Experiences on Stochastic π-Calculus

± 4 weeks after

Ing. Luca Gardelli
lgardelli@deis.unibo.it
lgardelli@unibo.it

DEIS - Dipartimento di Elettronica, Informatica e Sistemistica
Alma Mater Studiorum - Università degli Studi di Bologna
via Venezia 52, 47023, Cesena (FC) Italy
Ph: +39 0547 339244 Fax: +39 0547 339208

Foreword

- We will not discuss again about π-calculus and its stochastic extension ...
- You can easily refer back to the previous presentation

Outline

- What we can easily model in π-calculus
  - examples
  - desirable features
- Tools related to π-calculus
  - existing tools
  - desirable features
- Conclusions

What we can 'easily' model

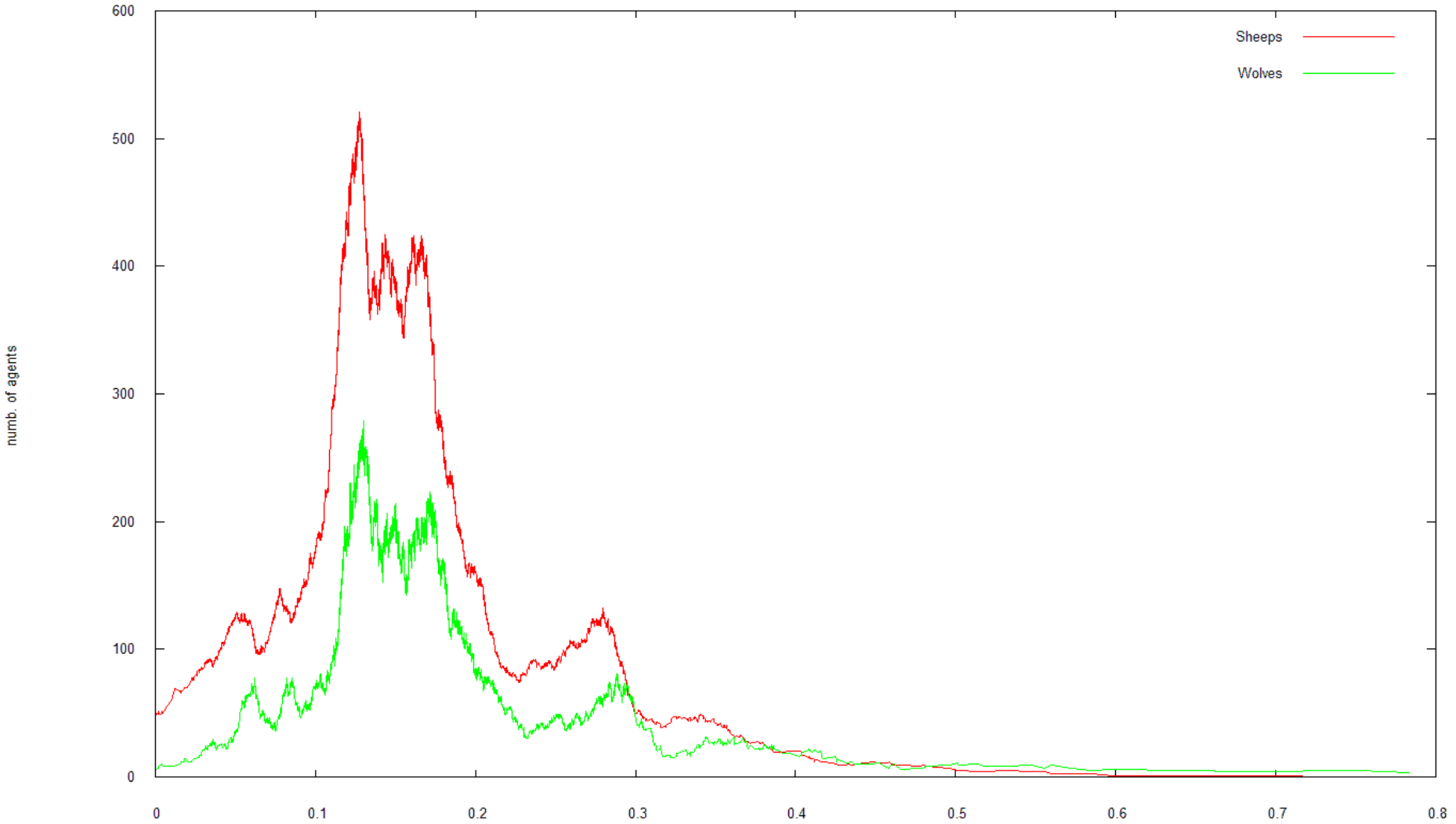
- From the further experience gained in this period...
- We can easily model systems
  - populated with a lot of agents, i.e. > 10^2
  - where the number of classes of agents is small, i.e. <10
  - that includes stochastic phenomena
  - that includes communication between processes
- We can simulate the model and observe self-organizing dynamics and emergent properties

Example: prey-predator system

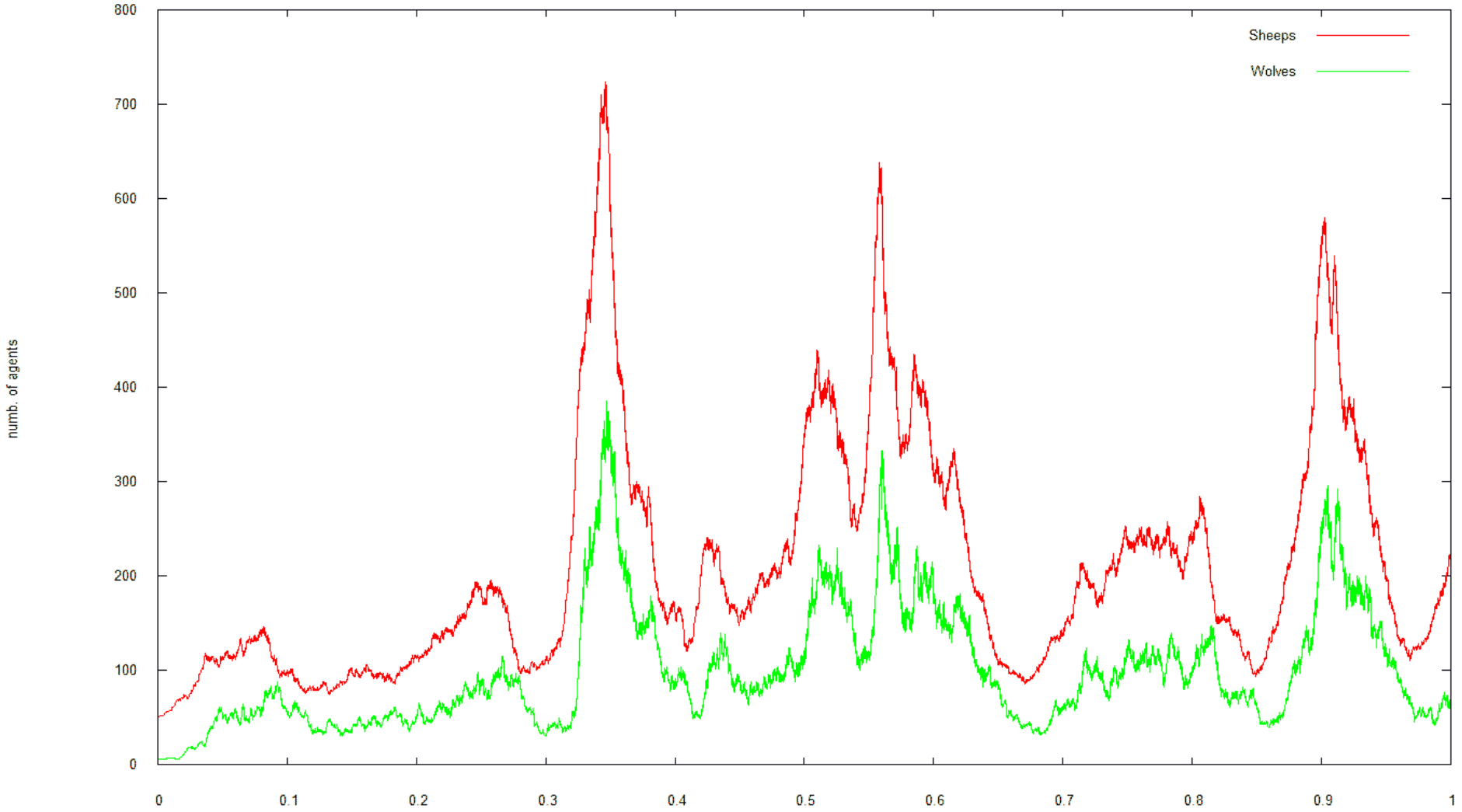
- The system is populated by 2 kinds of agents: Sheeps, Wolves
- The sheeps are able to reproduce or been eaten by wolves
- Wolves can reproduce, eat sheeps, starve
- We want to observe the overall behavior of the ecosystem when we tune the sheep growth rate

```
directive sample 1.0
directive plot Sheep(); Wolf()
new eatsheep@1.0 : chan
new sheepgrow@0.54: chan
new wastenergy@1.0 : chan
let Sheep() =
do ?eatsheep
or !sheepgrow; (Sheep() | Sheep() )
or ?sheepgrow; Sheep()
let Wolf() =
do !eatsheep; (Wolf() | Wolf())
or !wastenergy
or ?wastenergy
run 50 of Sheep()
run 5 of Wolf()
```

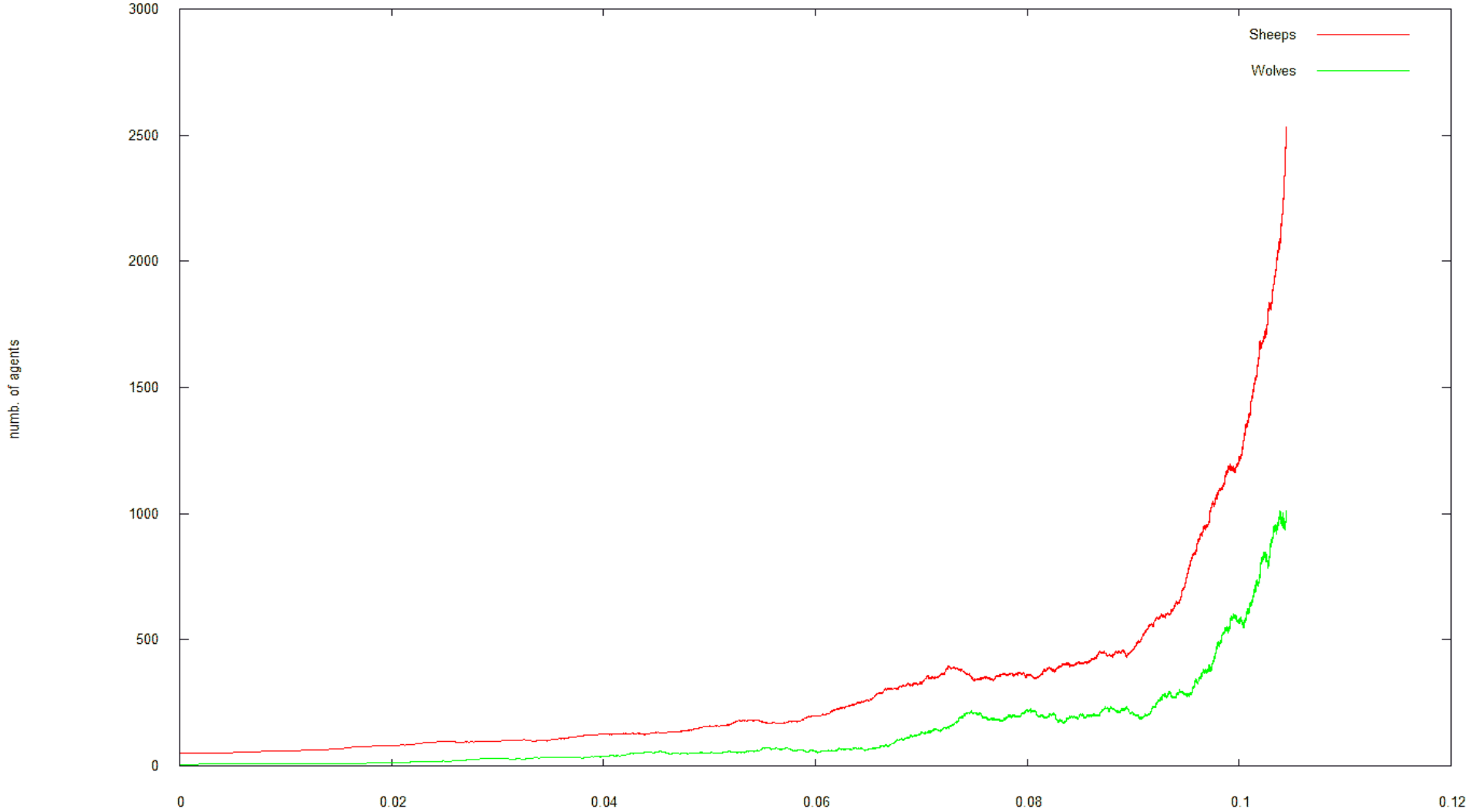
Example: prey-predator system



Example: prey-predator system



Example: prey-predator system

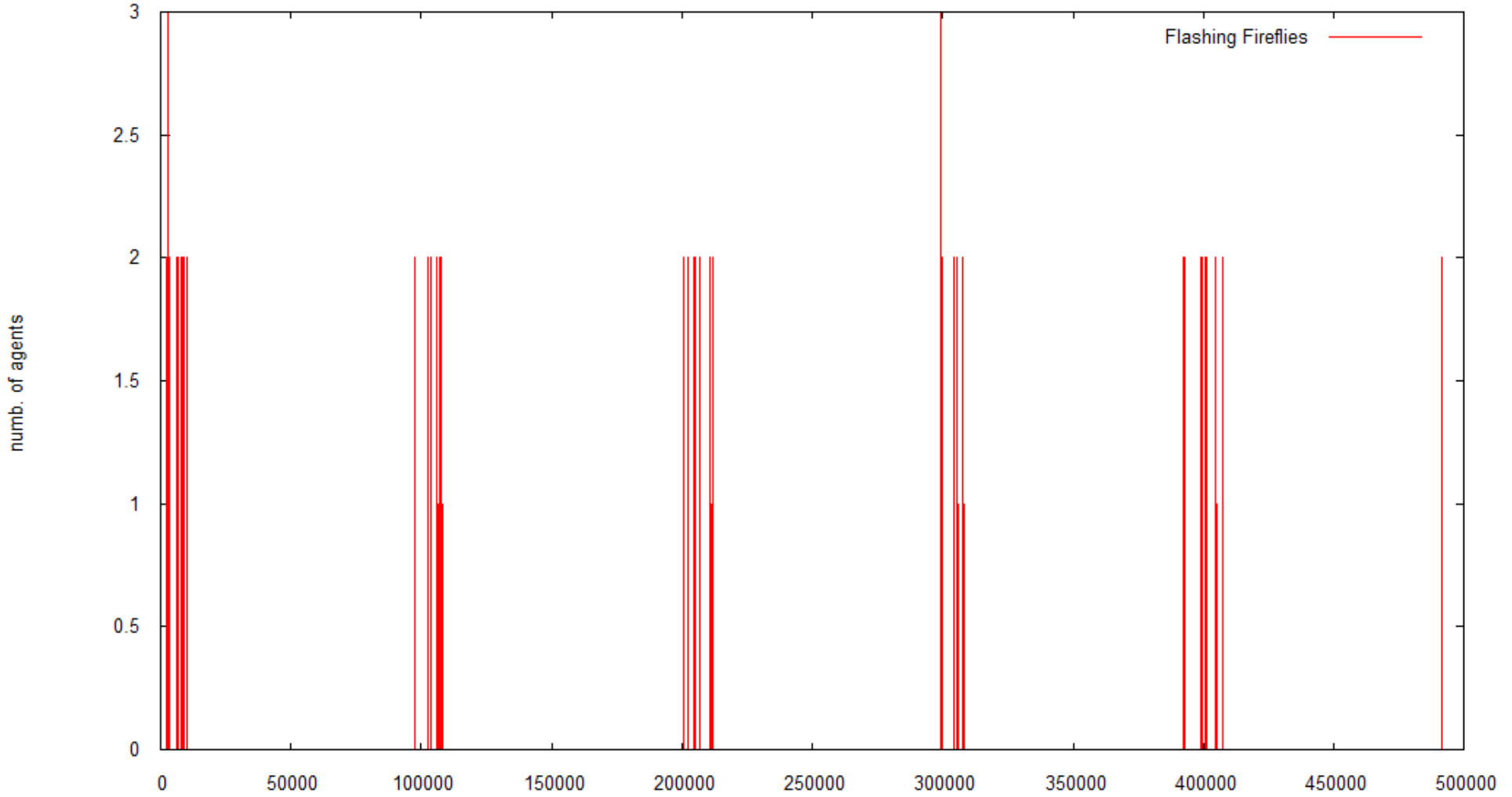


Example: Fireflies

- The system is populated by fireflies
- Every firefly counts using the same basis – not really necessary – and then flashes
- If a neighbor flashes, it flashes too and start counting from the beginning

```
directive sample 500000.0
directive plot !flashing as "Flashing"
val counter = 1000              val ritardo = 0.01
new flash@100000.0:chan
new flashing@100000.0:chan
let Fireflies(c:int) = (if c > 0 then Fireflies1(c) else Fireflies2())
and
Fireflies1(c:int) = (
 do delay@ritardo; Fireflies(c-1)
 or ?flash; ( !flashing | delay@ritardo; ?flashing;
Fireflies1(counter)))
and
Fireflies2() = (!flash | !flashing | delay@ritardo; ?flashing;
Fireflies1(counter))
run 2 of Fireflies(57)          run 2 of Fireflies(80)
run 2 of Fireflies(70)          run 2 of Fireflies(34)
run 2 of Fireflies(23)
```

Example: Fireflies



Remarks

- Notice that in these kind of systems situatedness can be abstracted away.
- We hypothesise that agents move and “statistically” come closer like in gas particles systems.
- The agents are distributed more or less uniformly in the evironment.

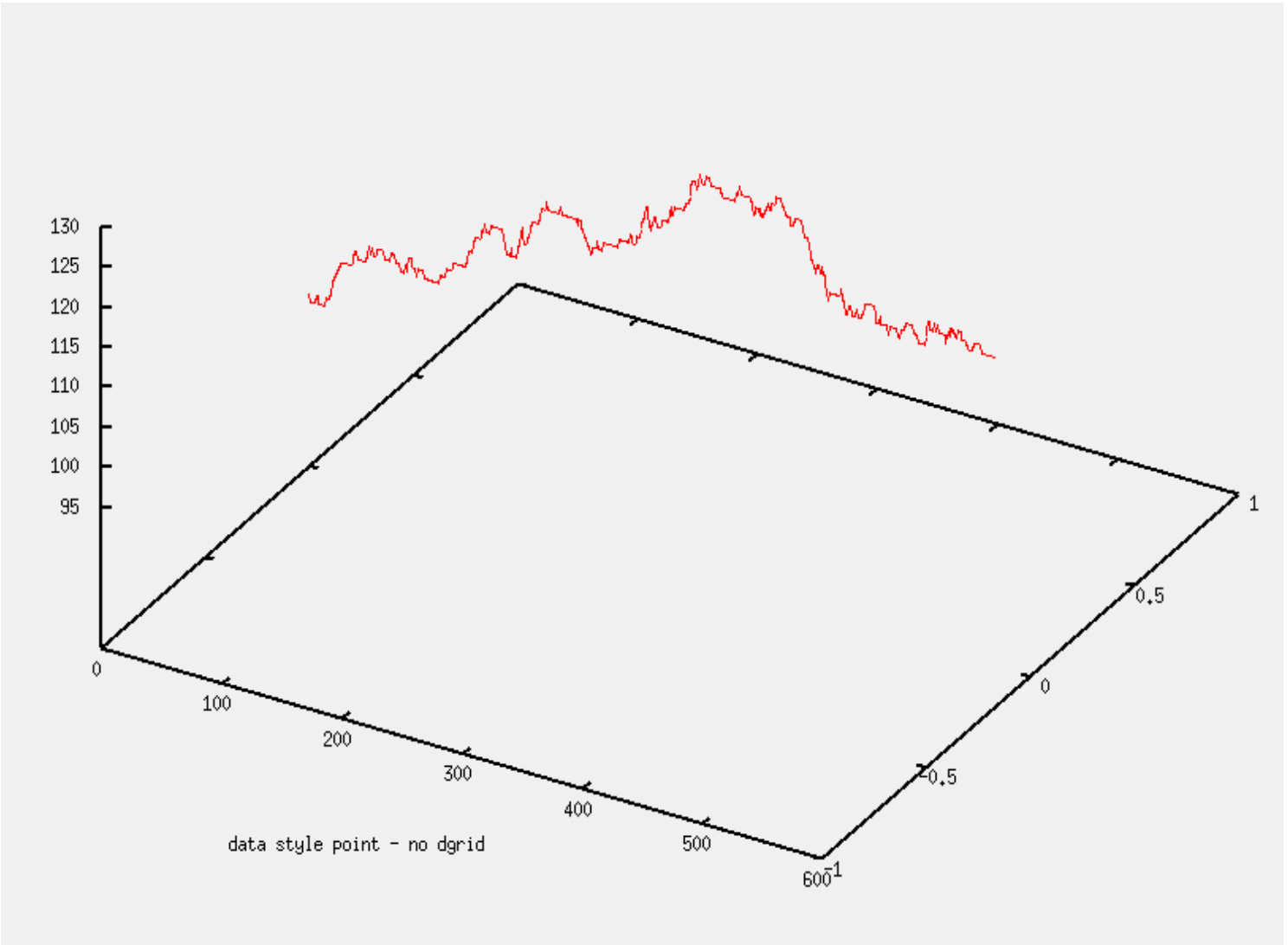
What is it difficult to model?

- Difficult systems to model include the ones with
  - a lot of different classes of agents
  - too many details of agents inner working
  - strong situatedness
- It is impractical to model systems where situatedness affects the way of communicating!

Example: ant wandering

```
directive sample 0.1          directive plot Tracker()
val left = 0        val right = 1      val top = 2        val bottom = 3          val front = 4      val back = 5
new reqLocation:chan       new sendLocation:chan(int,int,int)
let SpawnAgent() =(new move@1000.0:chan(int)  ( Ant(move) | Tracker(100,100,20,move) ))
and Tracker(x:int,y:int,z:int,move:chan(int)) =(
do ?move(dir); match dir
case 0 -> Tracker(x-1,y,z,move)              case 1 -> (Tracker(x+1,y,z,move))
case 2 -> (Tracker(x,y,z+1,move))            case 3 -> (Tracker(x,y,z-1,move))
case 4 -> (Tracker(x,y+1,z,move))            case 5 -> (Tracker(x,y-1,z,move))
or ?reqLocation; !sendLocation(x,y,z); Tracker(x,y,z,move) )
and Ant(move:chan(int)) =(
        do !move(left); Ant1(move)           or !move(right); Ant1(move)
        or !move(top); Ant1(move)            or !move(bottom); Ant1(move)
        or !move(front); Ant1(move)          or !move(back); Ant1(move))
and Ant1(move:chan(int)) = !reqLocation; ?sendLocation(x,y,z); println(show x + "," + show y + "," + show z);
Ant(move)
run 1 of SpawnAgent()
```

Example: ant wandering



How can we make (my) life simpler?

- We can provide facilities to handle situatedness for
  - local communication
  - modeling agents mobility
- These facilities may be implemented in 3 different ways
  1. patterns of usage / library of processes
  2. modifying the syntax @ programming language level
  3. modifying the syntax @ process algebra level

How can we make (my) life simpler?

- My “feelings” are the following
  1. is the easiest way, but for evaluation only (like the previously showed Tracker process)
  2. provides a better support without requiring formal proofs of correctness of the process algebra
  3. the hard-way: requires a lot of math and in general is not the best way for evaluate new concepts

Mobile Ambients – Cardelli 1998

- In mobile ambients the environment is represented as a tree of ambients.
- Branches of the tree can move upon agent request, but agents are confined within the ambients.
- It is mainly a model for exploring security issues.

Local Names – Bodei 1996

- Each name is prefixed by an address in the form x @ y.
- This type of locality is more similar to a locality of scope than locality of space.
- It is not suited for the type of systems we want to model.

Topology Information – Cardelli 1996

- Extends Stochastic π-calculus with a notion of topology.
- Processes run on top of a network of processors.
- This model is targeted to performance evaluation when resources are constrained.

Situatedness in the P.A. Literature

- There are a few attempts to extend π-calculus with locality issues.
- None of them are suitable for describing systems like the stigmergic ones.
- We would like to have a process algebra in which a communication happens in a “broadcast” fashion, but the message should be available only in the sender's neighborhood.

Possible extensions

- send @ 1000.0 : chan - channel syntax unmodified
  - relative neighborood defined by rate channel: in physical system higher frequencies attenuate faster than lower one
  - the location is provided by the agent performing the process
- send @ 1000.0 : chan - channel syntax unmodified
  - refer implicitly to an agent's capability rather than the environment
  - each agent has a cone of view and a location
- send ρ u30.0 @ 1000.0 : chan - channel syntax modified
  - the neighborhood radius ρ is defined by a distribution and parameter, e.g. uniform in a radius of 30.0 units
  - the location is provided by the agent performing the process

Comments on the extensions

- the 1st and the 3rd ones require modifications to the interpreter while in 2nd one act as a kind of pattern.
- the 1st one is not very versatile but can effectively express physical phenomena, e.g. wave propagation.
- the 1st and 3rd ones requires less effort from the programmer's viewpoint and lead to cleaner models.

Outline

- What we can easily model in π-calculus
  - examples
  - desirable features
- Tools related to π-calculus
  - existing tools
  - desirable features
- Conclusions

Tools related to π-calculus

- Pict – a programming language based on pi-caculus.
- p.s.a.: p-Calculus Static Analyzer – an academic tool for inferring the potential behavior of a system .
- KLAIM - a formalism that supports a programming paradigm where processes, like data, can be moved from one computing environment to another.
- TyCO - Typed Concurrent Objects, is an implicitly typed polymorphic concurrent language based on an extension of the asynchronous pi-calculus.

Tools related to π-calculus

- Symbolic Trace Analyzer - is a prototype tool for the analysis of security protocols.
- PEPA Workbench – is a tool for performance analysis, mainly suited for protocols.
- Mobility Workbench - tool for manipulating and analyzing mobile concurrent systems described in the pi-calculus, especially for proving mathematical properties.
- Concurrency Workbench – it offers more or less the same functionalities as the previous one but refers to CCS.

Tools related to π-calculus

- Stochastic Pi Machine (SPiM) - a simulator for the stochastic pi-calculus that can be used to simulate models of Biological systems.
- BioSPI – similar to SpiM.
- pi4SOA – open source project about web services orchestration.

Comments on tools

- There is no tools that is able to detect emergent properties.
- There are valuable tools for simulation purpose.
- Some tools are able to perform model checking, i.e. verifying system properties expressed in temporal logic statements.
- All of them use different “dialects” of pi-calculus, and sometimes there is no 1-1 mapping between them!
- Most of them still in the preliminary stage: command line interface, no debugging capabilities, a lot of them disappeared because they are no longer supported!

Outline

- What we can easily model in π-calculus
  - examples
  - desirable features
- Tools related to π-calculus
  - existing tools
  - desirable features
- Conclusions

Conclusions

- There's still a lot of work to do :)
  - the language is very basic and is not providing all the basic concepts for modeling self-organising systems
  - the tools are advanced for what it concerns math properties, but not as much advance for system analysis
  - tools do not provide enough support to the modeler / programmer

New Year's Resolutions

- Model a system where locality affect the way of agent's interaction using the “pattern approach”.
- Try to evaluate several possible extension for adding better locality support.
- Build my own prototype of Stochastic Distributed Pi Machine, at first for simulation purpose and later for more deep system analysis -- model checking and the like.