Constructive interactions: Abstract infrastructure for experiments in architecture and urbanism

Álvaro Castro Castilla

Universidad Politécnica de Madrid, School of Architecture (ETSAM)
C/Reina Victoria 15, 9ºB izq.
28003 Madrid (Spain)
alvaro.castro.castilla@gmail.com

Abstract

This paper outlines the on-progress work called *Interacciones constructivas* (*constructive interactions*) at medialabMadrid. It faces one of the most relevant concepts in the study of artificial intelligence and the mathematics of complexity: interactions.

Contemporary architecture and urbanism find theirselves strongly linked to the idea of interactions as generators of shape and space, though traditionally only the inverse relationship is more accepted.

The intention is to build a multidisciplinary model to make the design of such a concept possible, under the general case of abstract agents or particles interacting in an abstract space through a server/client computational model. Design is then emergent.

1. Introduction

Due to the purpose of the project, which tries to reach an open workflow among collaborators, the need to build a framework or infrastructure both general and specific, applicable to shape/space construction, seems fundamental. Multiagent design toolkits certainly solve the problem of interaction in most cases. However, they are generally oriented to experiment simulation. Consequently they don't fit our intentions in three aspects:

• Visualization is conceptualized as an instrument

• The separation between the design of the space and the particles is not conceptualized from the ground

• Concepts are thought for scientists, not artists

The second of the differences refers mainly the need to define at which point the independent clients (where the particles and their behavior are programmed) should be incorporated in a common ground for interaction (the space). Among the endless possibilities of this separation, we may introduce the multidisciplinary approach with ease. Interaction, then, is modeled by each client with minimal limitations, so the different conceptions can be joined and glued together in some way (defined partially in the abstract space and partially by the particles). As a trade-off for this openness, the result could be inevitably incoherent. The project *Interacciones constructivas*, investigates the multiple situations and architectural emergence that can arise at this experimental ground. The calibration of that incoherence is its primary objective.

The chosen option, then, was to redesign the framework needed for this simulation, instead of adapting the common toolkits, such as *Swarm* [1] or *Repast* [2]. We get, by this way, a more usable, simple and adaptable method to achieve the goal of designing architecture/urbanism; shape/space through the recreation of interaction at multiple levels: human, social, economical, aesthetical, physical...

2. Model

The model is made to suit our needs of constrained generality. Leaving the implementation to the next level of definition, it allows us to design “interaction” abstractly so we can, step by step, mold it to suit the experiment's needs.

2.1. Abstract dynamic space

The space-time variables, plus the attributes of an encapsulated object can be unified in a high dimensional space, built out of subspaces.

We define, for this model, a flexible space *S* which can virtually made out of an infinite stack of spaces with their own inner properties:

$$S^{\dots+i+\dots+j+\dots+k+\dots} = \dots * \mathbb{N}^i * \dots * \mathbb{R}^j * \dots * \mathbb{C}^k * \dots$$

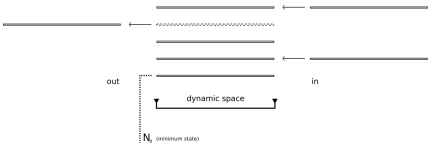
The inner properties of this space should be mathematically analyzed, but we consider that is not the objective of this paper, as the idea of the model is just the creation of an abstract space able to assimilate any other.

IMPORTANT: This document is outdated. It works on a pre-release basis.

Continuing with our pseudo-mathematical language, a typical *3-dimensional* space for computable simulations would simply be R^3, that together with *computational time* -continuous (1)-, or *computational steps* -discrete (2)-:

$$_{(1)}\, S = \mathbb{R}^3 * \mathbb{R}^+ \qquad _{(2)}\, S = \mathbb{R}^3 * \mathbb{N}$$

Nevertheless, the creation of the space relies, only, in the dynamic insertion of the particles. Every time we want to know how our space is, we have to check out which particles we have, and then, we have a new space; that is the reason we call it *dynamic space*.



To make this computationally possible, the most fundamental (minimum) state of this space must be:

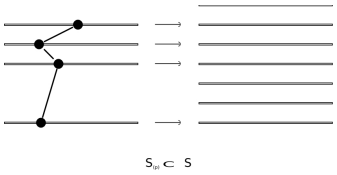
$$\forall S, \exists \mathbb{N}_0 \subseteq S$$

Corresponding to a a discrete natural numbers space which is usually identified with the computer steps, function evaluations or algorithm cycles run, so the state progression of the S space depends on a subset of itself that is the set N_0.

2.1. Abstract particles

As mentioned before, these entities are responsible for the state of the abstract dynamic space. They are conceived in a way that the limitation of the freedom of design by independent artists-researchers is minimized. The objective is that in one space, very different particles with very different degrees of interaction co-habit.

The particles *P* are mathematical subspaces of *S*. While ranging from dim=0 to dim=*n* (where *n* is the dynamic dimension of *S*) the value shows the number of dimensions that the particle doesn't define. Means, mathematically, that could take every value in that dimension, but to our model it reflects that the particle is not “aware” of the existence of it.

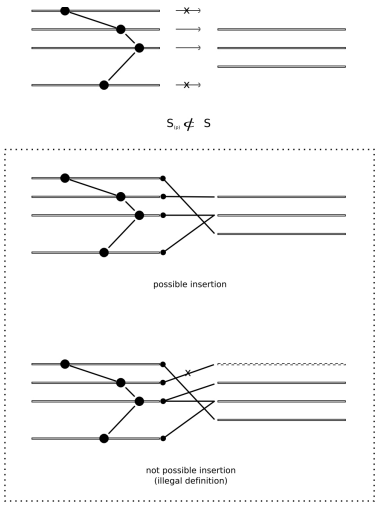


The problem is that this supposes that:

$$S_{(p)} \subset S$$

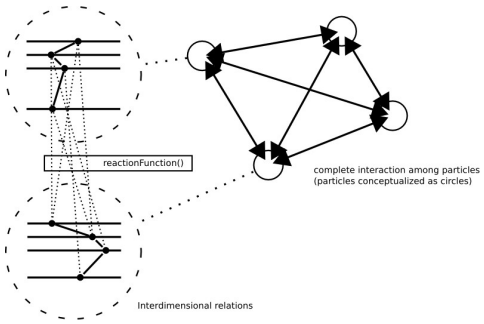
(where $S_{(p)}$ is the dynamic space claimed by the particle), which can only be true, when it is not a subset of *S*, if the dynamic *S* is readjusted. For the dynamically inserted particles to “fit” in the space, they only need to fulfill one condition: their dimensions cannot be defined from an undefined dimension (although we can define them together with the particle). While the opposite is perfectly acceptable, this situation would make it mathematically inconsistent. Following this requirement, the *n* can take a new value generating a new space *S'*.

$$S' = S_{(p)} \cup S$$



2.3. Abstract interactions

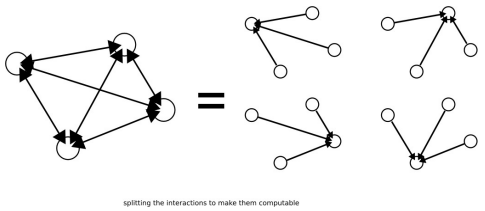
We will assume the most general case that interaction meets both action and reaction from all the particles in a set towards all the particles in the same set.



Interaction is defined as a joined combination of action and reaction, both of them functions, that can exist for the same values of *N,* consecutive values of *N* (would be the typical action-reaction behavior in time), or build a dependence function on the N_0 subset (which could be something like varying the particle's behavior through time). At the same time, the reaction function can be dependent on the action function. However, all this considerations are opened to the designer of the interactive particles, and our only purpose is to build the basic model to allow it.

Splitting away action and reaction, considering both of them functions (which can depend on *N*) reduces

everything to one assembled function for everyone of the particles at every point of N (that is, typically, the same general case of implementing a behavior function to all the particle objects which depend on the computational time and all the attributes of all the particles).



2.4. Emergent architectural shape/space

Consider S and S' two different states of the dynamic space and N_0 the fundamental subset. Then the emergent behaviors of the model will arise in the transition between S and S' if we simultaneously change the particles' value in the N_0 dimension.

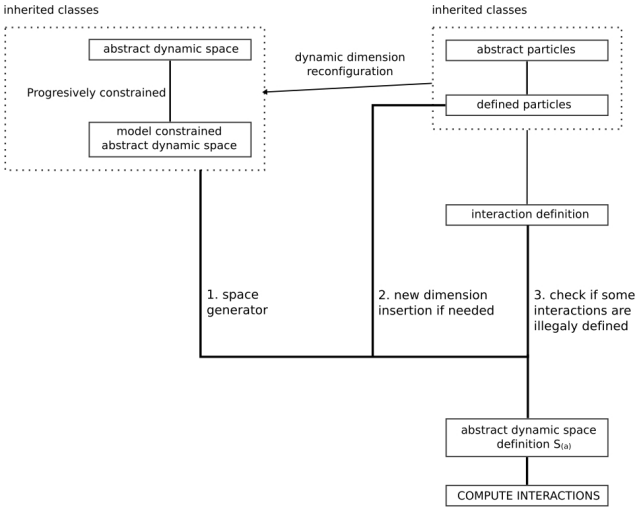
3. Algorithm implementation

Object-oriented languages are a perfect tool to build this model, mainly, for the following reasons:

- *Concept encapsulation:* the particle designers don't have to be aware of the space implementation
- *Inheritance:* this will allow us to make the model progressively concrete, so we can start the project from a scratch following the design pattern and end up coding the most particular cases and features of the project

3.1. Implementing space and particles

The design pattern will be built according to the following structure



3.2. Implementing interaction

The interaction takes the shape of a class that runs the ***step()*** function which executes the algorithm of a complete computational step. It processes all needed combinations and functions. The interaction model previously explained, should take this general procedure, which will evaluate all the possible action-reactions for all the particles towards all the particles:

// C/C++: All the variables are globally and dynamically created. The algorithm iterates through all the particles in the abstract dynamic space a*ds*:

```
void step()
{
   for (int i = 0; i =< ads.getsize(), i++)
      {
// every particle reacts to the all the other particles, and its reactions are saved in a temporary mirror space tms:
         tms[i] = applyReaction( i, chooseFunction(i) );
      }
// now the ads is overwritten with the completed tms, and tms is cleaned:
   ads = tms;
   tms = null;
}
```

The problem of this algorithm is that it runs extremely inefficiently, since the number of combinations quickly grows exponentially. The only solution for this is to model a $f_i(..., x, y,...)$ function that reduces the evaluations. Examples of this reduction are the common Von Neumann and Moore neighborhoods for cellular automata -which limit the affection range-, although we would need a more general approach. In any case, this relies only in the designer of the inserted particles, since they are the ones in which the interaction function acts upon (which belongs to the client part, as we will see).

3.3. Implementing emergent shape/space

The functions ***prestep()*** and ***poststep()*** are concepts borrowed from the Repast agent simulation toolkit, introduced here as the programmed events responsible of the "conceptual switching" done when visualizing data or transforming the abstract world until now into the "physical" level of architecture.

Introducing this two functions, the typical program structure to build our interactions model will be: prestep(), step(), poststep().

The code inside this functions affects inevitably all the interactions and particles designed by all the collaborators in the project. All the preprocessing done in the space by the *prestep()* will condition the interactions step, while the *poststep()* will likely be more adaptable for any kind of decision taken at the emergent level when the interactions are finished, such as graphics, data and visualization generation. However, although the translation into

perceptible space is done inside this function, the data flow could be send to an independent process dedicated exclusively to 3d visualization or any kind of interface working in its own.

4. Client/server approach

The last part of this computational model for architectural interactions is the response to one problem that arises from the original project idea: the openness needed for the additive inclusion of new components, both particles and interaction functions, would lead to a saturation of computation resources. Distributing the calculations with a server/client model lets us equilibrate the virtual space growth and the physical resources, as the addition of new particles and a new hardware client is done in simultaneity.

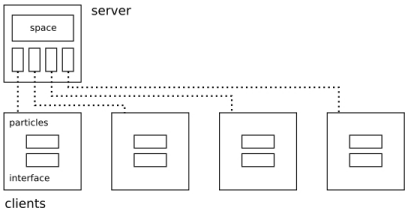
To make this dynamic generation and evaluation of spaces, particles and interactions possible in this new paradigm, we have to redesign the interaction algorithm and make possible a kind of communication among particles in different mediums and with the common dynamic space.

This is achieved via message packets sent through TCP/IP. Some languages are designed with that intention, like ACL, KIF, KQML and XML [3]. The last one is, nowadays, the de facto standard for applications communication. If the messages language is common to all clients and the server, it seems perfectly possible to build an extremely hybrid network, where the only rules to follow are the model's and the message formatting. The expected result is as follows:

• A central server (could, itself, be distributed), to be built with C/C++ under a GNU/Linux platform

• The clients network, developed under any programming language and platform

• The messages sent through the network XML formatted



Now, the interactions algorithm will be inserted in a conversation between the server and the clients. We will need to define a connection setup stage and a stable communication stage. Obviously, this is a simplification that will need to face the real problems of TCP/IP networking. Nevertheless, it is outlined as follows:

4.1. Communication setup stage

• Client:

1) I need ...x,...,z...,r... dimensions to work, do you have them?

2) I'm also going to insert ...a,...,b... dimensions in your dynamic space

• Server:

1) ok, I have that dimensions you need // oops, I don't have them, you will have to keep trying

2) Adds the new dimension to its dynamic space

3) Start managing the connection/identification of the client

4) Keeps in mind which dimensions are being used by the client, to avoid sending unused data over the network

4.2. Stable communication stage

• Server:

1) Executes prestep()

2) Sends particles in the list of requested particles to every client (would be good to compress this data).

• Client:

1) Reads users/interface input

2) Computes interactions with user input and received data, the step() function

• Server:

1) Receives the particles for the next step and waits until all the clients send their particles. Then the temporary mirror space is complete and overwrites the dynamic space.

2) Executes poststep()

3) Restart cycle

5. Current development and future work

The project is being developed at medialabMadrid, but was originated at GEP (A*rchitectural Exploration Group*, School of Architecture in Madrid). A workshop at the medialab will be essential for the development. Currently, active collaborators are to start the creation of external clients. The server and XML implementation is in alpha stage. The whole project pretends to take no more than one month of intense work at this point.

References

1. http://www.swarm.org
2. http://repast.sourceforge.net
3. http://www.s3.org/TR/REC-xml