An API-based Approach to Co-creation in Automatic Storytelling

Eugenio Concepción[1], Pablo Gervás[2], and Gonzalo Méndez[2]

[1] Facultad de Informática
[2] Instituto de Tecnología del Conocimiento
Universidad Complutense de Madrid
econcepc@ucm.es, pgervas@sip.ucm.es, gmendez@fdi.ucm.es

Abstract. The basic idea behind this paper is the development of a collaborative environment for generating stories. Hence, the authors put forward an architectural model for knowledge interchange between story generation systems, namely Propper, STella and Charade, in the interest of enhancing the interoperability and fostering the co-creation process. For this reason, this paper proposes an API Economy model based on the interchange of knowledge and services between story generation systems. The proposed architecture is based on an API-based microservices ecosystem connected according the REST architectural model. This approach aims at starting with a reduced set of services for extending it later with new capabilities.

Keywords: Computational creativity, Story generation systems, Software architecture, Service-Oriented Architecture

1 Introduction

Digital assets are increasingly becoming the most valuable resources that underlie much of the present economics. The digital artefacts are the key components in many organizations, whose businesses rely heavily on their ability to manage them. Making these key capabilities available by publishing them as APIs accelerates the innovation and provides uniform data and functionalities to internal and external actors. According to Willmott and Balas [28], an API Economy is defined as the emerging economic effects enabled by companies, governments, non-profits and individuals using APIs to provide direct programmable access to their systems and processes.

Automatic story generation is a part of a wider research area in Artificial Intelligence named Computational Creativity (CC), which aims to develop a creative behaviour in machines [26]. A story generator algorithm (SGA) refers to a computational procedure resulting in an artefact that can be considered a story [8]. A story generation system, also named storytelling system, can be defined as a computational system designed to tell stories [8].

From an architectural point of view, automatic story generation systems have been traditionally designed as monolithic systems. That means that a single

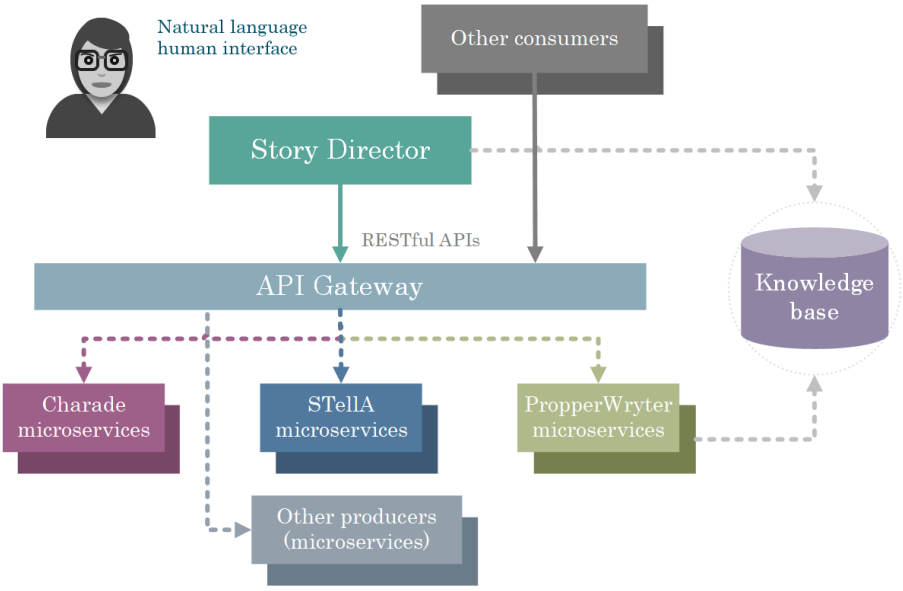

Fig. 1. An API-based architecture for storytelling.

application concentrated all the required functionality and assets. Obviously, this was a feasible solution for the earlier systems, which were built mainly for research purposes and implemented a limited-complexity functionality. As the story generation systems are becoming more complex, they are being designed in a much more modular way.

The big picture of the presented ecosystem relates to a service-oriented architecture [5, 20], and the microservices model [2]. This paradigm provides a convenient framework for organizing complex software systems. Applied to our particular research context, this approach, along with an API economy model, would allow the storytelling systems to create new functionalities and value. The resulting economy enables many new classes of applications with the potential to open new ways of hybridize algorithms, models and processes. This new ecosystem also entails the adoption of new roles, that is, API providers, API consumers, and the end user —as shown in Figure 1.

2 Related work

For the purpose of this paper, two research efforts need to be reviewed: a panorama of the architecture of the most relevant story generation systems – in order to understand how they operate and what type of architectural model they follow –, and existing approaches to combine story generation systems together – to consider what possible ways of combining them have been attempted.

Generally speaking, the architecture of a good part of the existing story generation systems usually fits with three main categories: those that are built over a planner [14, 4, 25, 23]; the systems developed by applying case-based reasoning (CBR) [25, 9]; and those builts over agent-based architectures [24, 15].

Despite there is not a vast amount of literature on the subject, several efforts concerning collaborative story generation have been carried out. Slant [17] can be considered a remarkable example of storytelling systems interoperating for producing an enhanced outcome. It is an architecture for creative story generation that integrates different types of story generation systems. It also provides a convenient framework with a view to other systems to integrate with it. Slant is the end result of an ambitious integration project that involves the integration of several different components: one based on Mexica [21], one based on GRIOT [11], and a new one specifically developed for the combined system. From a technical point of view, Slant consists of a blackboard architecture that allows different storytelling systems or components to create stories collaboratively. The goal is to allow the systems to influence each other for generating an enhanced result. The blackboard architecture for developing the story representation, and the Slant story XML format that is used, open up new possibilities for collaboration between creative literary systems, allowing models of creativity to be developed and added in different configurations.

In a wider context, still within the computational creativity area, it is also noteworthy the architecture proposed by Veale [26] for creative Web services. This model tried to combine both the academic and the industry needs in a solution for enhancing computational creativity systems. The architecture identified three types of services: *discovery services*, aimed at mining the knowledge contained in texts, and acquire emergent insights and novel perspectives on the expressed concepts; *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 proposed architecture is also accompanied by two specific Web services: Thesaurus Rex and Metaphor Magnet -as examples of creative functionality.

3 Scope

The idea behind the development of a collaborative environment for co-creating stories is pretty close to a practice referred by Veale [26] when he spoke of how organizations outsource their creative needs to external agencies. Such agencies act as option providers, in the sense that they create a universe of potential solutions, but leave others to make the final decision. In a co-creation scenario, several systems interact for creating a variety of feasible stories, but they require

the collaboration of one or more humans to evaluate the results and provide a feedback.

The involved systems, STellA [13], PropperWryter [9], and Charade [16], have been selected because they differ considerably from each other. The three systems focus on different aspects of storytelling. STellA is centred on causality, putting the stress on the causal order of events and actions. PropperWryter's thrust is the inner structure of the story. It works using the categories of characters functions defined by Propp [22]. Charade is basically oriented to the simulation aspect, giving as a result the evolution of the affinities between the characters. Thus, the combined operation of the three systems can be fruitful, especially if every component supplies the rest with its special features.

STellA can bring the basic causal structure of the story. This product can be refined by applying the functions defined in PropperWryter, giving a more cohesive plot. Charade can provide a more credible behaviour by incorporating the interaction between the characters of the story.

The choice for REST as the architectural style of the solution comes from the need of decoupling the distinctive features of each system from the communication architecture [12].

3.1 STellA

STellA (Story Telling Algorithm) [13] is a story generation system that controls and chooses states in a non-deterministically generated space of partial stories until it finds a satisfactory simulation of events that is rendered as a story.

STellA uses a custom representation for the knowledge it needs. It manages several different structures, including a matrix representation of the world in which characters live, and a set of rules for evaluating the range of results associated to the actions.

In STellA, the generation process involves an iterative creation of new states. Every simulation is modelled and implemented as a non-deterministic process in which every step can generate not only one but many others. This simulation requires the whole world domain to be explicitly represented as a simplistic view of a realistic environment. This approach provides a very detailed scenario that allows for a rich set of possibilities in generation. Each iteration generates a set of candidate versions of the current state, and then the process identifies the most likely ones by analysing their likelihood in terms of their plausibility and their narrative properties. This step is carried out by applying constraints and a generalized version of tension curves to drive story generation. These candidate partial stories are evaluated insofar they satisfy a given set of constraints and to what extent their tension curves fit with a set of target curves. The results of this process provide a criterion to decide if a partial story is promising and whether a story is finished.

The system requires an initial state and final conditions (which may also be a state to be reached). Basically, it generates from a starting point to a final condition. One of the most characteristic concepts managed by STellA is the *entropy*. The generator is able to generate many scenarios during the

reasoning process; and the more things are invented, the more entropy a story has. For example, if in the initial state there is a scenario expressed like *John loves Mary and they have a child*, and the child would be invented, it generates little entropy. Conversely, if the scenario expresses that *Mary has been abducted by the Martians*, it would be necessary to invent the *Martians*, who live on Mars, who want to take Mary (and why), and a few other things. That scenario would generate a lot of entropy.

Every state has entropy, and the state entropy is given by every generation cycle. The user determines how much entropy can be reached. The system outputs a very detailed sequence of snapshots of what happens at each moment. The result is a more or less narrative elaboration. Thus, the output is a list of states, in the same format as the input. Each state contains a timestamp. All the generated story, that is, all its states, are checked to see if they verify the established conditions.

3.2 PropperWryter

PropperWryter [9, 10] is a story generation system that creates Russian folktales according to Propp's generation rules [22]. These rules provide a very clear description of how the folktales morphology could be used for story generation. This approach has been previously used in other systems, like [27].

PropperWryter uses a set of abstractions for representing the essential concepts defined by Propp, especially the character function, and defines a procedure that first chooses a sequence of character functions to act as abstract narrative structure to drive the process, and then progressively selects instantiations of these character functions in terms of story actions to produce a conceptual representation of a valid story.

The generator can work in two forms: it can generate a story with no input, or it can take an input query (which can be a sequence of narrative tags or a sequence of actions), and generate a sequence of states. PropperWryter requires several work resources: a set of actions, a list of possible dependencies between actions, the mapping of each action to a high-level narrative label (Match, Return, Clash, Defeat, Prison, Release...), a list of possible dependencies between narrative labels (Departure-Return, Clash-Defeat, Prison-Release...), and the mapping of each variable that appears in an action to a narrative role (Hero, Villain, Victim...). The output of the system consists of a sequence of states, where each state is described by a set (not necessarily ordered) of predicates in which the characters are identified as variables. In general terms, this seeks to ensure that if the sequence includes an action that is an instance of a tag that has dependency on another tag, the sequence will also include an action that is an instance of this last tag. It is also intended that the assignment of narrative roles to each character appearing in the sequence (depending on the roles they play in the set of actions in which they appear) is consistent throughout the sequence.

3.3 Charade

The system developed by [15, 16] models the relationship between two characters using their mutual affinities, and applies it for generating stories. This system is an agent-based architecture developed using JADE. It consists of two types of agents: a Director Agent, that sets up the execution environment and creates the characters; and the Character Agents, one for each character of the story, whose interactions generate the story.

The main objective of the system were implementing an affinity model as decoupled as possible from the story domain, and testing it independently from other factors such as the environment in which the action takes place or the personality traits and emotional state of the characters. Due to this independence, it can be easily used to generate different kinds of stories.

The generator is based on a simulation of the characters' interaction. During the simulation, the characters perform actions between them, varying the affinity levels between them as a result. According to the affinity level, the characters can be a couple, friends, mutually indifferent, and enemies. Generation is independent of the domain; although, since it focuses on affinities, it works best in domains where this affinity makes sense. The simulation is not directed, so that it can not be considered to constitute a plot or a story by itself. The input includes a complete parametrization of possible actions, categorized according the type of relationship allowed for the characters, the simulated characters, and their relationships measured in terms of affinity. The output consists of a list of actions proposed by characters, and the response of their counterparts, that can accept or reject the proposals, with the variation of affinity between the characters involved. Despite no text is generated, it would be easy to use a template for generating a textual description.

4 Proposed solution

The proposed architecture aspires to articulate the operation of several automatic story generators in a way that allows the generation of higher quality stories by joining the capabilities of each system. The seed of this solution is the model composed by the three storytelling systems described before.

Every story generator provides a set of key operations in the form of services, so that each system can be considered as a module in the overall structure. The communications are based on the well-known REST architectural model [6]. This approach aims at simplifying the communication process by means of an easily achievable representation of the information. From a technical point of view, every involved system publishes their capabilities as REST-based services. Every service understands and generates JSON messages containing the required information in each case. Due to the fact that all these systems existed prior to the definition of the API ecosystem, everyone can be considered as a legacy system that must be adapted to this new purpose. This is the reason why the core capabilities or every system will be considered as the back-end, and a new

tier, specifically designed for publishing REST services, will be built for wrapping them.

4.1 Design considerations

Many of the existing story generation systems have been built in a such way that the collaboration between them is a really complex task. This happens because almost every system duplicates a considerable part of main storytelling functions. For example, the generation of the story in natural language is a typical stage in every story generator. If every storytelling system breaks its architecture into finer-grain components, such as microservices [29][18], these components could be used separately. Also, every microservice would be autonomous enough to be independently evolved according to new requirements, without affecting the rest of the architecture. But the most remarkable achievement of this approach would be the possibility of building hybrid coarse-grain services by composing the existing microservices. This new system would take advantage of using the best-of-breed for building a collaborative story generation architecture.

One of the key points of the architectural model is to ensure semantic interoperability. To develop a formalism for knowledge sharing in a collaborative architecture would make no sense if it is not possible to have an understanding by all parties of the information being exchanged. In this respect, it seems necessary a component for orchestrating the different microservices, and a repository to keep the shared knowledge that can be consulted by any microservice every time it has to interpret a request.

This model of knowledge base has been designed as centralized for two reasons: on the one hand, the concept-based framework applied must be necessarily shared between the various story generators, and on the other hand, it is necessary to avoid that the messages exchanged become too verbose. This last requirement is technical. If we understand that a microservice-based architecture is deployed in a distributed environment, this means that communications rely on the network, at all costs. If we assume that the communication model is truly REST, there is no state. This means that for each request it is necessary to send all the data that the server requires to be able to perform its work. This has the effect of sending the complete generated story in every request. In other words, the entire knowledge base required by the story is appended to the request data, making the communications inevitably become inoperative after a few requests between systems.

Another aspect to consider is that it is not possible to delegate to each system the definition of the entities or concepts that it handles. Take, for example, an action as simple as eating. In the case of Charade, this concept refers to a couple lunch, and it is an atomic action. Instead, in STellA this is a composite action, which refers to the physical act of eating and involves several steps such as bringing food to the mouth, chewing and swallowing. Clearly, a human being senses that the same concept is not being talked about, but a computer system needs to have precise definitions so that it knows what the system is referring to.

It is therefore necessary for the definition to become universal, and for systems to know what they are referring to by taking existing concepts to operate.

4.2 Methodology

The involved systems will be decomposed in its basic functionalities, that is, as microservices that will expose their capabilities as REST-based API [6]. Every service will understand and generate JSON messages containing the required information in each case. Due to the fact that all these systems existed prior to the definition of the collaboration architecture, some parts can be considered as a legacy system that must be adapted to this new purpose. This is the reason why certain core capabilities of the systems will be reconstructed, and a new tier, specifically designed for publishing REST services, will be built for wrapping them. A high-level component, namely the story director, will implement the orchestration of the whole system, establishing the order in which every system would make its part. For achieving a full syntactic and semantic interoperability, the exchanged messages between the different components will be based on a common knowledge representation model [1].

The steps for achieving the establishment of a well-grounded API Economy are widely discussed by specialized literature [7] [19].

Olson [19], suggested that organizations should treat their API as products it must nurture. In this regard, she proposed a sequence of steps for achieving this [19]. Notable among them are the importance of understanding the value chain, and the establishment of goals for every API strategy.

4.3 System design

The general model of joint operation of the three systems is based on the use of the key capabilities of each of them. The Figure 2 depicts the whole architecture and its components. Thus, the role of PropperWryter is to develop the main scheme of the plot, while STellA is responsible for simulating the development of the different low-level scenes, and Charade establishes the evolution of the relationships between the characters.

The role of the story director entails the orchestration of the whole system, establishing the order in which every system would make its part. This is the central component that will need to preserve the collective knowledge by means of the common knowledge base. Concerning this point, the need for a common representation arises. As stated above, every system focuses in a different aspect of story generation, so are its knowledge representation. Every published service must be considered to provide system-specific knowledge structure, so the adaptation step must be performed in the composer module.

The story director is also related to the maintenance of consistency in the story that is being generated. As already mentioned, Charade simulates the evolution of relationships (couple, friends or enemies). Let us suppose a story in which, in one of the scenes of the plot, the couple has a romantic dinner and the end result is that their love affinity increases. For Charade, a romantic

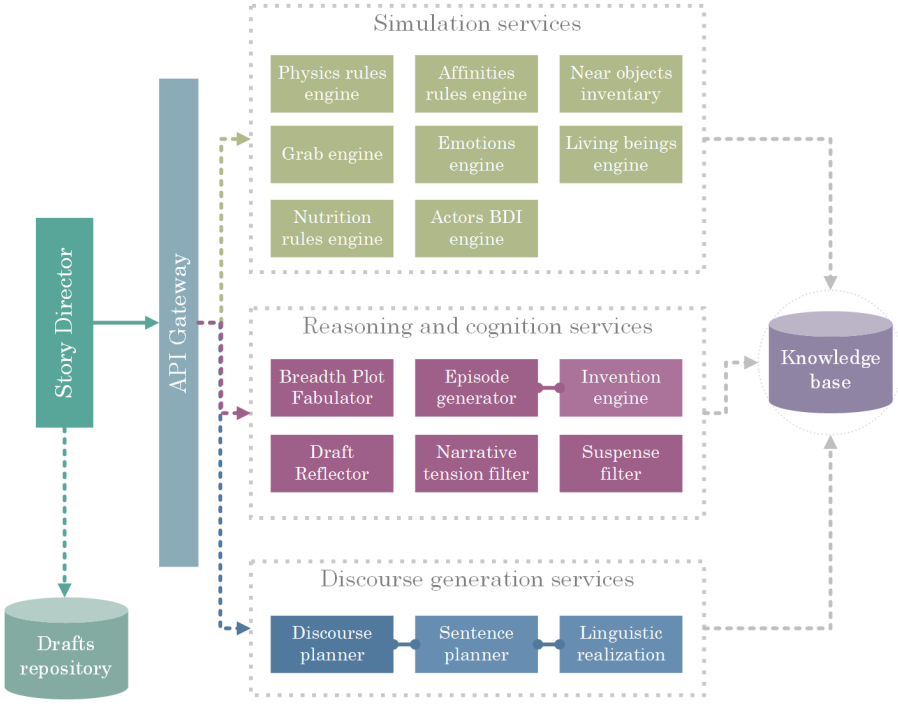

Fig. 2. Architecture of the proposed system.

dinner is an atomic action, it does not go into the detail of how it evolves. The story could then be passed to a service of STellA for developing in more detail the scene of the romantic dinner. During the simulation, STellA generates the actions performed by the characters, and it turns out that, at a given moment, the couple conversation becomes a discussion. This result would be clearly inconsistent with the final result calculated previously. At this point, the director's role becomes crucial. It must decide whether to discard the scene generated by STellA, if it changes the course of events in the relationship (as generated by Charade), or if it requests STellA to re-evaluate the situation so that after the discussion there is a reconciliation, and the dinner ends happily.

In a first approach, a REST-based interface is being defined for wrapping the original story generation systems. This step leads to the definition of a set of common concepts shared across the systems, so that the same entities are expressed in the same way. This step is essential for articulating a generation pipeline with all the systems. At this point, there are key concepts that must be managed by the director and clearly informed to the participants: state transitions, actions and results (understood as a causal chain), sequentiality...

The key component of the API Economy approach is the API Manager. This component provides the scaffolding for building the service-based API structure. Usually, an API Manager provides three essential features: a single entry point for

all the consumers requests (API Gateway), a central web-based tool for managing the various policies to be applied, and a marketplace for developers that allows them to easily find the APIs they need to consume (API Portal). All these component are normally completed by a centralized configuration service, and the elastic infrastructure for hosting the microservices. The roles and derived benefits from this infrastructure are multiple:

- Centralized configuration, a service that all applications use to specify and access their respective configuration information in a consistent way.
- Automatic deployment, a service that invokes and decommissions APIs and service implementations under administrator control.
- Single security enforcement point, usually provided by the API Gateway.
- Auditing and monitoring, provided by the API Gateway.

Another essential component in architecture is the repository of drafts (or stories). Both PropperWryter and STellA generate story trees from possible continuations. While the STellA model is less restrictive than the PropperWryter model, in both cases it is necessary to maintain a draft tree in progress. In order to avoid having an exchange of excessively large JSON messages, the idea is to reposition all the drafts, to recover them only when required. The formalism employed for representing these drafts is detailed in a specific paper [1].

Thus, the main services are the following:

- BreadthFabulator (PropperWryter)
- Reflector (PropperWryter)
- Simulation engines (STellA)
- EpisodeGenerationEngine (STellA)
- DiscourseGenerator (STellA)
- SuspenseFilter (STellA)
- NarrativeTensionFilter (STellA)
- AffinitiesEngine (Charade)
- StoryDirector

In the development of the plot, the actions of the characters and the events modify the global state of the narrative universe. In this sense, every action that takes place in the plot carries information related to the new state in which the universe remains.

The joint operation of the microservices ecosystem will be directed by the Story Director, who will act as an orchestrator of the generation process. It will request the APIs of the different services according to the generation process. This process will proceed iteratively, generating drafts that will be refined in each pass, until the established criteria for story completeness are met. The Reflector service will analyse the draft for ensuring the compliance of these criteria —as it originally did in PropperWryter.

In the case of STellA, it provides a detailed simulation for every scene generated by PropperWryter in the high-level plot. The service that provides this is the Episode Generation Engine, which receives as input a draft of the story,

which contains information about the characters, and what must happen in the scene at a high level. The service then generates a simulation to explore the universe of possible solutions. Unlike the original operation of STellA, which was unrestricted, in this case, there are restrictions to apply to the final state in which the scene must be found. This means that there will be solutions whose generation should be truncated by not reaching a state of the narrative universe compatible with the expected final state. The result of the simulation will be a new collection of drafts that will be persisted in the Drafts Repository. The director of the story, who is responsible for orchestrating the behaviour of the whole, will analyze the various drafts through by means of the two filtering microservices (Suspense and narrative tension). The objective is discarding the stuff that should not prosper in the next iteration. The different STellA simulation engines, as well as the knowledge base, will be used at convenience. This is also the case of the affinity engine of Charade, which will serve to calculate the evolution of the relationship of the characters as the development of the plot takes place. In this sense, it is important that a relationship be established between all the concepts that the three systems handle. For example, in the case of an action such as dining, which can be interpreted in different ways by each of the systems, especially in the case of STellA, which tends to a strong physical representation of the actions.

5 Conclusions and future work

The set out approach intends to establish a collaborative model that allows the free exchange of knowledge between the different storytelling systems in order to develop an iterative improvement process of literary creation. In addition to this objective, it promotes the development of a knowledge representation model for creating a common knowledge base that can be fed in the future with the outcomes of new storytelling systems, without the need to adapt locally their knowledge representation models.

The architecture exposed so far has several features to be developed in the next steps. The main pending task is related to the design and implementation of the composition service. It is necessary to develop, not only a technical model for aggregating services, but also a proper interface for interacting with the human participants in the process.

Currently, in every iteration, for every draft of the current population, all the possible continuations are generated and added to the population of the next iteration. On the generated population, a reflection process is applied (Reflector class), and drafts that are considered already finished are separated from the work population. This process continues until the work population is empty (all drafts are terminated) or a limit of iterations is reached (to guarantee completion). In the face of future work, the development of a service that helps to decide is pending. The process for deciding what is the most appropriate level of detail in each of the scenes is still pending. If we take as an example any novel, it can be seen that in each scene a different level of detail is handled —which greatly

influences the narrative rhythm, for example. Certain scenes are described at a high level, without going too deeply into the details, while other scenes related to very brief moments in time, are treated in detail, because they are very relevant in the narration. This component will become increasingly important as more and more systems are incorporated into the proposed ecosystem.

The most immediate roadmap focuses on developing and testing the described REST-based services. Once these services become available, the next step will involve the design of a formal representation for the persisted knowledge. On the basis of this knowledge, the composer could generate a human-readable output. As stated in [3], a suitable solution would be the use of a controlled natural language (CNL), which is naturally easier to understand by humans than formal languages, encouraging the co-creation cycle.

Once that all the participants have implemented and made available their services, the next step will be the development of the process for integrating them in a generation pipeline making use of the knowledge shared across them. It is still a matter of study how to apply certain local concepts, such as entropy, to the whole architecture for enhancing the global outcomes.

Acknowledgements

This paper has been partially funded by the project IDiLyCo: Digital Inclusion, Language and Communication, Grant. No. TIN2015-66655-R (MINECO/FEDER).

References

1. Concepción, E., Gervás, P., Méndez, G.: A common model for representing stories in automatic storytelling. In: 6th International Workshop on Computational Creativity, Concept Invention, and General Intelligence. C3GI 2017 (2017)
2. Concepción, E., Gervás, P., Méndez, G.: A microservice-based architecture for story generation. In: Microservices 2017 (2017)
3. Concepción, E., Gervás, P., Méndez, G., León, C.: Using cnl for knowledge elicitation and exchange across story generation systems. In: International Workshop on Controlled Natural Language. pp. 81–91. Springer (2016)
4. Dehn, N.: Story generation after tale-spin. In: IJCAI. vol. 81, pp. 16–18 (1981)
5. Erl, T.: Service-oriented architecture: a field guide to integrating XML and web services. Prentice Hall PTR (2004)
6. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
7. Gat, I., Succi, G.: A survey of the api economy. Cut. Consort (2013)
8. Gervás, P.: Story generator algorithms. In: The Living Handbook of Narratology. Hamburg University Press (2012), http://hup.sub.uni-hamburg.de/lhn/index.php
9. Gervás, P.: Propp's morphology of the folk tale as a grammar for generation. In: OASIcs-OpenAccess Series in Informatics. vol. 32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2013)
10. Gervás, P.: Reviewing propp's story generation procedure in the light of computational creativity. In: AISB Symposium on Computational Creativity, AISB-2014, April 1-4 2014. Goldsmiths, London, UK (04/2014 2014)

11. Harrell, D.F.: Walking blues changes undersea: Imaginative narrative in interactive poetry generation with the griot system. In: AAAI 2006 Workshop in Computational Aesthetics: Artificial Intelligence Approaches to Happiness and Beauty. pp. 61–69 (2006)
12. Khare, R., Taylor, R.N.: Extending the representational state transfer (rest) architectural style for decentralized systems. In: Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on. pp. 428–437. IEEE (2004)
13. León, C., Gervás, P.: Creativity in story generation from the ground up: Non-deterministic simulation driven by narrative. In: 5th International Conference on Computational Creativity, ICCC (2014)
14. Meehan, J.R.: Tale-spin, an interactive program that writes stories. In: In Proceedings of the Fifth International Joint Conference on Artificial Intelligence. pp. 91–98 (1977)
15. Méndez, G., Gervás, P., León, C.: A model of character affinity for agent-based story generation. In: 9th International Conference on Knowledge, Information and Creativity Support Systems, Limassol, Cyprus. vol. 11, p. 2014 (2014)
16. Méndez, G., Gervás, P., León, C.: On the use of character affinities for story plot generation. In: Knowledge, Information and Creativity Support Systems, pp. 211–225. Springer (2016)
17. Montfort, N., Pérez, R., Harrell, D.F., Campana, A.: Slant: A blackboard system to generate plot, figuration, and narrative discourse aspects of stories. In: Proceedings of the fourth international conference on computational creativity. pp. 168–175 (2013)
18. Newman, S.: Building microservices: designing fine-grained systems. " O'Reilly Media, Inc." (2015)
19. Olson, L.: The open api economy: What is it and how do i capitalize on it? In: International Service Technology Symposium (2012)
20. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on. pp. 3–12. IEEE (2003)
21. Perez y Perez, R.: MEXICA: A Computer Model of Creativity in Writing. Ph.D. thesis, The University of Sussex (1999)
22. Propp, V.: Morphology of the folk tale. 1928 (1968)
23. Riedl, M.O., Young, R.M.: Narrative planning: balancing plot and character. Journal of Artificial Intelligence Research 39(1), 217–268 (2010)
24. Si, M., Marsella, S.C., Pynadath, D.V.: Thespian: Modeling socially normative behavior in a decision-theoretic framework. In: Intelligent Virtual Agents. pp. 369–382. Springer (2006)
25. Turner, S.R.: Minstrel: A Computer Model of Creativity and Storytelling. Ph.D. thesis, University of California at Los Angeles, Los Angeles, CA, USA (1993), uMI Order no. GAX93-19933
26. Veale, T.: A service-oriented architecture for computational creativity. Journal of Computing Science and Engineering 7(3), 159–167 (2013)
27. Wama, T., Nakatsu, R.: Analysis and generation of japanese folktales based on vladimir propps methodology. In: New Frontiers for Entertainment Computing, pp. 129–137. Springer (2008)
28. Willmott, S., Balas, G.: Winning in the api economy. 3scale, octobre (2013)
29. Wolff, E.: Microservices: Flexible Software Architecture. Addison-Wesley Professional (2016)