How to Build a CC System
Paper type: System Description Paper

Dan Ventura
Computer Science Department
Brigham Young University
Provo, UT 84602 USA
ventura@cs.byu.edu

Abstract

Building a computationally creative system is a challenging undertaking. While such systems are beginning to proliferate, and a good number of them have been reasonably well-documented, it may seem, especially to newcomers to the field, that each system is a bespoke design that bears little chance of revealing any general knowledge about CC system building. This paper seeks to dispel this concern by presenting an abstract CC system description, or, in other words a practical, general approach for constructing CC systems.

Introduction

The broad field of computational creativity (CC) admits a range of autonomy, from creativity support tools to co-creative systems to fully autonomous artificial agents, and it is this last extreme which is the focus here. The notion of an (autonomous) creative agent has been instantiated in many different forms, and a variety of systems of varying degrees of sophistication and efficacy have been built by the CC community for creating artefacts in a broad range of domains. Many of these systems have been documented in the literature and their mechanisms described in some level of detail. The goal of this paper is to generalize multiple approaches from different domains into an abstract system description that provides a sort of blueprint for how to build a CC system for an arbitrary domain. The intent is to provide a fairly straightforward distillation of (one view of) what is involved in the construction of such a system, both to provide a hands-on guide for the newcomer wishing to build such a system and to stimulate discussion about what exactly *is* the right way to go about building such a system.

To begin with, in this paper, *computationally creative agent* means an agent whose behavior exhibits three characteristics: *novelty*, *value* and *intentionality*. Note that the first two are most commonly addressed with respect to product[1], while the third deals with process. For the purposes of this discussion, these characteristics will be defined as follows:

novelty: the quality of being new, original or unusual; this is relative to the population of artefacts in the domain in question and can apply in the personal or historical sense.

value: the importance, worth or usefulness of something; this would typically be ascribed by practitioners of the domain in question.

intentionality: the fact of being deliberative or purposive; that is, the output of the system is the result of the system having a goal or objective—the system's product is correlated with its process.

The goal here is to lay out how to build an autonomous CC system, but it is, of course, possible to selectively apply parts of what follows to build co-creative systems or even "simple" creativity support tools as well, with the differentiation (most) dependent on the level of creative responsibility with which the final system is endowed. It is important to note that the guidelines presented here are one current view of things, based on experience both building and reviewing multiple CC systems in widely differing domains, and while the general concepts are meant to be somewhat definitive, the examples and methodological suggestions given are meant to be representative rather than prescriptive.

Building the System

The goal of an autonomous CC system is to intentionally produce artefacts that are both novel and valuable in a given domain. Figure 1 gives a functional design for such a system, embedded in the target *domain*. The system is embedded in the domain for several reasons: it has a domain-specific *knowledge base*; it has a domain-appropriate *aesthetic*; and it has the ability to externalize artefacts that potentially can contribute to the domain. In addition, the system has an internal *representation* of artefacts that allows it to reason about domain-related concepts and manipulate these concepts to *generate* potential artefacts. It also has the ability, based on its aesthetic, to *evaluate* both its internal *conceptualization* and the *translation* of this conceptualization into the realization of an artefact in the domain. Each of these components will be considered in turn, while emphasizing that there is not a strict linear ordering to either their development or their deployment; rather, both system development and system operation are more likely semi-ordered, iterative processes.

[1]But also note that the use of "product" here is abstract, and that, in particular, the artefact produced might itself be a process.

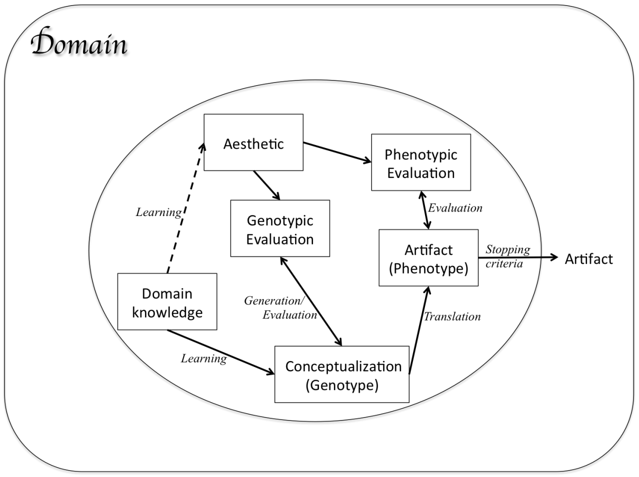

Figure 1: An abstract CC system is embedded in a particular domain of interest by incorporating both a repository of domain knowledge and a domain-appropriate aesthetic that together inform the production of artefacts that (potentially) contribute to that domain. Artefacts are represented internally as a genotypic conceptualization that is manipulated and evaluated internally and is eventually translated into an externalizable phenotypic representation that is further evaluated for its suitability before (potentially) being exported to the domain.

Domain

In contrast to traditional artificial intelligence tasks, which are most often characterized with an objective function that is to be maximized (or minimized), the kinds of problems that CC systems are built to solve are of an entirely different class. There is no such thing as a best song, or best theorem or best design. One cannot maximize a piece of visual art or a recipe or a poem. There are many interesting songs, theorems, designs, paintings, recipes and poems, and the goal is to find one or more of these. What constitutes a "solution" for these types of "problems" is nothing like an optimization problem, at least in the traditional sense. The first step in building a CC system is to choose a domain D for which it would be useful to build such a system. Because almost any domain of endeavor can be argued to require creativity to meet at least some of its challenges, from the artistic to the scientific to the mundane, the choice is really limited only by the imagination. Indeed, successful CC systems already exist for a large variety of domains, including culinary recipes (Morris et al. 2012; Varshney et al. 2013), language constructs such as metaphor (Veale and Hao 2007) and neologism (Smith, Hintze, and Ventura 2014), visual art (Norton, Heath, and Ventura 2013; Colton 2012), poetry (Toivanen et al. 2012; Oliveira 2012; Veale 2013), humor (Stock and Strapparava 2003; Binsted and Ritchie 1994), advertising and slogans (Strapparava, Valitutti, and Stock 2007; Özbal, Pighin, and Strapparava 2013), narrative and story telling (Riedl and Young 2010; Pérez y Pérez and Sharples 2004), mathematics (Colton, Bundy, and Walsh 1999), games (Cook, Colton, and Gow 2016; Liapis, Yannakakis, and Togelius 2012) and music (Bickerman et al. 2010; Pachet and Roy 2014). Of course, none of these domains can yet be considered "solved" by CC (indeed, for CC problems, it is not clear that the idea of "solving" even makes sense), so much work remains to be done even here. However, certainly there are many more domains for which the development of CC systems will prove beneficial: product design (a very general domain that could be further specialized to automobile design, electronics, clothing, software apps, etc.), architecture, drug design, protein synthesis, trip planning, robotics (physical systems, path planning, goal generation, etc.)—the list is endless.

Representation

Given a domain, it is necessary to next choose an appropriate representation for artefacts in that domain, and for the general case, that entails actually choosing two representations: a *phenotype* p that is an external, public representation, and a *genotype* g that is an internal, private representation.[2]

Given the domain D, the phenotypic representation is often at least somewhat prescribed. For example, for the domain of narrative, the phenotype must be some version of a story, for the domain of music it must be some kind of song, for visual art, a painting, for mathematics, a theorem. However, it may be convenient to choose some modification $P \simeq D$ as the phenotypic representation: a story outline, a lead sheet, a digital image, a sequence of predicates. Then, a specific phenotype p is the representation of an artefact in the domain D (or its surrogate P). So, $p \in P \simeq D$.

The genotypic representation G may be very different from both D and P; it should be a convenient form for knowledge representation, reasoning, and manipulation. Using the same four examples, for the domain of narrative, it might be entity-relationship graphs, schemata or plans; for music, it might be MIDI or lead sheets or viewpoints; for visual art, it could be arrays of pixel values, sequences of image filters or sets of image segments; for mathematics, it might be Prolog programs, trees or Boolean formulas. A specific genotype $g \in G$ is an encoding of a phenotype p that is used internally by the system.

It will also be important to decide on a representation for domain knowledge that will be used as the system's knowledge-base or experience. It may be convenient to adopt the genotypic representation for representing knowledge about the domain, or, it may be convenient to use the phenotypic representation, or both.

Knowledge Base

Having the question of domain knowledge representation settled, the next task is deciding on a way to collect representations of that knowledge into a knowledge base K. This will serve as the system's experience and provide it with its connection to the domain. Perhaps the most common means of populating K is by leveraging the web in some way: scraping websites and cleaning the obtained data,

[2]We appropriate these terms without intending to imply the normal biological or evolutionary connotations with which they are usually associated.

open-access or for-purchase corpora or databases, crowd-sourcing, etc. K can also be populated using experts to construct bespoke rules, prototypes, knowledge graphs, semantic networks and the like. In the case of a system for creating recipes, one might scrape recipe websites for ingredients, example recipes, their rankings, their categories, etc.; for a poetry creation system, resources such as Google n-grams, WordNet and ConceptNet might serve as a knowledge base; for a joke-writing system, the knowledge base might contain a set of rules constructed by professional humorists; for a system intended to invent board games, K could consist of a set of known board games (represented in an appropriate language, such as the Stanford GDL (Love et al. 2006)). This knowledge base K is used as a starting point for the rest of the system; in particular, it will be used to learn some kind of conceptual model of the domain and may also be used to learn an appropriate aesthetic as well.

Aesthetic

An aesthetic A is an abstract measure of quality for artefacts in the domain. Given the goal of producing artefacts that are valuable and novel, this quality should in some way be correlated with these. Continuing the second set of examples, aesthetic considerations for recipes might include

- appeal (is it tasty?),
- nutritional value (is it healthy?),
- cost/availability of ingredients (is it affordable?),
- surprising flavors (do these flavors somehow complement each other?), etc.;

for poetry, they might include

- semantic coherence (does it make sense?),
- interestingness of theme (will it hold the reader's attention?),
- metrical and/or rhyming considerations (is it interesting to read?)
- and cultural reference (does it apply?);

for jokes, good measures may include

- funniness (will it make people laugh?),
- accessibility (will people get it?),
- surprise (is it different than expected?),
- timeliness (does it reference pop culture or current events?),
- potential for shock or insult (will it make people angry?);

for board games, aesthetic factors might include

- playability (do the rules actually work?),
- winnability (can the game be won?),
- amount of time required (is it too long?),
- complexity (will people understand how to play?),
- enjoyability (is it fun to play?),
- and social considerations (how many people play? how do they interact?).

The most straightforward way to imbue the system with an aesthetic is to simply give it one—as the system designer, decide on some set of measures for the system to use.

Another approach is to have the system learn an aesthetic from the knowledge base, $A = \lambda_a(K)$. That is, given the information in K about (presumably both good and bad) artefacts from the domain, the system makes use of some learning function $\lambda_a : \mathcal{K} \to \mathcal{A}$, to infer an aesthetic $A \in \mathcal{A}$, where $\mathcal{K}$ is the set of all knowledge bases and $\mathcal{A}$ is the set of all aesthetic measures. This is appealing for its greater system autonomy, but it may be difficult to effect, and the result may not be interpretable (i.e., one may not know what aesthetic the system is using, though some will argue this as a positive advance).

Conceptualization

A conceptualization C of K is some kind of model of the knowledge that facilitates the understanding of, and thus the creation of, artefacts in the domain D (or the surrogate domain P). This model is constructed via some kind of learning process $\lambda_c : \mathcal{K} \to \mathcal{C}$, where $\mathcal{C}$ is the set of all conceptual models; so, $C = \lambda_c(K)$. The form of this conceptualization can vary widely, but ideally it should admit some method of reasoning about it and should be mutable. It should also facilitate generation of artefact genotypes. Representative examples include, for narrative, sets of characters, relationships and actions or an engagement-reflection model; for music, (hidden) Markov chains of transition probabilities between pitches, durations or chords or probabilistic context sensitive grammars; for visual art, a library of semantically clustered images or generative adversarial networks; for mathematics, axioms and operators or genetic programs; for recipes, a list of ingredients and their relative or absolute statistics or a model of chemical properties of ingredients; for poetry, templates or recurrent neural networks or n-grams; for jokes, templates or skip-thought vectors; for board games, a probabilistic grammar or set of cases.

Generation

The conceptualization should allow the system to generate artefact genotypes via some generative function $\gamma : \mathcal{C} \times \mathcal{S} \to G$, where $\mathcal{S}$ is a placeholder set meant to include anything that might be useful in the generation process: inspiration sources, randomness, examples, or even nothing at all. This generative process might be a natural extension of the conceptualization, or it might be quite distinct from it. Examples of the former include (hidden) Markov chains (just sample the chain for some length of sequence), probabilistic context sensitive grammars (sample a grammatical derivation), generative adversarial networks (cycle between generative network and adversarial network until output stabilizes/error is low), recurrent neural networks (just stimulate the network and record the output for the desired sequence length). Examples of the latter include combinatoric approaches, chaining pre- and post-conditions, logic programs, genetic algorithms, genetic programs, physical modeling or simulation, template filling, and nearest neighbor methods.

Genotypic Evaluator

Given the ability to generate candidate genotypes, the system requires some way to evaluate those genotypes—those judged to have quality will be converted to phenotypes that are further evaluated and may be released to the domain as successful creations; those judged unfit can be discarded or possibly modified to increase their quality. This evaluation should consist of both a domain-specific assessment of the value of an artefact and some kind of similarity measure for artefacts; these combined give a measure of both value and novelty—the value rule allows filtering for value potential and the similarity measure can be used to compare a candidate with the population of K as well as with artefacts the system itself has previously created. Thus, what is required is an evaluation function $\epsilon_g : G \rightarrow [0...1]$ which assigns some real-valued quality score to an artefact g by taking into account both the value and the novelty of G, perhaps most simply as a linear combination: $\epsilon_g(g) = \alpha \upsilon_g(g) + (1-\alpha)\nu_g(g)$, where $\upsilon_g : G \rightarrow [0...1]$ computes a value score for g, $\nu_g : G \rightarrow [0...1]$ computes a novelty score for g and $0 \leq \alpha \leq 1$.

The novelty score returned by ν_g should correlate in some way with a notion of distance from known artefacts (i.e. those in $K \cup Z$, where Z contains those artefacts [already] successfully created by the system), with a higher novelty score indicating a greater distance from known artefacts. The notion of distance will be representation-specific, of course, and could be something as simple as a Hamming distance (in the case of binary strings), as common as Euclidean distance for representations as points in $\mathcal{R}^n$ or as exotic as a generalized edit distance, where the notion of editing is appropriately defined. Though this distance will typically be explicitly designed based on the representation, there may be situations in which it could make sense for the system to learn a notion of distance (or even novelty directly) empirically from data in K.

The value score returned by υ_g should correlate with the aesthetic A. That is, there should be some mapping $\phi_g : \mathcal{A} \rightarrow \mathcal{E}$, where $\mathcal{E}$ is the set of valuation functions whose domain is G and whose range is $[0...1]$. This mapping will likely be something *ad hoc*, with the function υ_g, being explicitly designed as part of the system, though again, it may be possible to instead implement ϕ_g and allow the system to learn υ_g. Another way to think of this is that υ_g operationalizes A. Examples of a valuation function might include measure of affect, or sentiment (via dictionaries) or tension/resolution for stories; measuring melodic shape, number of key changes, pitch range, uniqueness of chord progression for music; affect, subject matter, color usage, style for visual art; generality or simplicity for mathematics; existence of unique ingredient combinations, complementary chemical taste profiles, number of calories or cost of ingredients for recipes; identifiable meter, rhyming, affect, sentiment, word usage for poetry; word usage, current event usage, sentiment, incongruity in jokes; number of turns required, rule complexity, number of players, number of ludic conditions, likelihood of a draw for board games.

Finally, it should be mentioned that it is possible in some cases to partially or completely incorporate the functionality of ϵ_g into the generative function γ, obviating the need for a separate evaluation of g. For example, if a certain metrical structure or rhyme scheme is highly valued in a poetic form, the generative process may constrain all outputs to follow that structure, or if the combination of dairy and meat products in a recipe were considered undesirable, the generative process may disallow the combination. Of course, this kind of constrained generation precludes exploring some parts of the domain, so it should be used cautiously.

Translation

Given that the system is working with an internal, genotypic representation g but that it must ultimately produce an external, phenotypic representation p, some method of translation is necessary. This translation mechanism may be thought of as a helper function $\tau : \{G \cup \perp\} \rightarrow \{P \cup \perp\}$, where $\perp$ represents the null artefact, which produces a phenotype from a genotype; thus, $p = \tau(g)$ and $\perp = \tau(\perp)$. Using τ, the system turns a schema into a story or a Markov chain into sheet music or a sequence of image filters into an image or a Prolog program into a proof or a binary string into a list of ingredients or a template into a poem or a bag of words into a joke or a probabilistic grammar into a board game (description). This may be one of the easiest parts of the system (e.g., obtaining an image from a sequence of image filters is usually a simple matter of composing a few well-defined function calls) or one of the most difficult parts (e.g., composing a punchline from a bag of key words to make a joke funny is completely ill-defined).

Phenotype Evaluator

Genotypes that are evaluated highly enough will be translated (via τ) into candidate phenotypes that must be evaluated in their own right, with a function $\epsilon_p : P \rightarrow [0...1]$. This evaluation should be qualitatively different than its genotypic counterpart because a) this is a different representation and b) there is nothing to be gained by re-using the same evaluation criteria. This is analogous to evaluating a piece of sheet music (genotype) by checking its agreement with music theoretic principles on the one hand and listening to how an audio track (phenotype) of the music sounds when played (translation). This function, too, should correlate with notions of value and/or novelty. And, again, like ϵ_g, ϵ_p may be designed as an(other) operationalization of A, or it may be learned by the system using a function $\phi_p : \mathcal{A} \rightarrow \mathcal{E}$.

While the evaluator ϵ_g is most often somewhat piecewise and (in some sense) cognitive in nature, ϵ_p may be more holistic and very often is based in some form of perception. Because of this, domains seem to vary widely in the ease with which a phenotypic evaluator can be developed. For some domains, we have a fairly good understanding of the perception necessary to construct a phenotypic evaluation: some kind of audio signal processing for music; computer vision-based techniques for visual art; chemical analysis for recipes; (simulated) game play for board games. For other domains, this perceptual understanding is much less developed, and it is less clear how phenotypic evaluation can be done: how do readers perceive a story as interesting? how do mathematicians see beauty in a theorem? how do listeners feel an emotional connection with a poem? how does an audience find a joke funny?

Putting it all Together

With all the pieces in place, it is now possible to summarize the entire process as follows. To build a CC system and produce an artefact, follow these steps:

1. Choose a domain D
2. Choose a genotypic representation G and a phenotypic representation $P \simeq D$
3. Collect data and build a knowledge base K
4. Choose a generator function γ
5. Choose aesthetic A [or build a function λ_a that allows the system to learn $A = \lambda_a(K)$]
6. Choose a novelty measure ν_g and a value measure υ_g [or build a function ϕ_g that allows the system to learn $\upsilon_g = \phi_g(A)$] and construct a genotypic evaluator ϵ_g from ν_g and υ_g
7. Choose a phenotypic evaluator ϵ_p [or build a function ϕ_p that allows the system to learn $\epsilon_p = \phi_p(A)$]
8. $C \leftarrow \lambda_c(K)$
9. While $\epsilon_p(p) < \theta_p$

```
    While $\epsilon_g(g) < \theta_g$
      $g \leftarrow \gamma(C, \rho)$
    $p \leftarrow \tau(g)$
```

10. return(p)

Looking at this more formally requires two additional helper functions, $\Theta_g : \{G \cup \perp\} \to \{G \cup \perp\}$ and $\Theta_p : \{P \cup \perp\} \to \{P \cup \perp\}$, defined as follows:

$$\Theta_g(x) = \begin{cases} \perp & \text{if } x = \perp \\ x & \text{if } \epsilon_g(x) >= \theta \\ \perp & \text{if } \epsilon_g(x) < \theta \end{cases}$$

where θ is some threshold of acceptability, and Θ_p is defined similarly. With these, the entire system operation for artefact creation can be expressed as

$$\begin{aligned} a &= \Theta_p(\tau(\Theta_g(\gamma(\lambda_c(K), \rho)))) \\ &= \Theta_p(\tau(\Theta_g(\gamma(C, \rho)))) \\ &= \Theta_p(\tau(\Theta_g(g))) \\ &= \Theta_p(\tau(g)) \\ &= \Theta_p(p) \\ &= p \end{aligned}$$

when $\Theta_g(g) = g$ and $\Theta_p(p) = p$, and $a = \perp$ otherwise.

Variations and Further Considerations

Simplifying

For some domains, it might be unnecessary to work with both phenotypic and genotypic representations. While, in general, this will likely limit the ability of the system to produce quality artefacts, in the case of simple tasks, prototyping or other less complex scenarios, it may be possible to work directly with only a phenotypic representation. This simplifies system design significantly (see Figure 2).

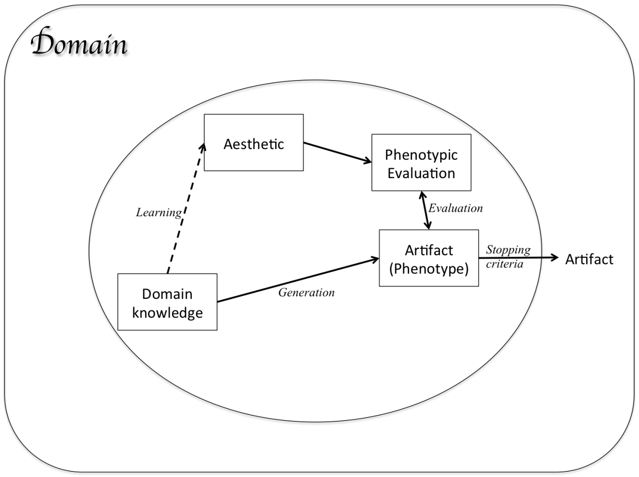

Figure 2: A simplified abstract CC system may eschew an internal representation, simplifying both the generation and evaluation processes. The system is still embedded in a domain with both a knowledge base and an aesthetic, but now its internal and external representations are equivalent, obviating the need for differentiated evaluation/generation mechanisms. It is likely that such a system will be limited in its ability to intentionally produce quality artefacts for most domains due to the lack of an ability for conceptualization.

It is still necessary to choose a domain D and phenotypic representation $P \simeq D$, collect a knowledge base K, produce an aesthetic A and its operationalization in the form of a phenotypic evaluation function ϵ_p; however, the list of components no longer required includes the genotypic representation G, the translation function τ, the genotypic evaluator ϵ_g, the conceptualization model C and the learning mechanism λ_c. In addition, because the system no longer has any conceptualization nor genotypic representation, the generation function γ must be modified so that it depends directly on K rather than on C and so that it outputs a phenotype p rather than a genotype g; thus, now $\gamma : \mathcal{K} \times \mathcal{S} \to P$.

Then, the simplified process is

1. Choose a domain D
2. Choose a phenotypic representation $P \simeq D$
3. Collect data and build knowledge base K
4. Choose a generator function γ
5. Choose aesthetic A
6. Choose a phenotypic evaluator ϵ_p
7. While $\epsilon_p(p) < \theta_p$

```
    $g \leftarrow \gamma(K, \rho)$
```

8. return(p)

and the formal description simplifies to

$$\begin{aligned} a &= \Theta_p(\gamma(K, \rho)) \\ &= \Theta_p(p) \\ &= p \end{aligned}$$

Complexifying

Of course, in many more situations, not only will it not be possible to simplify the original system as just discussed, but also it will likely be necessary to introduce further complexity to obtain satisfactory results. This complexification can come in many forms, and more of these will be discussed later, but the most natural next step is likely to introduce further domain-system interaction in the form of teaching and feedback. This ability allows the system to change over time, assimilating new domain knowledge as it becomes available and incorporating feedback about its creations. This new knowledge and feedback can directly affect the knowledge base K, directly or indirectly affect the aesthetic A and directly or indirectly affect the conceptualization C (see Figure 3). These effects can (and should) be propagated throughout the system, introducing the need for a time index t and obviating the need for some initial design decisions (e.g. even the knowledge base does not need to be fixed in advance). This facilitates dynamic knowledge acquisition, conceptualization, evaluation and generative abilities. Thus, the system can react to changes in the environment and become distanced from initial designer decisions.

Once again revisiting the process, a time index is now incorporated, but the overall flow remains recognizable:

```
1. Choose a domain $D$
2. Choose a genotypic representation $G$ and a phenotypic representation $P \simeq D$
3. Choose a generator function $\gamma$
4. Choose aesthetic learning function $\lambda_a$
5. Choose a novelty measure $\nu_g$
6. Choose genotype evaluation learning function $\phi_g$
7. Choose phenotype evaluation learning function $\phi_p$
8. $t = 0$
9. While not done
     $t = t + 1$
     Collect data and build knowledge base $K^t$
     $A^t \leftarrow \lambda_a(K^t)$
     $\upsilon_g^t \leftarrow \phi_g(A^t)$
     construct a genotypic evaluator $\epsilon_g^t$ from $\nu_g$ and $\upsilon_g^t$
     $\epsilon_p^t \leftarrow \phi_p(A^t)$
     $C^t \leftarrow \lambda_c(K^t)$
     While $\epsilon_p^t(p) < \theta_p$
       While $\epsilon_g^t(g) < \theta_g$
         $g \leftarrow \gamma(C^t, \rho)$
       $p \leftarrow \tau(g)$
     return(p)
```

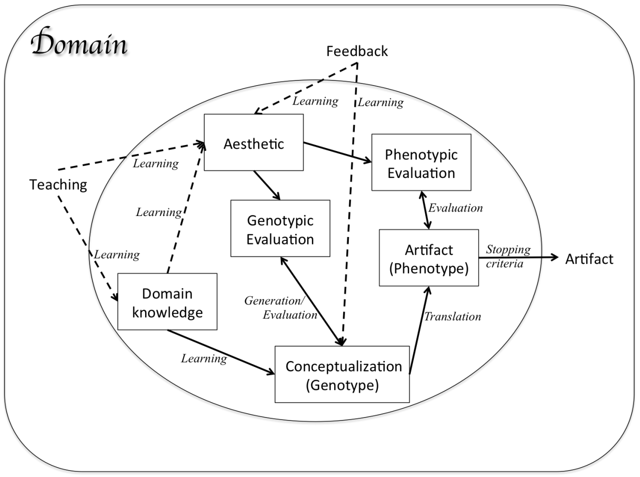

Figure 3: For increased autonomy, and therefore, presumably increased (potential) creativity, a CC system should not only be embedded in a domain but should also be capable of interacting with its embedding domain. This interaction will most often take form of teaching and feedback that allows the system both to dynamically update its background knowledge base and aesthetic as new domain information becomes available and to adapt its aesthetic, conceptualization and generation mechanisms based on feedback it receives in response to the artefacts it creates (and releases externally).

(The formal view of this does not change in an interesting way, so it will not be repeated here with only minor changes.)

Note the shift towards greater autonomy in the fact that the *a priori* design decisions (made initially, before the operational loop) are being made for more abstract entities (i.e., designing functions for learning components rather than designing the components themselves). Also, note that some of this shift towards greater autonomy is not strictly necessary in the sense that design decisions that make some system component time-independent are certainly still possible. On the other hand, it is also possible to consider designing additional abstract learning functions that allow additional components to become time dependent (e.g., γ, λ_a, λ_c, ν_g, τ, θ_g, θ_p).

Intentionality

Though novelty and value have been incorporated into the construction or learning of the evaluation functions, system intentionality has not yet been explicitly addressed. In what way can a system built using the proposed approach be claimed to be intentional?

The answer is first that, in a limited sense, the system has a goal to produce novel, valuable artefacts in the domain D. This may not be entirely satisfactory to critics who may argue (correctly) that the goal is imposed on the system by an external agent (the designer); however, the definition of intentionality offered in the introduction does not require that the system invent its own goals (see the discussion on Turtles below).

Another way the system may be said to exhibit intentionality is if its product (the artefacts it produces) is correlated with its process (how it produces them). This is certainly the case, by design, especially with respect to, for example, the aesthetic-based evaluation mechanisms—the system has intention to some extent because to some extent it "understands" what it is creating.

Yet another possible indicator for intention, and one not explicitly included in the CC system building process proposed here, is an ability of the system to explain its process and/or product—can the system justify in some way why it made the decisions it made and why the result is what it is? This is one example of the broad notion of *framing*, and though not required as part of the blueprint provided here, it is often desirable and sometimes not too difficult to incorporate some basic framing ability in CC systems (e.g., showing source material for musical inspiration, explaining perception of images, showing the connection between setup and punchline, etc.), and when this is done, it provides further support for a claim of system intentionality.

(External) Evaluation

Once the system is running and producing artefacts, at some point the question must be asked whether it is doing so satisfactorily. Of course, from the system's perspective, this question has already been answered in the affirmative—by design, any artefact produced has already been vetted (in two different ways) and found to meet aesthetic and uniqueness standards. However, this is not sufficient in the sense that creation and contribution to a domain are inherently social processes—a creator can not be the sole arbiter when it comes to the question of its creativity. Therefore, it becomes important to subject the system to external scrutiny. While how best to do this is still an open question, there have been multiple proposals for evaluation mechanisms for CC systems. Collectively, these can examine both system product and process and include Ritchie's suggestions for formally stated empirical criteria focusing on the relative value and novelty of system output (2007); the FACE framework for qualifying different kinds of creative acts performed by a system (Colton, Charnley, and Pease 2011); the SPECS methodology which requires evaluating the system against standards that are drawn from a system specification-based characterization of creativity (Jordanous 2012); and Ventura's proposed spectrum of abstract prototype systems that can be used as landmarks by which specific CC systems can be evaluated for their relative creative ability (2016).

A Note on Appropriation

Pablo Picasso is often credited with having said, "good artists borrow; great artists steal". In the context of building a CC system, it is perhaps stating the obvious to suggest that one should avoid re-inventing the wheel when possible. There are a great number of existing tools, web services, and APIs that can be incorporated into the architecture of a CC system as solutions for many of the components discussed here. In many cases, these tools are exactly what one is wanting and may be found with some little effort (e.g., the many extant NLP and computer vision tools freely available on the web); in other cases, useful resources exist, but may need to be discovered more serendipitously or cleverly re-purposed (e.g., using an online list of "Top Tens" as a source of pop culture references). The take away here should be that it is often worth spending significant time searching for existing resources instead of trying to build them from scratch—it is very often the case that someone has already done the work and done it well. Commonly, the main (engineering) contribution of a CC system is not in the implementation of components but in the system building and architectural work.

A Note on Turtles

It should be noted here that the proposed process for CC system building does not incorporate state-of-the-art thinking and address all the latest questions that are being asked in the field, and that is not its intention. However, it does perhaps come close to encapsulating the state-of-the-art with respect to actual working CC systems currently being built.

There is certainly a great deal more that can be (and is being) said about the topic, and, in particular, it should be mentioned that the field of CC is by nature begging for treatment at the meta-level. That is, it is fair game to consider the domain D of knowledge representations or conceptual models or aesthetics or learning functions or evaluation functions or goals, and to then attempt to build a CC system embedded in *that* domain—a system for inventing new and useful representations or conceptual models or aesthetics or learning/evaluation functions or goals. One could then consider building a hierarchical system that incorporates meta-level creativity to improve base-level creativity, increasing the overall system autonomy and further distancing the (eventual) creative system from the original designer.

Of course, this begs the question of meta-meta-level considerations, but it is not clear that such thinking is useful, either for CC systems, nor perhaps even with respect to human cognition, and in any case, it is beyond the scope of the current discussion.

Conclusion

The main contribution of this paper is a general recipe for the construction of an autonomous, computationally creative agent, with the intention being that anyone can follow the proposed process to build a CC agent for an arbitrary domain. Of course, domain-specific challenges are not addressed here and remain as opportunities for the system builders to act creatively themselves. A secondary contribution of the paper is the initiation of a conversation about whether or not the main contribution is realizable—is it possible to describe a useful, general-purpose approach to CC system building?

References

Bickerman, G.; Bosley, S.; Swire, P.; and Keller, R. M. 2010. Learning to create jazz melodies using deep belief nets. In Ventura, D.; Pease, A.; Pérez y Pérez, R.; Ritchie, G.; and Veale, T., eds., *Proceedings of the International Conference on Computational Creativity*, 228–237.

Binsted, K., and Ritchie, G. 1994. A symbolic description of punning riddles and its computer implementation. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, 633–638.

Colton, S.; Bundy, A.; and Walsh, T. 1999. HR: Automatic concept formation in pure mathematics. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 786–791.

Colton, S.; Charnley, J.; and Pease, A. 2011. Computational creativity theory: The FACE and IDEA descriptive models. In *Proceedings of the 2nd International Conference on Computational Creativity*, 90–95.

Colton, S. 2012. The Painting Fool: Stories from building an automated painter. In McCormack, J., and D'Inverno, M., eds., *Computers and Creativity*. Berlin, Germany: Springer-Verlag. 3–38.

Cook, M.; Colton, S.; and Gow, J. 2016. The ANGELINA videogame design system, part I. *IEEE Transactions on Computational Intelligence and AI in Games* to appear.

de Silva Garza, A. G., and Maher, M. 2000. A process model for evolutionary design case adaptation. In Gero, J., ed., *Proceedings of the Sixth International Conference on Artificial Intelligence in Design*, 393412. Kluwer Academic Publishers.

Jordanous, A. 2012. A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative. *Cognitive Computation* 4(3):246–279.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2012. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games* 4(3):213–228.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2006. General game playing: Game description language specification. Lg-2006-01, Stanford.

Morris, R.; Burton, S.; Bodily, P.; and Ventura, D. 2012. Soup over bean of pure joy: Culinary ruminations of an artificial chef. In *Proceedings of the 3rd International Conference on Computational Creativity*, 119–125.

Norton, D.; Heath, D.; and Ventura, D. 2013. Finding creativity in an artificial artist. *Journal of Creative Behavior* 47(2):106–124.

Oliveira, H. G. 2012. PoeTryMe: a versatile platform for poetry generation. In *Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence*.

Özbal, G.; Pighin, D.; and Strapparava, C. 2013. BRAINSUP: Brainstorming support for creative sentence generation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 1446–1455.

Pachet, F., and Roy, P. 2014. Non-conformant harmonization: the real book in the style of Take 6. In *Proceedings of the 5th International Conference on Computational Creativity*, 100–107.

Pérez y Pérez, R., and Sharples, M. 2004. Three computer-based models of storytelling: BRUTUS, MINSTREL and MEXICA. *Knowledge-Based Systems* 17(1):15–29.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research* 39(1):217–268.

Ritchie, G. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines* 17:67–99.

Smith, M. R.; Hintze, R. S.; and Ventura, D. 2014. Nehovah: A neologism creator nomen ipsum. In *Proceedings of the 5th International Conference on Computational Creativity*, 173–181.

Stock, O., and Strapparava, C. 2003. HAHAcronym: Humorous agents for humorous acronyms. *Humor - International Journal of Humor Research* 16(3):297–314.

Strapparava, C.; Valitutti, A.; and Stock, O. 2007. Automatizing two creative functions for advertising. In *Proceedings of 4th International Joint Workshop on Computational Creativity*, 99–105.

Toivanen, J. M.; Toivonen, H.; Valitutti, A.; and Gross, O. 2012. Corpus-based generation of content and form in poetry. In *Proceedings of the 3rd International Conference on Computational Creativity*, 175–179.

Varshney, L.; Pinel, F.; Varshney, K.; Schorgendorfer, A.; and Chee, Y.-M. 2013. Cognition as a part of computational creativity. In *Proceedings of the 12th IEEE International Conference on Cognitive Informatics and Cognitive Computing*, 36–43.

Veale, T., and Hao, Y. 2007. Comprehending and generating apt metaphors: A web-driven, case-based approach to figurative language. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, 1471–1476.

Veale, T. 2013. Less rhyme, more reason: Knowledge–based poetry generation with feeling, insight and wit. In Maher, M. L.; Veale, T.; Saunders, R.; and Bown, O., eds., *Proceedings of the Fourth International Conference on Computational Creativity*, 152–159.

Ventura, D. 2016. Mere generation: Essential barometer or dated concept? In Pachet, F.; Cardoso, A.; Corruble, V.; and Ghedini, F., eds., *Proceedings of the Seventh International Conference on Computational Creativity*, 17–24.