A Framework for Building Creative Objects From Heterogeneous Generation Systems *

Carlos León[1], Jorge Carrillo de Albornoz[2], and Pablo Gervás[3]

Departamento de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
[1]cleon@fdi.ucm.es, [2]jcalbornoz@fdi.ucm.es, [3]pgervas@sip.ucm.es

Abstract. There exist several types of generative systems which produce some artistic output not necessarily considered to be creative. However, the processes they follow can, in principle, be combined between them, possibly obtaining a creative product. Although nowadays giving a general description of how such systems must be implemented is far from being possible, this paper presents a framework for putting together generative systems in such a way that the output can be creative. The paper shows preliminar ideas and discusses the main conclussions obtained from them.

Keywords: Computational Creativity, Cinematographic Direction, Storytelling, Framework, Artificial Intelligence.

1 Introduction

Building generative systems which are not creative for artistic production is not difficult. For instance, rule based systems controlling a story generation program can be easy implemented, but, probably, the output will not be as creative as needed for that program to be considered really useful in general. The same ideas apply for any other generative system (like automatic camera direction).

However, complex creative objects, like movies, are composed of many heterogeneous systems, and for these systems to be considered creative, it is not necessary that every part of the creation follows a creative process.

Given these characteristics, to study how different processes are connected in a full complex system becomes interesting and useful in the field of the Computational Creativity, where computers have to follow models of creativity in order to obtain new objects that *would be deemed as creative if found in humans* [1]. This knowledge would allow us to put several heterogeneous systems together, bringing new possibilities in the productions of such systems.

* This research is funded by the Ministerio de Educación y Ciencia (TIN2006-14433-C02-01), Universidad Complutense de Madrid, Dirección General de Universidades e Investigación de la Comunidad de Madrid (CCG07-UCM/TIC 2803) and a grant from Caja de Burgos, Jóvenes Excelentes 2008 contest.

However, obviously it is extremely difficult to give a general description of how to put generative systems together. In fact, the authors are not sure if such description exist: probably it is not possible to define a general way of combining them. Basically, what we can do by now is to look for *ad-hoc* solutions, trying to figure out what functions are to be solved, and trying to give a valid solution to them. Or, at least, a valid solution for the domain or the particular application that is being developed.

In this paper a case-study of a framework for one of these systems is presented, showing how an application that creates 3D short stories, using four different systems could be created: an automatic story teller, and module that adapts stories to given cinematographic schemas, an automatic camera positioning system, and a camera motion director, as depicted in Figure 1. The focus of this particular research has not been put on the systems separately, but on the nexus between the four, to study how to put several heterogeneous systems together in order to obtain a creative scene from these not necessarily creative systems.

This study is oriented to the definition of a system for creating 3D movies called Spiel, which will try to implement all these ideas in order to get a working creative system.

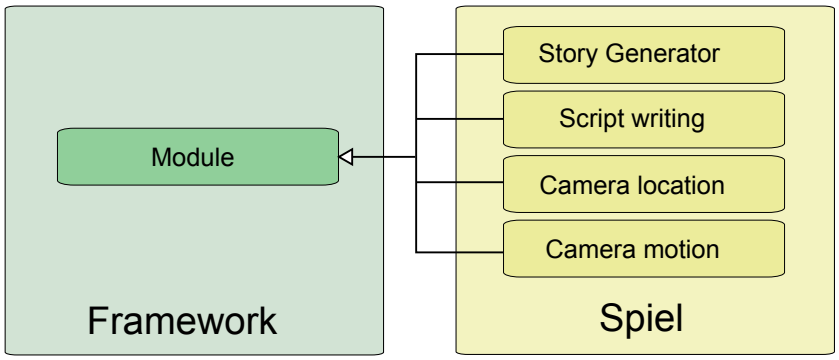

Fig. 1. Spiel's modules.

2 Previous Work

Computational Creativity tries to study processes related to human creative production, and is heavily influenced by Boden's work [2, 3]. The main issue about how to build creative systems is that *creativity* is not precisely defined [4]. However, building systems that could be considered creative is still possible and interesting.

There are several approaches to heterogeneous systems for cinematographic creation, but most of them are not focused on Computational Creativity. We can cite, however, some creative automatic systems from which several ideas have been taken to carry out the development of SPIEL.

Christianson et al. [5] use film idioms to adapt cinematographic techniques for portraying 3D animations. They formalise film idioms with their *Declarative Camera Control Language* (DCCL). As input the camera system use an animation trace representing the action that has already occurred and a number of sequences indicating which actor to film over each time. The system presented in [6], *The Virtual Cinematographer*, is a camera control system for a real-time virtual party that also use the concept of film idioms. The input is obtained directly from the state of the environment to choose the best idiom according to the actions. Each idiom is implemented as a finite state machine that describe the cinematographic techniques, which cover the current situation.

In the same line, the work presented in [7] shows a real-time system for interactive narration in virtual worlds. In [8] a similar approach is explained in a real-time computer game system. All this research is not focused on the creative processes involving the creation and visualization of a story, or in how the nexus between these components can enrich the interaction with humans.

Charles et al. [9] propose a system that combines an interactive storytelling system with an automatic camera system based on cinematographic techniques, which decides the correct idiom covering the actions of the characters, giving different weights according to the relevance in the situation.

Storytelling follows, however, a different research tradition. Creativity has been studied in several storytelling systems (MEXICA [10] or MINSTREL [11]). Results from those programs have been very influential on the development of Computational Creativity. They try, in general, to follow psychological models of human creativity, trying to apply a computational model of them to the generation of stories. But there also are other systems, (like TALESPIN [12], FABULIST [13] or UNIVERSE [14]), which, even creating good stories, are not focused on creative processes.

Next sections explain how this can be done: even without using creative systems, it is possible to build a creative object, by the interaction of the generative programs.

3 A Framework for Building Artistic Objects from Heterogeneous Modules

With all these ideas, a framework which tries to study how to automatically create a complex artistic object, like a 3D movie, from not necessarily creative systems could be possible, and an instantiation of this framework (SPIEL), are presented. This section explains in detail how the global system carries out the task of making several different heterogeneous systems work together. First an overview of the system, explaining the algorithm, is given, then some important details of the modules which instatiate the framework are explained.

3.1 How the Framework Works

The framework for creating artistic objects (mainly focused on movies) from heterogeneous modules is not a story generator, nor a camera direction system: it is a software whose application is focused on using several systems together, to create new information which the modules themselves could not create.

It is really difficult to design a system capable of carrying out this task for every possible set of systems. It would be necessary to define a language which every module should speak in order to intercommunicate those modules between them and with the main system. Also, functions for deciding operations like "when to apply each module" or "which operation must be done" should be defined in such a way that they would cover all possible types of modules, by giving a general description of those functions, or by covering all cases.

Although this is, in principle, possible, we are far from being capable of creating such a system. Instead, the logical approaches should follow a kind of *ad-hoc* solutions, trying to gather all possible information in order to gaining knowledge and experience for, some day, being able to develop a general system.

SPIEL, the framework's instantiation, is one of such *ad-hoc* systems. However, it is not only intended to cover a specific case, but it is fully focused on creating a "growing" nucleus, able to use, in the near future, different types of modules (stories, voice, 3D environments, plain text...). Basically, the framework's algorithm follows the schema shown in Algorithm 1. It is a very simplistic approach to an opportunist architecture. Several modules are present in the system, and the object which is going to be generated (a movie in the SPIEL instantiation), is being created step by step, by the sequential operation of the modules.

For directing what must be generated, modules are initialized with data guiding the generation. For instance, a story generation module should be initialized with constraints about what is to be created (a love story, perhaps).

What this algorithm is doing is explained in the list below. The algoritm is executed with a list of modules which implement the functions `validity`, `appropriate`, `new-solution` and `apply-solution` (what these functions are is explained), and it is assumed that the system's status is the partially built movie:

- **Lines 1-4**. The object which is going to be generated is initialized (s_0), and, running the `validity` function, each module yields a pair of values: the value for $moduleEvaluation_i$ is a real number in the interval [0..1], which is a measure of what the module thinks of the partial creation of the system (the partially generated movie). 0 means the module "thinks" the partial object is completely wrong, and 1 means that the partial object is valid enough to be the solution. This value is computed by each module in a different way, and considering different aspects of the generated object. The $moduleProblem$ value is a description of what the module considers to be wrong. This description is dependent on the particular instantiation of the system. The different incomplete parts of the partial object are defined in these steps.
- **Line 5**. After the system has computed every $\langle moduleEvaluation, moduleProblem \rangle$ pair of each module by calling its `validity` function, the framework applies

Algorithm 1 Using several modules in the framework for creating a single object

1: $s_0 \leftarrow$ *initial object*
2: **for all** $module_i$ in *modulelist* **do**
3: $\langle moduleEvaluation_i, moduleProblem_i \rangle \leftarrow module_i.validity(s_0)$
4: **end for**
5: $\langle evaluation, problem \rangle \leftarrow$ select the *most important pair* from the set of pairs $\langle moduleEvaluation_n, moduleProblem_n \rangle$
6: **while** $evaluation \neq$ valid **do**
7: **for all** $module_i$ in *modulelist* **do**
8: $moduleAppropriate_n \leftarrow module_i.appropriate(problem)$
9: **end for**
10: *chosen* $\leftarrow$ module with $max(moduleAppropriate_n)$, or *none* if no module is appropriate
11: **if** it has been possible to find a *chosen* module **then**
12: apply $chosen.newsolution()$ to s_j
13: **if** s_j has new information **then**
14: **for all** *othermodule* in *modulelist* **do**
15: $othermodule_i.applysolution(s_j)$
16: **end for**
17: **for all** $module_i$ in *modulelist* **do**
18: $\langle moduleEvaluation_i, moduleProblem_i \rangle \leftarrow module_i.validity(s_{j+1})$
19: **end for**
20: $\langle evaluation, problem \rangle \leftarrow$ select the *most important pair* from the set of pairs $\langle moduleEvaluation_n, moduleProblem_n \rangle$
21: **else**
22: discard the currently *chosen* module, and choose a different one in the next iteration
23: **end if**
24: **else**
25: exit program
26: **end if**
27: **end while**

a function, which can be defined in many ways, which returns the global system $\langle evaluation, problem \rangle$ pair. This function could return the problem from the module with the lowest *moduleEvaluation* value (which would be the pair with the "hardest" problem). Or it could return the mean value from the *moduleEvaluation* values of each module, and then a *problem* randomly chosen from the different modules. A discussion about some possible approaches is presented in Section 4. After this step, the problem (the incomplete part) which has to be solved is defined.

- **Line 6**. The system then checks whether the *evaluation* value is good enough to be considered valid. There are several possible ways of computing this boolean expression. Possibly, the most straightforward one is to check that the real value of *evaluation* is over a given threshold. This line guides a loop which will end once the object is created, or when it is considered impossible to create an object with the requisite from the user.
- **Lines 7-10**. Once we have the *problem* which has to be solved, each module runs its `appropriate` function. This function returns a real number in the interval [0..1], 0 meaning that it will not be able to solve a particular *problem* object, and 1 meaning it can perfectly solve the problem. Obviously, it is difficult to know whether a module can solve a problem before trying to solve it. What the `appropriate` function returns is a "possibility" value ($moduleAppropriate_n$). For instance, if the *problem* object returned by the computation of the `validity` function stores information about missing information of a story, a story generation module will return a higher value for the `appropriate` function than a module managing camera motion. However, even selecting a particular module, nothing ensures that it will be able of really solving the issues present in the *problem* object.

  After gathering all possible values for `appropriate` from each module, it is necessary to decide which module will be chosen. Initially it could be a good option to choose the module with higher `appropriate` value, but this value is only an approximation, and it can lead to local-minimum. That is why the framework adds random noise, and chooses, in some iterations (the frecuency of this noise can be parametrized), a module which is not the one with highest `appropriate` value. Section 4 discusses the benefits of this approach from the creativity point of view.

  If no module can be chosen (for instance, every module returns 0 as the value of the `appropriate` function), the system exits with a failure vaule: It has not been possible to create an object with the initial constraints.
- **Lines 11-12**. If it has been possible to find an appropriate (*chosen*), module, this chosen module applies its internal processes (what the module is really created for: a story, a new camera layout...), and generates a new object s_j applying its `new-solution` function (which returns partial information which completes the story: in SPIEL, a new set of facts for the story, for instance). If the `new-solution` function is not able to return a value, the algorithm will try to choose a different module. If no other module is *appropriate* (as explained in the previous list item), the generation fails.

– **Lines 13-16**. If the partial object (s_j) has been updated, The `apply-solution` function is called in each module. This function receives the description of the *solution* object just created by the choosen *module*, and updates the description of the partial whole object. For instance, if the chosen *module* was the story generator module, and it created three new facts for the story (which is only a part of the movie), the camera direction module would receive those three facts, and would update its information about the story. Obviously, this function assumes that every module knows how to apply information from other modules. So, although they can be heterogeneous, they can not be totally independent of each other.
– **Lines 17-20**. Finally the system checks again for the validity of the new partial object s_{j+1}.

The overall benefits of the approach followed in this algorithm are discussed in Section 4. Section 3.2 explains in detail how the previously commented functions are really instantiation.

3.2 Spiel: An Instantiation for Creating Movies

As it has been said before, four independent modules have been designed to instantiate this framework. They have been designed for interoperability. The next list details information about them, focusing on the instantiation of previously defined four functions for interacting with SPIEL, and, therefore, between them. These modules are not creative by any means, however, we want to discuss whether the output from these not necessarily creative modules can be creative or not.

– The *story generation module* outputs a list of first order logic-like predicates in this form[1]: `character(aragorn)`, `go(aragorn, forest)`. For instantiating the `validity` function, it accepts a story as valid if it is coherent with the user input (using hand-made logic rules). For example, if the user input is `kill(aragorn, sauron)`, and the resulting story-state is just `character(aragorn)`, `go(sauron, mordor)`, the story will be considered to be non-valid.
  The `new-solution` function returns, taking into account the current state of generation (the partial story, and the partial camera layout), a new story with logic facts.
  The `appropriate` function returns a valid value if the module can find, in its facts database, the set of facts necessary for completing the story, given the current problem. Finally, `apply-solution` updates module's state if the solution is a new story, and if the solution is a new camera location, it performs a mapping between the camera state and a set of "emotional facts" (`emotion(love, intense)`) in the story to create a new "emotional" state.

[1] We are working in The Lord Of The Rings domain.

- The script writing module adapts stories to typical story scripts. If, for example, a romantic scene is lacking the final kiss, this module adds that information to the set of facts (which has the same format as the previous module's output). The instantiation for the four functions follows approximately the same behaviour that for the story creating system (the rules vary, but the semantic is the same), but for the `appropriate` function: it computes this value by computing the distance between the script schema and the generated story. The less the distance (without being equal), the more appropriate the application of this module. That is, if there is only one different fact between the story and the schema, this module can solve the problem.
- The camera positioning module abstracts the cinematographic knowledge in order to determine which position and orientation, for a specific action or event in a story, are the most suitable, and it generates a new camera layout, creating a structure according to the story compose of scenes and sequences and the appropriate camera configuration for each sequence. It calculates the `validity` function output by computing a *specificity* value: whether the story shows a clear emotion within a predefined set, or match with one of the sequence predefined in the module, or the sequence of the scene don't lie in a cycle of camera configuration. It considers itself to be appropriate or not depending on how specific (as defined before) the partial story is: The higher the specificity, the higher the `appropriate` value. Finally, `apply-solution` updates module's state to generate a new structure with their appropriate camera configuration, if one solution is proposed by other modules. As a `new-solution` this module propuse new information like a movement for some character, or an emotional intensity for a sequence, or to introduce a new kind of predefined sequence in the story. With this information the new state of the modules will be update and checked again for correctness.
- The camera motion module works in the same way that the camera positioning module does. However, the output is composed of camera movements (track, pan, tilt) instead of positions.

4 Discussion

The framework and Spiel are not creative systems (the main algorithm is quite simple): the processes they run are rule based, without any creative idea. What the framework does is to handle, in a kind of opportunist and distributed way, different heterogeneous processes. What we want to discuss, basing on the main conclusions we have obtained during the development of the framework and its instantiation, is whether a creative object can be obtained by applying this kind of processes (trying to combine several generative systems), how can creativity emerge.

Although there are different computational systems focused on generating creative objects (like stories), and whole systems for creating 3D movies (or parts of them), there is not much focus on whole systems trying to generate

creative 3D movies. Of course, SPIEL is far from being able of such a creation, and this paper just shows the first steps towards this ambitious objective. The main point of this text is to discuss whether this is possible, and what processes could lead to creativity.

The main point we can argue is that this kind of frameworks can lead to results which would not be the output of a sequential system (that is, creating a story, and, after that, creating a camera layout). Modules provide feedback for other modules, which should, in principle, lead to more "enriched" results.

For this to be possible, it is important to create a good definition for choosing which module to apply and a the evaluation functions for the system. If these functions are just focused on choosing which module returns the highest values, probably the output of the system will not be much better. However, the study of how collaborative decission for this functions can lead to emergent creativiy is interesting.

For instance, if the result of the global *evaluation* value (as defined in Section 3.1) is computed taking into account every module's result (perhaps computing the mean value), the heterogeneous systems collaborate, and the global system gets enriched by this process, in the sense that a new evaluation function emerges, and this function is different, and not previously designed by the module creator (although obviously it is previously conceived by the framework's operation).

5 Conclusions and Future Work

A framework for creating compound creative systems, has been presented, and an instantiation for it, SPIEL. It is fully focused on how independent systems can work together to build a complex creative object. The main conclusion is whether, even having obtained new data from the use of this system, these new data can be considered creative, or just a product of feeding modules with additional data.

Next versions of the framework and SPIEL will focus on creating more modules, trying to build a real 3D creation system. It is planned to create modules not dependent on the framework (and adding a wrapper for it). Letting the system follow a state-space search, like a backtracking (trying many different possible options in each iteration), would improve the system output.

Also, the greedy approach is only intended to show the preliminary possibilities of such a system, but a blackboard architecture, or a backtracking state space search could be interesting for this purpose, letting the modules communicate in a more powerful way.

References

1. Wiggins, G.: Searching for Computational Creativity. New Generation Computing, Computational Paradigms and Computational Intelligence. Special Issue: Computational Creativity **24**(3) (2006) 209–222

2. Boden, M.: Computational models of creativity. Handbook of Creativity (1999) 351–373
3. Boden, M.: Creative Mind: Myths and Mechanisms. Routledge, New York, NY, 10001 (2003)
4. Ritchie, G.: The Transformational Creativity Hypothesis. New Generation Computing, Computational Paradigms and Computational Intelligence. Special Issue: Computational Creativity **24**(3) (2006) 241–266
5. Christianson, D.B., Anderson, S.E., wei He, L., Salesin, D., Weld, D.S., Cohen, M.F.: Declarative camera control for automatic cinematography. (1996) 148–155
6. wei He, L., Cohen, M.F., Salesin, D.H.: The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM (1996) 217–224
7. Amerson, D., Kime, S.: Real-time Cinematic Camera Control for Interactive Narratives. In: Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, Stanford, CA, USA, AAAI Press (March 26-28 2001) 1–4
8. Lin, T.C., Shih, Z.C., Tsai, Y.T.: Cinematic Camera Control in 3D Computer Games. In: Proceedings of 12th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2004), University of West Bohemia, Campus Bory, Plzen-Bory, Czech Republic (February 2-6 2004) 289–296
9. Charles, F., Lugrin, J.L., Cavazza, M., Mead, S.J.: Real-time camera control for interactive storytelling. In Mehdi, Q.H., Gough, N.E., eds.: GAME-ON. (2002)
10. Pérez y Pérez, R.: MEXICA: A Computer Model of Creativity in Writing. PhD thesis, The University of Sussex (1999)
11. Turner, S.R.: Minstrel: A computer model of creativity and storytelling. Technical Report UCLA-AI-92-04, Computer Science Department, University of California (1992)
12. Meehan, J.: The Metanovel: Writing Stories by Computer. PhD thesis (1976)
13. Riedl, M.: Narrative Planning: Balancing Plot and Character. PhD thesis, Department of Computer Science, North Carolina State University (2004)
14. Lebowitz, M.: Creating characters in a story-telling universe. Poetics **13** (1984) 171–194