EvoSpace-Interactive: A framework for Interactive Evolutionary Algorithms

author's names removed

ABSTRACT

This work presents EvoSpace-Interactive, 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 ii) Interactive, a Django framework application where end-users collaborate in a social network sharing, collecting, rating and ultimately evolving individuals. Individuals are presented as multimedia elements (images, animations, sound) using the Processing programming language. Details of the design are presented and two example applications are shown 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

1. INTRODUCTION

2. RELATED WORK

3. ARCHITECTURE

The goal of this work is to develop an open source framework for Web and Cloud-based C-IEA systems, using current web standards and libraries for mobile devices. Developers of C-IEA applications are liberated from the need of designing and programming 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; these are: 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; 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-Interactive are shown in Figure 1. The Interactive part is a Django [] 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; thous giving IEC 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

The EvoSpace model is presented in detail in [1], only brief description of the functionality related to this work is given next. EvoSpace is inspired on the Linda language by Gelernter and Carriero [2]. 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. EvoSpace consists of two main components (see figure 2): (i) the EvoSpace container that stores the evolving population and (ii) remote clients called EvoWorkers, which execute the actual evolutionary process, while EvoSpace acts only as a population repository. In a basic configuration, EvoWorkers take a random sample of the population, and use it as the initial population for a local EA executed on the client machine. When taken by an EvoWorker, individuals remain in a phantom state until their sample is returned. If an

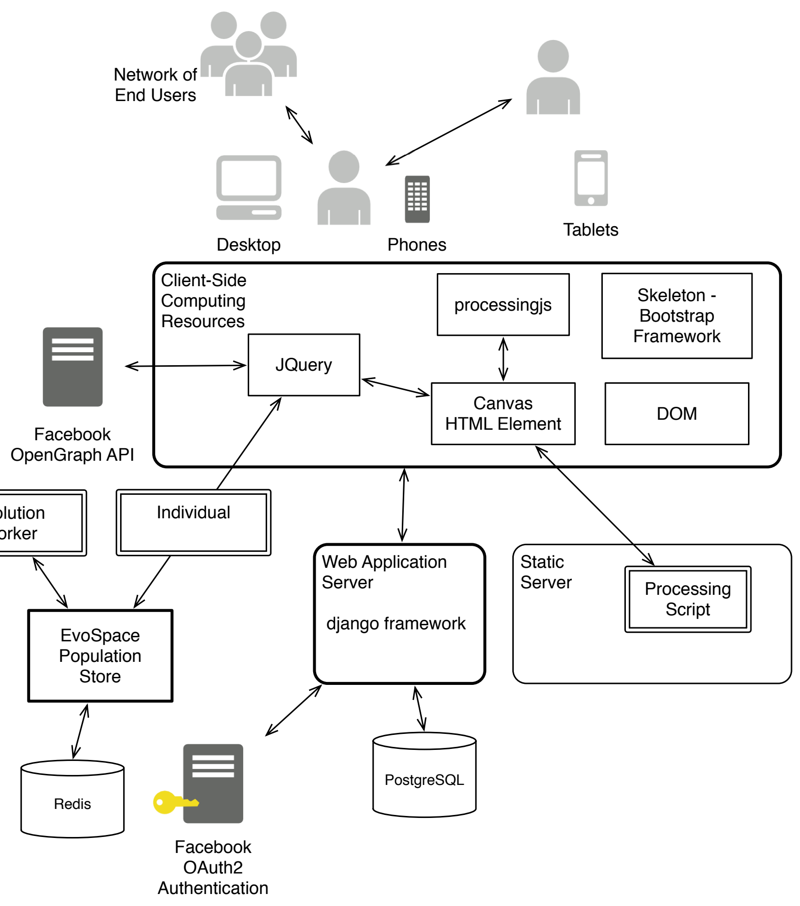


Figure 1: Main components of EvoSpace-Interactive.

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. Figure 2 illustrates the main components and dataflow within EvoSpace.

For this version of EvoSpace-Interactive, a new type of Worker is needed: Human users. Users are responsible for subjectively evaluating 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 EvoWorker 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 users, these are returned with their identity unchanged, other than the rating added by the current user. The internal representation of individuals is presented next.

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-Interactive 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 more than one time each individual. 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

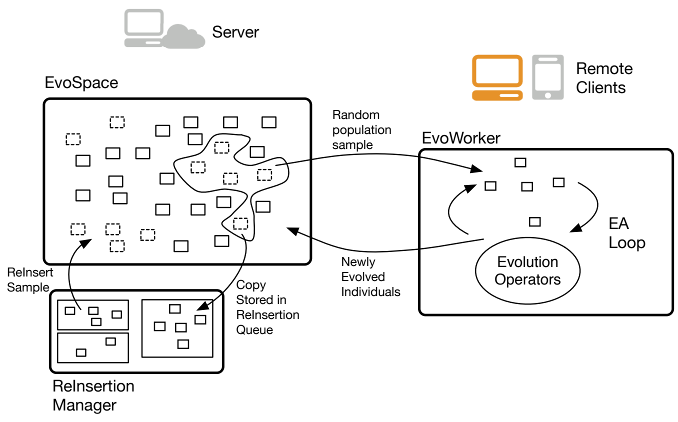
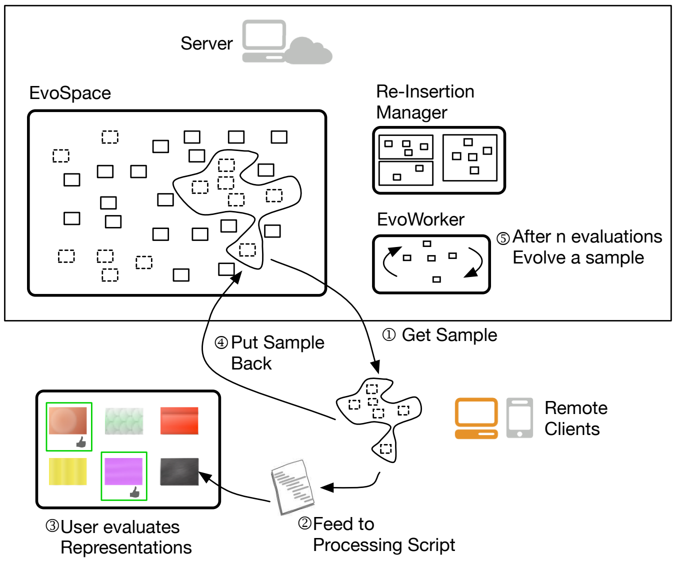


Figure 2: Main components and dataflow within EvoSpace.



Server
EvoSpace
Re-Insertion Manager
EvoWorker
⑤After n evaluations Evolve a sample
① Get Sample
④Put Sample Back
Remote Clients
③User evaluates Representations
②Feed to Processing Script

Figure 3:

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. The EvoSpace web service interacts is called directly by a JQuery script running in the client-side. 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.

5. INTERACTIVE EVOLUTION

In this section, the components needed to enable the interaction between users and individuals are presented. The first subsection the rendering and representation of individuals is discussed and next how fitness is assigned.

5.1 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 [5]. Processing is a subset of Java directed to novice programmers and generative artists [4], which are the intended users of the EvoSpace-Interactive framework. As a complement there 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; defined by the HTML5 standard, as an element that provides scripts with a resolution-dependent bitmap canvas which can be used for rendering graphics on the fly. 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.

5.2 Fitness

As stated before, the assignment of the fitness for each individual takes into account the evaluation given by several users. Initially in EvoSpace-Interactive 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 with only two likes, he 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, it is proposed to be 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.

5.3 Breeding Process

Once individuals have been evaluated by a minimum number of users, these can participate in the breeding process. As the genetic operators of the evolution process, will depend of the particular application; this algorithm must be implemented in python. Python libraries as DEAP or PyEvolve can be used for this purpose. 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 directly through a library. Only clients connect to the Web Service. There are a few additional parameters that are used for the `Evolve()` method: `EVOLUTION-INTERVAL` indicates the number of that must returned to trigger an `Evolve()`

call; the `SAMPLE-SIZE` parameter indicates how many individuals are taken from EvoSpace to participate in the Evolution, `MINIMUM-VIEWS` is the minimum views needed for each individual to participate in the breeding process.

6. COLLABORATION

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

6.1 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 returns the current group of individuals to EvoSpace, and brings back a new one. Each time a user 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.

6.2 Facebook API and OAuth2.0

Applications developed with the EvoSpace-Interactive 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.

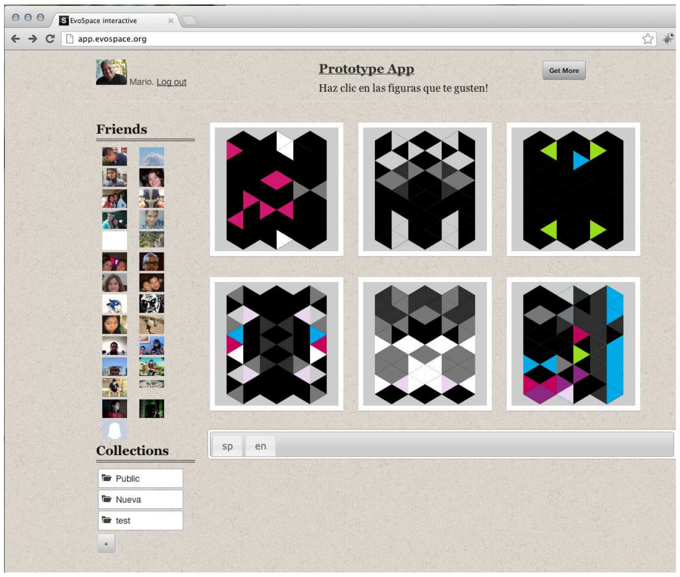


Figure 4:

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 [3].

6.3 Collections

7. EXAMPLE APPLICATIONS

8. EVALUATION

9. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

10. ACKNOWLEDGMENTS

This work is supported by projects 4616.12-P and 4617.12-P awarded by DEGEST-ProIFOPEP (Mexico), TIN2011-28627-C04-03 and -02 (ANYSELF), awarded by the Spanish Ministry of Science and Innovation, P08-TIC-03903 (EvOrq) awarded by the Andalusian Regional Government, project 83 (CANUBE) awarded by the CEI-BioTIC UGR (`http://biotic.ugr.es`) and CONACYT (Mexico) Basic Science Research Project No. 178323 and DGEST (Mexico) Research Project No. TIJ-ING-2012-110.

11. REFERENCES

[1] 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.

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

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

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

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