EvoSpace-i: A framework for Interactive Evolutionary Algorithms

author's names removed

ABSTRACT

Evolutionary art (EvoArt) encompasses a variety of research devoted to the development of evolutionary systems that can help produce artistic artifacts in an automated or semi-automated process. Given the difficulty of evaluating subjective artistic preferences, one of the main approaches used by EvoArt researchers is interactive evolution where user input guides the search. However, despite the growth of EvoArt over recent years the research area still lacks a comprehensive software tool that can help in the development of EvoArt applications. Therefore, this work presents EvoSpace-i, an open source framework for the development of collaborative-interactive evolutionary algorithms for art and design. The main components of the framework are: (i) *Evospace*, a population store for the development of cloud-based evolutionary algorithms, implemented using Redis key-value server; and an (ii) *Interactive web application* where end-users collaborate in a social network sharing, collecting, rating and ultimately evolving individuals. Individuals are presented as multimedia elements or artistic artifacts (images, animations, sound) using the Processing programming language, a development language specifically aimed at artists. EvoSpace-i is designed to be easy to use and setup, allowing researchers, and more importantly artists, to quickly develop distributed and collaborative EvoArt applications. This paper presents the main details of EvoSpace-i and two example applications to illustrate the potential of the tool.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Interactive evolutionary computation, Interactive Systems, Cloud-based platforms

GECCO'13, July 6-10, 2013, Amsterdam, The Netherlands.

1. INTRODUCTION

Currently, evolutionary computation (EC) techniques are applied in a wide variety of applications and problem domains. Among them, definitely one of the most intriguing application areas corresponds to what is broadly referred to as evolutionary art or EvoArt [1, 16]. EvoArt encompasses many works that seek to enhance the design abilities of researchers, or even artists, by exploiting the powerful search, optimization and learning capabilities of EC algorithms. In other words, EvoArt is an attempt by the EC community to enhance the creative process of artists and designers, or in a more ambitious scenario, reproduce a similar creative process in an autonomous system.

This goal is not trivial, particularly given the difficulty of modeling and expressing human artistic preferences in a computational system. Therefore, one important approach in EvoArt systems is to use interactive evolutionary algorithms (IEAs), where humans interact with the evolving population in a specific way, particularly in evaluating the quality of the evolving population [16, 15]. Through the use of IEAs many EvoArt applications have been successfully developed and deployed, as local applications or distributed over the web. Indeed, the promise in EvoArt and the quality of the research developed thus far is clear, given the large number of workshops and tutorials that are currently part of all of the major EC conferences.

However, there is much work and research questions that still lie in the horizon, issues that are in some instances theoretic while in others they are pragmatic. In this paper, we are concerned with the latter, in particular the current lack of a complete and integrated software tool for rapid development of EvoArt applications. The proposed platform is called EvoSpace-Interactive or EvoSpace-i for short, it is aimed at computer scientists who are interested in studying art and design, or at artists that want to exploit tools from computer science to enhance their abilities. Furthermore, EvoSpace-i provides a research and development tool that addresses the following specific issues within EvoArt.

Firstly, a common problem in IEAs is what is referred to as user fatigue [16], when an individual gets tired or simply bored with evaluating a large number of individuals, many of which will not be interesting at all. EvoSpace-i takes a collaborative approach, where preferences of multiple users are considered and integrated into the evolutionary process, thus EvoSpace-i is a collaborative IEA or C-IEA. This is done by exploiting the use of social networking and using a distributed computing model. A noteworthy feature of the approach taken in EvoSpace-i, is that it attempts to inte-

grate the way creative elements are shared between users, and how different preferences are integrated into a single search process. Thus, the term memetic evolution, as originally defined by Richard Dawkins [3], relying on both minds and computers could be finally a main component in the development of automatic artistic designs. Secondly, the distributed approach proposed in EvoSpace-i follows current trends in software and system development, where computational resources are shared across the web and applications are available on heterogenous computational devices. This is known as the Cloud computing model, where infrastructure, platforms and applications are shared across the internet. EvoSpace-i provides a platform that easily integrates into the the Cloud model, and can be used to develop EvoArt services that reside on the Cloud. Thirdly, an important aspect of EvoArt is the overlap between artists and computer science researchers, individuals that usually have very different academic backgrounds. Therefore, EvoSpace-i attempts to integrate design tools that can be used by both. In particular, EvoSpace-i exploits the Processing programming language to generate artistic designs, since Processing allows for easy representation of images, animations, audio, and data processing procedures that are intuitive to the non-programmer.

This paper provides a comprehensive presentation of the EvoSpace-i tool, describing the overall goals of the system, design principles and implementation details. Moreover, two example applications are reviewed, to illustrate how EvoSpace-i could be used for the deployment of distributed and collaborative EvoArt systems. The remainder of the paper proceeds as follows. Related Work is discussed in Section and the general architecture of the framework is presented in Section 3. Details about the components needed to enable the interaction between users and individuals are presented in Section 4 and collaboration between users is discussed in Section 5. Finally, two example applications are reviewed in Section 6, and final conclusions are given in Section 7.

2. RELATED WORK

The most common use of evolutionary algorithms is to search for solutions that are optimal with respect to a specific objective or fitness function that can be computed automatically. Conversely, as stated above, in EvoArt the goal is to evolve artistic designs where quality evaluation, almost by definition, will have a large subjective component. Therefore, over the years interactive algorithms have been developed and matured [16, 15]. However, interactive algorithms are not new, indeed some of the earliest EAs were open-ended interactive systems, such as the well-known Biomorphs program [4]. In what follows, relevant examples of interactive systems used in EvoArt are reviewed. In particular, we give examples of collaborative IEAs (C-IEAs) where many users interact and evaluate an evolving population, where the search is guided based on an aggregate of subjective preferences and considerations.

Langdon [11] developed one of the first C-IEAs, which evolves fractal representations of virtual creatures within a simulated environment. The proposed system is a distributed EA where the evolving population resides on a central server and individuals are distributed to remote web-clients using Javascript. Users evaluate individuals locally and preferred individuals are returned to the server, at which point they can be distributed to other clients.

More recent examples can be found in the work of Secretan et al. [14] and Clune and Lipson [2], both of whom use a web-based IEAs to evolve artistic artifacts using a generative encoding. In [14], the system is used to evolve static images and in [2] evolved objects are 3-D sculptures. Users of the systems can concentrate their search on specific artifacts, and collaboration is encouraged by allowing users to continue an evolutionary lineage created by others in a sequential manner. Both proposals offer webpages (see Picbreeder.org and EndlessForms.com), where users can select or create random individuals and evolve lineages of artifacts. Therefore, evolved artifacts can be the product of a collaborative search process. Another feature is that evolved artifacts can be rated by users, and since users can create individual accounts, the ratings provide a way to rank users, or to select previously evolved artifacts based on the particular style of each user. Furthermore, the collaborative process is captured by the system, since it is possible to visualize how, and when, different users influenced a genetic lineage.

Another example is the work of Kowaliw et al. [10], where ecosystemic models are evolved using a generative encoding based on multi-agent systems that generate high quality artistic drawings. Users interact with the system through a website and a Java applet, where they can evolve their own images and have the option to add evolved images to a central collection such that other users can see the resulting images.

The present work builds on previous proposals and extends the C-IEA approach. First, it promotes collaborative evaluations of artistic artifacts in a dynamic and parallel manner, instead of the sequential approach followed in [14, 2]. Second, it incorporates explicit user interactions by encouraging the use of social networking. Third, by exploiting the Processing programming language for graphics programming, it facilitates rapid development for artists with a limited computer science background. Fourth, it focuses on the evolution of artistic animations, not static pictures or paintings; another feature facilitated by the use of Processing. Fifth, it facilitates the ability to save and share promising artifacts. Finally, it emphasizes the use of a distributed model, to exploit current trends in software and hardware technology.

3. ARCHITECTURE

The goal of this work is to develop an open source framework for web and cloud-based C-IEAs, using current web standards and libraries for desktop and mobile devices. Developers of C-IEA applications are liberated from the need to design and program a platform for distributed user collaboration. Only three components of the framework must to be defined for each application (marked with double lines in Figure 1), namely: an *individual* representation; a *Processing script* that renders each individual; and a *worker* script that encodes the evolutionary operators will need to be defined according to the representation and problem domain. However, in future versions of the framework much of this work could be predefined, but also left open for advanced users to change as they require. What the framework offers for free is: a central repository for the population implemented as an EvoSpace service; and a Web Application script implemented using Django, a mature full stack Web Framework with a BSD license developed in Python.

The main components of the EvoSpace-i are shown in

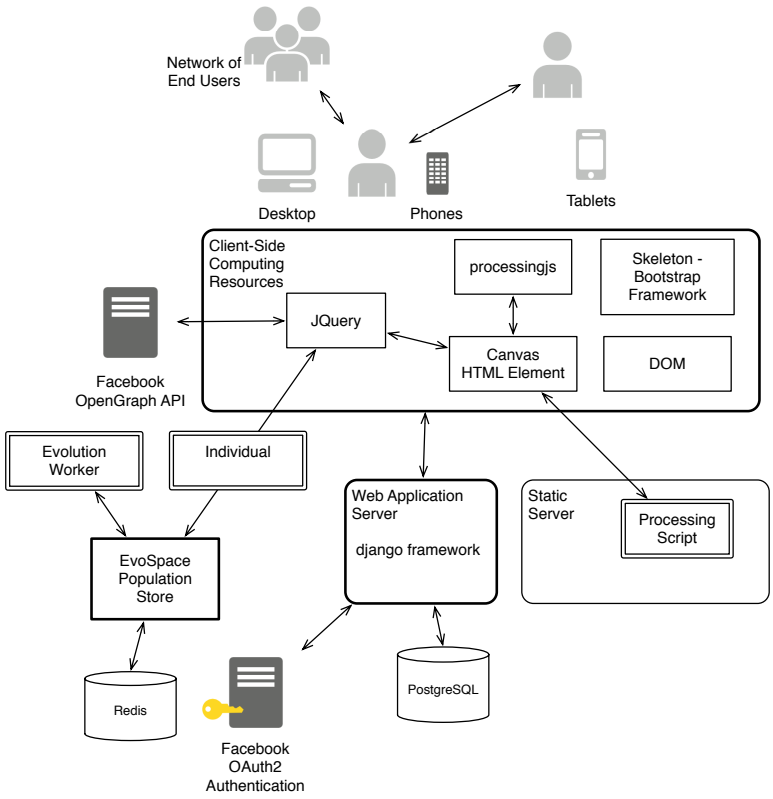

Figure 1: Main components of EvoSpace-i.

Figure 1. The Interactive part is a Django [9] web-based application, together with a client-side component implemented using JQuery and processingjs Javascript libraries. This application shows users a number of multimedia objects rendered in HTML5 Canvas elements. These multimedia objects are the phenotypes of individuals drawn from the Evospace population store. Evospace is responsible for storing and retrieving the data of each individual. An Evolution Worker is responsible for generating new individuals from a sample taken from EvoSpace. The evolution process is decoupled from the interactive application; thus giving designers the opportunity to define their own variations of the algorithm. In the following sections each component is described in detail, staring with the population store Evospace, and then the interactive and collaborative components.

4. EVOSPACE INTERACTIVE EVOLUTION

In this section, the components needed to enable the interaction between users and individuals are presented. EvoSpace is presented first, then the rendering and representation of individuals. Finally, the assignment of fitness to individuals and the current breeding process is described.

4.1 EvoSpace

The EvoSpace model is presented in detail in [6], only a brief description of the functionality related to this work is given next. EvoSpace is inspired on the Linda language by Gelernter and Carriero [7]; Linda is a model of coordination and communication among several parallel processes operating upon objects stored in and retrieved from a shared, virtual, associative memory called *tuplespace*. In a tuplespace, tuples are read and removed by processes; once a tuple is taken, no other process can read it until it is written back. Similarly EvoSpace consists of two main components (see figure 2):

- The EvoSpace container that stores the evolving population and
- Remote clients called EvoWorkers, which execute the actual evolutionary process, while EvoSpace acts only as a population repository.

Figure 2 illustrates the main components and dataflow within EvoSpace. In a basic configuration, EvoWorkers take a random population sample from EvoSpace, and use it as the initial population for a local EA executed on the client machine. After a certain number of local generations, the evolved population is returned to EvoSpace to replace the sample. When taken by an EvoWorker, individuals remain in a phantom state until their sample is returned. When individuals are in this phantom state, they cannot be taken by other workers. If an EvoWorker does not returns a sample in a certain amount of time, for instance because of a lost connection; individuals are re-spawned (re-inserted) by the ReInsertionManager. Similar to video games in which characters once killed are re-spawned after a certain time. This can also happen when the EvoSpace container starves or the population size is below some threshold.

For this version of EvoSpace-i, a new type of Worker is needed: Human users. Users are responsible for evaluating

the quality of individuals, the process is depicted in Figure 3: (i) first a random sample of six individuals are taken from EvoSpace, (ii) the chromosome of each individual parameterizes a Processing script, that renders to the user, (iii) users select those individuals they like, this is stored in each individual's data, (iv) finally the sample is returned to EvoSpace. The fitness assigned to each individual can depend on the ratings given by a certain number of users. In this case the Evo Worker process is replaced by an `Evolve()` method, that is executed after a certain number of samples have been returned. Unlike the normal operation of EvoSpace, when a user takes a sample of individuals, these are returned with their identity unchanged, other than the rating added by the current user. The internal representation of individuals is presented next.

4.2 Individuals

As stated above, the objects stored in EvoSpace are individuals in an EA. Explicitly, individuals are stored as *dictionaries*, an abstract data type that represents a collection of unique keys and values with a one to one association. In this case, keys represent specific properties of each object and the values can be of different types, such as numbers, strings, lists, tuples or other dictionaries. In the current implementation, individuals are described by the following basic fields. An **id** string that represents a unique identifier for each object. A **chromosome** string, which depends on the EA and the representation used. The **fitness** dictionary for each individual; In EvoSpace-i it stores pairs of user's ids and timestamp values, which represent that a user has rated the individual with a like. Currently a user can rate each individual more than one time. The **views** The number of times the individual has been presented to a user. If a user has seen an individual and did not assigned it a like, this can be used as a probable not-like rating. A **parents** dictionary with identifiers of the individual(s) from which it was produced. Finally, a **GeneticOperator** string that specifies the operator that produced it.

Individuals are stored in-memory, using the Redis key-value database. Redis was chosen over a SQL-based management system, or other non-SQL alternatives, because it provides a hash based implementation of sets and queues which are natural data structures for the EvoSpace model. For example, selecting a random key from a set has a complexity of O(1). The logic of EvoSpace is implemented as a python module exposed by the same Django framework used by the web-based application. This is done by a JQuery ajax request, and using the JSON-RPC protocol. The EvoSpace module is available with a Simplified BSD License from `https://github.com/evoWeb/EvoSpace`. The components required for the interaction between individuals and users are presented next.

4.3 Processing Scripts

Processing is a programming language and development environment initially created to serve as a software sketchbook and as a tool to teach fundamentals of computer programming within a visual context. Currently is used by artists, designers, architects, and researchers for visualization applications, games and interactive animations projects [13]. Processing is a subset of Java directed to novice programmers and generative artists [12], which are the intended users of the EvoSpace-i framework. As a complement there

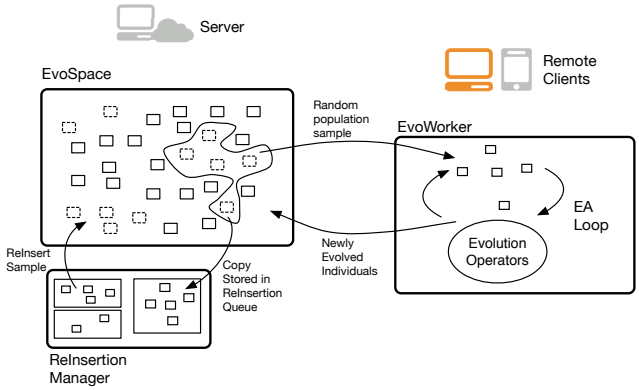

Figure 2: Main components and dataflow within EvoSpace.

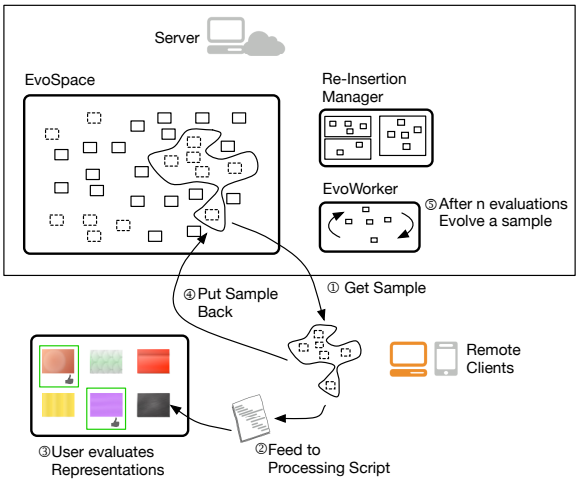

Figure 3: Evaluation process in EvoSpace-i.

is a javascript library *processingjs* that allows Processing scripts to be run by any HTML5 compatible browser. Processing scripts are responsible of rendering individuals which can involve animations, sound or even interactive artifacts. Before calling the `draw()` method of the processing script a local array of parameters are replaced with those of each individual's chromosome. Each individual's script has its own Canvas entity; Although the combination of an HTML5 Canvas element and a Processing script is supported by default, other combinations could be used. For instance, images, embedded audio, or other libraries capable of drawing in the Canvas. Also, a fallback implementation must be considered for applications intended for non-HTML5 capable browsers. To create a new C-IEC application, the rendering script must be defined, and its parameters encoded as a chromosome.

4.4 Fitness

As stated before, the assignment of the fitness for each individual takes into account the evaluation given by several users. In this version of EvoSpace-i users can only give positive evaluations explicitly when they select an individual, giving it a rating of *Like*. When a user evaluates a sample of individuals, some (or all) of them will not receive a Like, in

each case the `views` property will be incremented by 1. For instance, if an individual has a high number of views with only two likes, he is considered to be worse than an individual with two views and two likes. The ratio $Likes/Views$ is more informative, but it does not distinguish between an individual with many views and another with only one view if they both have zero likes; also views must be >=1 to avoid dividing by zero. Fitness, therefore, is given by $(Likes + 1)/(Views + 1)$. As a future work more options can be defined, for instance taking into account the number of "shares" or the times an individual has been stored in a collection.

4.5 Breeding Process

Once individuals have been evaluated by a minimum number of users, they can participate in the breeding process. Since the genetic operators of the evolutionary process will depend on the particular application, they must be implemented by the application designer. For this purpose, in the current version they are implemented in Python, allowing users to use libraries such as DEAP or PyEvolve. For example, both examples presented in this work were implemented using only the NumPy library for array operations. As this process is executed in the server, the communication with EvoSpace-Redis is done directly through a library. There are a few additional parameters that are used for the `Evolve()` method: `EVOLUTION-INTERVAL` indicates the number of samples that must returned to trigger an `Evolve()` call; the `SAMPLE-SIZE` parameter indicates how many individuals are taken from EvoSpace to participate in `Evolve()`, `MINIMUM-VIEWS` is the minimum views needed for each individual to participate in the breeding process.

5. COLLABORATION

Using their Facebook account, users can collaborate with their Facebook friends, sharing those individuals they like, or by taking individuals from the collections of friends. As an introduction to this section de general user experience is described next.

5.1 User Interface.

Users interact with the web interface depicted in Figure 4, which is composed of five elements. First, at the top left corner user login and authentication. Users can login with their Facebook account or participate as anonymous users. Second, if a user chooses to login a list of Facebook friends that have also linked their account with the C-IEA application is presented on the left, to encourage users to interact with the system. The third element is a central *Wall* area, where a population sample of n individuals is shown to the user. These are n random individuals taken from the EvoSpace server. Here, the user can interact with the system in two ways. He can click on the individuals he prefers, a clicked image is highlighted and this counts as a "like" for the individual. Additionally, a user can choose to add an image to one of their *Collections*. A collection is a special directory to store individuals a user prefers and wishes to save. After the user finishes interacting with the current crop of individuals on the Wall, he can choose to retrieve a new sample from EvoSpace. This is done with the fourth element of the interface, located at the top of the screen, the *GetMore* button. The button first returns the current group of individuals to EvoSpace, and brings back a new one. Each time a user

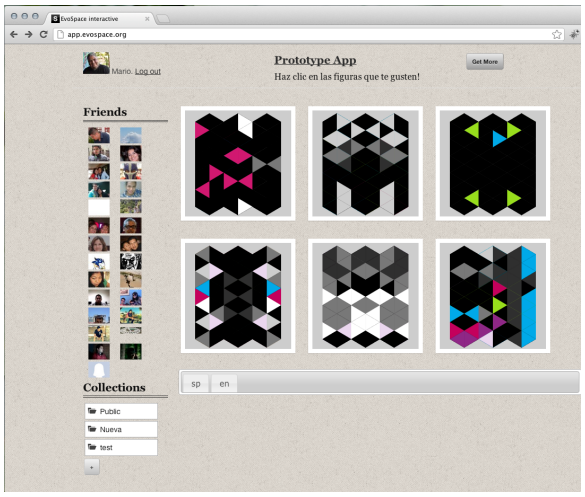

Figure 4: Current user interface of EvoSpace-i.

performs a *GetMore* click, it increments the number of samples returned, and this could trigger a server-side `Evolve()` method. The fifth element of the interface is shown at the bottom left corner, the *Collections* section. The user can create several collections, to group and organize his favorite artifacts. Moreover, a user can browse the content of each collection and from there share images through the social network. When a user browses over an individual a detail pane shows how many users have liked the individual. The pane also includes a link to the individual's details, the parents, genetic operators that created it, and genealogy information.

5.2 Facebook API and OAuth 2.0

Applications developed with the EvoSpace-i framework must be defined as Facebook Web Applications. Other social networks are going to be enabled in further versions of the framework; but initially Facebook was selected as a Social Network platform for the following reasons:

- Popularity. Facebook is currently the social network with more active users, with more than 1 billion.
- Applications. Facebook applications are also common, popular applications like Youtube, Pinterest, Netflix, XBOX Live and Spotify allow users to share their activities.

In order to enable the application's collaborative functionality, end-users must be authenticated with their Facebook account. This is done using the OAuth2.0 protocol [8]. The OAuth 2.0 login flow generates an access token, which allows the application to make API calls on behalf of a user. As part of this flow, users also give certain permissions to the application, so it can access their private data. Currently the framework uses the most basic permission, having access only to their public profile and list of friends. The public profile includes their profile picture, username, gender and locale. An important information is the list of friends, that includes which friends also have the application installed.

5.3 Django Framework Application

The Django web development framework [9], is a set of Python libraries, that provide high-level abstractions of

common web development patterns. In Django a web application consists of different python scripts following a Model-View-Controller (MVC) design pattern. Using a separation of concerns design principle, the application logic is separated mainly in four scripts:

- The `models.py` script contains a class based description of the database schema. These classes are Object-Relational mappers with methods to create, retrieve, update, and delete records in the database system using Python code instead of SQL.
- The `views.py` file contains the business logic for each web page defined as special "View" functions. These functions receive as parameters and http request data, do operations on the model (database) , and return the data through HTML generator templates.
- The `urls.py` file specifies which view is called for a given URL pattern.
- Various HTML template files that describes the design of the page.
- The `settings.py` file is where the configuration of the project is stored.

In Django, a project can be an aggregation of multiple reusable applications, each incorporating a particular functionality. Each application has its own model and views. For EvoSpace-i the database model is presented in Figure 5. There is a many to one relationship between Facebook sessions and users, this is because sessions and access tokens can expire. The database is stored in a PostgreSQL database system. Individuals are not stored in this database, they are stored in a Redis server as JSON text strings. The model for individual is presented in Figure 6

The main views for the application are:

`evospace()` This function receives JSON-RPC requests from JQuery code in the client. Main methods are `getSample()` and `putSample()`.

`home()` This view is for the main page, if the user is authenticated the list of friends and profile data is retrieved.

`individual_view()` This view returns the details of Individuals.

`facebook_login()` Initializes the OAuth 2.0 flow. The `facebook_get_login` is also related.

`dashboard()` Returns a JQuery dashboard page, for administration of EvoSpace.

Client-side scripting is used extensively by the framework. As mentioned earlier, JQuery is used to implement the evaluation of individuals as shown earlier in Figure 3, sending `getSample()` and `putSample()` requests. Other controls such as Modal Windows, Lists, Buttons are also implemented using JQuery-UI library.

5.4 Collections

An important feature of the framework, is that authenticated users can create collections where they can store individuals. These collections can be private or public. Public collections can be visited by Facebook friends and if they

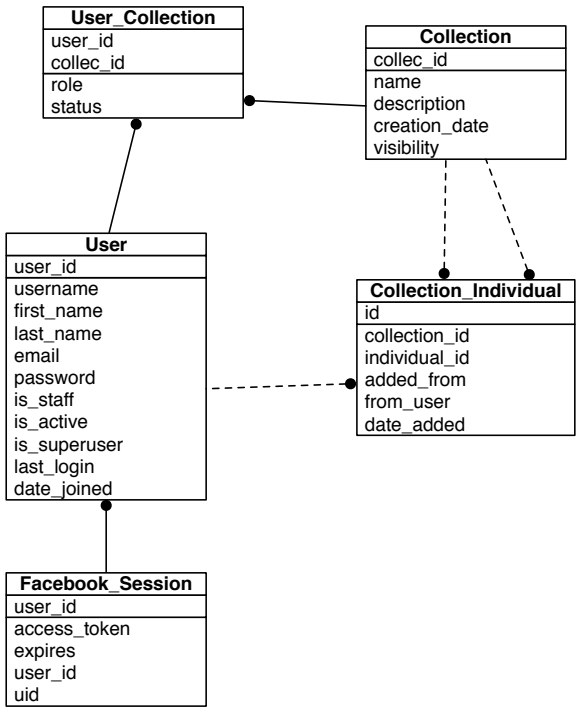

Figure 5: Django data model for the Interactive application.

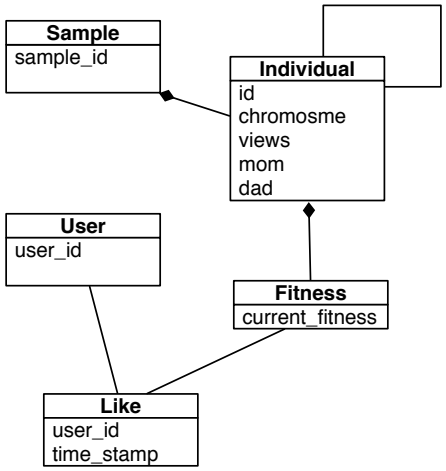

Figure 6: Data model of individuals.

wish, they can save individuals to their own collections. In future versions, users could also publish individuals to their Facebook Timeline or Photo Albums. Giving a sense of ownership (or in future versions authorship) to users, could be considered as a reward to their effort. Ã§ could also increase the voluntary participation of users. Information about how an individual is stored and shared, could be considered in the assignment of fitness. This way sharing can be part of the evolution. Collections are implemented as part of the web application, the database model is presented in Figure 5, there is a many to many relationship between users and collections, this way there can be shared collections, this is not yet implemented in the current version. A collection can

have many individuals, if the individual comes from another collection there is a relation with that collection.

6. EXAMPLE APPLICATIONS

As a proof of concept a C-EIA application was implemented with the EvoSpace-i framework, this application is detailed in [5]. The application is called *Shapes*. In *Shapes*, individuals represent a two dimensional 11 by 6 array of equilateral triangles, these arrays where inspired by Op-Art style paintings. Each triangle has a color drawn from a twelve color palette. The array is represented by a 66 element chromosome vector $\mathbf{v} = (v_1, ..v_{66})$, with $v_i \in \{1, 2, ..11\}$. The background of the painting is Light Gray, this can give the effect of a missing triangle when it has the same color. A processing script is used to render a static version of the image. The breeding process uses tournament selection of size 6 to select two individuals from EvoSpace, and generates two offspring. The offspring replace the worst individuals from both tournament groups. Crossover operators are used with crossover rate of 1, these are vertical and horizontal one-point crossovers. Several mutations are used with a mutation rate of 0.3, these are: (1) single *point mutation*; (2) vertical and horizontal *mirrors* at a random point; (3) *shuffle* that gives a new permutation of the chromosome. In nearly two weeks of operation, there wa a total of 70 active users, users who gave permissions to the *Shapes* application and haven't removed it from their Facebook account. Facebook dashboard reports that 74 percent of users accepted the permission request to use their credentials to login onto *Shapes*. Participation of anonymous users was permitted, but their number was not recorded. Basic instructions we're shown in the landing page, but part of the functionality was left to be explored by users. An auto-increment id was assigned to each individual, after two weeks the highest id number is 8379. A total of 17449 samples were taken from the EvoSpace server after the two week trial. A sample of artistic artifacts in a collection is depicted in Figure 7.

Another example application is called *Fireworks*. The goal of the Fireworks C-IEA is to evolve artistic animations of particle swarms in 3D. The animations are based on the open-source Processing script developed by Claudio Gonzales called Galactic Dust available at `http://www.openprocessing.org/sketch/8062` and licensed under Creative Commons Attribution-Share Alike 3.0 and GNU GPL license. The Galactic Dust script presents a virtual 3D canvas where a set of N particles can move about. In the original version of the script the particles are randomly positioned within the 3D canvas (or placed in an a priori pattern) and remain static. Then, when the user left clicks on a point within the canvas this specifies the position of a gravitational point P on the canvas and produces an attractive force on all of the particles towards P, proportional to their distance to P and mass which is also specified initially. Similarly, when a user right-clicks on the canvas a repulsive force is produced. Moreover, the particles continuously change color using small random steps and the user can toggle a tracing effect on or off, in which each particle leaves a visual trail of its movements that gradually disappears with time. The Galactic Dust script is the inspiration and basis for the artistic animations evolved by the Fireworks application. In particular, the goal of Fireworks is to begin with randomly placed particles and to evolve a pattern of movements and behaviors, searching for interesting

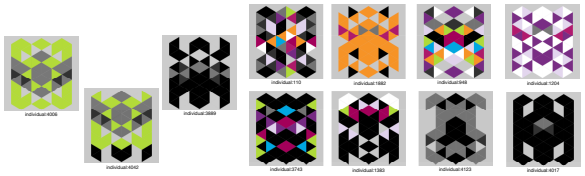

Figure 7: Artistic artifacts in a collection

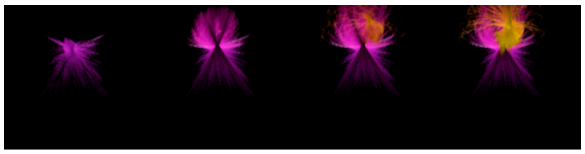

Figure 8: Sample frames of of an evolved animation.

visual animations. These animations are similar to popular screen savers or background visualizations for music players, however the authors feel that an illustrative description is to say that they resemble elaborate fireworks displays, thus the name.

Fireworks is based on a linear genetic programming algorithm, where the chromosome of each individual is encoded as a variable-length list. Each element, or gene, of the chromosome represents one of five basic operations, these are: $Attract_{flag}$, $Move_{flag}$, $Trace_{flag}$, $ForwardX_{flag}$, $ForwardY_{flag}$ and $Step_m$. The first four instructions are toggle operations for different script parameters that determine the characteristics of particle movements. $Attract_{flag}$ instructs the script to toggle between attraction or repulsion towards P. $Move_{flag}$ toggles between particle movement or particle deceleration at each frame of the animation. $Trace_{flag}$ determines if particle movement will produce tracing or if it will not. $ForwardX_{flag}$ and $ForwardY_{flag}$ flags determines the orientation of particle movements in the horizontal and vertical axis respectively. Finally, the $Step_m$ function determines the magnitude of step movements in each direction given by m pixels, with a random decision uniformly chooses from $u \in [1, 13]$.

Currently, the Fireworks applications has been evolving individuals for over three weeks, progressively generating novel animations. In general, it is clear that the genetic representation and evaluation criteria has allowed the algorithm to progressively incorporate the preferences of several users. There is no time-table to halt the open-ended evolutionary process, and the system is expected to run for at least one more month. It is now possible to access the application and interact with it at `http://app.evospace.org/`. It is important to point out that the Processing scripts evolved in this work are computationally costly, and push browser resources to the limit. Only the Chrome web browser offers an acceptable frame rate, using 200 swarm particles within each animation with canvas elements of 200 by 200 pixels in size.Figure 8 shows a series of frames from a singled evolved animation.

7. CONCLUSIONS

EvoArt is growing and expanding as a sub-field of EC, with dedicated workshops and tracks in all major EC conferences. It differentiates itself from other areas by incor-

porating truly varied perspectives, from subjective artists and designers and objective computer scientists. Moreover, EvoArt is more than just an intriguing application area of EC, it might be able to offer insights regarding how best to aid, model, and perhaps reproduce, probably one of the most studied and least understood human abilities: artistic creativity. Through EvoArt systems, it might become possible to develop systems where mental mems are distributed and reproduce in an artificial substrate.

This paper presents EvoSpace-i, the first computational tool specifically aimed at the EvoArt community, particularly those interested in C-IEAs. EvoSpace-i offers several general features that are desirable in an EvoArt system; such as: an interactive approach towards evaluation; a distributed system that exploits the web and can be integrates with current Cloud-based approaches towards computing; a collaborative system where evolution is guided by the preferences of all users, and where collaboration is encouraged by integrating social networking; finally, EvoSpace-i renders artistic artifacts using the Processing programming language, that reduces the steepness of the learning curve for non-programmers, since the tool is meant to promote and establish collaborative work between artists, designers and computer scientists.

Initial results are encouraging, the EvoSpace-i framework was successfully used to deploy two C-IEAs that users accepted and used to design artistic artifacts. However, the true possibilities have not been pushed to the limit by the Shapes or Fireowrks applications described here, one of the main areas of future work and research. Nonetheless, the paper presents the first attempt to build a tool specifically for the development of C-IEAs for evolutionary art.

8. ACKNOWLEDGMENTS

This work is supported by ***

9. REFERENCES

[1] P. Bentley. An introduction to evolutionary design by computers. In P. J. Bentley, editor, *Evolutionary Design by Computers*, chapter 1, pages 1–73. Morgan Kaufman, San Francisco, USA, 1999.

[2] J. Clune and H. Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life*, pages 144–148, 2011.

[3] R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, UK, 1976.

[4] R. Dawkins. *Climbing Mount Improbable*. W.W. Norton & Company, 1996.

[5] M. Garcia-Valdez, L. Trujillo, F. Fernández de Vega, J. Merelo Guervós, and G. Olague. EvoSpace-Interactive: A Framework to Develop Distributed Collaborative-Interactive Evolutionary Algorithms for Artistic Design. In P. M., M. J., and C. A., editors, *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, volume 7834 of *Lecture Notes in Computer Science*, pages 121–130. Springer Berlin Heidelberg, 2013.

[6] M. García-Valdez, L. Trujillo, F. Fernández de Vega, J. J. Merelo Guervós, and G. Olague. EvoSpace: A Distributed Evolutionary Platform Based on the Tuple Space Model. In A. Esparcia-Alcázar, editor, *Applications of Evolutionary Computation*, volume 7835 of *Lecture Notes in Computer Science*, pages 499–508. Springer Berlin Heidelberg, 2013.

[7] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, Jan. 1985.

[8] E. Hammer-Lahav, D. Recordon, and D. Hardt. The oauth 2.0 authorization protocol. *draft-ietf-oauth-v2-18*, 8, 2011.

[9] A. Holovaty and J. Kaplan-Moss. Introduction to django. In *The Definitive Guide to Django*, pages 3–9. Apress, 2008.

[10] T. Kowaliw, A. Dorin, and J. McCormack. Promoting creative design in interactive evolutionary computation. *Evolutionary Computation, IEEE Transactions on*, 16(4):523 –536, 2012.

[11] W. B. Langdon. Global distributed evolution of l-systems fractals. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming, Proceedings of EuroGP'2004*, volume 3003 of *LNCS*, pages 349–358. Springer-Verlag, 5-7 April 2004.

[12] M. Pearson. *Generative Art*. Manning Publications, pap/psc edition, July 2011.

[13] C. Reas and B. Fry. *A programming handbook for visual designers and artists*. The MIT Press, 2007.

[14] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evol. Comput.*, 19(3):373–403, Sept. 2011.

[15] Y. Semet. Evolutionary computation: a survey of existing theory. Technical report, University of Illinois, 2002.

[16] H. Takagi. Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation. *Proceedings of IEEE*, 89(9):1275–1296, 2001.