

Population Symbiotic Evolution in a Model of Industrial Districts

Ugo Merlone

Dipartimento di Statistica e Matematica Applicata “de Castro”, University of Turin, Italy

merlone@econ.unito.it

Michele Sonnessa

Dipartimento di Informatica, University of Turin, Italy

sonnessa@di.unito.it

Pietro Terna

Dipartimento di Scienze economiche e finanziarie “G. Prato”, University of Turin, Italy

pietro.terna@unito.it

Abstract

This paper considers a model of industrial districts where different populations interact symbiotically. The approach consists in the parallel implementation of the model with jESOF, JAS and plain C++. We consider a district decomposition where two populations, workers and firms, cooperate while behaving independently. We can find interesting effects both in terms of worker localization consequences and of the dynamic complexity of the model, with policy resistance aspects. Those effects are stressed by comparing the dynamics of both the populations when they are independent one from each other and when they co-evolve.

By using a multiple implementation strategy we compare the advantages of three modeling techniques and highlight the benefits arising when the same model is implemented on radically different simulation environments, with different implementation approaches; furthermore we discuss some results and future model improvements.

1. INTRODUCTION

In this paper we consider a model of industrial districts where different populations interact symbiotically. According to Becattini (2003) the industrial district “[...] first and fundamental decomposition is to be the one between the productive apparatus and the human community in which it is, so to speak, «embedded»”. In our approach the two populations we consider allow for this decomposition: the first one represents the productive apparatus, while the second one can be considered the human community. Several aspects about industrial districts have been examined in the literature, for an introduction the reader can refer to Garofoli (1981, 1991, 1992), Becattini et al. (1992), or Belussi and Gottardi (2000). Carbonara (2005) analyzes some common key features in

the literature on geographical clusters and in particular on industrial districts¹. She identifies, among the others, both "...a dense network of inter-firm relationships, in which the firms cooperate and compete at the same time..." and "...a dense network of social relationships, based mainly on face to face contact, which is strictly inter-connected with the system of economic relationships...". The first aspect looks extremely interesting and is one of the aspects we consider in our approach. In fact, starting from the definition given by Squazzoni and Boero (2002) where "industrial districts can be conceived as complex systems characterized by a network of interactions amongst heterogeneous, localized, functionally integrated and complementary firms", we introduce and model the role of workers interacting with firms.

The model of industrial district we obtain consists of two populations having different peculiarities and interacting in the same environment. In the literature, the representation of districts as communities of populations is not a new idea (see for instance Lazzeretti & Storai, 1999 and 2003), nevertheless to the best of our knowledge studies devoted to the dynamical evolution of these populations are still limited.

Ecological models of population dynamics for different species can be found both in mathematical ecology and in computer science literature. Ecology of populations examines the dynamics of a number of organisms. In this field the use of mathematical models is quite common in explaining the growth and behavior of population; for a first introduction the reader may refer to Hastings (1997). The most famous model is probably the well known Lotka-Volterra prey predator model (Lotka, 1925; Volterra, 1926); in this model, which is the simplest prey predator system, two species coexist with one preying on the other (for a concise mathematical discussion of the model the reader may refer to Hofbauer and Sigmund, 1998). For more recent contributions about mathematical models of population the reader may refer to Royama (1992).

In the model of industrial districts we consider, cooperation is in some sense more implicit, since the structure of the model assumes that workers and firms cooperate. In fact, each of the two species (namely, the workers and the firms), is necessary to the other. In this sense our model exhibits a sort of necessary symbiotic evolution of the two species. In order to emphasize this requisite we analyze the dynamics and eventually the equilibria² of both the populations, trying to isolate the effects each population produce upon the other. By fixing one population behavior we collect results to be compared with the outcomes generated by co-evolving scenario. Such reference data can help in understanding complex phenomena behind the model as well as the impact of relationships on its dynamics.

Starting from this approach we consider the dynamics of the populations of firms and workers and their evolution. In particular, we are interested in shedding light on the emergence of industrial districts when the mentioned decomposition is considered, showing that this simple interaction is sufficient for firms to form clusters. While this cannot be an exhaustive explanation of districts' behavior, it is an interesting insight.

Since the system is highly complex, a valid mathematical model of it is itself complex, precluding any possibility of an analytical solution. As it is common in these cases, the model must be studied by means of simulation; for further details on the role of simulation the reader may refer to Law & Kelton (2000). The simulation approach for the analysis of districts is not new. For example, Zhang (2003) uses agent based simulation to study the dynamics of high-tech industrial clusters, Squazzoni & Boero (2002) use computational techniques to focus on some evolutionary

¹ Carbonara (2004) points out how Italian geographical clusters are usually referred as Industrial Districts.

² In particular for firms, the equilibrium is expected to be similar to an economic cycle.

fundamentals of industrial districts modeling, and Brenner (2002) uses simulation to study some mechanisms of the evolution of industrial clusters.

While in the simulation of real systems several techniques are used in order to increase the model validity and credibility, not all of them can be used when implementing a theoretical model such as in our case. Software developers are well aware that computer programming is an intrinsically error-prone process, for example Becker (2005) claims "... As applications become more complex, their resource management requirements become more complex, and despite our best efforts, our designs often have holes in them, or we apply our designs incorrectly, or we make coding errors...". This is well known in the simulation literature, for example in Law & Kelton (2000) several techniques are suggested for the model verification, i.e., "...determining whether the conceptual simulation model (model assumptions) has been correctly translated into a computer program...". Nevertheless only recently the agent based modeling literature seems to be aware of the potential pitfalls, see for example Polhill et al. (2005). For these reasons we decided to use an approach similar to the one of Cerruti et al. (2005), i.e. to consider and compare different model implementations. To obtain completely identical implementations, however, it is not straightforward given the complexity of the considered model. Nevertheless, we found that the whole process of comparing and discussing the different implementations is extremely beneficial. In this sense while results replication is to be considered preliminary – at the moment we cannot obtain exactly the same results with the three implementations – yet the process of discussing and comparing different implementations details and results seems to be extremely beneficial and promising in terms of model verification. Finally, this approach looks promising in dealing with some important issues as those mentioned in Polhill et al. (2005).

2. THE MODEL

The model we consider simulates a small industrial district consisting of two interacting populations and four different components:

- the orders or productive tasks to be completed. The tasks come from the external world and, when completed, disappear from the model scope;
- the skills or abilities that are necessary to perform the different production phases of each order;
- the firms receiving the orders either from the market or from other firms and processing them;
- the workforce that, when hired by firms, allows them to process the orders.

In each time period new workers and firms are, according to some parameters, generated and randomly located.

Here we describe each of the four components and discuss their mutual relationships.

Each order contains a recipe, i.e. the description of the sequence of activities to be done by the firms in order to complete a specific product. The different activities or phases to be done belong to the skill set S but are not necessarily of the same kind, in this sense skills can be considered as technical abilities. This assumption is motivated by different studies of skill dynamics in manufacturing; for an empirical study on the Italian manufacturing firms the reader may refer to Piva et al. (2003). To explain how skills are modeled consider a district with three skills 0,1,2. A feasible example of order is '00120': this is a five-phase order where the first two phases need skill 0, the third needs skill 1, the fourth skill 2, and the last one skill 0. Each firm is specialized in a single skill and the same holds for workers. Obviously other approaches in modeling production processes using recipes are possible; for an example the modeling of production operations as recipes is adopted by Auerswald et al. (1998) or, for a complexity and knowledge based approach, the reader may refer to

Sorenson et al. (2004).

In the current version of the model specialization for firms and workers is randomly attributed and does not change. Firms can only process order phases denoted with their skill specialization and workers are preferably hired by firms with the same skill. The orders consisting of non-homogeneous phases need to be sent to different firms to be completed. The mechanisms workers are hired with and firms pass each other the orders rely on the social structure of the district: the environment is a (social) space with (metaphorical) distances representing trustiness and cooperation among production units (the social capital). While firms have a social visibility that increases according to their longevity, workers' visibility is fixed. Only mutually visible agents can cooperate, i.e., firms may hire only workers that are in their social network and orders can be passed between mutually visible firms. This aspect refers to the other key feature which is mentioned by Carbonara (2004), i.e., the network of social relationships. In our model, to keep the analysis simple, we do model the network in terms of distance. With these assumptions an important aspect of the model arises: cooperation is not optional; rather it is necessary for agents to survive.

At each turn of the simulation both firms and workers bear some costs, both hired workers and producing firms receive some revenue. In the current version net profits and wages are modeled simply assuming marginal costs to be not greater than marginal revenues for firms and a positive salary for workers. As a consequence, if for prolonged time either a worker is not hired or a firm has no worker, their balance may become negative. Workers and firms with negative balance disappear. Negative balance is the only cause of default for workers.

By contrast, other reasons for a firm default are either prolonged inactivity or the impossibility of sending concluded orders to other firms to perform the successive steps. While the interpretation of the first two default causes for firms is straightforward, the third one requires some explanation. First, we assume that a totally completed order is absorbed by the market; the rationale for this is that since the market generated this order there is demand for it. Second, recalling that orders may consist of different phases, when a partially completed order cannot be sent to a firm with the needed skill, this means either that such a firm at the moment does not exist or that it is out of scope. The latter may be interpreted as a lack of knowledge and trust, i.e., gaps in the social capital. It is worth to remark that all these aspects are consistent to the mentioned definition of industrial districts given by Squazzoni & Boero (2002). In all these cases the permanence of the agent on the market is economically unfeasible; for these reasons we summarize these situations with the broad term default.

All the firms and workers are located and operate on a two superimposed toroidal grids: one for the workers and one for the firms³.

From the populations perspective we can identify some specific distinctiveness. The workforce population is heterogeneous in terms of skills each individual is specialized in. Incomes of workers depend on how much the skill they offer matches with the firms' demand. Population dynamics is regulated by two coefficients, conveniently chosen: the birth frequency k of new workers and the probability r that a worker is hired by a firm. This factor is indirectly correlated with the death rate of workers. In fact, a worker is supposed to come out of the model in case her income balance is negative.

The dynamics of the firms population depends on many factors. We are interested in identifying which parameter combination can reproduce an equilibrium, without considering the workforce

³ In the jES Open Foundation version of the model we have also instrumental layers showing separately the presence of workers for each type of skill, but, obviously, the two implementations are equivalent in terms of modeling.

contribution. We take into account that firms survival is pledged by the possibility of trading production phases with other skill-complementary firms as well as by an adequate flow of orders in the system. The output we expect is an equilibrium shaped as an economic cycle, without explosions or district failures.

The separation of two population dynamics is obtained removing the element joining them: the workforce hiring process.

The Figures 1 and 2 well show the different processes when the hiring is enabled or not.

Trying to summarize the enter/exit process we can identify one cause of death for workers:

- when a worker is not employed for a long time;
- and two causes for firms:
- when a firm does not receive orders for some time;
- when it has to short social visibility so that cannot deliver orders to other firms.

Joining the two populations into a coevolving model introduces another cause of death for firms:

- when the production stock, generated by hired workers, is negative, the firm cannot complete the production step.

The novelty of this model structure is the introduction of the interaction “within” a model layer (the one containing the productive structures) while the classical Lotka-Volterra structure exploits only the consequences of the interaction “between” two different layers.

Firms, to produce, have to interact with other firms, with the constraint of considering mutually visible only the units sharing a portion of their visibility space, as a synthetic way for representing trustiness. The “within” interaction influences the dimension of the productive clusters, while the “between” interactions has the role of determining the spatial localization of the clusters. With this abstract but not unrealistic tool we can verify the emergence of well known phenomena and, in a parallel way, of new ones, which appear to be not obvious, but plausible, behaviors of the district structures.

3. THE COMPUTATIONAL APPROACH

Our model can be formalized as a discrete time dynamic system. Consider two sets of variables, the first one describing the worker, $w_i \in W_i$ and the second one describing the firm $f_i \in F_i$. Since workers and firms are located on two $p \times q$ superimposed toroidal grids, we can consider a single grid where each cell can: a) contain a worker and no firm, b) contain a firm and no worker, c) contain both a firm and a worker, d) be empty. In cases a), b) c) the cell state consists of the informative variable of its content, while in case d) all the informative variables are null. The state of cell at location i, j can be formalized as a vector $\mathbf{x}_{ij} = (\mathbf{w}, \mathbf{f}) \in W_1 \times \dots \times W_m \times F_1 \times \dots \times F_n$ with the convention that either \mathbf{w} or \mathbf{f} can be the null vector, when respectively either no worker or no firm are present. We define the time t state of the system as the vector of the states of cells at time t $\mathbf{x}^t := (\mathbf{x}^t_{11}, \mathbf{x}^t_{12}, \dots, \mathbf{x}^t_{pq})$. Finally, consider the following stochastic processes:

- $\{\tilde{O}^t, t \in \mathbb{N}\}$ describing the orders generation
- $\{\tilde{W}^t, t \in \mathbb{N}\}$ describing new workers entry
- $\{\tilde{F}^t, t \in \mathbb{N}\}$ describing new firms entry

The evolution of the system can be formalized as follows:

$$\mathbf{x}^{t+1} = \varphi(\mathbf{x}^t, \tilde{O}^t, \tilde{W}^t, \tilde{F}^t)$$

The direct consequence of non-linearity and complexity is that the theoretical analysis of the formal model is not straightforward. This is well known in the literature and, according to many authors (e.g. Carley & Prietula, 1994), many models are too complex to be analyzed completely by conventional techniques that lead to closed-form solutions. In order to obtain some results turning to simulation is natural and necessary.

Having described all the agents interaction in our model of industrial district, to introduce the structure of the simulation is immediate. For each turn, first new orders are created and allocated to firms, then, after updating the social network induced by the mutual visibility of firms and workers, the productive process begins. Successively compatible and visible workers are hired by firms, all the firms with orders perform productive phases, then costs and revenues are accounted for and balances are computed. Defaulted workers and firms are removed. Finally visibility for firms is updated and new workers and firms are created. At this point the state of the system is displayed both in terms of graphical output and in terms of relevant data.

The Figures 1 and 2 show the parallel interacting process of order completion, production/stocking and workforce hiring. Those processes are supposed not to be interconnected in figure 1, which is the case of different unlinked populations. On the contrary the figure 2 shows the case of coevolving symbiotic scenario where production depends on workers availability, and their lack can cause the firm to go out of stock (one more cause of death).

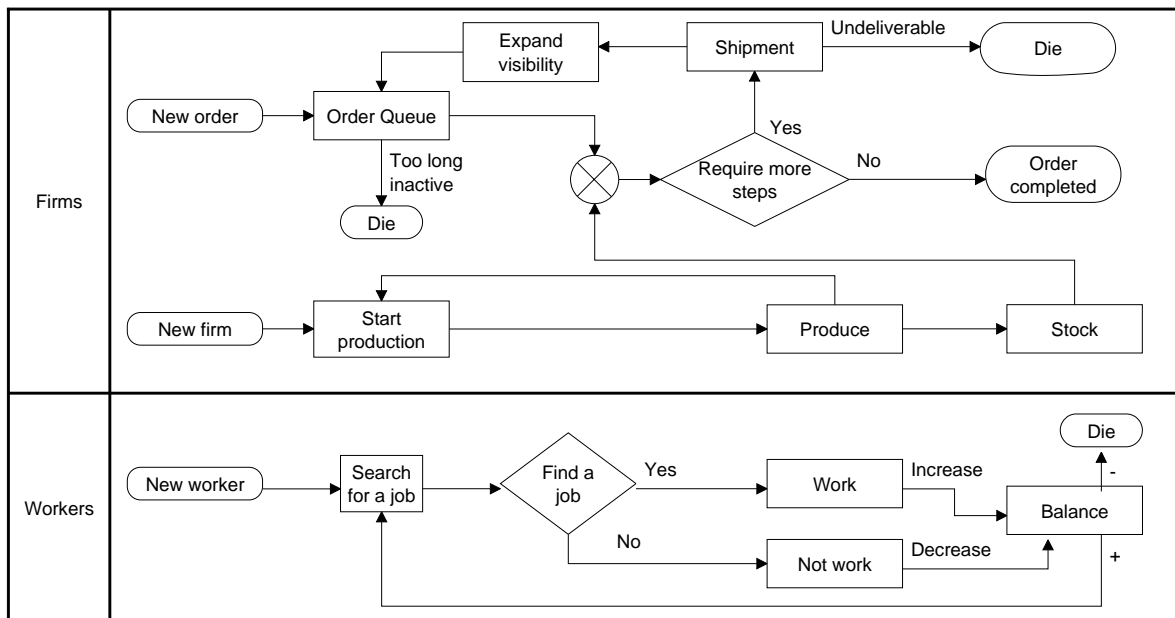


Figure 1 The dynamic process of independent populations.

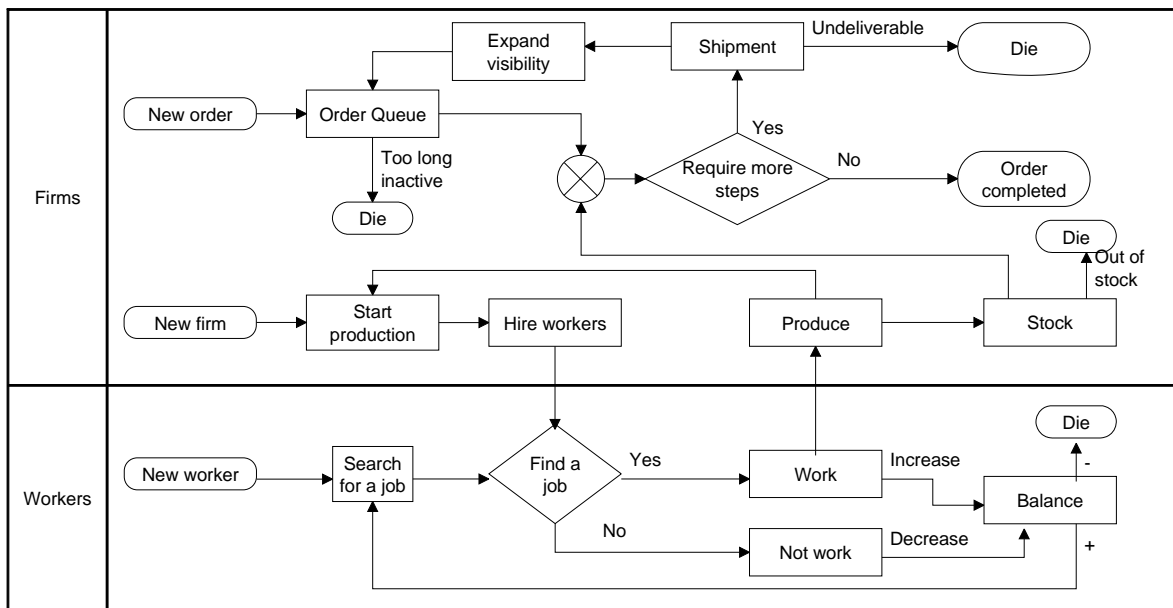


Figure 2 The dynamic process of co-evolving model.

4. THE COMPUTER IMPLEMENTATION

Different simulation tools are available to social scientists interested in agent-based simulations. Among the most widely used we recall Swarm, Repast, NetLogo; yet other approaches are possible. For example it is also possible to implement models by custom simulation platforms using high level languages. While the choice of the simulation platform should have no effects on the simulations results, this might not always be true (see for example Polhill et al., 2005). Furthermore, in science, repetition is what allows results to be validated and accepted by the scientific community.

Furthermore, in science, repetition is what allows results to be validated and accepted by the scientific community. Considering computer simulations, repetition can be intended just as experiment replication by other scientists or, more dramatically, as model reimplementaion. While experiment replication can be easily achieved, reimplementaion is a more rigorous process in which the model is deeply examined again. Given the complexity of the models considered, and the fact that computer programming is an extremely error-prone process, reimplementaion of the model may be extremely valuable in order to identify potential problems that may invalidate results.

We implement this model using jESOF, an enterprise simulator based on Swarm, JAS and a custom C++ implementation.

Our purpose is to compare the advantages of the three implementations and highlight the benefits arising when the same model is implemented on radically different platforms.

5.1 The jESOF/Swarm Implementation

We use the original package jESOF (java Enterprise Simulation Open Foundation, described at <http://web.econ.unito.it/terna/jes>). The package is built using the Swarm library for agent based models (described at <http://www.swarm.org>).

The purpose of the jESOF structure is to develop a two-side multilayer world, considering both the actions to be done, in terms of orders to be accomplished (the “What to Do” side, WD), and the structures able to execute them, in terms of production units (the “which is Doing What” side, DW). WD and DW can be consistent or inconsistent and the presence of social capital - expressed by the necessity of firms inter-visibility as a condition to exchange – introduces inconsistent situations reproducing real world occurrences.

The jESOF dictionary follows:

- unit: a productive structure; a unit is able to perform one of the steps required to accomplish an order;
- order, the object representing a good to be produced; an order contains technical information (the recipe describing the production steps);
- recipe, a sequence of steps to be executed to produce a good.

The central tool in this simulation environment is a proprietary scripting language used to describe units acting in the simulated world and to define actions to be done within the simulation framework by the recipes contained in the orders. The recipes can call computational functions written in Java code to make complicated steps such as creating new firms or workers, to hire workers, to account for income and consumptions of the different units.

In our case the scripting language uses two different sets of recipes included in orders.

- In the firm stratum we have recipes related to production, with sequences of steps describing the good to be produced.
- In the workers stratum, which is also the interaction place, recipes produce five kinds of effects: (i) new workers appear in the simulation context, either near to similar ones, or randomly distributed; (ii) firms hire workers and recipes modify workers and firms private matrixes; this is done accounting for both the availability of the labor production factor (firm side) and household income (workers side); (iii) firms make use of available labor production factors; (iv) firms either short of orders to be processed, or lacking adequate workers on the market, or being unable to deliver produced goods disappear from the economic scenario; (v) workers also disappears if unable to find a firm for prolonged time.

Recipes are able to perform complex tasks, such as those described above, and are developed via computational steps. These steps can be interpreted as calls to code functions (methods of a Java class) invoked by a scripting language.

As an example, the sequence ‘1001 s 0 c 1220 2 0 0 1 s 0’ is a recipe describing a task of type (ii) above, going from a unit of type 1001 (a firm) to a unit of type 1 (a worker) and invoking a computational step with id code # 1220.

An example of the Java code is given in ; note that the method uses both internal parameters and matrix references.

```

public void c1220(){

    if (pendingComputationalSpecificationSet.
        getNumberOfMemoryMatrixesToBeUsed() != 2)
        {
            System.out.println(« Code -1220 requires 2 matrixes ; « +
                pendingComputationalSpecificationSet.
                getNumberOfMemoryMatrixesToBeUsed() +
                " found in order # " +
                pendingComputationalSpecificationSet.
                getOrderNumber());
            MyExit.exit(1);
        }

    rd=4;

    // displacements for the unit memory matrixes coordinates
    mm1= (MemoryMatrix) pendingComputationalSpecificationSet.
        getMemoryMatrixAddress(1);
    layer=pendingComputationalSpecificationSet.
        getOrderLayer();
    urd0=(int) mm1.getValue(layer, 0+rd, 2);
    ucd0=(int) mm1.getValue(layer, 0+rd, 3);
    urd1=(int) mm1.getValue(layer, 0+rd, 4);
    ucd1=(int) mm1.getValue(layer, 0+rd, 5);

    checkMatrixNumber=false;
    c1120();
    checkMatrixNumber=true;
    rd=0;
    urd0=0 : ucd0=0 : urd1=0 : ucd1=0 :

```

Figure 3 Java Code implementation for a computational firm.

5.2 The JAS Implementation

The implementation we present is written using the JAS library (described at <http://jaslibrary.sourceforge.net>). It is compliant to the standard modeling approach often used in agent based modeling: the model-observer paradigm. Everything related to the description of the model is put into a section (a set of classes) called ‘model’, while the code relative to data analysis, graphical representation and interaction with user is put into the ‘observer’ section.

While JAS is very similar to the Swarm architecture, the former implementation is very different from the one presented here, since the jESOF layer introduces a well-defined language and structure to model organizations. It allows an higher level of abstraction in the model description.

JAS is used here only as a basic platform, offering the modeler some well tested libraries. The logic of the model has been designed from scratch, using object-oriented programming and Java formalism.

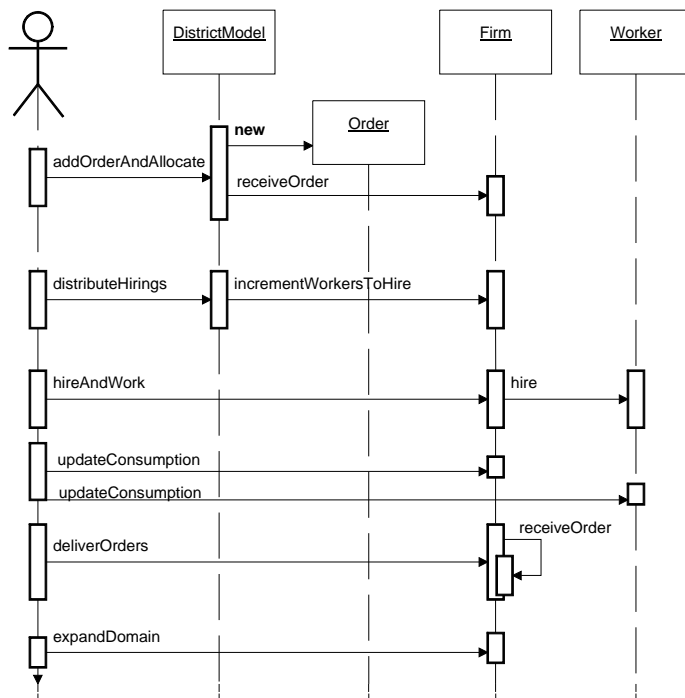


Figure 4 Time-sequence UML diagram of the model implemented with JAS.

The Figure 4 shows the dynamic structure of the model, representing the sequence of events which activate the agents. It is described using a modified UML sequence diagram, as described in Richiardi *et al.* (2006).

Thanks to the JAS built-in libraries the implementation is rather standard. In fact, the toroidal space representation as well as the random number generators are available in the package. The most tricky aspect of the model implementation has been represented by the management of intersections between firms and workers and among firms.

Taking advantage from the object oriented programming both the Firm and Worker classes has been implemented as subclasses of a generic class called DomainPlayer. Through its methods of all the *domain players* (firms and workers) are able to expand the domain visibility and computer intersection with other players. In particular, the intersection algorithm is shown in the Figure 5.

```

private Set [][] domain;

public Set getPlayersInMyDomain(Class classType) {
    Set fullSet = new HashSet();

    for (int xx = 0; xx < space.getXSize(); xx++ )
        for (int yy = 0; yy < space.getYSize(); yy++ )
            {
                final Set set = domain[xx][yy];
                if (set.contains(this) && set.size() > 1)
                    {
                        Iterator it = set.iterator();
                        while (it.hasNext())
                            {
                                final Object itm = it.next();
                                if (itm.getClass().equals(classType))
                                    fullSet.add(itm);
                            }
                    }
            }

    fullSet.remove(this);
}

```

Figure 5 The method `getPlayersInMyDomain()` from `DomainPlayer` class.

5.3 The C++ Custom Implementation

The implementation we present here is written for C++, in particular we use Borland C++ Builder 5.0. Our approach consists of two phases: first, the model is coded as set of classes; second, we decide what sort of information are displayed to the user. The first phase is independent from C++ compiler used, while the second may rely more on the used compiler and will involve technical aspects not so relevant here. For these reasons we shall focus our attention to the first phase.

The implementation we chose is hierarchical: we modeled a container class called *district* which contains pointers to objects belonging to classes *order*, *worker* and *firm*. The container class implements the methods performing each phase of a simulation turn, the graphical display methods and the interface with the main unit. The code for a turn of simulation is reported in .

For sake of brevity technical details are not discussed here, source codes are available from the Authors upon request.

```

// create order
mydistrict->CreateOrder();
// allocate order or destroy it
mydistrict->AllocateOrder();
// permutate firms and workers
mydistrict->PermutateFirms();
mydistrict->PermutateWorkers();
// compute intersections firm/firm and firm/worker
mydistrict->FindFirmsIntersection();
mydistrict->FindFirmWorkerIntersection();
// firms work
mydistrict->HirePWorkers();
// firms consumption
mydistrict->FirmConsumption();
// worker consumption
mydistrict->WorkerConsumption();
// check alive firms and reallocate orders
mydistrict->ReallocateOrders();
// expand firms
mydistrict->ExpandFirmDomain();
// expand population
mydistrict->ExpandPopulation(CBProxWG->Checked);
// Display
mydistrict->FastPaintFlatDistrict(FlagView);

```

Figure 6 C++ implementation code for a simulation turn.

5.4 A Discussion on the Strengths and Weaknesses of the Different Implementations

While the first two approaches rely on well tested libraries and most of the implementation details are hidden from the programmer, the third approach requires almost everything to be built from “ground zero”. As a result it is more time consuming and certainly more error prone. On the other side the definitive advantage of the second approach is flexibility both in terms of graphical output and of interactivity.

The three implementations follow an increasing abstraction approach. In fact, while the C++ implementation does not refer to any modeling framework, the JAS one is founded on a well known and accepted implementation pattern. It is the so called model-observer paradigm, which is largely adopted in many platforms like Swarm, Repast and JAS.

Not only a JAS-compliant model is easier to be debugged and read by others but it can get a more elegant way to declare some modeling specific aspects, like the time event structure. The model definition is self contained into a specific class (DistrictModel). The number, type and relationships of agents are declared at the model building procedure as well as the declaration of the events will be fired during the simulation execution.

The extreme abstraction is reached in the jESOF implementation. Even if it is based on Swarm, which is largely comparable with JAS, the jESOF layer provide a declarative approach in designing agent-based models. In fact, through the declaration of a set of unit capabilities and a sequence of recipes (the doing-what and what-to-do perspectives) the model is quite complete. The custom logic differentiating the model from the basic jESOF model is provided by some algorithms defined as computational steps.

Even if none of the three approaches does provide an agent-based specific language, as exemplified by the Starlogo/Netlogo experience, some of them provide a protocol in design process. The Swarm-like platforms are based on the following principles:

- the use of object-oriented programming language, with different objects representing different agents (and agent types);
- a separate implementation of the model and the tools used for monitoring and controlling experiments on the model (the so called “Observer”);
- an architecture that allows nesting models one into another, in order to build a hierarchy of “swarms”. One swarm can thus contain lower-level swarms whose schedules are integrated into the higher-level schedule.

Those three principles are followed in the three approaches, with a lot of differences. Taking into account the multi-object (or multi-agents) approach, it is naturally present in both the C++ and JAS implementations, while the jESOF one provides an abstraction which does not require to explicitly model an agent type. There is an implicit description of their capabilities. They are only logically modeled.

The separation of model and observer are naturally present in the jESOF and JAS implementation, since they are both based on platform providing a “model-observer” framework. The C++ one follows the principle, even if the separation is less rigorous.

The possibility of nesting swarms into swarms is possible in the three implementations, thanks to the flexibility of object-oriented programming languages. Obviously the jESOF platform has an embedded multi-layered architecture, providing a natural way of nesting models and sub-models. The other two approaches would require a specific implementation of such model separation. JAS automatically manages schedule integration, while C++ would require its custom management.

Moreover, the objective was rather to build a sufficiently reliable code even at the cost of some redundancies. Since the final goal of parallel implementation is the replication of the results, the whole process is certainly beneficial for the theoretical validation of the model. In fact the analysis and comparison of the implementation details resulted in the discussion of the assumptions of the whole model. Furthermore, while in general with parallel implementations some economies of scale in designing at least the “housekeeping” algorithms are to be expected, this did not happen in our case. The reason is to be found in the fact that the C++ implementation could not benefit from the Swarm or JAS libraries. On the other side, however, this may avoid the implementation of ill designed algorithms.

When comparing the three implementations, the results we obtain are qualitatively the same even if they are different in terms of exact replication. The reasons for these differences are several. Mainly they come from minor modeling differences in the three implementations, secondarily they come from more technical reasons such as the error accumulating when floating-point arithmetic operations are performed, and the different random numbers generators routines which are used in the two different codes.

As for the first point, while in our case the consequences of floating point arithmetic are not as serious as those described in Polhill *et al.* (2005), the fact that we perform several branching comparing floating point variables may be one of the reasons leading to different behaviors of our platforms. The other relevant aspect to be discussed is the one concerning random numbers generators. jESOF code uses well tested routines internal to the Swarm library, and devoted to integer or double precision number, quoting Swarm documentation “The generator has a period

close to 2^{19937} , about 10^{6001} , so there is no danger of running a simulation long enough for the generator to repeat itself. At one microsecond per call, it would take about $3.2 \cdot 10^{5987}$ years to exhaust this generator. For comparison, the age of the Universe is ‘only’ $2 \cdot 10^{10}$ years! This generator can be asked either for a real number (a variable of type double) between $[0,1)$ or for an integer, which will be uniformly distributed on the range $[0,4294967295] = [0,2^{32}-1]$. Even JAS uses a well tested and very fast algorithm for generation of random number⁴: the Mersenne twister. As explained in Matsumoto and Nishimura (1998), it provides for fast generation of very high quality pseudorandom numbers, having been designed specifically to rectify many of the flaws found in older algorithms.

With the C++ implementation we felt we could not rely on the internal random numbers generator. The reasons for our choice can be found in Press et al. (2002), and we decided to use a Minimal Standard generator with a shuffling algorithm which is reported there as “ran1”.

5. CONCLUSIONS

In this paper we examined and discussed a model of industrial districts where social agents cooperate symbiotically. The approach we used consisted in the parallel implementation of the model with jESOF, JAS and C++.

The multiple implementation approach allowed the results comparison and a complete discussion of the model assumptions. We were able to evaluate and compare different modeling approaches on the same model.

Another interesting aspect was the replicability of the results; while the results we obtained were qualitatively the same, there were differences in terms of exact replication. The reasons for these differences can be found both in the error accumulation process involved in sequences of floating point operation and in the different random generators used.

The comparison of results is not conducted with the aim of finding differences, but to understand which aspects of the model implementation are relevant on results as well. In fact, the results are supposed to be qualitatively the same, if the implementation is coherent with the model description.

So far, the multiple implementation is carried on to compare modeling paradigms, in terms of appropriateness to problem formulation and in terms of implementation time.

Finally an important prescriptive lesson arises from our experience. To maximize effectiveness in agent based modeling, the scientist should consider first to implement a prototype model by a fast development platform such as Netlogo or jESOF in order to assess its feasibility. After this phase the serious researcher should consider both someone else rewriting the model with a more customizable platform (either using a generalized ABM such as Swarm or using object oriented languages) and reengineering its structure avoiding the constraints which were imposed by the shell quoted above. These are important steps of modeling discussion and are obviously the first ones in order to encourage scientific repeatability.

The sharp idea that comes from the experience of implementing the same model with different approaches consists in the important impact that tools have in turning a model description into a

⁴ The generator is taken from the COLT library (<http://hoschek.home.cern.ch/hoschek/colt/>) and more precisely is referred to the class `cern.jet.random.engine.MersenneTwister`.

computer artifact. In fact, in order to obtain the same results, parameter sets, algorithm tuning and implementation architectures have to be carefully managed with a continuously improving and testing process.

Researchers potentially need to have a wide range of techniques in their modeling toolkit to be effective. An important point is that depending on the exact nature of the model we are developing a subset of those techniques and tools are required. Over time the variety of problems require to use each specific tool at some point.

The time between an action and the feedback on that action is critical, mostly when working with other people on a model. It is necessary that the implementation technique allows a near-instant feedback on modeling ideas.

In further research we plan to obtain the full replication with both implementations, since this should be the only way to be reasonably certain that most of the coding errors have been properly addressed. This technical goal can be achieved considering either a deterministic version of the model or, more reasonably, considering the same random generator, possibly one where sequences can be replicated. Furthermore it would be interesting to investigate the role of workers' clustering in district emergence.

6. REFERENCES

- Auerswald, P., Kauffman, S., Lobo, J., & Shell, K. (2005). The Production Recipes Approach to Modeling Technological Innovation: An Application to Learning by Doing. *CAE Working Paper*, 98-10.
- Becattini, G., Pyke, F., & Sengenberger, W. (Eds.), (1992). *Industrial Districts and Inter-firm Co-operation in Italy*. Geneva: International Institute for Labour Studies.
- Becattini G. (2003). From the industrial district to the districtualisation of production activity: some considerations. In F. Belussi, G. Gottardi, & E. Rullani (Eds.), *The Technological Evolution of Industrial Districts*. Dordrecht: Kluwer Academic Publisher.
- Becker, P. (2005). Bad Pointers. *C/C++ Users Journal*, 23(9), 37-41.
- Belussi, F., & Gottardi, G. (Eds.) (2000). *Evolutionary Patterns of Local Industrial Systems. Towards a Cognitive Approach to the Industrial District*. Sydney: Ashgate.
- Brenner, T. (2002). Simulating the Evolution of Localised Industrial Clusters- An Identification of the Basic Mechanisms. Presented at IG2002, Sophia Antipolis, France, January 18-19, 2002. Retrieved November 15, 2005 from <http://www.idefi.cnrs.fr/IG2002/papers/Brenner.pdf>.
- Bruno, G., & Otranto, E. (2004). Dating the Italian business cycle: a comparison of procedures. *ISAE, working Paper* 41.
- Carbonara, N., (2004). Innovation processes within geographical clusters: a cognitive approach. *Technovation*, 24, 17-28.
- Carbonara, N., (2005). Information and communication technology and geographical clusters: opportunities and spread. *Technovation*, 25, 213-222.
- Carley, K. M., & Prietula, M. J. (1994). *Computational Organization Theory*. Hillsdale. N.J: Lawrence Erlbaum Associates.
- Cerruti, U., Giacobini, M., & Merlone, U. (2005). A New Framework to Analyze Evolutionary 2x2 Symmetric Games. Proceedings of the *IEEE 2005 Symposium on Computational Intelligence and Games*, April 4-6 2005 Essex University, Colchester, Essex, UK.
- Fioretti, G. (2002). Individual Contacts, Collective Patterns - Prato 1975-97, a Story of Interactions. *Tinbergen Institute Discussion Paper*, TI-2002-109/3.
- Garofoli, G. (1981). Lo sviluppo delle aree "periferiche" nell'economia italiana degli anni '70. *L'industria* 3, 391- 404.

- Garofoli, G. (1991). Local Networks, Innovation and Policy in Italian Industrial Districts. In E. Bergman, G. Mayer & F. Tödtling (Eds.), *Regions Reconsidered. Economic Networks Innovation and Local Development in Industrialized Countries*. London: Mansell.
- Garofoli, G. (1992). Industrial Districts: Structure and Transformation. In G. Garofoli (Ed.), *Endogeneous Development and Southern Europe*. Avebury: Aldershot, 49-60.
- In F. Belussi ,G. Gottardi, & E. Rullani (Eds.), *The Technological Evolution of Industrial Districts* . Dordrecht: Kluwer Academic Publisher.
- Hall, R.E. (2005). Employment Fluctuations with Equilibrium Wage Stickiness. *American Economic Review* 95(1), 50-65.
- Hastings, A. (1997). *Population Biology. Concepts and Models*. New York: Springer-Verlag.
- Hofbauer, J., & Sigmund, K. (1998). *Evolutionary Games and Population Dynamics*. Cambridge: Cambridge University Press.
- Kaitala, V., Ranta, E., & Lindström, J. (1996). Cyclic Population Dynamics and Random Perturbations. *Journal of Animal Ecology* 65(2), 249-251.
- Law, A. M., & Kelton, W. D. (2000). *Simulation modeling and analysis*. Boston: Mc-Graw-Hill.
- Lazzaretti, L., & Storai, D. (1999). Il distretto come comunità di popolazioni organizzative. Il caso Prato. Prato: *Quaderni IRIS n.6*.
- Lazzeretti, L., & Storai, D. (2003). An ecology based interpretation of district “complexification”: the Prato district evolution from 1946 to 1993. In F. Belussi ,G. Gottardi, & E. Rullani (Eds.), *The Technological Evolution of Industrial Districts* . Dordrecht: Kluwer Academic Publisher.
- Lotka, A. J. (1925). *Elements of physical biology*. Baltimore: Williams & Wilkins Co.
- Matsumoto, M. & Nishimura T. (1998). *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*. New York: ACM Transactions on Modeling and Computer Simulation (TOMACS), Vol.8(1) p. 3-30.
- Piva, M., Santarelli, E., & Vivarelli, M. (2003). The Skill Bias Effect of Technological and Organizational Change: Evidence and Policy Implications. *IZA Discussion Paper, 934*.
- Polhill, J.G., Izquierdo, L.R., & Gotts, N. M. (2005). *The Ghost in the Model (and other effects of the floating points arithmetic)*. *Journal of Artificial Societies and Social Simulation*, 8(1), 5. Retrieved November 5, 2005 from <http://jasss.soc.surrey.ac.uk/8/1/5.html>
- Press, W.H., Teulkolsky, S.A., Vetterling, W.T., & Flannery,B.P. (2002). *Numerical Recipes in C++*. Cambridge: Cambridge University Press.
- Richiardi M., Leombruni R., Sonnessa M. & Saam N. (2006). *A Common Protocol for Agent-Based Social Simulation*. *Journal of Artificial Societies and Social Simulation*, 9(1). Online at <http://jasss.soc.surrey.ac.uk/9/1/15.html>
- Royama, T. (1992). *Analytical population dynamics*. New York : Chapman and Hall.
- Rullani, E. (2003). The Industrial District (ID) as a cognitive system. In F. Belussi ,G. Gottardi, & E. Rullani (Eds.), *The Technological Evolution of Industrial Districts* . Dordrecht: Kluwer Academic Publisher.
- Shimer, R. (2005). The Cyclical Behavior of Equilibrium Unemployment and Vacancies. *American Economic Review* 95(1), 25-49.
- Sorenson, O., Rivkin, J.W., & Fleming, L. (2004). Complexity, Networks and Knowledge Flow, *Working Paper*.
- Squazzoni, F., & Boero, R. (2002). Economic Performance, Inter-Firm Relations and Local Institutional Engineering in a Computational Prototype of Industrial Districts. *Journal of Artificial Societies and Social Simulation*, 5(1), 1. Retrieved November 15, 2005 from <http://jasss.soc.surrey.ac.uk/5/1/1.html>.
- Sterman, J.D. (2000). *Business Dynamics. System Thinking and Modeling for a Complex World*. Boston: Irwin McGraw-Hill.
- Swarm Development Group (2005). Documentation Set for Swarm 2.2, Random Library. Retrieved June 20, 2005, from

<http://www.swarm.org>.

Volterra, V. (1926). Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. Roma: *Mem. R. Accad. Naz. dei Lincei*. VI(2).

Zhang J., (2003). Growing Silicon Valley on a landscape: an agent-based approach to high-tech industrial clusters. *Journal of Evolutionary Economics* 13. 529-548.