Cogn Comput (2016) 8:187–203
DOI 10.1007/s12559-015-9338-8

Computational Drafting of Plot Structures for Russian Folk Tales

Pablo Gervás[1]

Received: 2 November 2014 / Accepted: 15 May 2015 / Published online: 27 May 2015
© Springer Science+Business Media New York 2015

Abstract The plots of stories are known to follow general patterns in terms of their overall structure. This was the basic tenet of structuralist approaches to narratology. Vladimir Propp proposed a procedure for the generation of new tales based on his semi-formal description of the structure of Russian folk tales. This is one of the first existing instances of a creative process described procedurally. The present paper revisits Propp's morphology to build a system that generates instances of Russian folk tales. Propp's view of the folk tale as a rigid sequence of character functions is employed as a plot driver, and some issues that Propp declared relevant but did not explore in detail—such as long-range dependencies between functions or the importance of endings—are given computational shape in the context of a broader architecture that captures all the aspects discussed by Propp. A set of simple evaluation metrics for the resulting outputs is defined inspired on Propp's formalism. The potential of the resulting system for providing a creative story generation system is discussed, and possible lines of future work are discussed.

Keywords Computational creativity · Computational narratology · Storytelling · Knowledge representation

Introduction

The concept of plot of a story is a useful abstraction. It refers to the skeleton of the story, its elementary structure, and how the material in it all comes together into a single, coherent whole. Everybody has intuitions on what is involved in such a plot: a sequence of events presented in a particular order and related to one another in such a way that if one were left out it would be missed and if an extra one were added it would be out of place. Of particular importance is the fact that it should all lead to an end point which rounds off the whole and that any issues raised during the sequence should have been resolved before the end. With this sketch of the concept of story plot in mind, the issues of how these plots come about, and which computational procedures might provide a good model of how to produce them becomes an interesting research question. The interest arises from the fact that such story plots are themselves artifacts of great potential in terms of their applicability if they can be produced automatically for a given set of constraints, and because the task of story construction is a relevant instance of the human ability to create novel artifacts of value.

The construction of stories is a cognitive task that is fundamental to the way humans understand the world and attempt to influence it. Yet for centuries, work on the analysis of narrative has focused on its structural properties and its semiotic nature, rather than the cognitive processes that lead to its emergence. The structural school of narratology focused on the analysis of literary works in terms of their structure and the larger structures they are part of. It is only recently [13] that attention has turned to considering how the human storytelling ability relates to human cognition. Pioneering work on automated storytelling dating back to the beginnings of artificial intelligence as a discipline had already considered dynamic modelling of the processes involved in creating stories [5, 15, 18, 19]. These efforts focused on making the most of available computational techniques—such as logic, planning, or case-based reasoning—to obtain story-like outputs. The idea to exploit structuralists account of narrative in computational systems

✉ Pablo Gervás
pgervas@ucm.es

[1] Instituto de Tecnología del Conocimiento, Universidad Complutense de Madrid, 28040 Madrid, Spain

capable of generating stories is not new. Propp's account of the morphology of the Russian folk tale [22] has the advantage of being simple, intuitive, and formally tractable. Many attempts have been made to use it as underlying theory for automated story generation [6–9, 12, 17]. The need to explore further the combinations of artificial intelligence and narratology has been defended elsewhere [11]. The present paper provides a description of the Propper system for the generation of Russian folk tales based on the morphology of the folk tale by Vladimir Propp. Within the computational framework provided by this system, the paper explores the particular question of how the dependency relations between different elements of a plot and the constraints imposed on the element that occurs at the very end of a plot—the ending—have to be taken into consideration when designing a constructive procedure for plots. These two questions are relevant to storytelling in a broader context, and the intuitions uncovered during empirical testing may be applicable beyond the Propper system and beyond the particular view of narrative structure proposed by Propp.

Previous Work

Before the proposed system can be described, a number of issues addressed by previous work must be presented: basic elements of Propp's morphology, Propp's description of how his morphology could be used to generate stories, a brief review of existing automated storytellers as relevant to this effort, and some basic points on computational creativity.

Elements of Propp's Formalism Relevant for Computational Implementation

Propp [22] set out to study a subset of a corpus of Russian folk tales collected by Afanasiev and concentrated on 100 of those tales to carry out this study. Over these tales, he identified a set of regularities in terms of character functions, understood as acts of the character, defined from the point of view of their significance for the course of the action. He concluded that, for the given set of tales, the number of such functions is limited, the sequence of functions was always identical, and all these folk tales could be considered instances of a single structure, an archetype of a folk tale.

The collection of tales that Propp focuses on involves stories built on combinations of a number of narrative ingredients: a protagonist sets out on a journey, usually triggered by a lack in his immediate environment or a villainy performed upon it, faces a villain, and in the process gets helped by a magical agent. A possible complication considered is the presence of an additional character that competes with the protagonist for the role of hero of the story, which involves additional ingredients such as a gradual unveiling of the hero's real role in the story, from initial presentation in disguise to the obtention of a reward towards the end, and usually involving recognition as a result of success on a difficult task.

The two cornerstones of Propp's analysis of Russian folk tales are a set of roles for characters in the narrative (which he refers to as *dramatis personae*) and a set of character functions. These two concepts serve to articulate the morphology as an account of the elementary structure of the tales. Both of these concepts are constructed specifically for the family of tales being considered. Therefore, the set of roles includes fundamental elements such as the hero (who sets out on a journey), the dispatcher (who dispatches the hero on his journey), the villain (that the hero faces during the story), the donor (who provides the magical agent to the hero), and the false hero (who competes with the protagonist for the role of hero of the story). The set of character functions includes a number of elements that account for the journey, a number of elements that detail the involvement of the villain—including the villainy itself, some possible elaborations on the struggle between hero and villain, and a resolution—a number of elements that describe the dispatching of the hero, a number of elements that describe the acquisition of a magical agent by the hero, and a number of elements concerned with the progressive unveiling of the hero's role in opposition to the false hero.

The sequence of character functions described by Propp is supposed to apply to all stories of the type described, so that any story will include character functions from this sequence appearing in the given order. With respect to the relative ordering, some deviation allowed in that tales may depart from it by shifting certain character functions to other positions in the sequence.

Character functions in a given narrative are related to one another by long-range dependencies related to motivation and coreference. Propp's analysis of this point has been discussed in [9]. These links are mostly concerned with particular instantiations of certain character functions being linked to instantiations of character functions that went before them. This is one of the ways in which overall coherence of the tale can be ensured: characters kidnapped at the beginning are freed towards the end, and so on. A computational procedure must take these links into account when deciding which characters to assign to particular roles in each new character function added to a story. If the sister of the hero was bewitched at the start, it is she that needs to be released from the spell towards the end.

Character functions are so named because, in Propp's understanding, they represent a certain contribution to the development of the narrative by a given character. When he talks about the set of characters of the story (or *dramatis personae*), Propp constantly reoccurs to a set of labels to describe particular roles played by characters in tales. They are gathered together in chapter VI where he discusses the distribution of functions among dramatis personae. For simplicity, I will refer to these as *role names*, though Propp does not. Some examples of these roles are as follows: the *villain*, the *donor* (who provides the hero with a magical agent), the *helper* (usually a magical agent, that helps the hero carry out his tasks), the *dispatcher* (who sends the hero on his mission), the *hero* (the protagonist of the story), and the *false hero* (who maliciously sets himself up to usurp the protagonist as hero of the story). Propp defines these in terms of the set of character functions that can be grouped around each one of them, as involving the same character. In the description of each character function in chapter III, Propp mentions how the character fulfilling a particular named role is involved in the various actions that can instantiate that character function (the villain carries out the villainy, the dispatcher sends the hero on his mission, the hero departs from home, etc.). If a procedural solution is sought that attempts to model closely the vision of tales that Propp had, these narrative roles must be explicitly defined, and some means of explicitly defining their participation in each type of character function should be provided, to ensure that these participations are instantiated by particular characters in a coherent manner throughout the tale.

Propp's Description of Tale Generation

Propp provides in his book a very clear description of how his morphology could be used for story generation:

> In order to create a tale artificially, one may take any *A*, then one of the possible *B*'s then a *C* ↑, followed by absolutely any *D*, then an *E*, the one of the possible *F*'s, then any *G*, and so on. In doing this, any elements may be dropped, or repeated three times, or repeated in various forms. If one then distributes functions according to the dramatis personae of the tale's supply of by following one's own taste, these schemes come alive and become tales. Of course, one must also keep motivations, connections, and other auxiliary elements in mind p. 111–112

In addition to this clearly procedural description he provides a number of constraints that a potential storyteller should obey and an enumeration of the points where a storyteller has freedom to decide.

The constraints on the story teller are as follows:

1. "The storyteller is constrained (...) in the overall sequence of functions, the series of which develops according to the above indicated scheme". p. 112
2. "The storyteller is not at liberty to make substitutions for those elements whose varieties are connected by an absolute or relative dependence". p. 112
3. "In other instances, the storyteller is not free to select certain personages on the basis of their attributes in the event that a definite function is required". p. 112

The points where Propp considers that a storyteller has a certain freedom are as follows:

1. "In the choice of those functions which he omits, or, conversely, which he uses" p. 112
2. "In the choice of the means (form) through which a function is realized". p. 112
3. in the assignment of story characters to particular slots in functions: "If one then distributes functions according to the dramatis personae of the tale's supply or by following one's own taste, these schemes come alive and become tales" p. 111–112 and "The storyteller is completely free in his choice of the nomenclature and attributes of the dramatis personae. Theoretically the freedom here is absolute". p. 112–113
4. "The story teller is free in his choice of linguistic means". p. 113

On the third point, Propp follows on to discuss in rather vague terms that people do not make wide use of this freedom, preferring to let personages recur much as functions do. So there is a typical villain and a typical donor. Given the level of uncertainty involving this description, it has been decided not to consider it in the present system. The fourth point surely underlies Propp's decision not to address linguistic issues in his morphology at all. We follow this decision in deciding not to address the linguistic rendering of the tales in the initial implementation of our system.

The remaining insights are considered in a computational implementation in "The Propper System: A Computational Solution for Proppian Story Generation" section.

A different point to consider is whether a sequence of functions generated in this way allows for a story with a satisfactory ending. This important point was not considered in detail by Propp, possibly due to the fact that his main goal was to propose an analytical framework to help classify folk tales. The proposal of a related generative procedure was a side product, and Propp never considered the problem of when to end a story. From a computational point of view, however, the need for a clear stopping condition on the construction procedure is paramount.

The most relevant mentions of endings in Propp's book occur in pages 58 ("A great many tales end on the note of rescue from pursuit".) and 64 (on the subject of the reward / marriage character function: "At this point the tale draws to a close".).

Existing Automated Storytellers

There have been several attempts to use Propp's formalism as a basis for story generation. However, with only one exception [9], most of these attempts either were loosely inspired by Propp (Lang's Joseph system [17], Turner's MINSTREL system [27]) or relied on the part of Propp's framework designed for analysing / describing folk tales, which they used to specify the building blocks for their systems but then combined with additional constructive techniques that had not been considered by Propp (such as case-based reasoning or interactive storytelling [6–8, 12]. Gervás [9] provides more detailed argumentation of how these various storytelling systems differ from Propp's description of the generative procedure he proposed based on his analytical framework.

An important concept related to the implementation of narrative systems is that of story actions as operators that change the world. Actions in a story are applicable if certain conditions hold in the state of the world before they happen, and after they happen they change the state of the world. This idea has been represented by defining actions with an associated set of preconditions and another of postconditions or effects. This approach to defining actions is important because it constitutes a possible way of capturing the causal dependencies that constitute a fundamental ingredient of narrative as it is understood by people [26]. It has become popular in story generation through the numerous research efforts that use planning techniques [1, 14, 23], which are inherently based on this concept. Even systems based on alternative generation technologies include the possibility of associating pre- and postconditions to actions, such as the information on emotional links between characters considered in the MEXICA system [21] or the preconditions added to the representation of actions in the Joseph system [17].

Computational Creativity

Wiggins [28] takes up Boden's idea of creativity as search over conceptual spaces [2] and presents a more detailed theoretical framework intended to allow detailed comparison, and hence better understanding, of systems which exhibit behaviour which would be called creative in humans. This framework describes an exploratory creative system in terms of a septuple of elements, which include elements for defining a conceptual space as a distinct subset of the universe of possible objects, the rules that define a particular subset of that universe as a conceptual space, the rules for traversing that conceptual space, and an evaluation function for attributing value to particular points of the conceptual space reached in this manner. Wiggins goes on to provide refinements that cover issues such as the differences between exploratory and transformational creativity expressed in terms of this framework. However, the systems considered in this paper are far too humble to require these refinements.

Ritchie [24] addresses another important issue in the development of creative programmes, that of evaluating when a programme can be considered creative. He does this by outlining a set of empirical criteria to measure the creativity of the programme in terms of its output. He makes it very clear that he is restricting his analysis to the questions of what factors are to be observed, and how these might relate to creativity, specifically stating that he does not intend to build a model of creativity. Ritchie's criteria are defined in terms of two observable properties of the results produced by the programme: *novelty* (to what extent is the produced item dissimilar to existing examples of that genre) and *quality* (to what extent is the produced item a high-quality example of that genre). To measure these aspects, two rating schemes are introduced, which rate the typicality of a given item (item is typical) and its quality (item is good). Another important issue that affects the assessment of creativity in creative programmes is the concept of *inspiring set*, the set of (usually highly valued) artifacts that the programmer is guided by when designing a creative programme. Ritchie's criteria are phrased in terms of: what proportion of the results rates well according to each rating scheme, ratios between various subsets of the result (defined in terms of their ratings), and whether the elements in these sets were already present or not in the inspiring set.

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 [4, 20] presents a framework to understand creative acts performed by software. It defines a creative act as a non-empty 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 aesthetic measure, or framing information. A *concept* is a procedure which 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 *aesthetic measure* is a function which 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 artefacts (generating instances of them) and processes (generating methods for producing instances).

The Propper System: A Computational Solution for Proppian Story Generation

The Propper system is composed by a set of modules, each addressing one of the decision points where Propp considers that a storyteller has a certain freedom of choice to exercise:

- A *plot driver generator* is in charge of building the sequence of character functions for a tale. It addresses the first freedom point in Propp's enumeration, and it must be guided by constraints 1 and 2.
- A *fabula generator* is in charge of instantiating the character functions in the resulting sequence with particular story actions, ensuring that they work well as a sequential discourse. It addresses freedom point 2, and it must satisfy the requirements of appropriate linking.
- A *casting module* is in charge of assigning particular characters to the arguments of the selected story actions. It addresses freedom point 3, and it must respect constraint 3.
- A *textual rendering module* is in charge of converting the final conceptual plan for a story into text.

The computational solution described in this paper has been partially implemented as a working prototype written in Java and operating over a small set of resources defined as plain text files. The partial implementation available so far includes full operational versions of the plot driver generator and the fabula generator and baseline solutions for the casting and the textual rendering module. Of these, only the plot driver generator is reported in the present paper. This development involved a very small effort of simple coding of the overall algorithmic procedures, but a considerable effort of knowledge engineering over the set of resources.

Knowledge Representation

The first step for considering Propp's formalism as a computational procedure would be to define specific representations for the concepts involved. In the description of Propp's formalism given in "Elements of Propp's Formalism Relevant for Computational Implementation" section, we have relied on two different concepts that would need to be assigned a conceptual representation: character functions, which are the basic ingredients handled by the plot driver generator, and story actions, which are the basic ingredients handled by the fabula generator.

Character Functions

The plot driver generator relies on the following specific representations for the concepts involved:

- A *character function*, a label for a particular type of acts involving certain named roles for the characters in the story, defined from the point of view of their significance for the course of the action
- A *plot driver*, a sequence of character functions chosen as backbone for a given story

Story Actions

To convert a sequence of character functions into a story, each of these functions must be instantiated with a particular story action. These *story actions* involve a number of *predicates* that describe events with the use of *variables* that represent the set of characters involved in the action.

To fully capture Propp's restrictions (constraint 3), story actions will also include non-narrative predicates which encode constraints on the specific choice of dramatis persona that can fill particular argument slots in the predicates of the story action; for instance, the fact that the author of a villainy must be the villain.

The set of story actions available for instantiating a given character function, as defined by Propp, includes several variants concerning the form of the action. For instance, a villainy can take the form of kidnapping a person, seizing a magical agent, ruining the crops, etc. Each of these would be represented in our proposal by an action with a set of preconditions and a set of postconditions. To keep track of the effects of these actions as they are added to the story, some form of representation of the context must be employed. As the simplest possible solution, a representation of the context is considered as a set of states, each one representing the state of the world before a certain story action took place. A *state* of the world is represented as a set of predicates describing the facts that hold in that state. The sequence of states for a given story we call a *fabula*.

We represent a *story action* as a set of predicates that describe an instance of a character function. Links with preceding story actions are represented as dependencies of the story action with predicates that need to have appeared in previous story actions (preconditions). Therefore, a story action involves a set of preconditions (predicates that must be present in the context for continuity to exist) and a set of postconditions (predicates that will be used to extend the context if the action is added to it). Some additional predicates not corresponding to events in the story are added to encode the sphere of action to which each story action belongs. These predicates explicitly link the corresponding narrative role to a particular variable in the story action. The predicates in a story action are defined over free variables as arguments. This ensures that relative instantiation of the various arguments in the predicates of a story action is coherent, as discussed later. Table 1 includes examples of story actions linked by preconditions.

Each successive state in a fabula contains all the predicates arising from the preceding actions that have not been retracted by a story action since they occurred. This is difficult to read. Also it is difficult to define over such a structure significant metrics on measures such as number of predicates in which a certain character appears (which we will need to consider when measuring the structural quality of a story). For this purpose, we define a final structure called a *flow* for a story, which is simply an ordered sequence of all the predicates in the fabula, such that each one appears only once, and grouped into subsets according to the particular state of the fabula in which they were first introduced.

The Overall Architecture

Based on this representation, the procedure originally sketched by Propp can be subdivided into the following stages, each one of which will be addressed by a different module in our proposed system:

- employ an algorithmic procedure for generating a sequence of character functions considered valid for a tale (*plot driver generator*)
- given a valid sequence of character functions, progressively select instantiations of these character functions in terms of story actions (*fabula generator*)
- given a fabula where all variables have been replaced by constants, produce a flow for the story (*flow generator*).

For each of these stages, a computational decision procedure must be selected. We are considering a possible computational implementation. For this purpose, we intend to consider in the first instance the simplest representation and the simplest procedures compatible with acceptable results. To this end, a number of computational options for some of these modules have been considered, together with a knowledge engineering effort to produce the required resources. The results have been empirically tested for fulfilment of Propp's constraints. The following sections report on the development, the evaluation procedures, and the results of the tests.

Given that the development effort has focused at a very abstract level of representation, evaluation has to be considered at a corresponding level to provide valid feedback for the improvement of the system. As the linguistic modelling of the stories has not been addressed, evaluation by human volunteers is plagued with difficulty. Introducing some kind of rapidly constructed stage for rendering the results as text by providing text templates for each story action (as done in some existing story generators [21]) is likely to introduce noise in terms of elements present in the text and not necessarily produced by the system. Asking human evaluators to rate the quality of an abstract representation as produced by the system runs the risk of judgements being clouded by the difficulty of interpreting the representation.

Additionally, evaluations by humans necessarily have to be restricted to a small number of instances of system output. The choice of which particular instances to test is left to the designer of the experiment, and there is a risk of focusing on examples that are not representative of system performance overall.

As an alternative, quantitative procedures have been defined to measure the specific qualities desired for each stage of the representation, at a corresponding abstract level. These procedures can be applied to a large number of system results, providing a measure of the quality of system output at the working level of abstraction and applicable to a broad range of system results, leaving no doubt as to their significance over the complete set of outputs.

The Propper Plot Driver Generator Module

The Propper systems rely on Propp's generative procedure for story construction as a blueprint for a computation solution to story generation.

The plot driver generator module operates on the following inputs:

- A reference sequence of character functions.

Table 1 Examples of story actions

Character function	*villainy*	*liquidation*
Preconditions	`married H Y`	`married H W`
	`hero H`	`sundered H W`
	`villain X`	**`hero H`**
Action	`makes_disappear X Y`	`resume_marriage H W`
Postconditions	**`victim Y`**	
	`sundered H Y`	

- A set of dependencies identified between character functions in the sequence.
- A selection of character functions that have been identified as likely options to end a story.

The constructive procedure for generating plot drivers traverses the reference sequence of character functions deciding at each point whether to add the character function under consideration to the draft of the plot driver. Further details on this constructive procedure are presented in "Computational Drafting of Plots: The Propper Plot Driver Generator Module" section .

The Propper Fabula-Flow Generator Module

The construction procedure for generating fabulae takes as input a plot driver, a set of story actions, and a mapping between character functions and story actions. It generates a fabula as a succession of states described by predicates, which is then converted into a flow, which lists a sequential discourse of predicates describing the story at a conceptual level. Both in a fabula and in a flow, characters appearing as arguments in predicates are referred only by a variable name acting as identifier.

A fabula generator receives a plot driver and selects story actions for the character functions given in it. To do this, the fabula generator has to define a fabula, a sequence of states that contain a chain of instances of character functions ideally somehow linked by having their preconditions fulfilled by the context. The initial state by default incorporates all predicates of the first action, and each valid action added to the fabula generates a new state that incorporates all predicates of the previous state, plus the predicates of the new action.

A mapping is established between the set of story actions and the set of character functions, so that each of the available story actions is considered a possible instantiation of a given character function.

To evaluate whether the preconditions of a story action are satisfied by the context, they are unified with the set of predicates that hold in that state. This serves two purposes:

- if the preconditions are not satisfied, an alternative story action will be considered
- unification allows any of the free variables in these preconditions to unify with those in the predicates holding in the fabula

A story action is considered a valid extension of a given fabula if the set of its preconditions can be successfully unified with the predicates in the latest state of the fabula. Once the story action is added, the next state is built by extending the preceding state with the action and the postconditions of the story action.

When the preconditions unify with the state in the fabula, any replacement of free variables in the preconditions is carried over to the rest of the story action before it is added to the context. This ensures that the story action become coherent with the rest of the predicates in the fabula, creating continuity.

The use of unification enables the system to model long-range dependencies between character functions. If the choice for a character function such as liquidation of misfortune or lack depends on which particular story action was chosen to instantiate the character function for lack, this procedure will both block non-appropriate instantiations for liquidation (as their preconditions will not be satisfied) and will ensure the appropriate assignment of variable names to ensure coherence (for instance, that the person that was kidnapped at the beginning be freed towards the end). The additional predicates encoding the sphere of action to which each story action belongs enforce a correct distribution of functions over dramatis personae. Overall, the use of unification models Propp's constraints 2 and 3.

Computational Drafting of Plots: The Propper Plot Driver Generator Module

The main contribution of this paper is the study of how the dependency relations between elements in a story and the particular constraints on the ending of the story affect the way in which story plots can be constructed. To shed light on these issues, we depart from the simplest possible computational implementation of the procedure for tale generation described by Vladimir Propp in his book—described in "Propp's Description of Tale Generation" section—and progressively incorporate additional heuristics where they can be empirically shown to rule out candidate plots that would be less successful in terms of satisfying expected dependency relations and providing a valid ending.

Resources for Plot Driver Generation

A *reference sequence of character functions* has been constructed following the matrix employed by Propp in Appendix III for tabulating his analyses of stories from his corpus. This sequence includes several possible placements of certain character functions in the sequence, to capture the accepted possibilities for inversion. The actual set of character functions employed as canonical sequence is given in Table 2.

Character functions are presented in two columns by their abbreviated name. A key point in the canonical sequence is the **villainy/lack** pair of character functions

Table 2 Set of character functions employed as canonical sequence

test by donor	*difficult task*
hero reaction	*branding*
acquisition magical agent	*victory*
villainy/lack	*task resolved*
hero dispatched	*trigger resolved*
begin counteraction	*return*
acquisition magical agent	*hero pursued*
departure	*rescue from pursuit*
test by donor	*unrecognised arrival*
hero reaction	*unfounded claims*
acquisition magical agent	*false hero exposed*
transfer	*transfiguration*
trigger resolved	*branding*
unrecognised arrival	*villain punished*
unfounded claims	*hero marries*
struggle	

written in bold. These differ from all the others in that only one of them is ever included in any single story, and all stories must contain either one or the other.

Dependencies can be of two types (Table 3). Some dependencies are such that a sequence is only acceptable if both character functions involved are present (for instance, if the hero is tested he has to react, and if he reacts it is because the has been tested, or a struggle and victory, or the setting of a difficult task and its resolution, or a pursuit and a rescue from pursuit). This is equivalent to each character function being a necessary and sufficient condition for the other. But there is also a different type of dependency where the presence of one character function suggests that another one may follow, but that one can occur without the previous one (for instance, if the hero is branded at some stage during the tale, it is very likely that he will be recognised by the brand later in the tale; however, he may also be recognised by some other means). In this case, the first character function is a sufficient (but not necessary) condition for the second one.

Some character functions have several possible dependents. For instance, the presence of character function *unrecognised arrival* (the hero arrives at a new place in disguise) early in the sequence suggest that the character function *unfounded claims* (a false hero tries to claim merit on some of the hero's actions) may appear later, but also *hero recognised* (the hero is recognised and his merits are recognised). This type of dependencies creates difficulties for simple ways of measuring satisfaction of dependencies.

Stopping conditions are crucial in any computational procedure. Story endings are also fundamental in the perception of the quality and the success of a story. In the process of building a sequence of character functions that will give rise to a story, it is important to consider whether the final character function of the sequence is likely to provide support for a satisfactory ending. Propp does not explicitly provide much information on the subject of endings. To obtain guidance on this issue, one must turn to the set of examples of folk tales he considers in his book. By studying these, we can come to some conclusions as to what character functions constitute suitable candidates to end a tale. The examples of tales in Propp's book come in two forms. One is the set of examples of analyses of tales given in Appendix II. The other is the set of schemes for tales tabulated in Appendix III.

Table 3 List of long-range dependencies between character functions: necessary conditions are indicated with a - sign and necessary and sufficient conditions with an = sign

test by donor	=	*hero reaction*
hero reaction	-	*acquisition magical agent*
villainy	-	*trigger resolved*
lack	-	*trigger resolved*
hero dispatched	-	*begin counteraction*
hero dispatched	-	*departure*
begin counteraction	-	*departure*
branding	-	*unrecognised arrival*
branding	-	*hero recognised*
unrecognised arrival	-	*false hero exposed*
unrecognised arrival	-	*unfounded claims*
unrecognised arrival	-	*hero recognised*
unfounded claims	-	*false hero exposed*
struggle	=	*victory*
difficult task	=	*task resolved*
departure	-	*return*
hero pursued	=	*rescue from pursuit*

Data have been collected for these two sources, and the results are presented in Table 4. Propp considers instantiations of his canonical scheme as the elementary unit for tales, which can be combined into more complex stories. Each instantiation of the canonical scheme is considered a move within the larger tale. The table lists both cases where character functions occur at the end of a tale and where character functions occur at the end of a move within a tale. It seems reasonable to assume that moves within a larger tale may finish in a character function that does not support a satisfactory ending. Yet suitability for ending a move may also be a merit in terms of ability to resolve a narrative thread.

Constructive Procedure for Plot Drivers

There are a number of points where the description of the procedure and/or the required operations given by Propp is

Table 4 Frequency data for character functions occurring in final positions for tale examples and schemes: FE are final moves in examples, FS final moves in schemes, AE any move in example, and AS any move in scheme

	FE	FS	AE	AS
hero marries	5	27	7	34
hero recognised	1	1	1	1
villain punished	1			1
acquisition magical agent	1		4	4
return	1	8	2	21
rescue from pursuit		6	1	12
trigger resolved	1		2	
unrecognised arrival			1	1
difficult task			1	
villainy/lack	1		1	5

vague. For this reason, the generative procedure as described by Propp has been extended with additional stages that account for aspects covered by Propp in his analysis but not in his procedure. In the particular case of generating a sequence of character functions, examples of such aspects include: the existence and management of dependencies between character functions and the need for a stopping condition to determine when a satisfactory sequence of character functions has been obtained. As explained above, Propp mentions dependencies in various ways, but he does not go into detail of how they may be treated during generation. On the subject of endings, he says very little. His proposed procedure does implicitly contain a solution: as it involves following the canonical sequence, deciding for each character function whether to include it or not, the procedure ends when the end of the sequence is reached. This will be our baseline solution, but we also want to consider whether more informed solutions might perform better with respect to the potential of the resulting sequence to support a satisfactory ending.

Several heuristics can be considered during the traversal of the canonical sequence in search for character functions to add to the draft of a plot driver to ensure that the final result has a potential for producing good stories.

Dealing with Long-Range Dependencies

The solution we have developed for taking into account long-range dependencies during the construction of plot drivers is based on the identification of possible dependencies between the character function being considered for addition at that point and character functions already in the plot driver draft. Given the set of dependencies for the character function being considered, the options to consider are as follows:

- *redundant* The character function is a follow-up to a character function that appears earlier in the draft but an instance of it has already been added to the draft before this point (resulting from consideration of a previous appearance of this character function in the canonical sequence)
- *incorrect* The character function is a necessary follow-up to a character function that appears earlier in the canonical sequence but which was not selected to include in the draft (if the current character function is included, it would result in incoherent stories)
- *compulsory* The character function is a necessary follow-up to an earlier character function and no previous instance of it occurs
- *optional* Either the character function has no dependencies or it has a weak dependency with character functions already appearing in the draft (so it may be added or not)

For each character function considered as possible addition to an ongoing draft, the procedure identifies the option that it falls under with respect to that draft, includes it if it is compulsory, rejects it if it is redundant or incorrect, and decides at random whether to include it if it is optional.

Dealing with Endings

To address the issue of whether a given plot driver has potential for producing stories with acceptable endings, we need to consider a stopping condition on the traversal of the canonical sequence. If the random selection procedure is applied strictly, the last character function added to the plot driver may not have the potential for a good ending.

Several different heuristics are considered for deciding whether to stop extending the plot driver draft, based on the character function that has been reached and the sequence of character functions already in the draft. The knowledge resources for story generation may be taken into account. The following heuristics are considered:

- a baseline greedy solution that simply stops when the first character function valid for ending is reached
- a baseline non-greedy solution that considers the possibility of exploring beyond the first such cutoff point reached
- a dependency-aware greedy solution that stops when a valid character function valid for ending is reached if all dependencies introduced by character functions in the draft have been closed
- a dependency-aware non-greedy solution that requires valid ending and dependencies to be all closed but considers the possibility of exploring beyond the first such cutoff point reached

Combined Strategies to consider

Specific constructive solutions have to be designed as combinations of the following basic computational tasks:

- a baseline constructive procedure that builds a sequences of character functions by randomly deciding whether or not to include character functions from the canonical sequence in the appropriate order (save for the trigger, which is forcefully included, either as a villainy or as a lack)
- a dependency-aware approach that constrains the addition of character functions as described above
- a stopping condition for the constructive procedure based on the validity of endings and/or the closure of dependencies

In the present paper, the following decisions have been selected for empirical consideration:

- how to decide when to add a particular character function from the sequence
- how to identify points in a draft where stopping might result in a valid ending
- how to decide whether to stop at a particular valid ending point or to continue beyond it in search of one further on

Evaluation of Plot Drivers

Three metrics have been developed to evaluate the quality of plot drivers obtained in this way:

- a metric for conformance to the reference sequence
- a metric for satisfaction of long-range dependencies
- a metric for the potential of a sequence to support a satisfactory ending

Plot driver generators must obey constraint 1 imposing a particular sequence of character functions, as described in "Propp's Description of Tale Generation" section. To establish the extent to which the various implementations fulfil this constraint, a measure of conformance to a reference sequence has been defined. The key measure to consider is, given a certain character function appearing in a candidate plot driver, how many of the functions preceding/following it in the plot driver are contained in the part of the reference sequence that goes before/after (the best scoring of) its appearances in the reference sequence. This value is normalised as a percentage over the length of the plot driver. This measure is 100 if all character functions before and after the one considered have the same relative order in the reference sequence. The measure for a complete driver is taken as the average value for all its functions. This is 100 if the plot driver satisfies perfectly the order in the reference sequence—as described in Table 2 of "Resources for Plot Driver Generation" Section—and degrades towards 0 if some of its character functions appear out of place with respect to the given sequence.

To measure satisfaction of long-range dependencies, we consider a metric that computes the number of dependencies that are actually satisfied out of the set that might have been satisfied. This is done by collecting the set of character functions present in the sequence that may have dependencies with other functions, and for each one, checking whether the character function that it depends upon is present in the sequence (before or after it, depending on the direction of the dependency). Bidirectional dependencies are counted twice if they are not satisfied. To normalise over a large set of tales, the metric currently returns 100 if there are no dependencies or if all dependencies are satisfied, and otherwise a number between 100 and 0 corresponding to the percentage of the dependencies present that have been satisfied.

To measure the potential of a sequence to support a satisfactory ending, we consider a metric that computes whether the sequence ends in a character function that has been recorded to occur at the very end of a tale (not at the end of internal moves). The metric assigns a score of 100 if the last character function is within the collected list, and 0 otherwise.

Evaluation of Strategies for Character Function Addition

Results for the two different strategies for the addition of character functions during generation of character sequences that have been tried are reported in Table 5. Each of the alternative implementations was run 100 times, and values were averaged over the results.

These data show some interesting results. Given that both the strategies employed are based on following Propp's canonical sequence, the fact that they achieve top score on the corresponding metric is no surprise. The baseline predictably gets a low score on satisfaction of dependencies. The dependency-aware strategy for taking long-range dependencies into account achieves very high results on dependency satisfaction as expected. The fact that it does not reach a 100 score is related to the fact that some character functions have multiple dependencies, and the current strategy for addition needs to be refined to consider these cases in more detail. There is also a noticeable increase in the average size of plot drivers. This is because the chance of adding a character function becomes higher than random when dependencies are considered, as the number of cases where character functions have to be added to satisfy dependencies is higher than the number of character functions that have to be left out based on the chosen heuristic.

Table 5 Results for different strategies for addition of character functions during the generation of plot drivers

	Random	Dependency-aware
Plot driver length	13.0	15.5
Conformance	100.0	100.0
Dependencies	43.7	96.3
Endings	61.0	59.0

Examples of sequences of character functions resulting from these strategies are presented in Table 6.

Sequence 1 was produced by the baseline strategy following Propp's procedure strictly. It obtained a score of 100 % on conformance to the canonical sequence, 0 % on dependency satisfaction, and 100 % on potential for satisfactory endings. The low score on dependency satisfaction can be understood seeing that, for instance, the hero is tested and he acquires a magical agent without his reaction to the test being mentioned, the hero is dispatched but he does not actually leave, a victory is mentioned but no preceding struggle is described, a task is resolved without it being set beforehand, the hero is rescued from pursuit without first being pursued, and unfounded claims are made and not resolved. The fact that the sequence ends in a marriage explains the high score on potential for satisfactory endings. But this is the result of random choice rather than a merit of the addition strategy. The fact that the character function in question occurs towards the end of the canonical sequence plays an important role.

Table 6 Examples of sequences of character functions for the two addition strategies

Sequence 1	Sequence 2
lack	*test by donor*
hero dispatched	*hero reaction*
test by donor	*acquisition magical agent*
acquisition magical agent	*villainy*
transfer	*begin counteraction*
branding	*departure*
victory	*test by donor*
task resolved	*acquisition magical agent*
rescue from pursuit	*unfounded claims*
unfounded claims	*struggle*
hero marries	*branding*
	victory
	return
	hero pursued
	rescue from pursuit
	unrecognised arrival
	unfounded claims
	false hero exposed
	transfiguration

Sequence 2 was produced by the strategy imposing satisfaction of all dependencies. It obtained a score of 100 % on conformance to the canonical sequence, unsurprisingly 100 % on dependency satisfaction and 0 % on potential for satisfactory endings. This sequence shows how the imposition of the dependencies forces very coherent sub-sequences of character functions, even though dependencies are stated only in terms of pairs of functions. This is because the pairs sometimes chain up to produce longer subsequences. An interesting example of this is the sequence of *begin counteraction*, *departure*, and *return*—which stretches over a significant part of the story—or the sequence for *test by donor*, *hero reaction* (reaction of the hero to the test), and *acquisition magical agent* (as a result of a positive result to the test), which form a complex interrelated sequence. The sequence reoccurs later in the sequence without the reaction of the hero. This occurs because the reaction of the hero is already present in the preceding draft, and the current version is not capable of taking the relative ordering into consideration. Problems such as these will be addressed in further work. Other examples of pairs of character functions linked by dependencies appearing in this sequence are as follows: *departure-return*, *struggle-victory*, *hero pursued-rescue from pursuit*, *unfounded claims-false hero exposed*. This sequence is a fair example of how plot drivers become longer as a result of the consideration of dependencies. The issue of repetition of character functions is not currently considered by the procedure. The fact that character functions reappear is not necessarily a negative feature. Propp considers that a very common feature of Russian folk tales is the reoccurrence of some event types in sets of three, known as *trebbling*. This might be considered as a further feature for future extensions of the system. The low score on potential for satisfactory endings is explained by the fact that it ends with the character function for *transfiguration*. This occurs late in the canonical sequence but is followed by other character functions more suited to end the tale (namely *hero marries*, representing marriage/reward, which in this particular case happen to have been omitted by the random decision procedure).

Evaluation of Strategies for Stopping Condition of Constructive Procedure

Results for the four different strategies for the stopping condition on the constructive procedure that have been tried are reported in Table 7. Each of the alternative implementations was run 100 times, and values were averaged over the results. All solutions conform 100 % with

Propp's sequence, so results for the conformance metric are not included in the table. As above, results reported for each solution correspond to averages over 100 runs.

The analysis of these results prompts the following observations. The best performer with respect to the metric for valid endings is the baseline greedy approach (BG). This is because it is the most conservative strategy with respect to endings, securing the very first one reached. However, it pays a high price in terms of the metric on dependencies, as many of the dependencies introduced by early character functions get no chance to be resolved due to premature closure of the procedure. It also leads to very short plot drivers. Results reported here indicate a dramatic drop in the length of resulting plot drivers with respect to those for ending agnostic approaches given in Table 6. A slight improvement on these metrics can be obtained by allowing the first valid solution on endings to be skipped, in search of later alternatives. This is what the baseline exploratory approach (BE) does. By skipping the first valid option, the chance of dependencies being resolved increases, with the value on the dependency metric rising to 75.3 %. This is coupled with an increase in results for length of plot driver. In contrast, there is a slight decrease in the metric for satisfactory endings. Because no look ahead and no backtracking are contemplated, this approach runs the risk of skipping the last possible valid ending in search of later alternatives that do not exist. The dependency-aware greedy (DG) approach provides an interesting balance. By considering the dependency-aware strategy for character function addition, it ensures a high value (96.8 %) for the dependency metric. This is comparable with the values obtained when the potential for valid endings is not considered. The price to pay in this case is that some potential candidates for valid endings may be bypassed because at the corresponding point not all dependencies already introduced in the draft have been resolved. In some cases, this leads to situations where the end of the canonical sequence is reached without having found a satisfactory ending. In those cases, the resulting plot driver scores poorly on the metric for valid endings, bringing the average score down to 72.0 %. The dependency-aware exploratory (DE) approach fares no better. By foregoing the opportunity to close on valid solutions, the exploratory version exacerbates the problem. Average score for the metric for valid endings drops further to 60.0 %.

Table 7 Results for different strategies for stopping the procedure during the generation of plot drivers

	BG	BE	DG	DE
Plot driver length	5.7	9.5	16.4	16.2
Dependencies	58.7	75.3	96.8	96.0
Endings	100.0	95.0	72.0	60.0

BG baseline greedy, *BE* baseline exploratory, *DG* dependency-aware greedy, *DE* dependency-aware exploratory

Examples of sequences of character functions resulting from these strategies are presented in Table 8.

Sequence 1 corresponds to a plot driver generated by the baseline greedy approach. All such plot drivers are characterised by the fact that they finish either with a trigger resolved character function or with acquisition of a magical agent character function. This is because those are the first two that occur in the canonical sequence. The choice between one and the other comes about depending on whether the randomness in the procedure reaches one or the other first. But no plot driver can be generated with any other ending beyond one of those character functions. This accounts for the brevity of these plot drivers. The example given scored 67 % on dependencies, because although the character functions for the subsequence of acquiring a magical agent are all present, the plot driver includes earlier instances of *departure* character function—which should be coupled with a return character function—and, more significant, a *villainy*—which should be coupled with a *trigger resolved* character function to achieve the happy ending expected of the genre.

Sequence 2 is produced by the baseline exploratory approach. It has a 92 % score on dependencies and a 0 % score on endings—because it ends oddly with the hero being branded. It has been chosen to illustrate the dangers of the approach. In constructing it, the procedure has bypassed two possible points: one when the *trigger resolved* is reached, and one when the *rescue from pursuit* is reached—as both of these are considered valid potential endings for a tale. It has also skipped other possibilities and has reached the end of the sequence with a *branding* as the last character function added. The less than perfect score on dependencies—in spite the fact that many coupled character functions are included—arises from the fact that the last occurrence of *branding* has no associated character function of *hero recognised*.

Sequence 3 is an instance produced by the dependency-aware greedy approach. Although the approach does sometimes produce less optimal solutions, in this case a good performer has been chosen: this plot driver achieves top scores on all counts. The tale starts with a villainy, the character functions involved can be interpreted as a connected sequence that follows coherently from one to the next, and it ends with the villain being punished and the hero getting married.

Sequence 4 is a result by the dependency-aware exploratory approach. It achieves a top score for dependencies and for endings, and it is longer than previously considered plot drivers. The interesting point about this example is that if the greedy approach had been followed instead of the exploratory one, it might have been closed

Table 8 Examples of sequences of character functions for the different stopping strategies

Sequence 1	Sequence 2	Sequence 3	Sequence 4
villainy	*test by donor*	*villainy*	*test by donor*
begin counteraction	*hero reaction*	*hero dispatched*	*hero reaction*
departure	*acquisition magical agent*	*begin counteraction*	*acquisition magical agent*
test by donor	*villainy*	*departure*	*lack*
hero reaction	*departure*	*test by donor*	*begin counteraction*
acquisition magical agent	*trigger resolved*	*hero reaction*	*departure*
	unfounded claims	*acquisition magical agent*	*trigger resolved*
	branding	*transfer*	*unrecognised arrival*
	return	*trigger resolved*	*unfounded claims*
	hero pursued	*branding*	*difficult task*
	rescue from pursuit	*return*	*branding*
	unrecognised arrival	*unrecognised arrival*	*task resolved*
	unfounded claims	*unfounded claims*	*return*
	false hero exposed	*false hero exposed*	*unrecognised arrival*
	transfiguration	*villain punished*	*unfounded claims*
	branding	*hero marries*	*false hero exposed*
			villain punished

with top scores on reaching character function *return*. The last four-character functions added to the plot driver correspond to an exploratory jump beyond a valid ending already reached, which in this case has paid off.

Discussion

The issue of long-range dependencies between character functions is addressed in the Propper system by two different mechanisms. The first of these is the use of unification/accommodation, which enables the system to partially model long-range dependencies between character functions. If the choice for a character function such as liquidation of misfortune or lack depends on which particular story action was chosen to instantiate the character function for lack, this procedure will both block non-appropriate instantiations for liquidation (as their preconditions will not be satisfied) and will ensure the appropriate assignment of variable names to ensure coherence (for instance, that the person that was kidnapped at the beginning be freed towards the end). An earlier attempt to model dependencies only in this way was shown in Gervás [9]. This captured dependencies correctly when two related character functions appear in a given plot driver (hero sets out and returns). In these cases, the unification process ensures that the choice of story action and the instantiation of its variables satisfy the dependency. However, problems arose when one of the related character functions was missing from the given plot driver, due to the fact that the mechanism for generating the driver did not take them into account. As a result, it could happen that a villain maims a certain victim, but the story neither resolves this villainy nor punishes the villain. The introduction of additional computational means of taking the long-range dependencies into account during construction of the plot driver improves the performance of the system on this aspect. This solution will only translate into successful fabulae or flows if the procedure for instantiating the plot drivers within the fabula generator module is capable of correctly unifying the story action for a dependent character function with the story action already introduced for the depending character function.

The chosen architecture shares with all knowledge-based solutions to plot generation their dependence on the availability of appropriate knowledge. As the paper has strived to demonstrate, there are a number of features of how we think about events and stories that need to be taken into account when building plots. These include the fact that actions have preconditions and effects, the fact that certain types of actions have dependencies with other types of actions that span significant segments of the story, and the fact that certain actions work better as ways to end a story. These facts need to be captured in the form of knowledge that the system can use during the construction process. A significant engineering process is involved in acquiring and encoding these facts, but their impact on the quality of the results has been shown to be significant.

Although the current initiative strove to build a system as faithful as possible to Propp's formalism, all references to Propp's material in the resulting implementation occur only within the set of plain text files that constitute the knowledge resources. The choice and names of the character functions, and the restrictions imposed on how they may combine, are captured jointly in the canonical sequence of character functions used as reference, and the set of long-range dependencies established between character functions. The set of story actions and the relationships between stated in terms of preconditions are also written in a separate text file. The actual Java code that exploits these resources is independent of Propp's particular solution for Russian folk tales. As a result, it would be possible to write an alternative set of resources to use a completely different reference canonical sequence, over elements of a similar nature but which need no longer be called character functions, and which combine according to different rules, or which get instantiated with a completely different set of story actions, and assigned to different characters.

Because of this property, the approach presented in the paper has the potential for being ported to different domains (for instance, to science fiction stories by changing the set of story actions and the set of characters) or adapted to account for different structural analyses of narrative (by changing the reference canonical sequence into one that covers, for instance, Campbell's account of the hero's journey [3] or Lakoff's account of the structure of fairy tales [16]). This is a significant different with other systems based on computational combination/constraint satisfaction.

In terms of the criteria defined by Ritchie, the metrics presented in this paper for the evaluation of sequences of character functions are clearly instances of ratings of typicality, rather than novelty of these sequences. Conformance to a canonical sequence, satisfaction of dependencies, and provision for satisfactory endings constitute valuable features of an acceptable story. In fact, stories are more likely to be considered novel the further away they are from a canonical sequence, or if they allow for some unresolved dependencies, or if they opt for an unconventional ending. In this sense, the framework described in the present paper is unlikely to be considered creative by any standards. Nevertheless, it addresses fundamental issues concerning computational attempts to generate stories.

The framework proposed by Wiggins can be used to analyse the type of creative system that is being considered. The procedures described in this paper for generating sequences of character functions can be understood as defining a conceptual space of sequences of character functions. For each different strategy, the resulting conceptual space is different. The description of the strategy itself constitutes an instance of the traversal function that Wiggins defines to traverse the conceptual space. The definition of the conceptual space is implicit in the description of each strategy, as each one rules out different kinds of sequence of character function. For instance, all the strategies discussed in this paper rule out sequences of character functions that do not conform to Propp's canonical sequence (as shown by the results given in Table 5).

The metrics defined over sequences of character functions constitute instances of evaluation functions as defined by Wiggins. These metrics assign different values to elements of these conceptual spaces. The fact that the metrics are applicable to elements of all the different conceptual spaces is an important insight. In truth, these metrics are defined in such a way that they pick out elements of particular conceptual spaces by assigning high scores to them. In particular, the metric for satisfactory endings assigns top scores to sequences from a particular conceptual space that would only include sequences with satisfactory endings. An additional point of interest is that the metric for satisfactory endings is perfectly applicable to sequences that do not conform to Propp's canonical sequence, and which would therefore be outside the conceptual space of solutions being considered at this particular point.

The described system models the task of drafting plot structures at a level that is more abstract than recent attempts to model creativity at a cognitive level [29], which involve a parallel view of mental computation based on statistical simulation of aspects of memory and perception such as sequence. It is also yet far from the levels of autonomy in terms of personal motivation and social interaction discussed in [25]. Nevertheless, the Propper system represents a computational model of the craft involved in the creation of plot structures, and it extends prior models with consideration of important issues such as long-range dependencies across different part of a plot and the need for appropriate closure.

As mentioned above, the procedures for generating character functions described in this paper exhibit very low indices of creativity. However, they constitute an elementary exploration of what the conceptual spaces are for this particular kind of artefact, what traversal functions can be defined, and what metrics might be useful to capture features that humans consider typical for this domain. Once these basic elements have been established, more elaborate generative procedures may be explored. These can involve systematic exploration of how these basic elements can be progressively modified. The degree of transgression of the modifications considered would likely determine the perception of creativity arising from the results obtained. Along these lines, Propp himself considers some simple transgressions of his framework as possible when he talks about the possibility of having *inverted sequences* (p. 107), where character functions occur out of order but this is not considered a transgression of the basic rule. More elaborate transgressions are likely to lead to conceptual spaces further away from those considered here. As this happens, more refined evaluation functions will be required. This type of progression can be observed between the system presented by Gervás [9] and the one presented in this paper. Where the paper by Gervás addresses the construction of sequences of character functions, the strategies he defines determine a conceptual space of character functions different from those considered here (except for the baseline solution following Propp's procedure strictly, which is similar in both). To the extent that the present paper addresses issues not considered by Gervás [9], such as long-range dependencies and endings, new extensions of the metrics are needed. The conceptual spaces defined by the strategies presented here constitute subsets of the more generic conceptual space defined by the baseline.

With respect to Colton's FACE model, Propp's generative procedure would constitute a *concept* of the process type. The particular implementation of that procedure described here would constitute an *expression of that concept*, different from the expression of that same concept described in Gervás [9]. Interestingly enough, each sequence of character functions obtained by either of these procedures would itself be a concept of the artefact type, susceptible of being expressed in different ways. The procedure for fabula/flow generation from a sequence of character functions would be a concept of the process type. Similar considerations can be made about the final stage in the construction of a story, that of rendering the set of predicates as text. This final text would itself be an artefact with an associate process to produce it.

Given that the development effort has focused at a very abstract level of representation, evaluation has been considered at a corresponding level to provide valid feedback for the improvement of the system. The alternative of evaluation by human volunteers was not considered due to the difficulty of the linguistic modelling of the stories and the fact that evaluations by humans necessarily have to be restricted to a small number of instances of system output. The choice of which particular instances to test is left to the designer of the experiment, and there is a risk of focusing on examples that are not representative of system performance overall. As an alternative, quantitative procedures have been defined to measure the specific qualities desired for each stage of the representation, at a corresponding abstract level. These procedures can be applied to a large number of system results, providing a measure of the quality of system output at the working level of abstraction and applicable to a broad range of system results, leaving no doubt as to their significance over the complete set of outputs.

The development of the remaining modules of the Propper system is ongoing work. Initial results on the fabula generation module were reported in [9]. The interfacing of the fabula representation of a story—which instantiates the character functions in a plot with particular story actions—is likely to require a further step of narrative composition [10], which would handle the presentation of the created fabula as a sequential discourse phrased in terms that are apprehensible to the reader in the sense that they match his intuition of how characters perceive the space surrounding them and each other. This may require reformulating the terms in which the plot is described to handle issues like character perception, focalisation of the action on particular characters, perceived relative movement, and the fact that characters may have only partial knowledge of what occurs elsewhere in the story. This process of narrative composition may also be applicable to model the way certain stories, as described by Propp, are made up of more than one *move* understood as a subplot produced by a single pass over the reference sequence of character functions.

Conclusions

The theoretical account of Russian fairy tales provided by Vladimir Propp has been revisited as potential source for a procedure for story generation. By considering the simplest possible implementation of these procedures, a framework for story generation has been developed that takes full advantage of the intuitions behind Propp's account but which is built in a modular and declarative manner so that particular details arising from Russian folk tales can later be replaced with material from alternative knowledge sources.

The long-range dependencies between elements in the story have been shown empirically to have a very significant impact on the quality of the resulting plots. In this particular instantiation of the plot construction process, the elements being considered are character functions of the type proposed by Propp. However, we have every reason to believe that the nature of these long-range dependencies is independent of the particular abstraction being used to define an element in the story. Similar long-range dependencies could be established for story actions, or even for predicates occurring in a fabula or a flow. In this sense, the heuristics presented in the paper for taking dependencies into consideration during construction could be extrapolated to constructive procedures that operate on representations of stories at a different level of granularity, provided the dependencies themselves can be identified between elements at that level.

The constraints on valid endings for a story take two forms. One type of constraint arises from the nature of the element to be placed at the end of the story—a character function when considered at this level of granularity—and distinguishes between elements that are good to end a story with and elements that are not. This constraint may have a genre-specific aspect, in the sense that positive elements may be preferred in genres that prefer happy endings. But it also includes a generic aspect, in that it has preference for elements that do not require a continuation. This type of constraint has been modelled in the present paper by the set of character functions considered to be valid as endings of a plot driver. A different type of constraint arises from the expectation that all issues raised within a story should be resolved before the end. In this particular case, the concept of issue raised has been associated with the initial element of a dependent pair being added to a plot driver, and the resolution of the issue is associated with the final element of the dependent pair being added as well. This type of constraint has been modelled in the present paper by requiring that no unresolved dependencies remain at the point when a plot drive is to be closed.

The approach suffers from the limitations inherent to any knowledge intensive approach, in the form of a heavy knowledge engineering effort required to kick start the necessary set of resources. Preliminary results show strong coupling between the quality of these resources and the quality of the resulting stories. This can be seen as a weakness in terms of a deep adaptation curve for any new domain or new application, but also as a significant strength, in terms of the possibility of extending it to other domains and the possibility of carrying out a process of targeted refinement until a desired level of quality is reached for a given domain.

The overarching framework of Propp's generative procedure and the particular solutions presented in this paper have been analysed in terms of relevant theoretical advances in computational creativity. Although the proposed system has no claim to being considered creative, it has been argued that it constitutes a first step in a long road towards understanding the procedures involved in story generation, with a view to finding how and where these procedures can be infused with the spark of creativity.

The Propper system is ongoing work, and further advances on the fabula generation module, the casting module, and the text rendering module are expected in the future. Further work will also include refinement of the knowledge engineering for story actions and extension of the set of resources to cover new domains.

Acknowledgments The research reported in this paper was partially supported by the Ministerio de Educación y Ciencia (TIN2009-14659-C03-01) and the Project WHIM 611560 funded by the European Commission, Framework Programme 7, the ICT theme, and the Future and Emerging Technologies FET programme. The author would like to thank the referees of earlier versions of this paper for their comments which helped improve the present effort.

References

1. Bae B-C, Young RM. A use of flashback and foreshadowing for surprise arousal in narrative using a plan-based approach. In: Proceedings of the ICIDS 2008; 2008.
2. Boden M. Creative mind: myths and mechanisms. New York: Routledge; 2003.
3. Campbell J. The hero with a thousand faces. 2nd ed., Bollingen seriesPrinceton: Princeton University Press; 1968.
4. Colton S, Charnley J, Pease A. Computational creativity theory: the face and idea descriptive models. In: 2nd international conference on computational creativity; 2011.
5. Dehn N. Story generation after tale-spin. In: Proceedings of the IJCAI 1981; 1981. p. 16–8.
6. Fairclough C, Cunningham P. A multiplayer case based story engine. In: GAME-ON conference; 2003. p. 41
7. Chris F, Padraig C. A multiplayer o.p.i.a.t.e. Int J Intell Games Simul. 2004;3(2):54–61.
8. Gervás P, Díaz-Agudo B, Peinado F, Hervás R. Story plot generation based on CBR. Knowl Based Syst. 2005;18:235–42 Special Issue: AI-2004.
9. Gervás P. Propp's morphology of the folk tale as a grammar for generation. In: Workshop on computational models of narrative, a satellite workshop of CogSci 2013: the 35th meeting of the cognitive science society, Universität Hamburg, Hamburg, Germany, 08/2013 2013. Schloss Dagstuhl - Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany, Schloss Dagstuhl - Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.
10. Gervás P. Composing narrative discourse for stories of many characters: a case study over a chess game. Lit Linguist Comput. 2014;29(4):511–31.
11. Gervás P, Lönneker B, Meister JC, Peinado F. Narrative models: narratology meets artificial intelligence. In: International conference on language resources and evaluation. Satellite workshop: toward computational models of literary analysis, Genova, Italy; 2006. p. 44–51.
12. Grasbon D, Braun N. A morphological approach to interactive storytelling. In: CAST 2001. Living in mixed realities : conference on artistic, cultural and scientific aspects of experimental media spaces, 21–22 Sept 2001, Schloss Birlinghoven, Sankt Augustin; 2001.
13. Herman D. Storytelling and the sciences of mind. Cambridge: MIT Press; 2013.
14. Jhala A, Young RM. Cinematic visual discourse: representation, generation, and evaluation. IEEE Trans Comput Intell AI Games. 2010;2(2):69–82.
15. Klein S, Aeschliman JF, Balsiger D, Converse SL, Court C, Foster M, Lao R, Oakley JD, Smith J. Automatic novel writing: a status report. Technical report 186, Computer Science Department, The University of Wisconsin, Madison, Wisconsin, Dec 1973.
16. Lakoff GP. Structural complexity in fairy tales. Study Man. 1972;1:128–50.
17. Lang RR. A formal model for simple narratives. PhD thesis, Tulane University; 1997.
18. Lebowitz M. Story-telling as planning and learning. In: Proceedings of the IJCAI 1983, vol 1; 1983.
19. Meehan JR. Tale-spin, an interactive program that writes stories. In: Proceedings of the IJCAI, 1977; 1977. p. 91–8.
20. Pease A, Colton S. Computational creativity theory: inspirations behind the face and the idea models. In: 2nd international conference on computational creativity; 2011.
21. Pérez y Pérez R. MEXICA: a computer model of creativity in writing. PhD thesis, The University of Sussex; 1999.

22. Propp V. Morphology of the folktale. Austin: University of Texas Press; 1968.
23. Riedl M, Young M. Narrative planning: balancing plot and character. J Artif Intell Res JAIR. 2010;39:217–68.
24. Ritchie G. Some empirical criteria for attributing creativity to a computer program. Minds Mach. 2007;17:67–99.
25. Saunders R. Towards autonomous creative systems: a computational approach. Cogn Comput. 2012;4(3):216–25.
26. Trabasso T, van den Broek P, Suh SY. Logical necessity and transitivity of causal relations in stories. Discourse Process. 1989;12:1–25.
27. Turner SR. Minstrel: a computer model of creativity and storytelling. PhD thesis, University of California at Los Angeles, Los Angeles, CA, USA; 1993.
28. Wiggins G. A preliminary framework for description, analysis and comparison of creative systems. Knowl Based Syst 2006;19(7):449–58.
29. Wiggins G. The mind's chorus: creativity before consciousness. Cogn Comput. 2012;4(3):306–19.