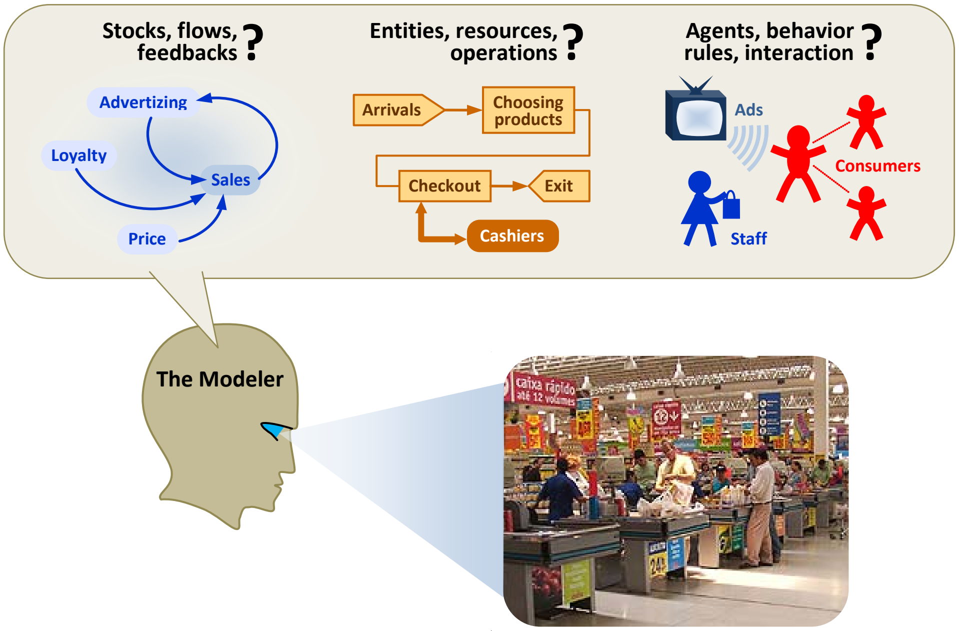
The three methods in simulation modeling

By *method* in simulation modeling, we mean a general framework for mapping a real world system to its model. A method suggests a type of language, or "terms and conditions" for model building. To date, there exist three methods:

- System Dynamics
- Discrete Event Modeling
- Agent Based Modeling

The choice of method should be based on the system being modeled and the purpose of the modeling – though often it is most heavily influenced by the background or available tool set of the modeler. Consider the Figure where the modeler is deciding how best to build a model of a supermarket. Depending on the problem, he may: put together a process flowchart where customers are entities and employees are resources; an agent based model, where consumers are agents affected by ads, communication, and interaction with agents-employees; or a feedback structure, where sales are in the loop with ads, quality of service, pricing, customer loyalty, and other factors.



The modeler chooses a modeling method

Sometimes, different parts of the system are best (meaning most naturally or elegantly) modeled using different methods. In this case, a multi-method model is built.

System dynamics

System dynamics is a method created in the mid-1950s by MIT Professor Jay Forrester, whose original background was in science and engineering. Forrester's idea was to use the laws of physics, in particular the laws of electrical circuits, to describe and investigate the dynamics of economic and, later on, social systems. The principles and the modeling language of system dynamics were formed in the 1950s and early 1960s, and remain unchanged today. Most of the definitions below are taken from www.systemdynamics.org, Wikipedia, and the book *Business Dynamics* by John Sterman.

- **System dynamics is a method of studying dynamic systems. It suggests that you should:**
  - **Take an endogenous point of view. Model the system as a causally closed structure that itself defines its behavior.**
  - **Discover the feedback loops (circular causality) in the system. Feedback loops are the heart of system dynamics.**
  - **Identify stocks (accumulations) and the flows that affect them. Stocks are the memory of the system, and sources of disequilibrium.**
  - **See things from a certain perspective. Consider individual events and decisions as "surface phenomena that ride on an underlying tide of system structure and behavior." Take a continuous view where events and decisions are blurred.**

To feel the spirit of system dynamics (especially in the context of comparing different modeling methods), consider a shop with a counterman serving the shop's clients. The more people that come to the shop per hour, the longer the queue grows. You can build a discrete event model that will give you the length of the queue as a function of the clients' arrival rate and the service time. However, in a real shop, as the queue grows longer, some clients may decide not to join the queue, and instead leave the shop. Others may decide to leave the queue after having waited longer than they expected to. In other words, the length of the queue feeds back to inhibit the rate of queue growth. The results of the "straightforward" model (sometimes called open-loop models in the system dynamics community), will not be valid unless it addresses these circular causal dependencies. One of the key advantages of the system dynamics approach is to readily and elegantly identify such feedback loops and include them into the model.

This section should not be considered an introduction to system dynamics. This is just a glimpse into a complex area of study and we will provide just one possible scenario of model building to illustrate the methodology. The recommended reading is John Sterman's *Business Dynamics* , one of the best books on modeling in general ever written.

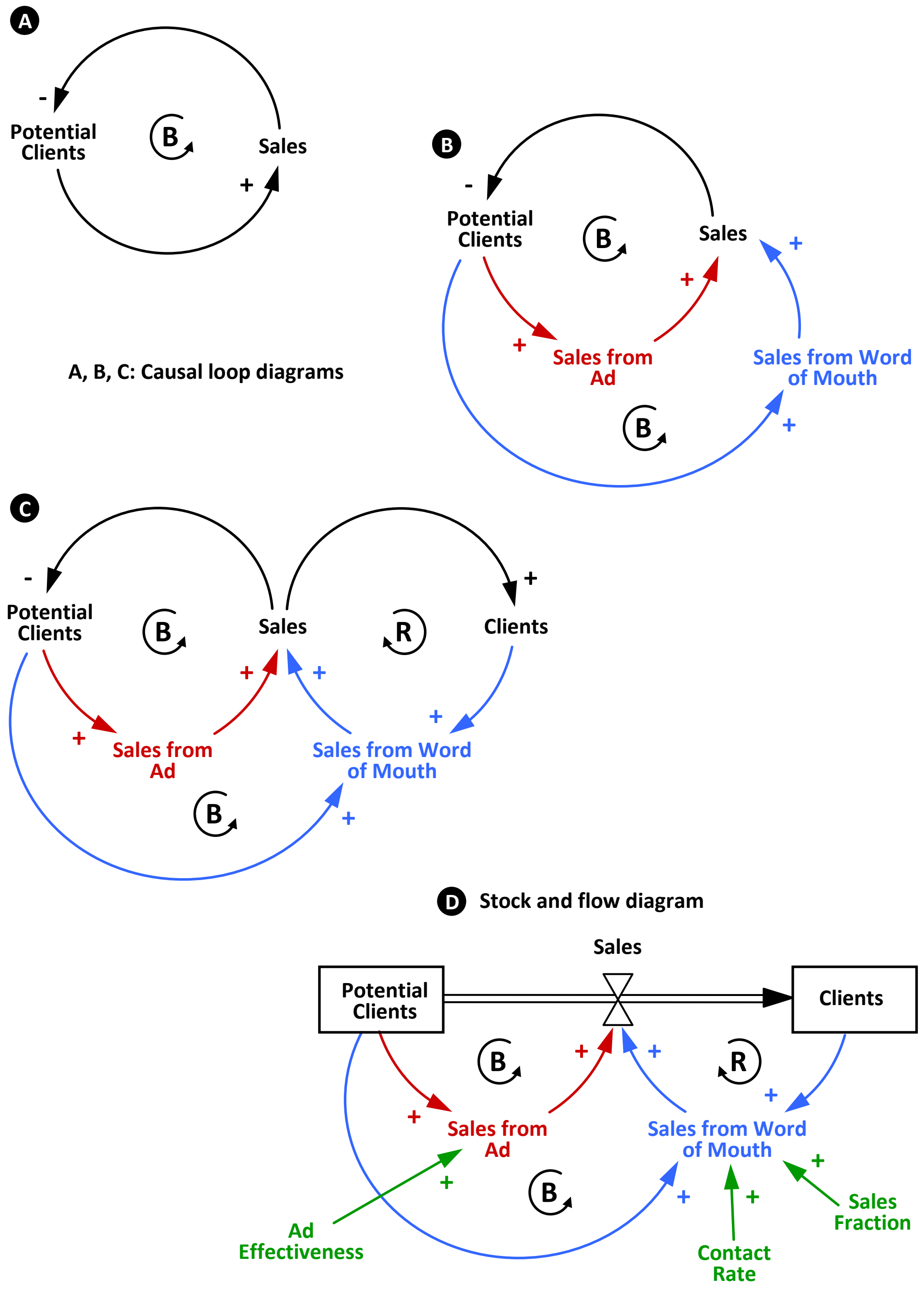
Example: New product diffusion

Consider a company that starts selling a new consumer product. The addressable market has a known size, which does not change over time. Consumers are sensitive to both advertising and word-of-mouth. The product has an unlimited lifetime and does not need replacement or repeated purchases. A consumer needs only one product. We are to forecast the sales dynamics.

We will start with identifying the key variables in our model, and will iteratively draw *causal loop diagrams*. In a causal loop diagram, variables are connected by arrows showing the causal influences among them, with important feedback loops explicitly identified. Causal loop diagrams are good for quickly capturing your hypotheses about the causes of the dynamics in the system, and communicating it to others [Sterman].

In our system, one of the variables is obviously **Sales** – the number of people who bought our product per time unit, e.g. per week. The number of **Potential Clients** will be the other variable. The bigger the market, the greater the sales; therefore, we can draw a causal dependency from **PotentialClients** to **Sales** with positive polarity (see Figure A). On the other hand, as potential clients buy the product, they stop being potential clients, so there is another influence from **Sales** back to **PotentialClients**, this time with negative polarity. The feedback loop we have just created is a *negative*, or *balancing feedback,* loop: it works for reaching a certain goal. In our case, we ultimately will sell the product to all potential clients, and both variables will become zero.

What determines the sales rate? According to our assumptions, consumers are sensitive to ads and to what other consumers say. So, we will distinguish between sales from advertising, and sales from word of mouth. We introduce two new variables and create two balancing loops instead of one, see Figure B. The **SalesFromWordofMouth** depend on the number of (hopefully happy) owners of our product – our **Clients**. The number of clients grows with **Sales**. We draw another feedback loop, this time *positive, or reinforcing* (see Figure C).



System dynamics model of a new product diffusion

While the causal loop diagram we have drawn shows variable interdependencies and feedbacks, it misses the clear mathematical interpretation, and therefore cannot be simulated directly. One of the things that we need to do on our way to the mathematical model is to identify *stocks and flows* among the variables in our system. Stocks are accumulations, and characterize the state of the system. Flows are the rates at which these system states change. Units of measure can help identify stocks and flows. Stocks are usually quantities such as people, inventory, money, and knowledge. Flows are measured in the same units per time period; for example, clients per month, or dollars per year.

In our model, the stocks are **PotentialClients** and **Clients**, and the flow between them is **Sales**. We can now draw a *stock and flow diagram* and write equations for our model. The diagram is shown in Figure D. The equations behind that diagram are:

$$\frac{d(PotentialClients)}{dt} = -Sales$$

$$\frac{d(Clients)}{dt} = Sales$$

$$Sales = SalesFromAd + SalesFromWordofMouth$$

The first two equations are *differential equations*. They define how the stock values change over time. For example, the number of **Clients** grows at the **Sales** rate. The third equation tells that the sales rate consists of two sources, and those sources are independent. The equations for those sources, however, are not clear from the causal loop diagram, and we need to make more assumptions in order to define them.

How big are the sales from advertising? Frank Bass, who originally developed this model, assumed that the probability that a potential client will decide to buy as a result of exposure to advertising is constant at each time period. Therefore, at each period, advertising causes a *constant fraction of potential clients* to buy the product. To reflect this in the model, we introduce a parameter, **AdEffectiveness**, which is a fractional rate. The equation for the sales from advertising, therefore, is:

$$SalesFromAd = PotentialClients \times AdEffectiveness$$

The equation for the sales generated by word of mouth is common for any model of diffusion, be it infectious disease, innovation, a new idea, or a new product. We assume that the people in the community come into contact at a certain rate -- say, each person contacts **ContactRate** people per time unit. If a client contacts a potential client, the latter will decide to buy with a certain constant probability, which we will call **SalesFraction**. So, at each time unit, each client generates

$$ContactRate \times \frac{PotentialClients}{PotentialClients + Clients} \times SalesFraction$$

sales. The middle component in this formula is the fraction of potential clients among all people in the community -- that is, the probability that a person being contacted is a potential client. The formula for the sales from word of mouth, then, is the following:

$$SaleFromWordofMouth =$$

$$Clients \times ContactRate \times \frac{PotentialClients}{PotentialClients + Clients} \times SalesFraction$$

Now the mathematical representation of the model is completed, and we are almost ready to run the simulation.

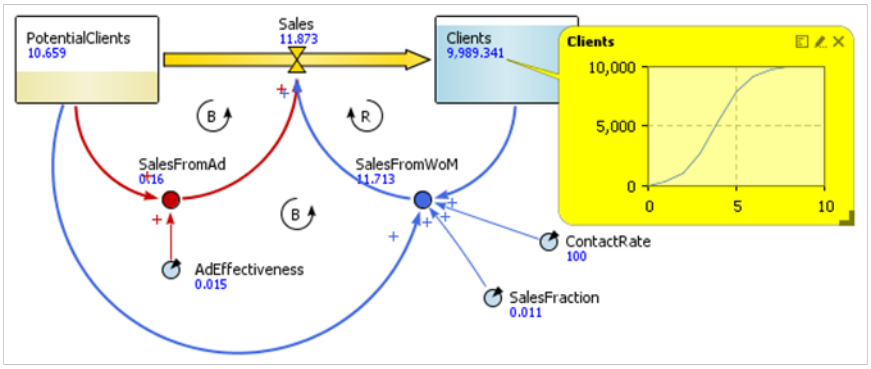
The model-building scenario described above is just one of many possible scenarios. For example, a modeler can start thinking in terms of stocks and flows right away and not use the causal loop diagram at all, or generate it afterwards from the stock and flow diagram. One could start with the positive feedback loop describing the word of mouth effect, and then add the advertising influence. The choice of additional assumptions and the corresponding parameters could also be different.

Before running the simulation, however, we need to specify the values of the parameters and the initial values of the stocks. While the size of the market is roughly known, and the number of clients is known for sure, the three parameters that we have introduced in the last phase of model building cannot be measured directly in the real system. One of the ways to determine the values of such parameters is through *model calibration*. Calibration assumes you have reference (historical) data, with which you can compare and fit the simulation output by varying the model parameters.

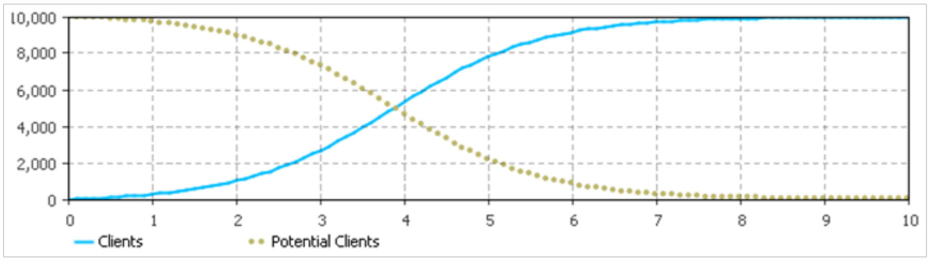
The Figure shows the simulation results for the following parameter values:

- The size of the market is 10,000 people
- At each time unit, 1.5% of potential clients buy the product because of advertising
- Each person contacts 100 other people per time unit
- If a client contacts a potential client, the latter will buy the product with the probability of 1.1%.

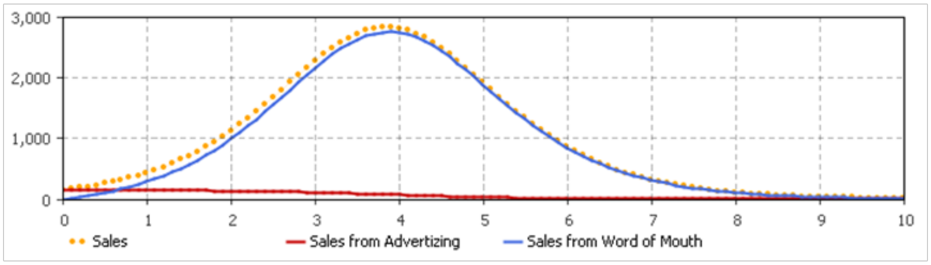
AnyLogic system dynamics model at runtime



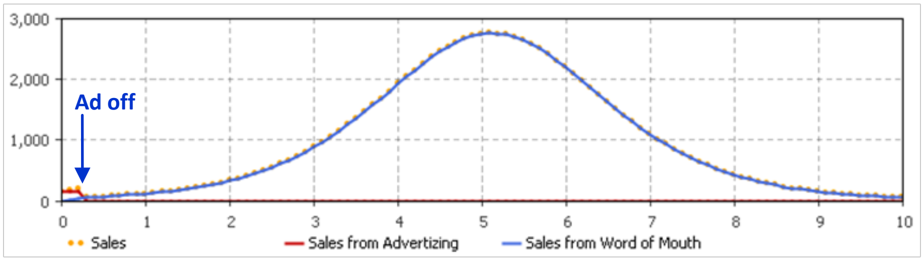
S-shaped curve of Client base (= cumulative Sales)



Bell-shaped curve of Sales and its components



Sales curve if avertizing is stopped shortly after the product launch



Output of the new product diffusion model

The curve of the client base over time is S-shaped. This is a result of the interaction of the two feedback loops. In the beginning, while the size of the market is still large, the reinforcing word-of-mouth effect causes the exponential growth. Then, as the market gets closer to saturation (meaning the system approaches its limit growth limit), the balancing feedback loop starts to dominate and brings the system to the equilibrium state: everybody who could buy the product has now bought it, and sales are at zero .

The sales curve (sales is derivative of the client base) is bell-shaped. You can compare two scenarios; in the one at the very bottom of the Figure, the advertising is turned off shortly after the launch of the product, which has a surprisingly small effect on sales.

Underlying mathematics and simulation engine

Mathematically, a system dynamics model is a system of coupled, nonlinear, first-order differential equations [Wikipedia]

$$\frac{d(X)}{dt} = F(X, P)$$

where X is a vector of stocks, P is a set of parameters, and F is a nonlinear vector-valued function. Simulation of system dynamics models is done with *numerical methods* that partition simulated time into discrete intervals of length *dt* and step the system through time one *dt* at a time.

While numerical methods may be very sophisticated in the modeling tools used by natural scientists and engineers, in particular using adaptive variable time step, the numerical methods used in system dynamics are simple, fixed-step methods: Euler and Runge-Kutta. In addition to differential equations, the simulation engine must be able to solve algebraic equations that appear in the models with *algebraic loops*.

Unlike discrete event and agent-based models, system dynamics models are deterministic, unless stochastic elements are explicitly inserted into them.

Abstraction level

System dynamics suggests a very high abstraction level, and is positioned as a strategic modeling methodology. In the models of social dynamics, epidemics, or consumer choice, individual people never appear as well as individual product items, jobs, or houses – they are aggregated into stocks (compartments) and sometimes segmented into gender, education, income level, etc. (You have probably noticed non-integer values such as 10.659 people or 0.24 cars in system dynamics models during runtime.) Similarly, individual events like a purchase decision, leaving a job, or recovery from a disease, are not considered – they are aggregated in flows.

Although the language of system dynamics is very simple, if not primitive, compared to other methods, thinking in its terms and on its level of abstraction is a real art. System dynamics models are inevitably full of notions that do not have direct material or measurable equivalents in the real world -- for example, morale, awareness, knowledge, and the impact of advertising. The choice of those notions, and drawing the corresponding feedback structures, is not, in many cases, as straightforward a task as the design of a process flowchart or agent behavior.

Software tools

System dynamics modeling is supported by four software tools: Vensim™, AnyLogic™, iThink™/STELLA™, and Powersim™. The modeling language of system dynamics is well-defined and minimalistic -- there are no application-specific dialects. Model conversion is possible between most pairs of tools, and vendors are discussing using a standard XML-based language to be supported by everyone. The tools differ in details. For example, iThink offers special types of stocks like Queue or Oven, which are not present in other tools; Vensim and AnyLogic offer more powerful and flexible arrays; etc.

Discrete event modeling

Discrete event modeling is almost as old as system dynamics. In October 1961, IBM engineer Geoffrey Gordon introduced the first version of GPSS (General Purpose Simulation System, originally Gordon's Programmable Simulation System), which is considered to be the first method of software implementation of discrete event modeling. These days, discrete event modeling is supported by a large number of software tools, including modern versions of GPSS itself.

- **The idea of discrete event modeling method is this: the modeler considers the system being modeled as a process, i.e. a sequence of operations being performed across entities.**

The operations include delays, service by various resources, choosing the process branch, splitting, combining, and some others. As long as entities compete for resources and can be delayed, queues are present in virtually any discrete event model. The model is specified graphically as a process flowchart, where blocks represent operations (there are textual languages as well, but they are in the minority). The flowchart usually begins with "source" blocks that generate entities and inject them into the process, and ends with "sink" blocks that remove entities from the model. This type of diagram is familiar to the business world as a process diagram and is ubiquitous in describing their process steps. This familiarity is one of the reasons why discrete event modeling has been the most successful method in penetrating the business community.

Entities (originally in GPSS they were called *transactions*) may represent clients, patients, phone calls, documents (physical and electronic), parts, products, pallets, computer transactions, vehicles, tasks, projects, and ideas. *Resources* represent various staff, doctors, operators, workers, servers, CPUs, computer memory, equipment, and transport.

Service times, as well as entity arrival times, are usually stochastic, drawn from a probability distribution. Therefore, discrete event models are stochastic themselves. This means that a model must be run for a certain time, and/or needs a certain number of replications, before it produces a meaningful output.

The typical output expected from a discrete event model is:

- Utilization of resources
- Time spent in the system or its part by an entity
- Waiting times
- Queue lengths
- System throughput
- Bottlenecks
- Cost of the entity processing and its structure

Example: Bank

Consider a bank with an ATM inside. The process in the bank is described as follows:

- On average, 45 clients per hour enter the bank.
- Having entered the bank, half of the clients go to the ATM, and the other half go straight to the cashiers.
- Usage of the ATM has a minimum duration of 1 minute, a maximum of 4 minutes, and a most-likely duration of 2 minutes.
- Service with a cashier takes a minimum of 3 minutes and a maximum of 20 minutes, with a most-likely duration of 5 minutes.
- After using the ATM, 30% of the clients go to the cashiers. The others exit the bank.
- There are 5 cashiers in the bank, and there is a single shared queue for all the cashiers.
- After being served by a cashier, clients exit the bank.

We need to find out the:

- Utilization of cashiers

- Average queue lengths, both to the ATM and to the cashiers, and the
- Distribution of time spent by a customer in the bank.

With this problem definition, building a discrete event model is more or less a straightforward task. Clients obviously are entities, and the cashiers are resources. The flowchart of the bank is show in the Figure. The block **ClientsArrive** generates clients at the rate 0.75 per minute (45 per hour). Having appeared in the model, 50% of the clients go to the cashiers, and 50% to the ATM. The usage of the ATM is modeled by a **Delay** block **ServiceAtATM**, preceded by a **Queue** block. Service at cashiers is modeled by a pair **Service** with triangularly distributed service time and **ResourcePool** **Cashiers** with capacity 5. The flowchart ends with the **Sink** block **CleintsLeave**.

Despite the example above is not always easy to identify what are entities and what are resources, especially in systems where there is no obvious continuous process flow (many manufacturing systems fall into this class). Sometimes you need to introduce a "virtual" entity to establish causality and the sequence of operation. If this is so, it may make sense to consider the agent-based modeling method and represent the system as a set of interacting active units. This is discussed later in this chapter.

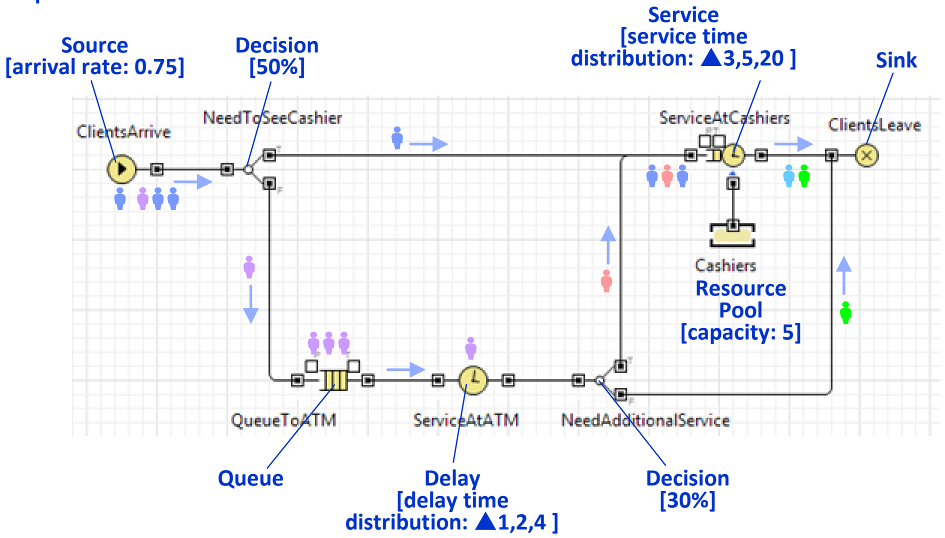
The output data is generated as the model is running. Statistics are collected at the blocks, as well as by the entities as they move through the process flowchart. For example, the cashiers' utilization is a part of the standard report offered by the **Resource Pool** object. Each entity (client) measures time spent in the bank by making a timestamp at the entry and then comparing it with the current time at the exit.

Abstraction level

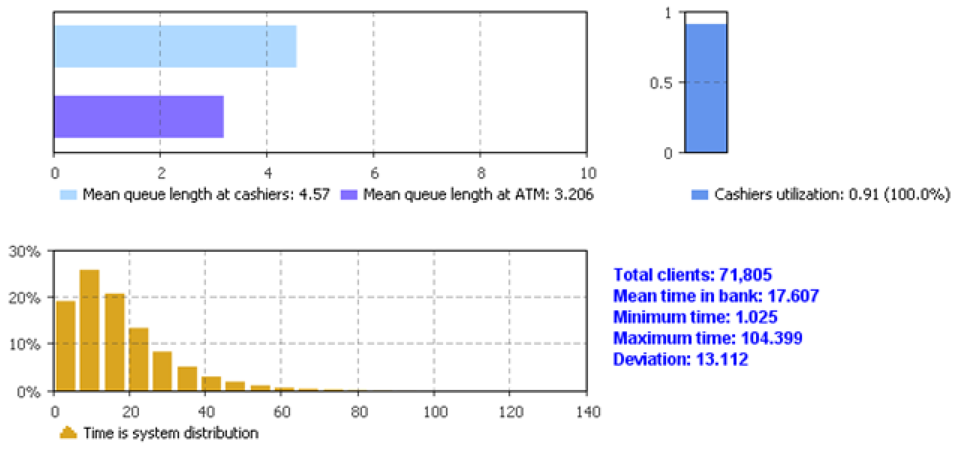
As you can see, the level of abstraction suggested by discrete event modeling is significantly lower than that of system dynamics; the diagram mirrors sequential steps that happen in the physical system. While in system dynamics we aggregate individual objects and talk about the dynamics of their quantities, in discrete event modeling each object in the system is represented by an entity or a resource unit, and keeps its individuality. Entities and resources may have attributes, may differ from each other, and can be treated differently by the process (the "perfect mix" assumption of a system dynamics model is dropped). In discrete event models, time delays can be deterministic or stochastic with any probability distribution. In system dynamics, "natural" delays have an exponential distribution and deterministic delays are special constructs.

Another important fact about process models—and this is the main difference with the agent based models we'll discuss next —is that both entities and resource units are *passive*: they have no behavior "on their own", they just carry their data. Anything that happens to them is defined by the process flowchart.

The processs flowchart



The model output



Discrete event model of a bank

Underlying mathematics and simulation engine

The mathematics behind discrete event simulation are based on *discrete time.* The model clock is advanced only when something significant happens in the model -- namely, when an entity starts or finishes an operation. Any change in the model is associated with those events; continuous changes are approximated by instantaneous ones. Discrete time, and its implementation in the simulation engine, is discussed in more details in the chapter Discrete events and event model object.

Software tools

Unlike system dynamics, discrete event modeling is supported by tens (if not hundreds) of software tools. There is no uniformly accepted language for specifying discrete event models; interoperability is not possible, and not even planned by software vendors. There are some standardized languages for specifying subclasses of processes -- for example, business processes -- but even those languages are not directly supported by simulation modeling tools. Each tool offers its own set of blocks and its own scripting language. Some tools are general-purpose, while some are specifically designed for a particular application area. They may look very different, but behind the GUI each one of them has a simulation engine that supports a discrete clock and event queue, and moves entities through the process flowchart.

Agent based modeling

Agent based modeling is a more recent modeling method than system dynamics or discrete event modeling. Until the early 2000s, agent based modeling was pretty much an academic topic. The adoption of agent based modeling by simulation practitioners started in 2002-2003. It was triggered by:

- Desire to get a deeper insight into systems that are not well-captured by traditional modeling approaches
- Advances in modeling technology coming from computer science, namely object oriented modeling, UML, and statecharts
- Rapid growth of the availability of CPU power and memory (agent based models are more demanding of both, compared to system dynamics and discrete event models)

Agent based modeling suggests to the modeler yet another way of looking at the system.

- **You may not know how the system as a whole behaves, what are the key variables and dependencies between them, or simply don't see that there is a process flow, but you may have some insight into how the objects in the system behave individually. Therefore, you can start building the model from the bottom up by identifying those objects (agents) and defining their behaviors.**
- **Sometimes, you can connect the agents to each other and let them interact; other times, you can put them in an environment, which may have its own**

dynamics. The global behavior of the system then emerges out of many (tens, hundreds, thousands, even millions) concurrent individual behaviors.

There are no standard languages for agent based modeling. The structure of an agent based model is created using graphical editors or scripts, depending on the software. The behavior of agents is specified in many different ways. Frequently, the agent has a notion of state, and its actions and reactions depend on its state. In such cases, behavior is best defined with statecharts. Sometimes, behavior is defined in the form of rules executed upon special events. In many cases, the internal dynamics of the agent can be best captured using system dynamics or discrete event approach. In these cases, we can put a stock and flow diagram or a process flowchart inside an agent. Similarly, outside agents and the dynamics of the environment where they live are often naturally modeled using traditional methods. We find that a large percentage agent based models, therefore, are multi-method models.

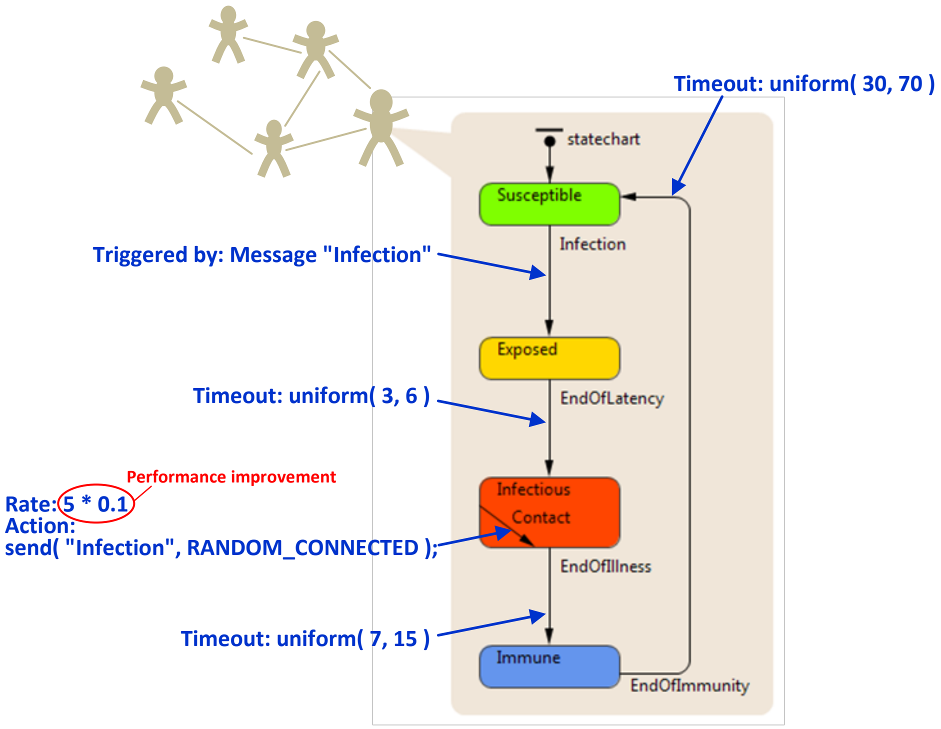
Example: Agent based epidemic model

As an example we will build an agent based model of the spread of contagious disease. Here is the problem statement:

- Consider a population of 10,000 people. They live in an area measuring 10 by 10 kilometers, and are evenly spread throughout the area.
- A person in the area knows everybody who lives within 1 kilometer of him, and does not know anybody else.
- 10 random people are initially infected, and everybody else is susceptible (none are immune).
- If an infectious person contacts a susceptible person, the latter gets infected with probability 0.1.
- Having been infected, a person does not immediately become infectious. There is a latent phase that lasts from 3 to 6 days. We will call people in the latent phase *exposed*.
- The illness duration after the latent phase (i.e. the duration of the infectious phase) is uniformly distributed between 7 and 15 days.
- During the infectious phase, a person on average contacts 5 people he knows per day.
- When the person recovers, he becomes immune to the disease, but not forever. Immunity lasts from 30 to 70 days.

We are to find out the epidemic dynamics -- namely, the number of exposed and infectious people over time.

The terminology and the overall structure of the problem is taken from the compartmental models in epidemiology -- namely, from the SEIR (Susceptible Exposed Infectious Recovered) model. The SEIR problem was originally solved using differential equations; the approach is similar to system dynamics. We, however, are adding details that are not well captured by compartmental (aggregated) models: space, communication dependent on space, and uniformly distributed phase durations. The rationale behind using the agent based approach is its *naturalness*: we may not know how to derive global equations for a particular disease, but we know the course of the disease and can model this easily and in a straightforward manner at the individual level.



Individual behavior of a person in the agent based epidemic model

According to the problem specification, a person can potentially be in one of four phases, and different things happen to him in each of the different phases. This type of behavior is best modeled with a statechart. Look at the Figure. The four states of the statechart correspond to the four phases of the disease. Initially, the person is in the **Susceptible** state, and will remain susceptible unless he gets infected. Infection can only be passed during a contact; it is modeled by a message sent from one agent to another. The transition from **Susceptible** to **Exposed** is triggered by the receipt of the message

"Infection". Once the person gets to the **Exposed** state, the timeout for the next transition **EndOfLatency** is evaluated, and the transition gets scheduled. Thus, the time spent in the **Exposed** state is drawn from the uniform distribution with the range [3, 6] days. Similarly, the time spent in the **Infectious** and **Immune** states is defined by the corresponding stochastic timeouts.

Finally, while the person is in the **Infectious** phase, he periodically contacts other people and can pass infection. This is modeled by an internal transition **Contact** that is executed cyclically at a given rate. When that transition is taken, a message **"Infection"** is sent to a randomly chosen "friend".

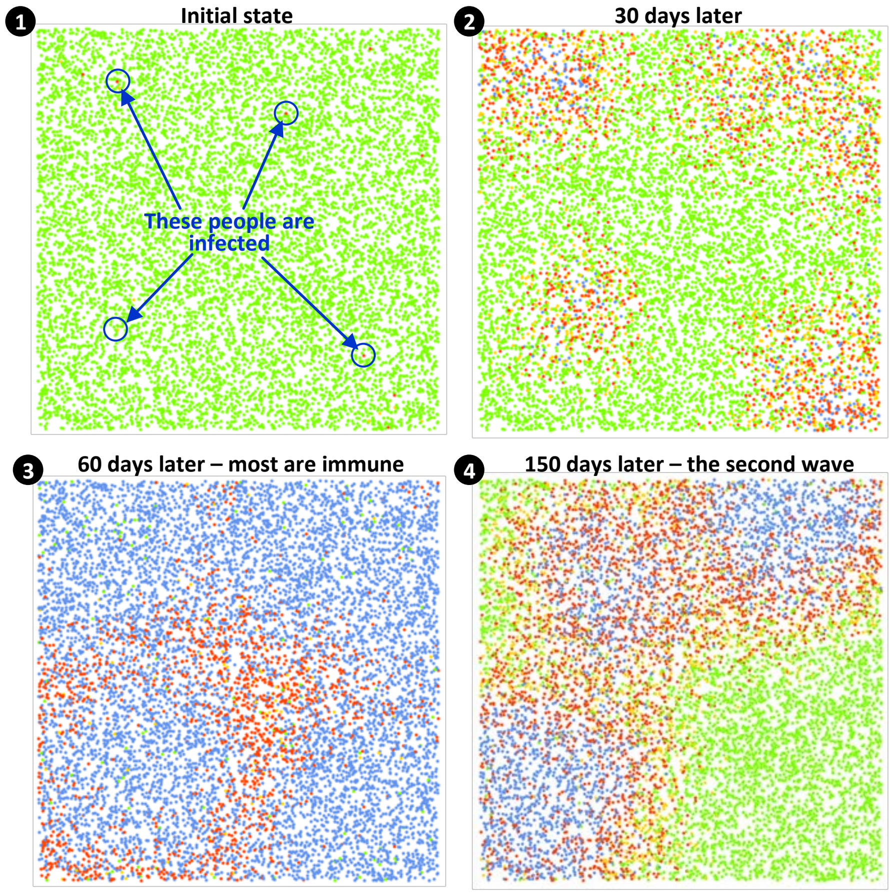
Here we do a simple simulation performance improvement. Instead of explicitly modeling each contact and then deciding whether the other person was infected, we model only "successful" contacts, which are far less common. We multiply the contact rate by the infection probability, so that if the latter is 10%, the contacts will occur 10 times more rarely, but will pass infection for sure.

Now that we have defined the behavior of a person, we need to:

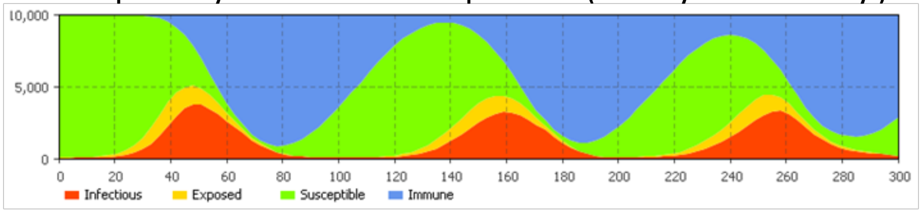
- Evenly populate the 2D space with 10,000 people (in object-oriented terminology, we would say *instances* of class **Person**),
- Establish network connections based on distance (this type of network is supported by most agent based modeling tools)
- Infect 10 random people (e.g. by manually sending them the message **"Infection"** at the beginning of the simulation)
- Run the model.

The dynamics of the model are fascinating to watch. Look at the Figure. Shortly after the model initializes, you can observe the clusters of sick people around the initially infected ones. After the initial peak at about day 50, the epidemic goes down. However, because the immunity of the recovered people does not last very long, they get infected again by those people who are still sick. The second wave of the epidemic starts, and it all repeats again.

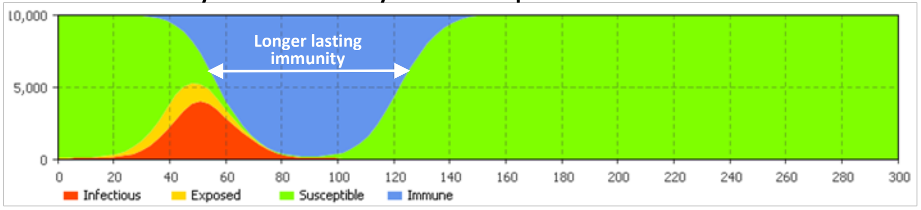
It is exciting to play with the parameters. If we run the same model with a longer and more deterministic immunity phase, say from 60 to 70 days, we will observe only one outbreak of the epidemic and no oscillations (see the bottom chart in the Figure). If we then change the type of network to one with some long distance links, the epidemic will again last forever.


1
Initial state
These people are infected
2
30 days later
3
60 days later – most are immune
4
150 days later – the second wave

The epidemic dynamics with the default parameters (immunity lasts 30 to 70 days)



Here immunity lasts 60 to 70 days – and the epidemic ends after the first wave



Animation and output of the agent based epidemic model

The agent based model we have created is very easy to modify to represent new assumptions. Different types of social networks, households, contacts at work and at home, treatment, impact of vaccination, etc. – all can be captured in the agent based model, naturally and incrementally.

Abstraction level

Agent based modeling does not assume any particular abstraction level. If agents are individuals, then the agent based model is certainly more detailed than a segmented system dynamics model where individuals are aggregated based on characteristics. Agent, however, can be developed with high level of abstraction. For example the agents may be competing projects, companies, or even ideas or philosophies.

Deciding what, exactly, should be modeled as an agent is not always as trivial as in the epidemic example. Even if we are considering people, an individual person should not necessarily become an agent. For example, in the model of an automobile market, agents may be households and not individual people, because the decision about which car to buy is mainly made at the household level and depends on household parameters.

For those who have read books and papers on agent based modeling

Academics are still discussing what kind of properties an object should have to “deserve” to be called an “agent”: pro- and re-activeness, spatial awareness, ability to learn, social ability, “intellect”, etc. In applied agent based modeling, however, you will find all kinds of agents: stupid and smart, communicating with each other and living in total isolation, living in space and without a space, learning and adapting and as well as not changing their behavior patterns at all, and so on.

Here are some useful facts aimed to ensure that you are not misguided by academic literature and various "theories" of agent based modeling:

- **Agents are not the same thing as cellular automata** and they do not have to live in discrete space (like the grid in The Game of Life). In many agent based models, space is not present at all. When space is needed, in most cases it is continuous -- sometimes a geographical map or a facility floor plan.
- **Agent based modeling does not assume clock "ticks"** or "steps" on which agents test conditions and make decisions (*synchronous discrete time*). Most well-built and efficient agent based models are *asynchronous* (conditions are tested and things happen only when they need to). Continuous time dynamics may also be a part of agent or environment behavior.

- **Agents are not necessarily people.** Anything can be an agent: a vehicle, a piece of equipment, a project, an idea, an organization, an investment, etc. For example, a model of a steel converter plant where each machine is modeled as an active object and steel is produced as a result of their interaction is an agent based model.
- **An object that seems to be absolutely passive can be an agent.** For example, even a pipe segment in a water supply network can be modeled as agent. We can associate it with maintenance and replacement schedules, cost, breakdown events, and so on.
- **There can be many, or very few, agents in an agent based model** (compare the model of the American automobile market and the model of the steel converter plant). Agents can be of the same type, or can be all of different types.
- **There are agent based models where agents do not interact at all.** For example, in the area of health economics, there are models of alcohol usage, obesity, chronic diseases where individual dynamics depend only on personal parameters, and, sometimes, on the environment.

Underlying mathematics and simulation engine

Most agent based models work in discrete time -- interaction, decision making, and state changes are instant. In this respect, at the low level the simulation engine should not be much different from the one used for discrete event modeling. At a higher level, it is desirable that the engine supports:

- A large number of concurrent activities, including their dynamic creation and destruction.
- Correct handling of multiple instantaneous events, in particular deterministic and random execution. This is important for synchronous models.
- Networks and communication.
- 2D, 3D, and geographical space, and space-related functionality.

If the agent internal dynamics or environment dynamics have continuous-time elements, such as differential equations, the simulation engine must include numerical methods and support hybrid discrete/continuous time.

Software tools

AnyLogic remains the only professional agent based modeling tool. It supports statecharts and action charts, object oriented-ness and Java, and has the ability to use system dynamics and process flowcharts inside and outside agents to allow the building of industrial strength agent based models. Other software for agent based modeling falls in one of the two classes: graphical tools with limited capabilities for rapid building of toy models, or Java or C++ libraries where you can do more, but have to code a lot.