Computational Intelligence

AN INTERNATIONAL JOURNAL

Computational Intelligence, Volume 33, Number 1, 2017

DECONSTRUCTING COMPUTER POETS: MAKING SELECTED PROCESSES AVAILABLE AS SERVICES

PABLO GERVÁS

Universidad Complutense de Madrid, Madrid, Spain

The article explores the potential of redefining as services a number of functionalities involved in poetry generation systems to better serve the challenge of working toward a yet unknown successful computational model of the creative task being addressed. This is performed by considering how some of the processes currently modeled in existing systems for poetry generation might be deconstructed into a set of services susceptible of being recombined in different ways to be integrated in other developments. Building on prior attempts to propose an evolutionary architecture that allows integration of a number of artificial intelligence technologies in combination, this article explores the advantages of service-oriented architecture for reimplementing as publicly available services some selected functionality of the Wishful Automatic Spanish Poet (WASP) poetry generation system.

Received 30 August 2013; Revised 17 March 2015; Accepted 6 April 2015

Key words: computational creativity, poetry generation, articulation of generative systems, service-oriented architecture.

1. INTRODUCTION

The development of computer software that exhibits creative behavior is faced with a number of challenges. Out of these, two stand out as particularly relevant to the development task. First, the development of creative software usually involves modeling cognitive functions that are very complex in themselves, such as linguistic performance, emotional perception, or analogy. This implies that complex software models of these functions may need to be developed and combined together with domain-specific solutions for particular creative tasks. Second, such development efforts mostly explore uncharted territories of human capability, for which no agreed models exist. This forces the development into the realm of tentative exploration, where solutions for particular problems need to be hypothesized, tested for conformance with desired behavior, and either adopted or rejected on the basis of this testing. Both of these challenges constitute significant barriers to researchers considering the possibility of working in the field. A body of computational models of human cognitive functions or empirical implementations of equivalent behavior is progressively being built by the concerted effort of the scientific community, and this may lower the perceived barrier. Yet even if such computational models of particular functions were available as reusable resources, the task of computational creativity research still faces the challenge that many different combinations will have to be considered before a satisfactory model of particular creative tasks is achieved. However, for this to occur, such models would need to be available in a form that is easy to access and available at an acceptable level of abstraction, so that scientists willing to use them need not become experts in their low-level detail before they can apply them. Such a form should allow a reasonably detailed description of the functionality to enable other users to identify easily what is being offered, and whether it suits their needs. Additionally, the desired form should also allow easy composition with other software modules, to facilitate exploratory construction.

Address correspondence to Pablo Gervás, Universidad Complutense de Madrid, 28040 Madrid, Spain; e-mail: pgervas@sip.ucm.es

This article explores this possibility by considering how some of the processes currently modeled in an existing creative system, the WASP poetry generation system, might be deconstructed into a set of services susceptible of being reused as operational modules of alternative solutions to the poetry generation task. To this end, some reflection is required on how the designers of a generative system approach the task of breaking down the overall tasks into smaller elements that can play a significant role in the broader context of an encompassing system.

The designer of a generative procedure for any particular artifact needs to define some way of understanding the desired type of artifact in terms of properties it must satisfy, a structure it must follow, or ingredients that may be used in its construction (Maeda 2001). When the artifact is for a particular purpose, the designer may, during this process, focus on particular elements that are more relevant to this purpose and pay less attention to other elements. As a result of these choices, the complexity and the versatility of the resulting generative procedure are affected. This is a well-known problem in natural language generation, where system designers have a broad range of options, from reusing canned text if there is just a small set of messages to be conveyed repeatedly, relying on templates for message structure to be filled in with appropriate terms in each instance, or devising a more elaborate characterization of the subset of language to be generated if better coverage and fluency are desired (Reiter and Dale 2000). This initial analysis of the target artifact with a view to selecting a particular frame for understanding and decomposing it into parts that can later be used to assemble equivalent instantiations of the same type has been called *articulation* (Gervás 2013a). This captures the concept of different parts being joined together in a whole but also covers the concept of allowing the parts to move with respect to one another, and even the concept of appropriately conveying a desired meaning.

In the case of poetry generation, the problem of articulation is compounded by the importance attributed to the form of the resulting text, added on top of the underlying complexity of language. This opens up two possible approaches to defining the understanding of poetry: from the point of view of language (grammar, vocabulary, and semantics) and from the point of view of poetic form (stanzas and verses). Depending on the degree of articulation of the generation procedure, some systems limit themselves to selecting a particular textual template with which the poems are produced, starting from a limited vocabulary, reusing a predetermined set of sentences or verses. The different systems for poetry generation described in Section 2.3 apply different levels of articulation.

The degree of articulation captures the idea of how different levels of representation may be used for content and form in each case. Content can be considered simply at the level of texts (different texts have different content) or at an additional semantic level (a semantic representation is used for meaning of a given text, which allows different texts to have the same meaning). Form has historically been considered at many different levels: as metric restrictions on the output (stress patterns and length in syllables for verses and number and length of verses for stanzas), as poem templates to be filled (Oulipo 1981; Colton et al. 2012), as sets of verses to use (Queneau 1961), as sets of lexical items to use (Gervás 2001), as a language model to follow (obtained from a reference corpus) (Kurzweil 2001; Gervás 2013a, 2013b), and as a semantic grammar (Veale 2013a). It is clear that many of these ways of restricting form carry an associated set of restrictions on content. Additionally, some of the reviewed examples consider specific ways of restricting content: as an input sentence (Gervás 2001), a specification of the target semantics in first-order predicate logic (Manurung 1999, 2003), as a word association network (Toivanen et al. 2012), as a mood established from the newspaper of the day (Colton et al. 2012), and as a set of semantic resources (Veale 2013a).

Articulation is also relevant at a higher level, if understood as the mechanism behind the deconstruction that has to be applied to an existing process to reimplement it as a combination of smaller modules, susceptible, for instance, of being deployed as services. In a way, the deconstruction of the computational model of an existing process into submodules that can be recombined to reconstruct the whole is itself a process of articulation in which the artifact that is being decomposed into pieces is a generating procedure itself. In the particular case of poetry generation, many different combinations will have to be considered before a satisfactory model of the poetry generation task is achieved.

There are therefore two axes of articulation when poetry generation systems are considered: one to describe how each system decomposes existing poems into basic units that can be recombined to build new poems and one to describe how the system itself is decomposed into modules that collaborate in the production of the resulting poems. In the following section, these double axes of articulation are considered for a number of poetry generation systems.

2. PREVIOUS WORK

This section presents an abstract high-level description of the type of services that are being considered as targets of the modularization process, outlines some useful references in terms of computational creativity that provide a basic vocabulary to discuss the phenomena under study, and reviews a number of automated poetry generators from which the techniques to build the system in the article are drawn.

2.1. Services as a Unit of Modularization for Software Systems

Services have become a popular means of organizing complex software systems (Ding and Sølvberg 2004; Duan et al. 2005; Mahajan 2006; Dietrich et al. 2007; Dai et al. 2007). This type of modularization is based on the concept of a service-oriented architecture. A *service-oriented architecture* (SOA) is a way of reorganizing a portfolio of siloed software applications and support infrastructure into an interconnected set of services, each accessible through standard interfaces and messaging protocols (Papazoglou 2003). This definition involves several aspects that need to be considered separately. First, it addresses the need to break away from information silos, understood as insular systems incapable of reciprocal operation with other, related information systems (Ensor 1988). A SOA is designed to make (selected) functionality that was previously only accessible internally to a given system available beyond its borders. Second, it involves a process of reorganizing the implementation of the existing functionality in new terms better suited to the new purposes. Third, the conceptual unit employed for this reorganization is that of a service, which implies a number of constraints that will be described in the succeeding text. Fourth, specific interfaces and protocols have to be defined explicitly for this type of organization to succeed.

In more general terms, a SOA can be understood as an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing.

Within this view, a service is an atomic unit of modularization that provides a functionality, which is formally described and publicly available on the Web. Services have been defined as self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications (Papazoglou 2003), although there is also some debate as to whether an acceptable definition can be established (Jones 2005).

For the purpose of this article, only a broad characterization is required. Services can be combined to build more elaborate functionalities, based on a number of basic properties:

- Services share standardized contracts: These contracts may need to specify all the interfaces that a client of this service expects as well as the service interfaces that must be provided by the environment into which the service is assembled/composed (Papazoglou 2003).
- Services are loosely coupled: The service contract should not refer to particular characteristics of the service consumers or of the underlying service logic and implementation.
- Nonessential information is abstracted: Users of the service need not know how the service has been designed or implemented.
- Services are reusable: They are intended to be reused and engaged in new transactions.
- Services are autonomous: Ideally, a service should have control over its run-time environment, and it should be possible to evolve the service without impacting its consumers.
- Services minimize statefulness: A stateless service is one that provides a response after your request and then requires no further attention. A stateful service is one where subsequent requests to the service depend on the results of the first request. Services where neither the client nor the service provider needs to keep track of the state of one another are easier to manage. The need to consider state requires the addition of transaction management.
- Services are discoverable: Service providers publish descriptions of their services, and clients can search over these descriptions to select the most appropriate service for their need.
- Services are composable: They represent meaningful functionality that can be assembled into larger and new configurations depending on the need of particular kind of users.

The previous description constitutes an abstract high-level characterization of a service-oriented architecture. Several technological solutions have been proposed for implementing this abstract view. The article addresses the issue at a conceptual rather than technological level to discuss the merits of the concept of service-oriented solutions for the particular problem of developing computational creativity systems in the domain of poetry generation. For this reason, no details on technological implementation of the proposed approach to modularity are considered. Of the four basic aspects involved in the definition given for a SOA, the key points for this article are the need to make available functionality currently embedded within existing creative systems, the proposal of a manner of reorganizing them to achieve this, and the concept of service as unit of articulation for this process. The provision of specific interfaces and protocols will be left for further work, because it ideally should involve joint work with other researchers in the field.

2.2. Computational Creativity

Computational creativity has evolved significantly from its early stages at the turn of the century, and as its body of knowledge grows, two ideas become ever clearer: the difficulties in reaching consensus on a definition of the field and its scope and the importance of researching the topic in spite of this. The problems of defining creativity with enough rigor have been addressed by recent reviews of the history of the field (Cardoso et al. 2009). The most useful definition provided so far as a guiding principle for computational creativity research development is the following:

> The philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviors that unbiased observers would deem to be creative. [Colton and Wiggins (2012)]

An important point in this definition—also touched upon in (Cardoso et al. 2009)—is the fact that creativity lies on the eye of the beholder: objective measurement of computational acts in isolation is unlikely to result in valuable judgments on their creativity unless some kind of observer volunteers an opinion (ideally an unbiased one). This presents a difficulty for pragmatic research because it implies a need for human evaluators to be involved in assessing the degree of creativity of any given computational act. It also opens the field to explore the creation of artifacts without necessarily modeling human behavior. The definition speaks about responsibilities, but these responsibilities may be undertaken in forms or procedures that a human might never have employed. A current of thought is progressively gathering strength in favor of exploiting the possibilities for creativity that machines provide, which are radically different from those available for humans. These observations suggest that computational creativity would benefit significantly from advances that allow its products easy exposure to a large population of observers, and advances that allow it to draw on technological solutions that are not available to human creators. The implementation of creative functionalities as composable services fulfills both of these requirements.

For the analysis of complex creative acts in terms of their constituents elements, some recent theoretical proposal for understanding computational creativity software will be useful. The FACE model (Colton et al. 2011; Pease and Colton 2011) presents a framework to understand creative acts performed by software. It defines a creative act as a nonempty tuple containing exactly zero or one instances of eight types of individual generative acts. The eight types are defined in terms of four different target types: the expression of a concept, a concept, an esthetic measure, or framing information. A *concept* is a procedure that is capable of taking input and producing output, the *expression of a concept* is an instance of an (input, output) pair produced when a concept is run. An *esthetic measure* is a function that takes as input a concept or an expression and outputs a numerical score. *Framing information* is a comprehensible explanation (in natural language) of some aspect of the tuple. The eight types arise by distinguishing, for these four target types, between artifacts (generating instances of them) and processes (generating methods for producing instances). An interesting aspect of the FACE model for the present endeavor is that it contemplates the possibility for certain generative acts in a tuple to be undertaken by a third party.

Many efforts over the recent years that address the study of creativity from a computational point of view acknowledge as a predecessor the work of Margaret Boden (1990). Boden proposed that artificial intelligence ideas might help us to understand creative thought. One of Boden's fundamental contributions was to formulate the process of creativity in terms of search over a conceptual space defined by a set of constructive rules. Sharples (1996, 1999) brings together Boden's computational analysis of creativity with insights on the task for writing, understood as a problem-solving process where the writer is both a creative thinker and a designer of text. The account instantiates the various elements in Boden's analysis as ingredients in the domain of the writing activity. For Sharples, the universe of concepts that can be explored in the domain of writing could be established in a generative way by exhaustively applying the rules of grammar that define the set of well-formed sentences. The conceptual space over which a writer operates is a subset of this universe identified by a set of constraints that define what is appropriate to the task at hand.

Sharples identifies creativity in writing with the application of processes that manipulate some of these constraints, thereby exploring and transforming the conceptual space that they define. Some of these concepts will revisited in the discussion in Section 4.

2.3. Articulating the Representation of Poetry in Existing Systems

Poetry generation systems explore a conceptual space characterized by form and content. The rules of the language being employed interconnect these two dimensions, in as much as any specific content, when phrased in a particular way in a particular language, thereby acquires a particular form. For the purposes of this article, it is useful to analyze existing poetry generation systems in terms of two different aspects: how they constrain the form of the poems they generate and how they constrain the content of these poems. The manner of constraining in each case can also be characterized by the level of abstraction at which the constraints are established. The setting of constraints can range from imposing the reuse of particular fragments of text—phrases, lines, sentences, or templates of ready-written text with gaps left to be filled during construction—to requiring particular features, which can themselves be formulated at different levels: phonetic, lexical, syntactical, and semantic, but also prosodic, metric, emotional, or rhetorical. Because language is a complex instrument interconnected at many levels, all of these ways of constraining the output can affect form and content, and some particular ways have more impact on one than the other. Constraints established at the phonetic, syntactic, metric, and rhetorical levels will have a more pronounced effect on the form of a poem, but they may also restrict the particular content that can be conveyed, because they may, for instance, rule out particular lengths of text or make it difficult to introduce longer words, which will affect content. Constraints at the emotional, lexical, and semantic levels are more likely to affect what content is conveyed, but they may bring in words that carry their own phonetic and metric restrictions, which will affect form. These various features will be used to analyze the systems reviewed in the succeeding text, because they jointly affect which aspects of the text need to be represented in each case, and this in its turn affects how these systems might be articulated differently so as to be deconstructed into services. It is important to note that this criterion induces an order of presentation of existing systems that does not respect their chronology.

Early approaches to systematizing the construction of poetry occurred in the literary word independently of the drive for computational creativity. The work of Queneau (1961) constitutes a combinatorial approach to poem generation where a large number of poems is obtained simply by interchanging the lines of a set of poems, always respecting the same relative position of each line within a set template stanza. The lines in each particular position were all carefully crafted so as to be compatible with all other possible continuations. The system in this case relies on a basic functionality for providing fragments of text that satisfy given metric constraints, namely, being valid as lines in a sonnet. The system combines these fragments into poems.

The *rimbaudelaires* constructed by Oulipo (1981) explores a different procedure, where words or phrases are cut out of poems by Rimbaud to be replaced by metrically matching words from poems by Baudelaire. In both of these cases, form and content are constrained by reusing particular fragments of text, with little or no abstraction. Here, the system relies on two different functionalities: one for providing textual templates suitable as poem skeletons and with gaps at particular places and one for providing words of a particular metric and corresponding to a particular part-of-speech (POS) tag.

Very similar procedures have been implemented computationally in more than one poetry generation system. Computational solutions of this kind constrain form by relying heavily on the reuse of fragments from a corpus of poetic texts but usually add some means for constraining content based on a separate source. Two different approaches can be found: those that reuse the text directly and those that reuse the structure of the text in terms of its implicit sequence of POS tags.

The FloWr system (Charnley et al. 2014), intended for implementing creative systems as scripts over processes and manipulated visually as flowcharts, has been used to build poetry generation systems by combining word selection strategies with retrieval strategies, key phrase extraction techniques, filtering strategies, rhyme or footprint matching and strategies for collating the resulting fragments into stanza-like groupings. Word selection strategies find words within a selected range of frequency of use, trim the set down to a particular syntactic category, further restrict to words with a particular sentiment. Retrieval strategies retrieve tweets containing a word picked randomly from the filtered set. Filtering strategies disregard tweets with profanities, or not containing personal pronouns. Rhyme or footprint matching identifies fragments with suitable form. This procedure establishes constraints on content in terms of the initial filtering of what words to use for seeding the retrieval, uses as candidate forms the result of filtering the obtained tweets, and defines constraints on form by a combination of the conditions used to filter and the target stanza-like forms used to drive the compilation process. However, ultimately, it is relying on reuse of complete tweets, or sentences extracted from tweets, with results emerging on the basis of the various procedures employed for selection and filtering the input material. The FloWr system relies on a two-tier approach to poem building: one process for building candidate lines and one for collating selected candidates into stanzas. In this case, there is a functionality for retrieving suitable fragments of text—here based on several possible retrieval strategies—and a functionality for combining these into poems.

The corpus-based approach of Toivanen et al. (2012) also reuses text, but it introduces small changes in terms of word substitution. It relies on two different corpora, one to establish constraints on form and one to establish constraints on content. To set constraints on content, a word association network is built from a corpus consisting of the Finnish pages of Wikipedia. To set constraints on form, fragments are selected from a corpus of old Finnish poetry. A word substitution method is applied: start from a given fragment from the poetic corpus, replace selected words with syntactically compatible words obtained from the word association network. Because word substitution is applied to the set of fragments obtained from the corpus, the operation here is equivalent to a process of template extraction followed by a process of selecting appropriate words to fill the gaps. This approach is refined further by Toivanen et al. (2014) where additional constraints are imposed on the words being used to replace: Candidate fillers must come from foreground associations obtained for a given input news item—words in the current news item not already associated with a prior news item on the same topic. When associations are employed to select the desired words, an additional level of semantic-related criteria is being used as filter for retrieval of words.

A similar solution is applied in the Pemuisi system (Rashel and Manurung 2014) where templates for lines are obtained by abstracting content words from a corpus of Indonesian poems, then filled with keywords extracted from newspapers (an intermediate semantic expansion process is applied to improve variation). Selection of slot fillers is treated in this case as constraint satisfaction problem; constraint satisfaction (at the poem level) is also used to select a set of lines to combine into a poem. A schema similar to those followed in the previous text is being followed: template construction followed by template completion from a set of candidate words. Pemuisi also follows articulation in two tiers: The task of building a poem is divided into a procedure for constructing lines and a procedure for combining valid lines into poems.

This two-tier approach to poem construction had already been employed in the PoeTryMe system (Gonçalo Oliveira 2012), which considered constraints on content specified at a semantic level. In PoeTryMe, constraints on form are set by employing line templates that are actual lines from a corpus of poems, with gaps—tagged with gender and number agreement information—for two or more words related to one another by a particular semantic relation. Constraints on content are established by picking candidate fillers from a semantic relation graph—nodes are words, edges are semantic relations between them—so that the selected words are in a relative vicinity of a given set of seed words. The actual poem is constructed by combining a set of filled-in line templates with a poem template that can establish constraints on number of lines per stanza, number of syllables per line, and/or rhyme schemes for the stanza. The poem template is filled in according to one of a set of strategies. These strategies are based on searching for candidates that optimize a scoring function that considers satisfaction of the constraints of the poem template. The set of strategies includes generate and test and evolutionary methods. Template construction followed by constraint-driven selection of words to fill the templates is again in evidence. This time, semantic criteria are used to filter the candidates words.

The approach of Colton et al. (2012) is also based on template extraction, but it presents a much richer process of construction, involving up to three tiers of recombination, and it includes a very elaborate inventive process for building the initial seeds for content. It also introduces emotional constraints in the form of a mood set at the start of the process—from the analysis of newspaper articles for a particular day—that influences later decisions. The system constrains the content by starting from a collection of similes—obtained by a complex process of expanding an existing database of similes and then selecting from the result those that satisfy basic correction understandability criteria—taken together with a set of keywords extracted from newspaper articles. The form of the poems is constrained during a three-stage process: The selected words are used to build phrases, a first tier of templates is used to combine phrases into larger fragments of text, and a second tier of templates to build stanzas and/or poems from fragments of text. The process of creation of phrases is itself described in terms of three stages: two initial ones of *retrieval* and *multiplication* (that together constitute the process of building the collection of similes used as seed) and *combination* (which involves combining the resulting similes with keyphrases to fill the gaps in the first tier of templates). The process of combining the phrases resulting from this combination process with the second tier of templates is called *instantiation*. Although the processes of constructing the templates and recombining the intermediate results into larger poems are more complex than in previous instances, overall, their basic nature is still one of template construction/filling plus recombination of fragments based on poetic criteria. An important innovation is the possibility to consider emotion-based constraints.

A different approach involves stripping all the words from the reference fragment for form, and retaining only its sequence of POS tags, then filling those in with words from a different source, with criteria for appropriateness to drive the process.

An example of this approach is the early version of the WASP system (Gervás 2000), which draws on prior poems—to provide plausible sequence of POS tags for lines—and a selection of vocabulary provided by the user to generate a metrically driven recombination of the given vocabulary according to the line patterns. The plausible sequence of POS tags for lines, together with the selection heuristics based on metrics constrain form, and the vocabulary provided by the user constrains content at a lexical level. This system introduces the concept of a specific module for evaluating poetic form, which can be considered a poetic expert, capable of returning the metric analysis of a given text fragment, numerical values for certain poetic features of the text, and/or a score for its metric quality. This is also the approach followed by Agirrezabal et al. (2013), which relies on extracting POS tag sequences for lines in a given corpus of poems—using different corpora for different lengths of line in the stanza—finding the most commonly used such POS tag patterns for lines, and filling those chosen patterns with new ones, based on criteria at two levels: one syntactic—matching POS tag and morphology—and one semantic. Several semantic criteria are considered, but the best performance is obtained with replacing only nouns with other semantically related nouns, ensuring that morphological information from the original word is transferred to the substitute. These approaches can be seen as an extreme version of template construction, where all the words in the seed fragments are stripped down to just their POS tag, to be refilled with new words satisfying the constraints.

A similar procedure but with more refined constraints on form is used by Toivanen et al. (2013). Instead of taking an actual fragment and replacing some of the words in it with new ones corresponding to the desired content, the selected fragment is stripped down to a skeleton consisting only of the POS tags of each line, and words corresponding to the desired content are used to fill this skeleton in, while obeying a complex set of constraints. Constraints can be established based on rhyme, number of syllables per line, occurrence of particular words, or even syntax, and they can be loosened so that rather than being binary, they allow for grading of the solutions in terms of how well they satisfy the constraints. The solution is then sought by applying answer set programming to search for optimal assignments of candidate words that optimize this grading. The set of constraints being considered here has been refined substantially, but the overall procedure still corresponds to an abstract process of extracting a template and filling it with words based on a set of constraints.

An evolution of the WASP system (Gervás 2001) used case-based reasoning (Aamodt and Plaza 1994) to build verses for an input sentence by relying on a case base of matched pairs of prose and verse versions of the same sentence. Each case was a set of verses associated with a prose paraphrase of their content. An input sentence was used to query the case base, and the structure of the verses of the best-matching result was adapted into a verse rendition of the input. The syntactic structure of the solution—the sequence of POS tags for the verse version, including the corresponding line breaks—was used as template to be filled with words from the user query, guided by metric restrictions and falling back on the actual words of the poem in case of mismatch. Because solutions not necessarily matched full stanzas or even complete lines, an additional tier of construction was included to combine the verse renditions of each input sentence into complete poems. Although there is a more elaborate procedure here for selecting the most appropriate template to realize a given input text, the procedure employed to actually generate the poem follows the lines outlined earlier: template extraction followed by template filling.

An approach that relies on word-based specification of content combined with emotional aspects but introduces a different way of constraining form and a different approach to the combination of modules dealing with different aspects of poem generation into a single system is presented by Misztal and Indurkhya (2014). This is a poetry generation system that relies on a multiagent blackboard approach to create poems that employ a wide range of literary tropes and aim to convey a particular emotion. The system relies on a set of expert modules that each focus on a particular aspect, and which interact by sharing results on a blackboard. Types of experts include *word-generating experts* that contribute word matching a given topic or emotion, *poem-making experts* that arrange words from the common pool into phrases or sentences guided by context-free grammars, and *evaluating experts*. This system includes a functionality for word selection already encountered, but it combines it with a new functionality that allows combination of words driven by a grammar, rather than the simple template filling seen so far. A theme and an emotion for the poem are extracted from an input text provided by the user. A *control component* determines which experts obtain access to the blackboard and in which order. Specific experts contribute particular tropes such as epithets, comparisons, metaphors, oxymoron, rhetorical questions, exclamation, or repetition. Other experts provide pertinent vocabulary, grammatical expertise, selection criteria based on number of syllables, or inspiration—by processing the input text. This approach constrains content in terms of a particular topic and an emotion extracted from the input text and constrains form through the implementation of the various experts. This type of solution allows a fine-grained control on many aspects of the form, including explicitly many literary tropes. It is important to note that in this particular approach, specific literary forms are introduced explicitly by the set of system modules, rather than arising from the reuse of an existing corpus of poetic texts.

Another system heavily influenced by semantic information used to drive the poetry generation process is described by Veale (2013a). This system exploits poems as summarization and visualization devices for the set of properties and feelings that are evoked when a certain subject T is viewed as M. That is, it explores the conceptual space of poems built around the metaphoric view of T as an M. It achieves this by relying on a rich semantic knowledge base mined from three resources: a large roster of stereotypes, a large body of normative relationships between these stereotypes, and the Google ngrams. The initial set of stereotypes is progressively elaborated—enriched with information on relations between stereotypes, and combined into complex conceptual blends—to provide semantic input pregnant with insight and wit. This material comes out as a set of pairings between elements from the domain of T and elements from the domain of M. It is exploited in poetic form by the use of a semantic grammar for the poem, which comes with gaps in particular lines for the two elements of a pair, and indications of the trope that should be used in different parts of the poem, how each line is to be rendered—as an assertion, an imperative, a request, or a question—and whether it should be framed positively or negatively. In this case, the content is constrained by the input terms—T and M—and the corresponding set of pairs as returned by the knowledge base, and the form is constrained by the semantic grammar, which is a template enriched with additional information used to guide the instantiation process. This system presents a more elaborate process for selecting words in related pairs, together with a grammar that can be used to combine them into text fragments.

Some of the systems described so far include constraints on the content at the semantic level. However, these semantic constraints take the form of either semantic relatedness of the content to a given word or having a certain semantic relation hold between certain parts of the content. Considering an actual message to be conveyed by the poem, specified in terms of a semantic representation, constitutes a significantly more difficult challenge. This challenge was addressed in initial work by Manurung (1999), who applied a generate and test approach based on chart generation but added an important restriction: that poems to be generated must aim for some specific semantic content, however vaguely defined at the start of the composition process. The approach relied on chart generation, taking as input a specification of the target semantics in first-order predicate logic, and a specification of the desired poetic form in terms of meter. Words are chosen from a lexicon that subsumes the input semantics, and a chart is produced incrementally to represent the set of possible results. At each stage, the partial solutions are checked semantically to ensure that no sentences incompatible with the original input are produced. Additionally, partial results are checked for compatibility with the desired poetic form. In this system, the process of grammar-based generation is driven by a semantic input, and the validation of the results relies on a poetic expert.

Manurung (2003) went on to develop in his PhD thesis an evolutionary solution for poetry generation—now described by Manurung et al. (2012)—that also aimed for a specific semantic target. Manurung's MCGONAGALL used a linguistic representation based on lexicalized tree-adjoining grammar over which operated several genetic operators—from baseline operators based on lexicalized tree-adjoining grammar syntactic operations to heuristic semantic goal-directed operators—and two evaluation functions—one that measured how close the solutions stress pattern was a target meter and one that measured how close the solutions propositional semantics was to the target semantics. Both of these systems by Manurung included constraints on content in terms of particular meaning to be conveyed, represented semantically, and constraints on form formulated at the metric level and at the syntactic level—both systems required that outputs be syntactically correct with respect to a given grammar.

Yet another approach to reusing text to constrain the output of poetry generators is the use of ngrams to model the probability of certain words following on from others. This corresponds to reusing fragments of the corpus of size *n*, and combining them into larger fragments based on the probability of the resulting sequence. This approach introduces a new approach to the generation of text, beyond template filling and grammar-based construction: the generation of text from an ngram-based language model.

The Poetic Machine (Das and Gambäck 2014) for generating Bengali poetry employs a line-based construction procedure, driven by a given rhyme pattern to be matched, with ngram-based constraints used for selecting the final candidate for a line.

Combining ngram modeling and evolutionary approaches, a redesigned version of the WASP poetry generator (Gervás 2013a, 2013b) has been built using an evolutionary approach to model a poet's ability to iterate over a draft applying successive modifications in search of a best fit, and the ability to measure metric forms. It operates as a set of families of automatic experts: one family of content generators or *babblers*, which generate a flow of text that is taken as a starting point by the poets; one family of *poets*, which try to convert flows of text into poems in given strophic forms; one family of *judges*, which evaluate different aspects that are considered important; and one family of *revisers*, which apply modifications to the drafts they receive, each one oriented to correct a type of problem, or to modify the draft in a specific way. These families work in a coordinated manner like a cooperative society of readers/critics/editors/writers. All together, they generate a population of drafts over which they all operate, modifying it and pruning it in an evolutionary manner over a number of generations of drafts, until a final version, the best valued effort of the lot, is chosen. In this version, the overall style of the resulting poems is strongly determined by the accumulated sources used to train the content generators, which are mostly ngram-based. This system combines an ngram-based text generation module with a poetic expert that acts as fitness function in an evolutionary context. It also includes a new type of module: a text rewriting functionality that receives text as input and generates a different text obtained by rewriting the original in some way. Several versions of this system have been developed, covering poetry generation from different inspirational sources as different sets of training corpora are used: from a collection of classic Spanish poems (Gervás 2013a) and a collection of newspaper articles mined from the online edition of a Spanish daily newspaper (Gervás 2013b). Readers interested in a full description are referred to the relevant papers. However, two specific aspects of this implementation are relevant for this article. First, the various judges assign scores on specific parameters—on poem length, on verse length, on rhyme, on stress patterns of each line, on similarity to the sources, fitness against particular strophic forms, and so on—and an overall score for each draft is obtained by combining all individual scores received by the draft. A specific judge is in charge of penalizing instances of excessive similarity with the sources, which then get pushed down in the ranking and tend not to emerge as final solutions. Second, poets operate mainly by deciding on the introduction of line breaks over the text they receive as input. These are aspects that will play a role in the proposed deconstruction to be implemented as services.

A more refined attempt at generating poetry based on ngrams and including more abstract constraints into a tractable and complete search procedure is the work of Barbieri et al. (2012). Relying on constrained Markov processes to generate texts in the lyrics in the style of an existing author, it integrates the constraints on grammaticality, rhyme, meter, and, to a certain extent, semantics into the search procedure itself. The system start from a given word and considers as semantically acceptable outputs those that include the *n* words in the corpus most closely related to the chosen word. This approach basically enriches an ngram-based solution for text generation, imbricating the constraints that drive the process into the construction procedure itself.

The material covered so far indicates that there are many different approaches to the task of poetry generation, and that each approach considers different representations of poetry and that the resources required to produce it come specified at different levels of abstraction. This affects the way in which solutions for poetry generation can be deconstructed into services.

2.4. Articulating the Modularity of Creative Computational Systems

In addition to the possible articulation of the poetry generation task in terms of its constituent element, one needs to consider the way in which these systems—and others designed for similar or related tasks—are themselves articulated into modules that cooperate.

Veale (2013b) proposes an architecture for creative Web services intended to act as a force magnifier for computational creativity, both for academic research and for the effective deployment of real computational creativity applications in industry. The architecture combines three types of services: *discovery and insight services*, aimed at mining diverse corpora to acquire emergent insights and novel perspectives on everyday concepts; *idea composition services*, designed to suggest, elaborate, and comprehend conceptual metaphors, analogies, and blends, as well as services for accessing the large store of commonsense knowledge that these composition services will crucially rely upon; and *framing services*, which can package the conceptual conceit that underpins a creative act for an audience in a concise, easily appreciable, and memorable form, such as a linguistic metaphor, simile, joke, name, slogan, short story, poem, picture, piece of music, or a mixture of these forms. The proposal includes description of two specific Web services—*Thesaurus Rex* and *Metaphor Magnet*—for providing creative functionality.

The architecture of the PoeTryMe system (Gonçalo Oliveira 2012) is presented with the hope that it might serve a generic as a platform for the automatic generation of poetry. This architecture includes the following modules: a *relations manager* that operates over a semantic graph, a *grammar processor*, which relies on a grammar, a *contextualizer* that uses a context grammar, a *sentence generator*, and a module that establishes the *generation strategy*, which draws upon poem templates and seed words provided as input. The flexibility of the platform lies in the possibility of replacing the various resources and modules with alternative solutions. This has in fact been tried out successfully in porting the PoeTryMe system—originally developed for the Portuguese language—to generate Spanish poetry (Oliveira et al. 2014). This effort involved the replacement of the resources for Portuguese with equivalent versions for Spanish, and slight adaptation of some of the modules. The resulting system does indeed generate poetry in another language, but the procedure employed to do so is very much the same as in the original system.

Along similar lines, the blackboard architecture proposed by Misztal and Indurkhya (2014) also promises a flexible approach, incorporating as it does explicit representation of multiple aspects of poetry generation—as so many expert modules—and also an explicit arbitration procedure between them in terms of its control component.

The FloWr system (Charnley et al. 2014) described in the previous text is an instance of a system designed to encourage third-party developers of creative functionality to contribute material, which becomes available within the general framework for recombination with existing modules and resources. This approach partakes in the spirit of a service-oriented approach to development but expects contributors of services to develop them within a particular framework and adopting a particular implementation based on scripting and flowcharting.

ClowdFlows (Kranjc et al. 2012) is an open cloud-based platform for composition, execution, and sharing of interactive data-mining workflows, based on the principles of service-oriented knowledge discovery. It enables users to seamlessly integrate and join different implementations of algorithms, tools, and Web services into a coherent workflow that can be executed in a cloud-base application. This framework combines the visual graphical manipulation of processes as workflows with the modularity and composability of services, including means for integrating those deployed as Web services.

Pattern (Smedt et al. 2014) is a Python toolkit for Web mining, natural language processing, machine learning, network analysis, and data visualization that includes a module that can be used to set up Web services. The authors defend that users can work on different tasks and share their results without having to reinvent algorithms from published pseudo-code, without having to deal with installation instructions or adopt new programming languages.

OntoHub (Kutz et al. 2014) is a repository of ontologies based on the *distributed ontology language* that provides the capability for specifying domain diagrams in such a way that they can be automatically combined using conceptual blending. This is a creative service operating over the Web, but the current version still relies on distributed ontology language files as input.

The Slant system (Montfort et al. 2013) is a storytelling system that combines a number of existing systems with focus on different aspects of the storytelling process and integrates them using a blackboard architecture. The authors explain that the crucial issue they had to face to design the integration was to establish a shared representation for the material being considered. In particular, because not only entire complete stories need to be represented, but also partial stories, the composition of which is still in progress. Another important insight contributed by the Slant system is the need to establish a termination condition for the construction procedure: some criterion for deciding when the accumulated result can be considered complete. The architecture of the Slant system combines an initial stage where several systems operate jointly on a shared representation placed on a blackboard, and a second stage where the result of the first stage is passed on to a generative pipeline that processes the accumulated information to produce the final result.

3. DEVELOPING POETRY-RELATED PROCESSES AS SERVICE

In view of the work reviewed above, deconstruction of a computer poet would involve the following stages. Section 3.1 presents an attempt to digest this information into a tentative proposal of a high-level description of a set of services that might be valuable for researchers interested in poetry generation. Section 3.2 presents the rationale for two particular examples that can be developed by reusing functionality of an existing creative system, the WASP poetry generator. These examples are described in Sections 3.3 and 3.4.[1]

3.1. A Proposal for Poetry-Related Services

The review of existing systems presented in Section 2.3 allows us to compile a basic specification of some of these services in terms of the type of input they can receive and the type of output they can provide. It is beyond the scope of this article to propose particular

[1] Correction added on 28 September 2016, after first online publication: an additional header entitled "DEVELOPING POETRY RELATED PROCESSES AS SERVICES" and a brief introduction were inserted as Section 3 to better explain the structure of sections in the paper; succeeding sections and subsections have been renumbered.

implementations for all of these types of services. For each one of these functionalities, a standardized contract would need to be defined before it can be implemented as a service.

3.1.1. Poetic Fragment Provision. A number of the systems reviewed in Section 2.3 rely on modules that can provide snippets of text satisfying constraints of various types. These constraints can be phrased in terms of semantic relatedness to other words (Gonçalo Oliveira 2012; Colton et al. 2012; Veale 2013a; Toivanen et al. 2014), POS tags (Gervás 2000, 2001; Toivanen et al. 2014), emotion (Colton et al. 2012; Charnley et al. 2014; Misztal and Indurkhya 2014), metrics (Gervás 2000, 2001; Agirrezabal et al. 2013; Toivanen et al. 2012; Rashel andManurung 2014), rhyme (Gervás 2000, 2001), or emotion (Colton et al. 2012; Charnley et al. 2014; Misztal and Indurkhya 2014). The functionality of these modules might be usefully made available as a service. Such services would take as input a specification of the type of constraint desired, and they would return a set of snippets satisfying the constraints in the given specification. Such systems may return text fragments consisting of a single word, providing reusable solutions for all the systems that search for appropriate words to fill particular gaps in templates. They may also provide line-sized fragments, in which case they would be useful in systems that build poems by recombination of valid lines.

3.1.2. Template Provision. Many of the systems described rely on the use of templates for the construction of poetic fragments of text. Some of them include a stage that processes a corpus of texts and automatically extracts templates of different kinds to drive the generation process (Colton et al. 2012; Gonçalo Oliveira 2012; Toivanen et al. 2013; Rashel and Manurung 2014). The criteria for extraction can be based on line breaks, based on POS tags, based on semantic relations between words, or based on syntactic structure. These templates are obtained by taking an input fragment of text and abstracting away some of its components (usually single words, but possibly also longer phrases, or all of the words). The components that are abstracted are replaced by a placeholder specification that describes the constraints that the abstracted element satisfied. Any candidate to fill the resulting gap should satisfy the same constraints.

In some cases, the poetry generation systems described include functionality to automatically generate these templates from a given corpus of text. Such functionality might be very useful to developers of other systems if made available as a service. The service would receive as argument a specification of the unit to be employed as basis for the template. This can range over a large set of elements, including sentences, lines, paragraphs, or poems. Different instances of this type of service are possible, depending on how the selection of what to abstract is made. In such cases, an additional input argument should be provided to describe the type of abstraction that is desired. Possible types of abstraction to consider include abstract all words in the chosen fragment down to their POS tags; abstract only content words; abstract only words a particular category, such as nouns, for instance; and abstract only pairs of words connected by a semantic relationship.

A related type of service can simply provide hand-crafted templates on demand. This type of service would only need the arguments specifying the basic element on which the template should be constructed and the type of abstraction desired.

3.1.3. Grammar-Based Generation. A number of systems rely on grammars of some kind for at least part of their generative process resulting in text fragments or phrase. The types of grammar considered vary widely and include template-like semantic grammars (Gonçalo Oliveira 2012), context-free grammars (Misztal and Indurkhya 2014), or tree-adjoining grammars (Manurung 1999, 2003). In each case, the grammar generation module receives as an input a basic specification of the basic material to be considered and outputs a grammatically valid text that includes as much as possible of that material in related sentences. The specification of the material to use can come either as a set of words to include (Misztal and Indurkhya 2014), as a set of words interconnected by a given semantic relation (Gonçalo Oliveira 2012), or as full semantic description in terms of first-order logic (Manurung 1999, 2003).

3.1.4. Ngram-Based Knowledge. A different type of system relies on ngram-based language modeling at different levels, for analysis of candidate solutions, training, generation, and probability rating (Barbieri et al. 2012; Gervás 2013a; Das and Gambäck 2014). This would correspond to a set of potential services of different types, and each one would be defined in terms of its contract and implemented accordingly. Possible instances of these services might be service for returning an ngram-based language model for a given corpus, service for returning the probability of a given sequence of words with respect to a given language model, or services for providing a random sequence of text valid with respect to a given ngram model.

3.1.5. Queriable Knowledge Bases. A large number of the systems reviewed rely on knowledge bases that provide them with answers to particular queries. These queries may concern semantic relations between words (Gonçalo Oliveira 2012; Toivanen et al. 2014), rhetorical pairings (Colton et al. 2012; Veale 2013b), semantics to syntax mappings (Manurung 1999, 2003), or emotional connotation of particular words (Colton et al. 2012; Charnley et al. 2014; Misztal and Indurkhya 2014). These type of knowledge bases would be very useful if available as services. Solutions already exist out there that can be seen as partial answers to this need. Existing Web services for creativity, such as Thesaurus Rex or Metaphor Magnet (Veale 2013b), would actually fit into this description.

3.1.6. Poetic Expertise. Although many of the reviewed systems rely directly on their construction procedures to guarantee the poetic quality of their results, there are some that include a module capable of producing judgments on poetic quality for a given text input. This type of module has been described in the review in Section 2.3 as a poetic expert. The functionalities that involve receiving a text fragment and outputting some type of judgment on its poetic quality would be very welcome additions to a set of services for the development of poetry generation systems. Such functionalities might include providing syllable counts, identifying rhymes, checking for satisfaction of rhyming patterns characteristics of certain stanzas, scansion of lines into feet, or combinations of these features into more abstract judgments on poetic quality.

Examples of this type of service are described in Section 3.3.

3.1.7. Rewriting Capabilities. Many of the functionalities for which poetry-related services have been proposed had been identified as recurring throughout the set of existing poetry generation systems reviewed. The ability to rewrite a given fragment of text in a different form that might better match certain poetic criteria occurs less frequently. It is described as a module in (Toivanen et al. 2012) and in the WASP system. Toivanen et al. (2012) describe a word substitution method that involves rewriting fragments of the corpus by replacing some words with others. In the WASP system, there are at least two different modules for rewriting a given draft: one that simply inserts line breaks at appropriate places and one that modifies a given draft by either replacing or eliminating certain spans of the text, from single words to complete sentences. Additionally, the mutation and crossover operations involved in the evolutionary solution in (Manurung et al. 2012) can also be considered rewriting operations. There are two reasons to include this type of functionality in our set of services. First, because it is an operation that all beginners at poetry writing employ frequently. Second, because, although most of the other systems reviewed do include specific modules covering this functionality, the overall operation of many of them could be considered an instance of this rewriting ability: Filling the gaps in a template with new words is equivalent to applying a rewriting operation of this type to the original text from which the template was extracted.

Examples of this type of service are described in Section 3.4.

3.1.8. Conclusions on the Proposed Services. This set of services is difficult to classify in terms of the three variants described by Veale (2013b). Whereas some are clearly discovery and insight services, others would fall in the category of framing services. Yet, Veale's description of this category seems to be very coarse-grained, in the sense that poetry generation is mentioned singly as a member, and many of the types in the preceding text operate at a much lower level of granularity.

Ongoing attempts at providing infrastructure for combining modules from different sources, such as the FloWr system (Charnley et al. 2014) or the ClowdFlows system (Kranjc et al. 2012), would provide very useful functionality for combining existing services. Indeed, ClowdFlows already includes functionality to this effect, combined with the visual display in terms of workflows.

It is beyond the scope of this article to propose a single service-oriented architecture for the task of poetry generation. The field and the understanding of the problem are not yet ripe to undertake this endeavor. Nevertheless, a number of patterns can be seen to emerge from the review of existing systems.

The most elementary poetry generation systems rely on a combination of poetic fragment provision and as subsequent set of recombination. More elaborate solutions rely on a template-provision stage, followed by a poetic fragment provision stage to fill the gaps in the templates, and a final stage of recombination into poems that can be multilayered. Some of these solutions are driven by a poetic expert module that provides feedback on the metric quality of the results. The more refined solutions rely on either grammar-based or ngram-based generative solution in place of fragment provision and/or template-based generation, with similar layers of poetic expertise and recombination placed on top.

3.2. Designing Examples of Useful Poetry-Related Services

To test the initial hypothesis of this article, we will attempt to reconstruct part of the functionality of an existing poetry generator in terms of modules that could be made available as services. To this end, each of the modules must undertake a task that can be described in an understandable way, that can operate autonomously of the other modules, and that has been validated as a useful contribution to the overall task of poetry generation. The case study that we have chosen is the WASP poetry generator, in its most recent version. The purpose of this effort is to show that there are a number of subtasks in any approach to poetry generation that are related to the elementary properties of poetic text, which take basic representation as input—namely, clear text, possibly with the line breaks being relevant to the form—and return either text of the same kind as output or a numerical score represented as an integer or double. These are also tasks that are independent of the particular evolutionary procedure originally being applied in the parent system, and that do not rely on any particular internal representation, in the sense that, where they do, the details can remain hidden inside the implementation of the service and need not be taken into account by a client.

The major task modeled by the WASP system is that of identifying the most appropriate poetic form for a given text. This is achieved mainly by distributing coherent fragments of the input text over a set of verses, each one observing metric restrictions on a number of syllables, stress placement, and possibly rhyme of the final word. The selected fragments may also be modified in different ways, possibly departing from the original style or even its meaning, if a greater poetical effect is achieved as a result.

Two basic tasks are involved in this process.

First, a system attempting to find appropriate poetic form for a text requires some means for evaluating the poetical effect of any given draft. If this functionality were encapsulated in an independent module, it would surely be of use to any developer aiming to produce poetry. This would correspond to a service on poetic expertise, as described in Section 3.1. It is therefore our first candidate for an independent service. Within the WASP system, this particular task is carried out by the judges.

Judges are individual agents that can score a given draft according to specific criteria. They collectively conform the fitness function employed in the evolutionary process. Although judges might be implemented as individual services, so that different developers can compose different evaluation functions by selecting different sets of judges, in this article, we have considered only a single service that provides a homogeneous evaluation criterion representing the combined expertise of the available WASP judges.

The details of this combination, and the experiments carried out to calibrate it, are described in Section 3.3.

Second, a system attempting to find appropriate poetic form for a text requires some means for providing an initial distribution of the given text into verses. It is possible that inserting line breaks in appropriate places enhances the poetic effect of a given text. The ability to identify what these places may be would constitute a useful addition to a poet's toolkit. This subtask is undertaken in the WASP system by the poets. Poets receive a plain text, with no line breaks, and produce an initial poem draft, where the text has been broken down into a number of verses. This would correspond to a service of rewriting capabilities, as described in Section 3.1. Although this may seem trivial, it is an important step of the process because it establishes relevant parameters such as the verse length, the rhyme scheme, or the number of verses, which determine which type of stanza is being targeted. For any such given combination, the evaluation procedure described in the preceding text must be able to establish a valuation, possibly in the form of a numerical score, or some other means that allows the ranking of different candidates.

The functionality of a *versifier* as described may also be implemented as a service, to be invoked by systems hoping to generate poetry. Attempts at doing this are described in Section 3.4.

3.3. A Poetry Evaluation Service

An important task in the generation of poetry is the recognition of poetic form and the ability to rate a particular effort in terms of its poetic quality. Poetic quality in general terms is a very elusive concept, involving both form and content at all levels, whether linguistic, semantic, evocative, or emotional. The subset of this overall evaluation of quality that is concerned with form is easier to model computationally than other aspects. A number of characteristics universally recognized as relevant to poetic effect are easily identifiable from the surface form of the poems. These include number of syllables in a verse, number of verses in a poem, and rhyme. More elaborate aspects such as alliteration, or the use of rhetorical tropes, could be considered but are left out of the current effort for simplicity.

Even over this restricted set of characteristics, the task is sufficiently complex. Treatises exist that describe the accepted requirements of poetic form for given languages. In the case of Spanish, which is the language for which our service will be designed, a classic reference is Quilis (1985). Nevertheless, such treatises describe these requirements in historical terms as described by grammarians, not necessarily convenient for computational implementation. Furthermore, they usually focus on describing accepted fixed forms for classic poetry and leave open the description of free poetic forms that are known to be of high esthetic value and actually form a high percentage of poetic literature in some languages. The challenge is to devise a set of heuristics that capture both the requirements of classic poetic form and the characteristics of rhythm and rhyme that can constitute a significant part of the appeal of poems in free form. To be useful in computational settings, these heuristics need to be expressed in a numerical form. This would correspond to the definition of an esthetic measure as described by Colton et al. (2011). This challenge has been addressed by collecting a number of identifiable features that together capture the characteristics in question. These features are based on the elementary phenomena that compose metrical form—breakdown into syllables, stressed syllables, and rhyme—but rely on generic structural properties—such as two rhymes or two verses of similar length appearing consecutively in a poem draft, or appearing separated by a number of intervening verses—rather than on collections of named rhyme patterns—which treatises on the subject tend to prefer. A number of different metrics has been developed, each capturing a specific requirement on a particular feature. This results in a number of metrics concerning rhyme, a number of metrics concerning verse length in syllables, and so on.

These metrics could be made available as separate services, for client users to combine as they see fit. Foreseeing the need to combine more than one of these metrics together into a single overall score, each one of them has been normalized to a top score of 100. In each case, the normalization procedure has been devised to ensure that the resulting score remains significant when compared across poetic drafts of different characteristics.

Clients would also have the option to devise their own way of combining these metrics together. Many possibilities exist, from a simple average to weighted linear combination, or more complex mathematical combinations. For the results presented in this article, simple average has been used.

The following set of metrics have been considered:

- *Stress Placement* scores each verse in terms of whether the stressed syllables occur in an appropriate position for the length of that verse.
- *Line Break Placement* assigns low scores to poems where line breaks occur right after a word whose stress does not compute.
- *Consonant Rhyme* counts the number of consonant rhyming pairs—rhymes that appear more than once—and scores the poem with the ratio between the count of rhyming pairs and the length of the poem.
- *Asonant Rhyme* is a similar metric for asonant rhyming pairs.
- *Rhyme Patterns* computes the number of instances of N consecutive verses rhyming together—for $N = 2$, 3, and 4—and the number of instances of two verses rhyming together across a span of N verses with different rhyme—for $N = 1$, 2, and 3. These scores are computed for both consonant and asonant rhymes, but the judge selects the best of the two resulting scores to be assigned to the draft. This is based on the intuition that a poem should be built either aiming for consonant or asonant rhyme; thus, one should avoid the possibility that the score for one should drag the other down. This way, the highest scoring type of rhyme overrides the other one.
- *Verse Length Repetition* checks how many types of verse lengths appear in the poem and scores it, taking into account the ratio of the number of verses that share the most frequent length over the number of verses in the poem.

TABLE 1. Comparative Scores by the Set of Metrics for Valued Poems (VP) and News Articles (NA).

	SP	LB	CR	AR	RP	VI	VR	TS
VP	98	100	62	75	59	96	77	81
NA	100	0	0	0	0	0	0	14

Scores for metrics cover stress placement (SP), line break placement (LB), consonant rhyme (CR), asonant rhyme (AR), rhyme patterns (RP), verse length irregularity (VI), verse length repetition (VR), and total score (TS).

- *Verse Length Irregularity* scores by penalizing each verse that is of unique length within the poem. This scores highly for most known stanzas and decreases as irregularities creep in. This scores better the stanzas that combine two different verse lengths—which are heavily penalized by the *VerseLengthRepetitionJudge*—as long as each verse length occurs more than once.

The set of metrics has been calibrated by comparing the scores that it assigns to two different sets of texts: a selection of classical poems (understood to be instances of highly valued samples) and a set of news articles for a given day taken from a Spanish online daily newspaper[2] (understood to be not necessarily poetic in any way). The selection of classical poems employed to calibrate included seven poems of different lengths, different poetic forms, and from different historical and stylistical periods. The set was designed to include all major lengths of line used in classical Spanish poetry and free-form verse, both poems in fixed stanzas and in free form, and both short poems of fixed number of lines and long poems of arbitrary length—although usually constrained to an even number of lines. This set was deemed to cover a diverse enough set of different types of poetry to constitute a reliable indication of the generic applicability of the metrics beyond particular types of metric. An equivalent number of texts of similar sizes was obtained from the website of the newspaper, which consisted of texts with no poetic intent and no line breaks.

The hypothesis is that the metrics will fulfill their intended function if they allow discrimination between the two sets of texts.

The average scores for each of the metrics, plus the total combined score, are listed for both sets of texts in Table 1. These experimental results are provided solely for the purpose of illustrating the metrics that have been described.

The differences in scores between the two sets of text are considered significant enough to validate the hypothesis.

3.4. A Versifier Service

Two different approaches are possible: either one expects a specific form for the final results (in terms of metric restrictions), or one is willing to accept the best possible form for the given text. In the first case, the content will have to be modified to match the desired form. In the second case, the form is selected among a number of possibilities by the constraints on the content. Each one of these approaches will give rise to a different procedure to solve the task. Different heuristics are possible.

[2] Diario El Pais, 2013-05-21

TABLE 2. Comparative *Average* Scores for Different *Versifiers*: Rhyme Driven (RD), Rhyme Aware (RA), and Rhyme Aware Slack (RS).

	SP	LB	CR	AR	RP	VI	VR	TS
RD	87	98	55	74	30	94	23	66
RA	100	100	16	27	20	85	76	60
RS	89	97	26	52	29	90	47	61

SP, stress placement; LB, line break placement; CR, consonant rhyme; AR, asonant rhyme; RP, rhyme patterns; VI, verse length irregularity; VR, verse length repetition; TS, total score.

3.4.1. Free Form. If the specific poetic form of the result does not have to be constrained in any way—for instance, by providing specific preset values for parameters such as verse length, poem length in verses, choice of rhymes, or pattern of rhyme schemes—the system is free to explore all possible combinations.

One possible approach is to study the rhymes already available within the input text. Any rhyming word to be retained as a rhyme of the final draft has to appear at the end of a verse. A simple way to achieve is to insert the first tentative set of line breaks right after the words chosen as suitable rhymes.

Several *versifiers* can be devised by following this principle. The main problem that they face is that the desired rhymes will normally not appear in the input text with a suitable spacing between them to give rise to a valuable sequence of verse lengths. This problem can be addressed either by developing more refined composition strategies, or by establishing a less than optimal solution as a tentative draft to be later improved during revision.

Three possible strategies for line break insertion based on rhymes are considered:

- *Rhyme Driven* inserts a line break right behind every word in the text that has a rhyming word elsewhere in the same text.
- *Rhyme Aware* considers the distances between candidate rhymes, identifies the most propitious one—the one that appears most either as single stretches between rhymes or as combinations of smaller stretches—and attempts to break down the text into verses enforcing that length and the break points where rhymes have been identified.
- *Rhyme Aware Slack* is as the previous but allowing verse lengths within a margin of deviation of the identified optimal verse length.

In all cases, the set of rhymes to be considered is established simply by collecting the rhymes of any words in the input that have a matching rhyme elsewhere in the same text.

Validation of a *versifier* involves showing that, for a set of texts with low poetic effect, the results of applying the *versifier* show a significant increase in poetic effect. This is tested by applying every reviser to a fixed set of texts. The set of texts chosen as case study is the same set of news articles used during the calibration of the metrics presented in Section 3.3 to represent low poetic effect.

The average scores for the population of drafts produced by each of the *versifiers*, plus the total combined score, are listed in Table 2. These experimental results are provided solely for the purpose of illustrating the effect that the application of the poetic revisers may have on the values of the metrics for given poems.

The resulting scores for this set of *versifiers* show a considerable improvement—almost 40 percentage points on average—with respect to the original scores for the set of negative test samples, as shown in Table 1, even if they do not reach the levels of the positive test

TABLE 3. Example of Poem Produced by the Rhyme-Aware *Versifier*.

Esa documentación fue incautada
por la policía, hace más de
dos años, en una nave que utilizaba
la red corrupta para guardar material
diverso. La fiesta posterior, alguno
de cuyos gastos asumió la trama
según su propia contabilidad,
tuvo lugar en una finca dedicada
a la cría de caballos llamada
los Arcos del Real, propiedad de
un amigo de José María Aznar

TABLE 4. Score for the Poem Produced by the Rhyme-Aware *Versifier*.

SP	LB	CR	AR	RP	VI	VR	TS
80	78	**26**	**65**	40	97	53	62

Relevant values of the rhyme-related scores are highlighted in bold.
SP, stress placement; LB, line break placement; CR, consonant rhyme; AR, asonant rhyme; RP, rhyme patterns; VI, verse length irregularity; VR, verse length repetition; TS, total score.

samples. Although improvements are possible, the concept of composers as services useful to the poetry composition task can be considered validated in view of these data.

An example of a poem produced by the rhyme-aware *versifier* is given in Table 3, and the corresponding scores are given in Table 4. Note that lines 1, 8, and 9 have consonant rhyme in *ada*—as reflected by a consonant rhyme score of 26—and lines 1, 3 , 6, 8, 9, and 11 have asonant rhyme in *a-a*—as reflected by an asonant rhyme score of 65. This is in marked contrast to the zero scores on both counts reported for news articles not processed by the rhyme-aware *versifier*. The number of rhymes that can be found by this procedure is limited by the number of rhyming words available in the input text.

3.4.2. Target Form. If the specific poetic form of the result has to be constrained a priori to comply with specific preset values for metric parameters, the system can rely on these values to guide its exploration processes. Parameters that can be considered in this way include verse length, poem length in verses, choice of rhymes, or pattern of rhyme schemes.

In these cases, the main problem is that the input text received prove unsuitable for adopting a poetic configuration according to the desired parameters. In these circumstances, the resulting poetic effect after composition is likely to be poor. Subsequent improvement will have to rely on powerful revision procedures.

The following set of *versifiers* allow specification of target values for some of these parameters:

- *Verse Length* splits the given text according to a given target verse length.

TABLE 5. Comparative Scores for *Versifiers* with Different Verse Length Targets (TL) for the Same Initial Population.

TL	SP	LB	CR	AR	RP	VI	VR	TS
5	70	77	42	77	28	96	41	61
7	68	76	41	74	30	94	42	60
8	73	78	37	70	26	93	41	59
11	75	79	24	61	28	96	37	57
14	70	78	23	61	31	92	80	62

SP, stress placement; LB, line break placement; CR, consonant rhyme; AR, asonant rhyme; RP, rhyme patterns; VI, verse length irregularity; VR, verse length repetition; TS, total score.

TABLE 6. Comparative Score for *Versifiers* with Different Number of Verses in Poem (NV).

NV	SP	LB	CR	AR	RP	VI	VR	TS
50	65	75	30	60	34	96	70	61
40	65	74	30	56	36	94	77	61
30	67	76	18	45	31	95	77	58
20	63	78	18	39	24	95	85	57
10	75	70	14	29	40	80	80	55

SP, stress placement; LB, line break placement; CR, consonant rhyme; AR, asonant rhyme; RP, rhyme patterns; VI, verse length irregularity; VR, verse length repetition; TS, total score.

TABLE 7. Example of Poem Produced by the Fixed Line Length *Versifier*.

(..)
con Gaza, se encontró por
el contrario con un régimen
vecino que introduce armas
de contrabando y organiza
atentados terroristas
contra el vecino estado judío.

- *Number of Verses* computes the number of words in the draft, divides them over the desired number of verses, and inserts line breaks to implement that distribution.

An interesting point to consider is the extent to which the establishment of different values for the parameters affects the score for poetic effect of different input texts. To explore this point, the same initial population of drafts in need of improvement in poetic effect is subjected to several versions of a given *versifier*, each with different values of the relevant parameter.

The average scores for the population of drafts produced by different versions of the *versifier* for different verse length targets, plus the total combined score, are listed in Table 5.

The average scores for the population of drafts produced by different versions of the *versifier* for different targets of number of verses for poem, plus the total combined score, are listed in Table 6.

TABLE 8. Score for the Poem Produced by the Fixed Line Length *Versifier*.

SP	LB	CR	AR	RP	VI	VR	TS
83	82	12	45	16	**95**	**58**	55

Relevant values of the line length–related scores are highlighted in bold.
SP, stress placement; LB, line break placement; CR, consonant rhyme; AR, asonant rhyme; RP, rhyme patterns; VI, verse length irregularity; VR, verse length repetition; TS, total score.

In both cases, scores have improved significantly after the application of the *versifier* with respect to the source text. For both cases, the improvement is comparable with the free-form solution. As expected, none of the strategies for poetic form selection seems to offer significant advantages over the others. And, in all cases, revision procedures would be required to improve results to the levels shown by the target set of samples.

An example of poem produced by the fixe line length *versifier* is given in Table 7, and the corresponding scores are given in Table 8. The target length of the line is fixed at eight syllables. In this case, the length of the original poem makes it impractical to include it completely. Note that lines in positions 1, 2, 3, and 5 have the required number of syllables, and lines in positions 4 and 6 have nine and ten syllables, respectively. The repeated occurrence of lines of eight syllables warrants a high score of verse length irregularity—95—but the proliferation of lines of different length brings the verse length repetition score down to 58. This is due to the fact that instances where the target syllable count cannot be reached exactly produce erratic values around the target length. The overall results are in marked contrast to the zero scores on both counts reported for news articles not processed by the rhyme-aware *versifier*. The scores that can be obtained for this feature are limited by the additive counts of length in syllables of the sub-sequences of tokens present in the input text. The score of nine syllables for the line in position 4 is actually due to a miscount arising from the interaction between two vowels surrounding a *y* that the system incorrectly interprets as a semi-vowel.

4. DISCUSSION

The advantages for a partial reimplementation as services of selected functionalities in the poetry generation domain arise from the need to explore very different combinations of them. The actual argument in favor of this is that many different combinations will have to be considered before a satisfactory model of the poetry generation task is achieved. The issue of whether these additional process of combination might also be automated is open, but it is very likely that many years of hand-crafted explorations of different combinations be required before a successful outcome is achieved. Until this happens, service-based implementations of necessary or useful functionalities will provide the advantage of allowing each researcher to employ them without having to reimplement them from scratch. However, until a shared understanding of the task is reached, each researcher is likely to want to define a combination of his own. This may include the possibility of combining the functionalities available as services with other functionalities that are not yet available as services, or indeed with functionalities that are difficult to implement as services. The nature of the problem is such that it would be foolhardy to forego the advantages of service-oriented solutions, but to the same extent, it would be foolhardy not to consider as part of the solution other functionalities even if they are incompatible with service orientation.

The decision to exemplify the deconstruction procedure with examples of judges and poets and not to include babblers or revisers is due to the additional complexity that would be involved in defining clear interfaces for these cases. A reimplementation of a babbler as a service would correspond to an ngram-based knowledge service that provided a text draft built from a given language model. A choice would have be made on whether to accept arguments specifying which language model to use, or to accept that each implementation of the service would be tied to a given language model, defined internally. A reimplementation of a reviser as a service would correspond to a more elaborate text rewriting service. As described in Section 3.1, this would require definition of a clear interface to indicate what type of revision might be required, or to accept that particular implementations of the service would carry out specific rewriting operations.

Each one of the modules described in Section 3.2 fulfills better than the original modules of the WASP system the requirements for a Web service described in Section 2.1. They are now more loosely coupled than the original evolutionary operators, which makes them more autonomous. They are reusable outside their original context of operation. They can still be composed, as shown by the experiments that play one of them against the other described in Section 3.4. In the case of both services, nonessential information is now abstracted and encapsulated within the implementation, with only functional results being apparent from without the service. Their statefulness is minimal, because neither the service nor any of its clients needs to retain information about the state of the interaction to reach successful outcomes.

From the point of view of the WASP system, the effort to isolate and refine the functionality selected for deployment of services has required a slight redesign of the functionalities involved so that they would provide valuable results beyond the original settings employed for the initial version of the WASP system. The initial versions of the judges were designed to identify and score specific types of line lengths and stanzas that were popular in sixteenth-century Spanish classical poetry: 11 syllable lines known as *endecasílabos* and 8 syllable lines known as *octosílabos*, and stanzas such as *cuartetos* and *tercetos*. The judges redesigned as services are now based on abstractions that capture more generic aesthetic features based on principles of symmetry and repetition, such as several lines having the same length in syllables, or a rhyme occurring more than once in a given stanza. The poets have also been redesigned to include additional strategies such as dividing the text into lines based on possible rhymes rather than just on line length. This opens the door to revision strategies based on lengthening or shortening the line to place an existing rhyme at the end, as alternatives to the original strategy of replacing the word at the end of the line when the rhyme did not match. In addition, calibration procedures have been devised and applied that had not been considered for the original submodules. This is in line with the known advantages of service-oriented development.

These services are not yet available publicly over the Web. To achieve this final stage, an additional research effort would be required, related to two remaining requirements. Some means would have to be established to describe the available functionality in some standard way, possibly in terms of one of the description languages available as Web technologies. This would fulfill the missing requirements of actually having standardized contracts to describe the functionality, and the need for these services to be discoverable.

The poetry evaluation service proposed would be of use to any of the existing systems as an additional evaluation mechanism for their poem outputs, because it provides a numerical score designed to rate the poetic quality of a text beyond particular choices of metrical form. In this sense, it may serve as a way of mining the conceptual space of outputs for possibly valuable results not covered by the particular scoring function being used in each case. The *versifier* service provides means for reconsidering output that is ill-formed with respect to a given target form—lines too long or too short, too many or too few lines—and recast the actual text into a more valuable poetic form. This could help some of the existing systems, those whose procedures are tightly fixed on the line as a construction unit, to reach out into parts of the conceptual space of possible poems that are currently falling outside the scope of their traversal functions. It would also help them to recover instances of valuable or interesting texts that would otherwise have been rejected solely on the grounds of ill-fitting poetic form.

Overall, the large degree of variation in procedure and representation across the set of systems reviewed suggests that all of these approaches have something to contribute, and that progress toward human-level performance in this field may require the combination of several or all of these into a single system. Such an approach would be made significantly more plausible if the various subtasks involved were developed and deployed as services.

The concept of articulation outlined at the beginning of this article may help to refine some of the concepts of computational creativity described in Section 2.2 in the particular context of poetry generation. The process of analysis of the target poems, and the definition of a particular representation, which we have called articulation, can be seen as actually determining the conceptual spaces over which the system is going to search for solutions to the poetry generation task. Sharples's example of defining the universe of concepts in a generative way based on a grammar would correspond to a particular choice for articulation. If the selected articulation is based on ngrams, a different conceptual space would result. If templates or complete verses are chosen as means of articulation, the resulting conceptual spaces would be more restricted. Search would be easier, but coverage of possible resulting poems would also be extremely reduced.

The described services as they stand do not constitute a full-fledged poetry generation system. In terms of the terminology of the FACE model, they can be seen as elements to be combined into one such system. The poet evaluation service described in Section 3.3 can be understood as an esthetic measure, and the *versifier* of Section 3.4 as a concept. The specific results (poem drafts) produced by the *versifier*, paired together with the input text in each case, can be considered as expression of the concept. If these systems were to be combined with further services, or simply invoked by a larger system, they would be acting as third-party contributions to a more complex tuple. In this sense, the composition of Web services could be understood as a means of implementing creative systems described by more complex tuples (possibly with humans acting as third parties in some cases). An example of evolutionary programming in which humans contribute as third parties is the PicBreeder system (Secretan et al. 2011), an evolutionary system for creating visual art where humans play the role of fitness function.

5. CONCLUSIONS AND FURTHER WORK

A set of possible types for services relevant for poetry generation research has been proposed based on a thorough review of existing systems. Within this set of types, two particular services have been proposed as a redesign of original functionality of the WASP poetry generator. Although the actual implementations are very simple and do not exploit the full possibilities of the original model, they show the feasibility of using service-oriented computing to combine different functionalities in the proposed way. The deconstruction process has uncovered a number of issues that need to be faced when undertaking this type of effort.

First, existing decompositions of a system into separate modules may not be optimal for their implementation as Web services. To fulfill the requirements imposed by a service-oriented architecture, a different breakdown may need to be designed. The concept of articulation, both in terms of representation of poetry and in terms of system analysis, needs to be considered as a guiding principle during this process of redesign.

Second, the publication of services of this type requires the development of appropriate means of describing their functionality in a way that allows the establishment of standardized contracts and a discoverable interface. The main challenge to be faced here is the development of a language rich enough and expressive enough to capture the subtle nuances involved in the description of poetry in a formal format that can be interpreted by machines. For the particular domain of poetry generation, such means had not been discovered at the time of submission of this article, although an initial contribution was attempted by Díaz-Agudo (2002). The development of specific ontologies for the domain of poetry, and their formalization in standard languages and their public deployment in services such as OntoHub (Kutz et al. 2014), would be a significant step forward in this sense. This is considered as an interesting avenue for further work.

Third, some of the functionalities that need to be contemplated for the adequate evaluation of novelty in creative generation, such as keeping a record of prior results to avoid repetition or production of unoriginal results, are incompatible with the requirement to minimalize statefulness expected of Web services. By definition, a procedure for checking whether each subsequent result is similar to previous results involves some way of keeping track of the accumulated state of the system either in terms of outputs already seen or in terms of solutions already attempted. This is considered undesirable behavior in a service-oriented architecture. Additionally, it would incur difficulties related to distinguishing between what Boden (1990) describes as *p-creativity*—which in this context might correspond to a perception of novelty particular to a given client but possibly not apparent to the service provider—and *h-creativity*—which would correspond to novelty in more absolute terms, possibly defined in terms of every service provided by the system, and therefore inapprehensible to specific clients who have only seen results served to them.

In summary, the deconstruction of creative systems into service-oriented solutions would be valuable in several respects. It would make available to the general community of researchers each and every effort at tackling particular aspects of the domain in question, thereby broadening the view of the community. It would promote joint exploration of several approaches in tandem, as opposed to the current tendency toward fragmentation of research effort into simple solution focused very tightly on particular simplifications of the domain. Under the circumstances, further work will be devoted to addressing this vision at different levels, from pressing on with the development of services based on existing solutions to the integration of existing creative services into existing working prototypes for particular tasks.

ACKNOWLEDGMENTS

The research reported in this article is partially supported by the Ministerio de Educación y Ciencia (TIN2009-14659-C03-01). The project ConCreTe acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 611733. The author wants to thank the anonymous reviewers for the helpful comments that have led to significant improvements.

REFERENCES

AAMODT, A., and E. PLAZA. 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications, **7**(1): 39–59.

AGIRREZABAL, M., B. ARRIETA, M. HULDEN, and A. ASTIGARRAGA. 2013. POS-tag based poetry generation with WordNet. *In* Workshop on Natural Language Generation (ACL 2013) (14th ed.).

BARBIERI, G., F. PACHET, P. ROY, and M. D. ESPOSTI. 2012. Markov constraints for generating lyrics with style. *In* Ecai. *Edited by* L. D. RAEDT, C. BESSIRE, D. DUBOIS, P. DOHERTY, P. FRASCONI, F. HEINTZ, and P. J. F. LUCAS, Vol. 242 of Frontiers in Artificial Intelligence and Applications. IOS Press: Amsterdam, the Netherlands, pp. 115–120.

BODEN, M. 1990. Creative Mind: Myths and Mechanisms. Weidenfeld & Nicholson: London.

CARDOSO, A., T. VEALE, and G. A. WIGGINS. 2009. Converging on the divergent: the history (and future) of the International Joint Workshops in Computational Creativity. AI Magazine, **30**(3): 15–22.

CHARNLEY, J., S. COLTON, and M. T. LLANO. 2014. The FloWr framework: automated flowchart construction, optimisation and alteration for creative systems. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 315–324.

COLTON, S., J. CHARNLEY, and A. PEASE. 2011. Computational creativity theory: the face and idea descriptive models. *In* 2nd International Conference on Computational Creativity, Mexico City, Mexico, pp. 90–95.

COLTON, S., J. GOODWIN, and T. VEALE. 2012. Full-FACE poetry generation. *In* Proceedings of the International Conference on Computational Creativity 2012, Dublin, Ireland, pp. 95–102.

COLTON, S., and G. A. WIGGINS. 2012. Computational creativity: the final frontier? *In* Ecai. *Edited by* L. D. RAEDT, C. BESSIÈRE, D. DUBOIS, P. DOHERTY, P. FRASCONI, F. HEINTZ, and P. J. F. LUCAS, Vol. 242 of Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 21–26.

DAI, W., P. MOYNIHAN, J. GOU, P. ZOU, X. YANG, T. CHEN, and X. WAN. 2007. Services oriented knowledge-based supply chain application. *In* Proceedings of the 2007 IEEE International Conference on Services Computing, Salt Lake City, UT, pp. 660–667.

DAS, A., and B. GAMBÄCK. 2014. Poetic machine: computational creativity for automatic poetry generation in Bengali. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 230–239.

DÍAZ-AGUDO, B. 2002. Una aproximación ontológica al desarrollo de sistemas de razonamiento basado en casos. Ph.D. Thesis, Universidad Complutense de Madrid, Madrid, Spain.

DIETRICH, A. J., S. KIRN, and V. SUGUMARAN. 2007. A service-oriented architecture for mass customization—a shoe industry case study. IEEE Transactions on Engineering Management, **54**(1): 190–204.

DING, H., and I. SØLVBERG. 2004. Exploiting extended service-oriented architecture for federated digital libraries. *In* Proceedings of the 7th International Conference of Asian Digital Libraries, Shanghai, China, pp. 184–194.

DUAN, Z., S. BOSE, P. A. STIRPE, C. SHONIREGUN, and A. LOGVYNOVSKIY. 2005. SOA without Web services: a pragmatic implementation of SOA for financial transactions systems. *In* Proceedings of the 2005 IEEE International Conference on Services Computing, Orlando, FL, pp. 243–250.

ENSOR, P. 1988. The functional silo syndrome. AME Target, **16**: 16.

GERVÁS, P. 2000. WASP: evaluation of different strategies for the automatic generation of Spanish verse. *In* Proceedings of the AISB-00 Symposium on Creative & Cultural Aspects of AI, Birmingham, UK, pp. 93–100.

GERVÁS, P. 2001. An expert system for the composition of formal Spanish poetry. Journal of Knowledge-Based Systems, **14**(3–4): 181–188.

GERVÁS, P. 2013a. Computational modelling of poetry generation. *In* Proceedings of the AISB13 Symposium on Artificial Intelligence and Poetry, Exeter, UK, pp. 11–16.

GERVÁS, P. 2013b. Evolutionary elaboration of daily news as a poetic stanza. *In* Proceedings of the IX Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados - MAEB 2013, Madrid, Spain, pp. 229–238.

GONÇALO OLIVEIRA, H. 2012. PoeTryMe: a versatile platform for poetry generation. *In* Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence, C3GI 2012, Montpellier, France, pp. 16–24.

JONES, S. 2005. Toward an acceptable definition of service. IEEE Software, **22**(3): 87–93.

KRANJC, J., V. PODPECAN, and N. LAVRAC. 2012. Clowdflows: A cloud based scientific workflow platform. *In* ECML/PKDD (2). *Edited by* P. A. FLACH, T. D. BIE, and N. CRISTIANINI, Vol. 7524 of Lecture Notes in Computer Science. Springer: Heidelberg Berlin, pp. 816–819.

KURZWEIL, R. 2001. Ray Kurzweil's Cybernetic Poet. Available from: http://www.kurzweilcyberart.com/poetry/rkcp_overview.php. Accessed July 3, 2013.

KUTZ, O., F. NEUHAUS, T. MOSSAKOWSKI, and M. CODESCU. 2014. Blending in the hub: towards a collaborative concept invention platform. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 297–305.

MAEDA, J. 2001. Design by Numbers. MIT Press: Cambridge, MA.

MAHAJAN, R. 2006. SOA and the enterprise – lessons from the city. *In* Proceedings of 2006 IEEE International Conference on Web Services, Chicago, IL, pp. 939–944.

MANURUNG, H. M. 1999. Chart generation of rhythm-patterned text. *In* Proceedings of the First International Workshop on Literature in Cognition and Computers, Tokyo, Japan, pp. 15–19.

MANURUNG, H. M. 2003. An evolutionary algorithm approach to poetry generation, Ph.D. Thesis, University of Edinburgh, Edinburgh, UK.

MANURUNG, R., G. RITCHIE, and H. THOMPSON. 2012. Using genetic algorithms to create meaningful poetic text. Journal of Experimental & Theoretical Artificial Intelligence, **24**(1): 43–64.

MISZTAL, J., and B. INDURKHYA. 2014. Poetry generation system with an emotional personality. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 72–81.

MONTFORT, N., R. P. PÉREZ, D. F. HARRELL, and A. CAMPANA. 2013. Slant: a blackboard system to generate plot, figuration, and narrative discourse aspects of stories. *In* Proceedings of the Fourth International Conference on Computational Creativity. *Edited by* M. L. MAHER, T. VEALE, R. SAUNDERS, O. BOWN, and A. SYDNEY, pp. 168–175.

OLIVEIRA, H., R. HERVÁS, A. DÍAZ, and P. GERVÁS. 2014. Adapting a generic platform for poetry generation to produce Spanish poems. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 63–71.

OULIPO. 1981. Atlas de littérature potentielle, Collection Idées, vol. 1. Gallimard: Paris, France.

PAPAZOGLOU, M. 2003. Service-oriented computing: concepts, characteristics and directions. *In* Proceedings of the Fourth International Conference on Web Information Systems Engineering, Rome, Italy, pp. 3–12.

PEASE, A., and S. COLTON. 2011. Computational creativity theory: inspirations behind the face and the idea models. *In* 2nd International Conference on Computational Creativity, Mexico City, Mexico, pp. 72–77.

QUENEAU, R. 1961. 100.000.000.000.000 de poèmes, Gallimard Series. Schoenhof's Foreign Books, Incorporated. Gallimard: Paris, France.

QUILIS, A. 1985. Métrica española. Ariel: Madrid, Spain.

RASHEL, F., and R. MANURUNG. 2014. Pemuisi: a constraint satisfaction-based generator of topical Indonesian poetry. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 82–90.

REITER, E., and R. DALE. 2000. Building Natural Language Generation Systems. Cambridge University Press: Cambridge, UK.

SECRETAN, J., N. BEATO, D. B. D'AMBROSIO, A. RODRIGUEZ, A. CAMPBELL, J. T. FOLSOM-KOVARIK, and K. O. STANLEY. 2011. Picbreeder: a case study in collaborative evolutionary exploration of design space. Evolutionary Computation, **19**(3): 373–403.

SHARPLES, M. 1996. An account of writing as creative design. *In* The Science of Writing: Theories, Methods, Individual Differences, and Applications. *Edited by* C. M. LEVY and S. RANSDELL. Lawrence Erlbaum: Hillsdale, NJ.

SHARPLES, M. 1999. How We Write: Writing as Creative Design. Routledge: London.

SMEDT, T. D., L. NIJS, and W. DAELEMANS. 2014. Creative Web services with pattern. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia, pp. 344–346.

TOIVANEN, J. M., O. GROSS, and H. TOIVONEN. 2014. The officer is taller than you, who race yourself! Using document specific word associations in poetry generation. *In* 5th International Conference on Computational Creativity, Ljubljana, Slovenia.

TOIVANEN, J. M., M. JÄRVISALO, and H. TOIVONEN. 2013. Harnessing constraint programming for poetry composition. *In* Proceedings of the International Conference on Computational Creativity 2013, Sydney, Australia, pp. 160–167.

TOIVANEN, J. M., H. TOIVONEN, A. VALITUTTI, and O. GROSS. 2012. Corpus-based generation of content and form in poetry. *In* Proceedings of the International Conference on Computational Creativity 2012, pp. 175–179.

VEALE, T. 2013a. Less rhyme, more reason: Knowledge-based poetry generation with feeling, insight and wit. *In* Proceedings of the International Conference on Computational Creativity 2013, Sydney, Australia, pp. 152–159.

VEALE, T. 2013b. A service-oriented architecture for computational creativity. Journal of Computing Science and Engineering, **7**(3): 159–167.