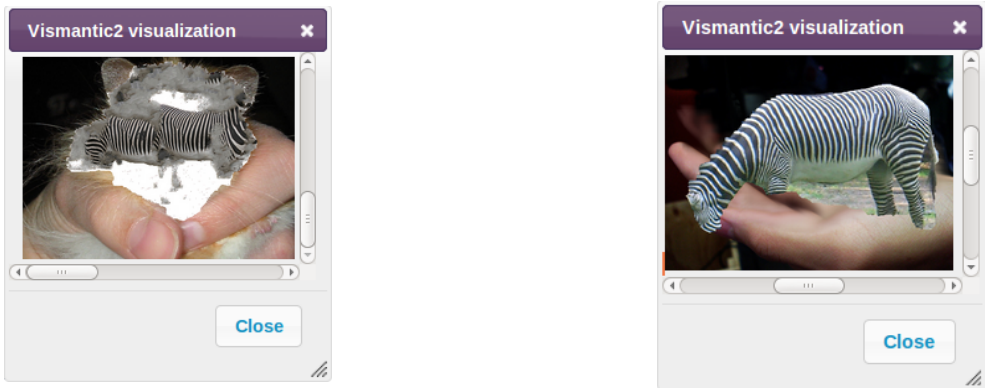
The visual module can also take input from the concept blender, which indicates a specific way of blending. Specifically, the input may indicate a choice between the replacement and fusion operations. For instance, a frequent conceptual blend is placing an object in an unusual environment, which suggests that the replacement operator shall be used.

In ConCreTeFlows, such blending is available in two versions (generations) as Vismantic and Vismantic2 widgets.

Text Generation

In order to generate a textual description of the blends obtained, a *Text Renderer* widget called Textifier has been added to the workflow. Textifier is a natural language generation tool that transforms data represented in a graph into a natural language text. It carries out stages of content determination, document planning and surface realization (Reiter and Dale 2000) and then translates the result into plain text. Content determination processes input to select and adapt what might be rendered. The input graph must contain pairs of source and target nodes with information, and the system will create all possible paths and represent them in a tree. Textifier first groups related information that refers to the same concept by combining nodes that contain the same subject and discarding duplicated nodes. Nodes that represent information with granularity inappropriate for textual rendering – such as verb-preposition groups represented as single strings – are rewritten to make all information explicit in the knowledge structure. Lastly, Textifier can prune the tree if only branches of a certain length need to be considered. Currently we are working with branches that are three nodes long, after detecting that they tend to contain more promising information. Document planning is basic at present but will play a larger role once the graphs of blends are processed. The surface realization stage transforms the tree into text. Figure 6 shows an example of Textifier in operation over a graph constructed from a given input text.

[11] https://github.com/vismantic-ohtuprojekti/qualipy



(a) (b)

Figure 5: Two (out of four combinations of exchanging context and texture) outputs of the Vismantic2 widget for the example of blending the concepts of *hamster* and *zebra*. Figure 5a shows result of exchanging texture: hamster with a zebra's texture. In Figure 5b is an example of exchanging context: zebra is put in the usual visual context of a hamster.

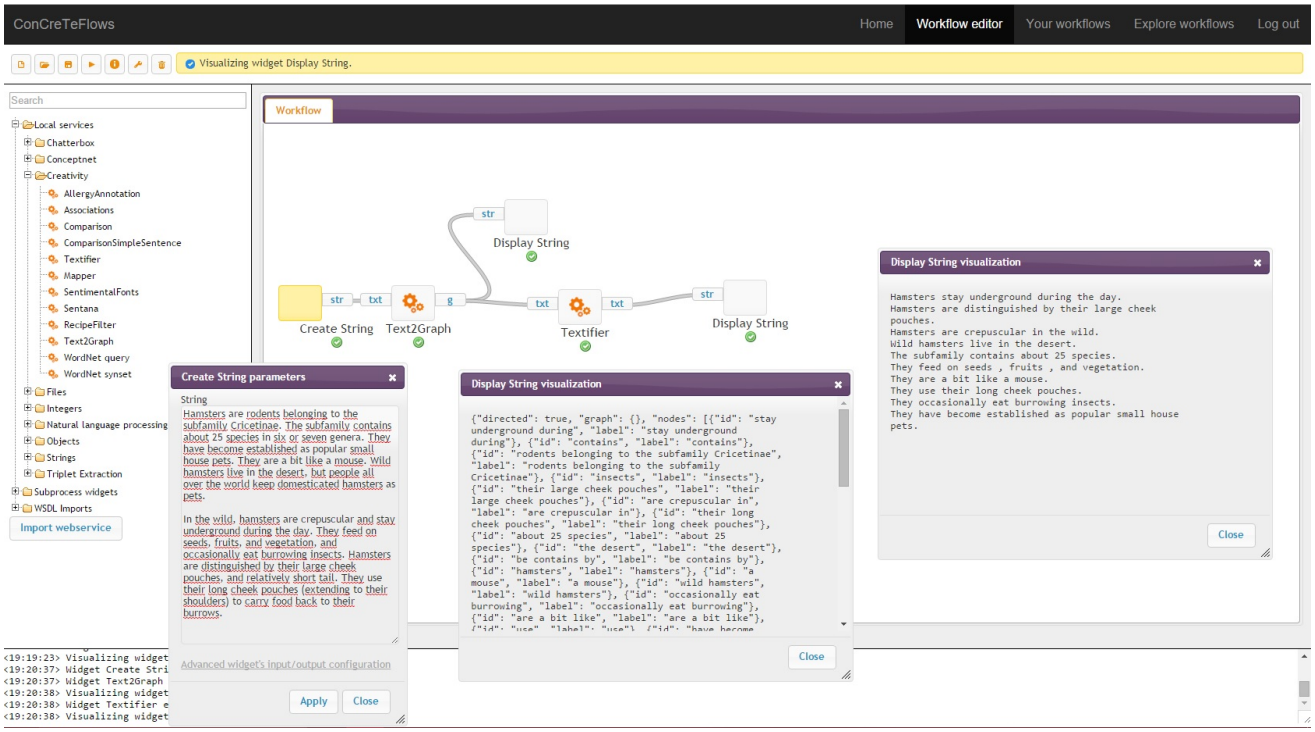


Figure 6: Example of Textifier working on input graph obtained from text.

Integration in a Workflow

The software components that implement the functionalities sketched in Figure 4 were implemented and integrated in ConCreTeFlows either as internal (Python) functions, wrapped standalone programs or as Web services. In addition, we implemented some additional components that support the user interaction and data processing. These are: (I) components for Web page content retrieval and filtering and (II) components for graph reformatting and visualization.

By connecting these software components, we composed a ConCreTeFlows workflow that conducts a basic conceptual, textual and visual concept blending. The workflow is presented in Figure 7 and is publicly available from:
`http://concreteflows.ijs.si/workflow/137/`
where it can be executed, changed and appended with additional functionality.

In this workflow, two textual inputs are transformed into conceptual graphs by a series of the Download web page, Boilerplate removal and Text2Graph widgets. The first one

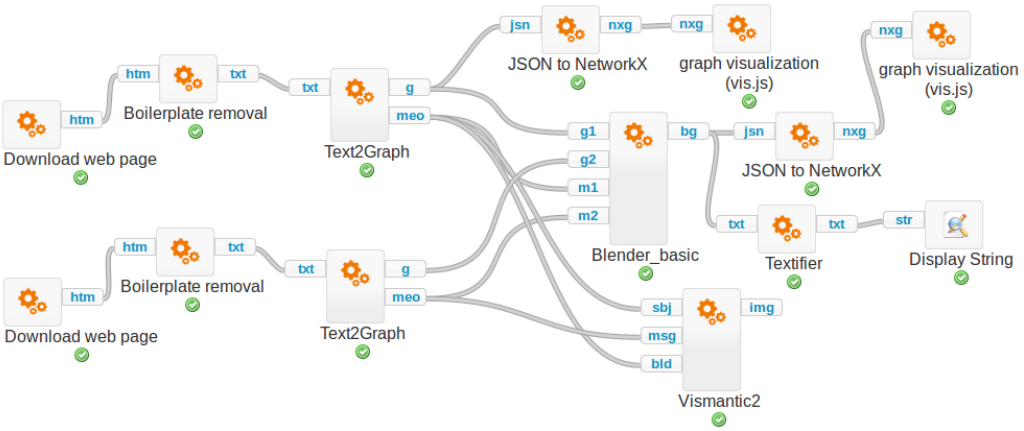


Figure 7: Workflow implementation in ConCreTeFlows (available at: `http://concreteflows.ijs.si/workflow/137/`).

obtains the Web page source from a given URL. In the example presented in this paper, these are the Wikipedia pages for two animals: hamster and zebra. The second widget removes the headers, menus, navigation and similar non-relevant content from the source. Finally, Text2Graph transforms the textual content into conceptual graphs (output *g*), which are available to other widgets with separately provided main entity (output *meo*). In the workflow, one of the graphs is reformatted and visualized with the graph visualization widget. All outputs of Text2Graph widgets enter the Blender_basic which blends the two graphs together and outputs a combined blended graph (output *bg*). This one gets served to the Textifier widget, which produces a textual description of the blend. Its output is presented by a standard Display String widget. The two main entities from Text2Graph widgets enter also the Vismantic2, which either changes the texture of one to the texture of the other (see Fig. 5a), or puts one in the usual surroundings of the other (Fig. 5b). This way it creates four candidates for visual blends. This widget takes somewhat longer to run, as it is in fact a call to a computationally intensive Web service. Upon completion, the outcome is shown in an output similar to the ones shown in Figure 5.

Workflow dissemination, reuse and extension

Any ConCreTeFlows workflow can either remain private or be made public for the purposes of dissemination and reproducibility of work. The workflow from the previous subsection is available from a public URL. This means that anyone can open it in ConcreTeFlows. Everytime this happens, a dedicated copy of the original workflow is made for that particular user. This allows any user not only to run the workflow with its original data and parameters, but also to change the inputs, parameters and redesign the structure of software components without affecting the original workflow.

Changing and extending a workflow is easy, but it requires some insight on the format of data that is exchanged among the widgets. This is usually made available in widget documentation, but can also be seen by observing the raw results of a widget (right-click and *Results*).

In the following we describe a simple exemplary extension of our workflow from Figure 7.

Exemplary addition: PoeTryMe widget The system named PoeTryMe (Gonçalo Oliveira and Cardoso 2015) is a poetry generation platform with a modular architecture that may be used to produce poetry in a given form, based on a set of seed words. Semantically-coherent lines are generated using the seeds or related words, and are produced by exploiting the knowledge in a semantic network and a grammar with textual renderings of the covered relations. A generation strategy selects some of the produced lines and organises them to suit the given form.

The PoeTryMe widget is limited to some of the features of the full system. Nevertheless, it can produce poetry in three languages (Portuguese, Spanish and English), given one of the available target forms (block of four, sonnet, ...), an open set of seeds, and a surprise factor, between 0 and 1, with implications on the selection of more or less semantically-distant words.

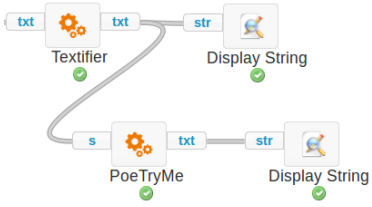


Figure 8: Addition of the PoeTryMe widget to the workflow.

In our workflow, the PoeTryMe widget can be appended to the Textifier widget (as shown in Figure 8) in order to get also a poem inspired by the resulting blend. Here is an example of a poem from a blend of *hamster* and *zebra*:

when the coat paints the water white and black
stadiums here make song each stand has his
have not yet grown by the familiar crack
will mine and leave where the great love is

Discussion and Future Work on Components

In the following, we discuss some of the encountered issues and shortcomings of the components and processes that are presented in this paper, as well as present some ongoing work on their improvements and additions.

Graph representations and formats The use of graphs for representing knowledge presents advantages in as much as it is a simple format with significant expressive power. In this sense it acts as a useful communication format for the various components in the flows envisaged. However, it has certain disadvantages in the sense that the graphs as considered at present do not have a unique semantic interpretation. Some of the modules produce graphs where relations are represented as edges between nodes representing objects, and others rely on graphs that represent relations as nodes occurring in the path of the graph between nodes representing objects. Even when the same approach to knowledge representation in a graph is used, problems may arise depending on the type of string used to label the nodes. Examples of problematic cases are: inflected verb forms used as well as verbs in infinitive, nouns used in singular and/or plural form, complex actions of the form `stay_at_home`... At present the content determination stage of the Textifier module is carrying out complex transformations to handle these various inputs in a uniform fashion when it comes to the final rendering. This requires the development of different version of the content determination stage for receiving input from different modules. It would be beneficial to make progress towards a unified approach to graph representation to allow blending operations to be carried out fruitfully between outputs generated by different modules. However, a certain flexibility is desirable in these content determination modules, so that they can tolerate inputs not altogether conforming to expectations. This is largely due to the open nature of the ConCreTeFlows platform, which may see the addition of new modules that do not conform to any standards set on graph representation, but also because the results of conceptual blending operations may not always produce output conforming to standards, even when the inputs to the conceptual blending process do conform.

TextStorm Conceptual Maps TextStorm (Oliveira, Pereira, and Cardoso 2001) is an NLP tool based on a Definite Clause Grammar (DCG) to extract binary predicates[12] from a text file using syntactic and discourse knowledge, not needing any preview knowledge about the discussed domain. The resulting set of predicates constitute a Conceptual Map. This tool can be used as an alternative to the Text2Graph.

TextStorm receives text as initial base of the open information extraction. After applying Part-of-Speech tagging and querying WordNet (Miller 1995), it builds predicates that map relations between two concepts from parsing of sentences. Its goal is to extract from utterances like Cows, as well as rabbits, eat only vegetables, while humans eat also meat, the predicates eat(cow, vegetables), eat(rabbit, vegetables), eat(human, vegetables), eat(human, meat) which will form its concept map. Since concepts in text are not named every time the same way, TextStorm uses WordNet's synonymy semantic relationship to identify the concepts that were already referred before with a different name.

Textstorm operates as a standards compliant SOAP Web service and as such can be imported on-the-fly to ConCreTeFlows (see Figure 9).

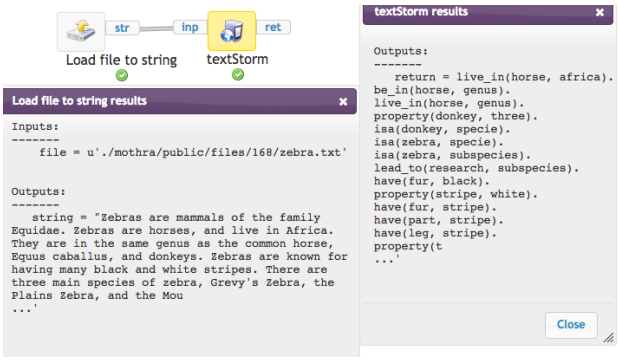


Figure 9: Imported Textstorm SOAP Web service used in ConCreTeFlows.

Divago concept blender The concept blending method that is currently used in the workflow from Figure 7 is very basic.We are currently working on the adaptation of a more elaborated blender, the pre-existing Divago (Pereira 2005), to offer its main functionalities as webservices in ConCreTeFlows. This blender adopts the same graph format as TextStorm, i.e., the Conceptual Map format, for the input and blended mental spaces.

The new blender, the *DivagoFlow*, is itself a flowchart composed of two modules, the *Mapper* and the *Blend Factory* The first is responsible for finding analogy mappings between two Input Spaces using structural alignment. More precisely, it computes the largest isomorphic pair of subgraphs contained in the Input Spaces. The output mapping is, for each pair of sub-graphs, the list of crossover relations between nodes of each of the input spaces. The *Blend Factory* takes these mappings as input, as well as the Input Spaces and a Generic Space. For each mapping, it performs a selective projection into the Blend Space, which leads to the construction of a *Blendoid*, an intermediate graph that subsumes the set of all possible blends. This Blendoid feeds an evolutionary process that explores the space of all possible combinations of projections of the Input Spaces taking into account the Generic Space. This module uses an implementation of the CB theory optimality principles (a set of principles that ensure a coherent and integrated blend) as fitness measure. When an adequate solution is found or a pre-defined number of iterations is attained, the Blend Factory stops the execution and returns the best Blend.

[12]These predicates have the common Prolog form: functor(Argument 1, Argument 2).

Conclusions

We have presented the ConCreTeFlows platform for online composition of computational creativity solutions. It is entirely Web based and does not require installation for its use. New processes in the platform can be designed as workflows of software components, which are either made available in the platform or even imported on-the-fly in case of SOAP Web services[13]. Workflows can be either private or shared, which makes for an elegant solution to dissemination and reuse of one's work and repeatability of experiments.

The main focus of the paper is on a use case, which shows how the platform can be used in practice and presents several computational creativity software components that were combined in a collaborative effort to implement an interesting conceptual blending solution. Namely, the resulting blends are not only conceptual but also visual and textual. The benefits of a unifying workflow for blending are twofold: a user can get blends of various kinds through the same user-interface and the components can affect one another to produce a more coherent and orchestrated set of multimodal blending results. While some of the presented components are currently being updated from implementing basic to more elaborate methods, the presented prototype solution is fully operational and serves as a proof of concept that such an approach to multimodal conceptual blending is possible. Potential for use of such an approach is for example in creation of news stories. Such a tool could form an entire automated article on a funny and humorous or a serious and thought-provoking blend of topics. All the components of an article are there: the text, the picture, as shown, one could even add a poem. Other potential uses of the approach could be in art, advertising and human creativity support.

To make these things possible, as described in section *Future Work on Components*, our future work will include improvement of the components and the workflow presented in this paper. We will also continue with development and improvement of the presented platform to make creation of this and other computational creativity solutions further more efficient, collaborative and fun.

Acknowledgments

This research was partly funded by the Slovene Research Agency and supported through EC funding for the project ConCreTe (grant number 611733) that acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission.

References

Bird, S. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, 69–72. Association for Computational Linguistics.

Charnley, J.; Colton, S.; and Llano, M. T. 2014. The FloWr Framework: Automated Flowchart Construction, Optimisation and Alteration for Creative Systems. In *Fifth International Conference on Computational Creativity (ICCC-2014)*, 315–323.

Fauconnier, G., and Turner, M. 2002. *The way we think: Conceptual blending and the mind's hidden complexities*. Basic Books.

Fauconnier, G. 1994. *Mental Spaces: Aspects of Meaning Construction in Natural Language*. New York: Cambridge University Press.

Gonçalo Oliveira, H., and Cardoso, A. 2015. Poetry generation with PoeTryMe. In Besold, T. R.; Schorlemmer, M.; and Smaill, A., eds., *Computational Creativity Research: Towards Creative Machines*, Atlantis Thinking Machines. Atlantis-Springer. chapter 12, 243–266.

Kranjc, J.; Podpečan, V.; and Lavrač, N. 2012. Clowdflows: A cloud based scientific workflow platform. In Flach, P. A.; Bie, T. D.; and Cristianini, N., eds., *ECML/PKDD (2)*, volume 7524 of *Lecture Notes in Computer Science*, 816–819. Springer.

Mausam; Schmitz, M.; Bart, R.; Soderland, S.; and Etzioni, O. 2012. Open language learning for information extraction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CONLL)*.

Miller, G. A. 1995. Wordnet: A lexical database for english. *Commun. ACM* 38(11):39–41.

Novak, J. 1998. *Learning, Creating, and Using Knowledge: Concept Maps as Facilitative Tools in Schools and Corporations*. Mahwah, NJ: Erlbaum.

Oliveira, A.; Pereira, F. C.; and Cardoso, A. 2001. Automatic reading and learning from text. In *Proceedings of the International Symposium on Artificial Intelligence*, 69–72.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research* 12:2825–2830.

Pereira, F. C. 2005. *Creativity and AI: A Conceptual Blending approach*. Ph.D. Dissertation, Dept. Engenharia Informática da FCTUC, Universidade de Coimbra, Portugal.

Reiter, E., and Dale, R. 2000. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.

Schorlemmer, M.; Smaill, A.; Kühnberger, K.-U.; Kutz, O.; Colton, S.; Cambouropoulos, E.; and Pease, A. 2014. COINVENT: Towards a computational concept invention theory. In *Proceedings of the 5th Int. Conference on Computational Creativity, ICCC-14, Ljubljana, Slovenia*, 288–296.

Thagard, P., and Stewart, T. C. 2010. The AHA! experience: Creativity through emergent binding in neural networks. *Cognitive Science* 35(1):1–33.

Veale, T., and O'Donoghue, D. 2000. Computation and blending. *Cognitive Linguistics* Special Issue on Conceptual Blending:253–282.

Xiao, P., and Linkola, S. 2015. Vismantic: Meaning-making with images. In *Proceedings of the Sixth International Conference on Computational Creativity*, ICCC2015, 158–165.

[13] REST services can currently only be wrapped into native widgets.