SCHNAPS: A Generic Population-based Simulator for Public Health Purposes

Audrey Durand
Dép. de génie électrique et de génie informatique
Université Laval, Québec (Québec), Canada G1V 0A6
audrey.durand.2@ulaval.ca

Christian Gagné
Dép. de génie électrique et de génie informatique
Université Laval, Québec (Québec), Canada G1V 0A6
christian.gagne@gel.ulaval.ca

Marc-André Gardner
Dép. de génie électrique et de génie informatique
Université Laval, Québec (Québec), Canada G1V 0A6
marc-andre.gardner.1@ulaval.ca

François Rousseau
Dép. bio. moléculaire, biochimie médicale et pathologie
Université Laval, Québec (Québec), Canada G1V 0A6
francois.rousseau@mac.com

Yves Giguère
Dép. bio. moléculaire, biochimie médicale et pathologie
Université Laval, Québec (Québec), Canada G1V 0A6
yves.giguere@crsfa.ulaval.ca

Daniel Reinharz
Dép. de médecine sociale et préventive
Université Laval, Québec (Québec), Canada G1V 0A6
daniel.reinharz@fmed.ulaval.ca

Abstract

In this paper, we present SCHNAPS, a generic simulator designed for health care modelling and simulations, parametrizable by configuration files and usable by non-programmers such as public health specialists. SCHNAPS is a population-based simulator, using hybrid-state agents to simulate time-driven models. Its software architecture integrates some fundamental object oriented concepts to facilitate further developments and extensions. The proposed approach aims at narrowing the gap between the simulation model and the conceptual modelling made by public health specialists. Current work on osteoporosis, under evaluation by health care specialists, is also presented as a real use case.

1. INTRODUCTION

Given that health care systems are characterized by great variability, the complete set of possibilities for a certain problem in that domain is often too large and complex to be dealt with easily by decision makers. That is why simulation has been used since the 1960s to assist them in assessing and comparing the various options [2]. However, as Eldabi, Paul and Young [6] explained, even if the use of simulation in health care decision-making has greatly increased after the year 2000, the impact of simulations on policy-making or managerial decision-making is weak. They suggested that although there are many interfaces and packages that implement basic methods, there is still a lack of robust and easy-to-use tools that could easily be used by clinicians and hospital managers.

The aim of SCHNAPS (SynCHroNous Agent- and Population-based Simulator) is to provide users with a robust, versatile and convenient simulation framework that will be used, among others, for public health purposes. In order to achieve this, our simulator has been developed jointly with health care specialists and decision makers, and is validated using real decision-making problems. However, the simulator is not targeting only the health care domain: it is designed to be extensible, supporting most types of simulations applied to populations.

Sec. 2. presents the public health application context in which the simulator has been developed. Then, Sec. 3. explains the concepts underlying its characterization as population-based, hybrid-state and time-driven. It also presents the internal modelling of a simulation and its execution process. Sec. 4. describes the object oriented aspects of its internal architecture. Then, the configuration of a simulation is detailed in Sec. 5., including the presentation of the graphical user interface. Finally, a case study is presented in Sec. 6. as an example to support previous concepts.

2. GENERIC SIMULATOR FOR PUBLIC HEALTH

The motivation of our project is to provide a general simulation framework for supporting public health decision makers in the design of screening and prevention strategies. Indeed, there is an interest to provide decision makers with quantitative results comparing expected outcome of different public health policies. Otherwise, decisions might be based on low quality results, opinions, political pressures, and/or pharmaceutical lobbying. Moreover, current models of health service delivery are too complex to be evaluated through analytical methods. Therefore, simulation tools allowing execution of different models of screening and prevention strategies on populations evolving in a specific environment might be of great interest.

Public health is concerned with a great variety of activities, as screening for preventable diseases or conducting vaccination campaigns. Therefore, a flexible simulation engine is required. Given that we cannot anticipate all possible models/uses that would be needed by public health decision makers, we opted for developing a generic simulator software framework, allowing models of any kind – even models not related to public health – with the common denominator that simulations are conducted on populations.

Robinson [15] classified simulation software into three categories:

Spreadsheets are general software such as Excel, that provide some basic computation capabilities for manipulating numbers in 2D grid of cells. Up to some point they can be used by non-programmers, but are of limited use and not convenient for good computing practices such as code reuse and modularization.

Programming languages allow implementation of any simulation models, but require good programming capabilities. Obviously, they support common practices in software engineering for code reuse, abstractions and modularization.

Specialist simulation software allow simple configuration of simulations, often through a visual interface, but are specific to an application domain, and usually difficult to extend or adapt to other fields. With usual commercial software, it is not possible for the user to extend the tool beyond the application interface, as the software internals are controlled by its owner.

The concept of the simulation software we are proposing is a hybrid between those types of software: a generic engine where the simulation models are given through a configuration file using a general format. It is similar to a programming language, as the models are given in an explicit form that works like a computer program. Moreover, it is possible to extend the instruction set with new functions and to create modules of some domain-related instructions. Models are built from such domain-specific building blocks, which is somehow comparable to spreadsheets, where the available instructions are restricted to what has been loaded, such that the user will not be overwhelmed by the possibilities. Finally, the configuration file format is designed to be easily generated and editable through a general graphical user interface, bringing non-programmers to a higher-level view of their simulation models. This is quite close to a specialist simulation software, while having an open architecture that provides as much possibilities as a programming language.

In previous work published by our team [8, 10], simulations were based on programming languages only, with custom applications made for each case studied. The models were provided by public health specialists, in the form of decision trees that were translated into computer programs by computer specialists (in our case, undergraduate and graduate students). It was a very long process for the public health specialists – which are not trained in computer programming – to validate these models. As a consequence, such validation was mostly done empirically, on small set of results, not allowing to detect some errors that may happen in the conversion of the high-level models to code. The concept proposed addresses this by providing a high-level representation of the simulated models, narrowing the gap between the conceptual and the computer model. We therefore expect that any public health specialists will be able to thoroughly validate the simulation models, and even modify or enter models with little or no help from programmers.

3. POPULATION-BASED HYBRID TIME-DRIVEN SIMULATOR

Systems are defined by Cassandras and Lafortune [3] as an interaction of components that must execute a specific function. The dependency of their output on the input history differentiates a static from a dynamic system. They can also be categorized as time-varying or time-invariant depending on whether the output changes on clock ticks when input stays the same. Systems are also defined according to their state, which consists in measurable variables that describe them at a particular moment.

Population-based simulator When each entity in a simulation is explicitly represented and has its own state, made of several descriptive variables, the simulator is said to be agent-based. SCHNAPS uses a set of homogeneous agents/individuals making a simulated population, hence the qualification of our model as *population-based simulations*. The population-based approach is already known in health care evaluation modelling, defined by Cooper, Brailsford and Davies [4] as the use of an initial population of interest subjected to incident cases of study as time progresses. They present it in opposition to the cohort approach, in which a group of individuals with a particular health condition are followed over their lifetime, without having any new individuals entering the simulation. The population-based approach is said to be more adapted to health care evaluation [12] although it is more complex to model [5].

Because the population evolves in conjunction with its environment, which is modelled as an agent, the simulation is said to be multi-agent. A multi-agent population could also be simulated by defining the state of individuals in some way that it could contain all variables that compose all kinds of agents, but use only the ones related to the actual concerned agent type. Nevertheless, the simulator was not planned for further extensions toward pure multi-agent population simulations.

Hybrid-state representation The kind of variables contained in the state of the simulated systems also contributes to classify the simulator. Indeed, a state is said to be discrete if the domain of all variables' values is finite, and it is said to be continuous if their domain is defined by an infinite set. SCHNAPS uses a hybrid-state representation, which can contain a mix of discrete and continuous variables, providing significant flexibility to the simulator in the modelling of individuals. This was required in order to support models of human individuals used in public health simulations, which may contain variables made of different computer types (string, integer, double, etc.), possibly organized into data structures (compositions, lists, etc.).

Time-driven simulations Finally, SCHNAPS is said to be time-driven as the state changes only on clock ticks and as the time is a natural independent variable which appears as the argument of all input, state, and output functions [3]. This is in opposition to event-driven simulations, where the state changes according to asynchronous events, like computer interrupts. Time-driven simulations fit well with the main objective of simulating processing of individuals along a timeline. It is possible to model the strategies according to the most common models in health care simulations: decision trees, Markov models and discrete event simulation [4].

3.1. Simulation Modelling

The simulation framework intends to provide users with tools required to build any model needed by health decision makers. In order to achieve this, the simulator structure relies on several important concepts common in simulation research. Its basic components are illustrated in Fig. 1.

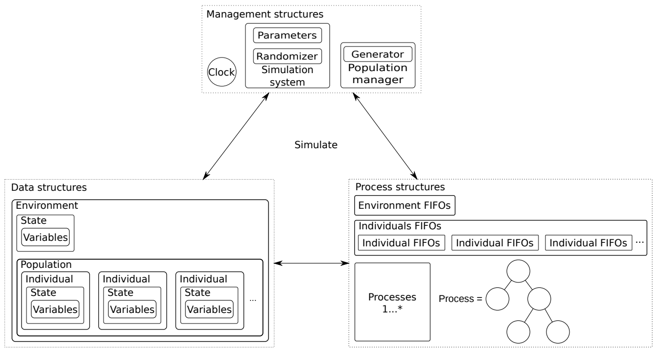

Figure 1. Overview of the simulation framework.

Time representation First, the model contains a clock used to synchronize events. The simulated time, as described by Balci [1], is initialized to zero, which corresponds to any real time value when the simulation begins. It is increased as the simulation advances and can be stopped. In our implementation, a clock tick corresponds to an increment of time t to time $t + \Delta t$. It is then possible to set the size of Δt according to the precision needed. However, as Balci [1] explained, the accuracy and the execution speed of the simulation greatly depend on the selection of an appropriate Δt. If it is too large, some events might occur at the same time while they should not and if it is too small, it will slow down the execution without any gain in the simulation accuracy.

State representation Agents are described by their state, which is constituted of variables. Since the states are hybrid, some of the variables might be discrete while others might be continuous. Our simulator contains two types of agents: individual and environment. Each individual has its own state, comprised of the same variables as the other individuals but with different values, which is characteristic of population-based simulations [4]. The environment has its own state made of specific variables. An alternative view is to consider the environment as the only simulation entity, with a state composed of all states of every individual plus some extra environment-specific variables.

Processes Processes are another important concept which consists in any operation that modifies individual or environment states. It is always represented by primitives, i.e. Boolean, mathematical and logical operations or any other building blocks included in a typical programming environment. Processes can call (for immediate execution) or push (for delayed execution) each other while specifying a target: the environment, all individuals (sequentially) or current individual (default).

A process can also be an observer, which means that it is triggered immediately for execution when the observed value is modified. An observer can watch the clock, so it is triggered on each tick, or it can watch one or many individuals or environment variables. The target of an observer must also be specified: environment, all individuals (sequentially) or current individual (default).

Some processes are defined as scenarios. A scenario is a process that is called at the beginning of the simulation, giving the initial drive by calling other processes for execution, which constitute all together the simulation.

Execution queues In order to manage the execution of all processes, the simulation framework employs several execution FIFO (first-in, first-out) queues per time unit, which allows a process to be triggered with a delay. There is one set of FIFO queues per individual and one set for the environment.

3.2. Simulation Execution

A simulation execution always begins with all FIFOs empty. Scenarios are the first processes executed. They can be assigned a target, like observers or called processes, or they can call or push other targeted processes. This happens on time value zero. After the scenario processing, the clock starts and on each clock tick, each of the clock observers are called for immediate execution. Then, the environment is processed, followed by each individual of the population.

When processing an individual (including the environment), each process in the FIFO for the current clock time is treated until the queue is empty. When a process called for execution targets an individual other than the one processing (for example, targeting all individuals while processing the environment), the current data is stored, the process is executed on the new target, then the data is restored and the execution continues. Algorithm 1 presents the pseudo-code for a step-by-step execution of a simulation.

Algorithm 1 Simulation step-by-step

```
execute scenario
for each clock tick do
    execute all clock observers
    while current_environment_FIFO not empty do
        current_process = pop current_environment_FIFO
        execute current_process
    end while
    for each individual in population do
        while current_individual_FIFO not empty do
            current_process = pop current_individual_FIFO
            execute current_process
        end while
    end for
end for
```

3.3. Information Flow

Information is exchanged by the different parts of the simulator throughout its execution. Fig. 2 illustrates that general information flow. On clock ticks, each observer is pushed into its target FIFO, which can be the environment or all individuals. This is characteristic of time-driven systems. Then, for each individual, starting with the environment, all processes contained in its current FIFO are executed sequentially until none remains. As stated before, a process has access to the current clock value or any environment variable. It can also read and write values of the current individual processing, causing its observers to be pushed into their target FIFO. Finally, a process can call another one for immediate execution or push the process into a FIFO (the current one or with a delay) of its target.

4. OBJECT-ORIENTED ARCHITECTURE

We wanted SCHNAPS to be reusable and extensible and thus we designed it to take advantage of the object oriented properties of C++. Its foundations were greatly inspired by

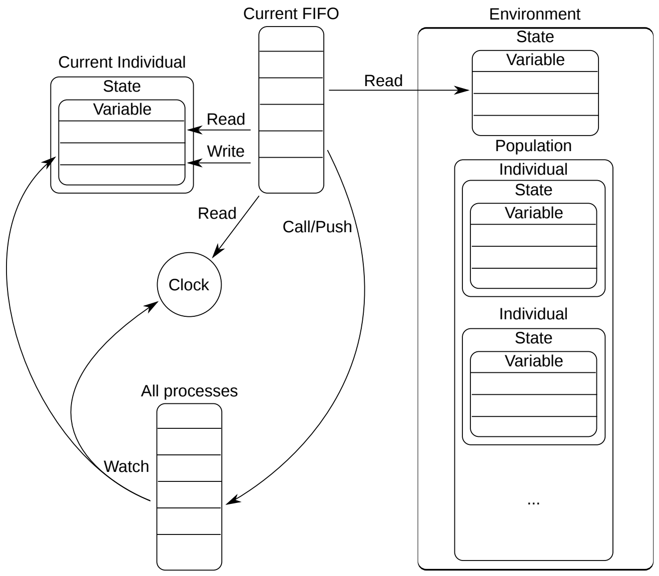

Figure 2. General information flow.

the architecture of Open BEAGLE [7], an open source framework for Evolutionary Computation, also programmed in C++.

First of all, the class hierarchy illustrated in Fig. 3 relies mostly on the inheritance and polymorphism features offered by the language. Everything in the framework inherits from the common root class `Object`. Even the fundamental data types in C++ are redefined as an `Atom`. The subgroup `Number` joins together numerical data types with similar functions and operators. The `Container` corresponds to the definition of vector from the Standard Template Library (STL). It is redefined for each object as a `Bag`, which defines a vector of pointed objects of that type.

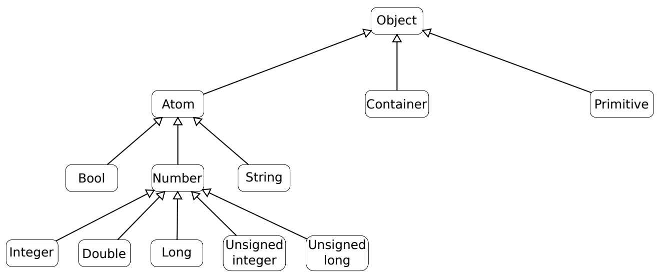

Figure 3. Basic types hierarchy in the simulator.

This class hierarchy brings a coherent set of types to the framework, such as in Java programming language, which should simplify further developments. That goal is also achieved by introducing some memory management concepts. These consist of the use of a reference counter and smart pointers, with a handle type defined for every object.

Through the inheritance mechanism, a handle on an object can reference any other classes inheriting from it, as shown in Fig. 4. This allows emulation of real C++ pointers. Moreover, casting of handles can be managed and verified in the same style as the C++ casting functions. Memory management is ensured by the reference counter, whose function is to keep track of all dynamically allocated objects and their references. It guarantees that these objects will be destroyed when the last handle referring to it is dereferenced. Therefore, when an object is allocated on the heap (using `new`) and assigned to a handle, the user need not be concerned with `delete` since the reference counter is already in charge of it. This is a form of garbage collecting, common in high-level programming languages such as Java or Python. Thus, we can take advantage of C++ execution speed while having access to high-level languages structures.

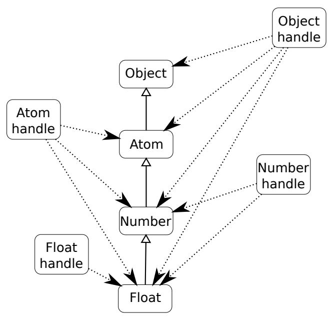

Figure 4. Smart pointers inheritance example.

The object oriented structure also includes the concept of an `Allocator`: a type defined for every classes that can create, clone or copy an instance of that class. It is a key element of the simulator factory mechanism, an entity intended to create objects dynamically without specifying their concrete class. This is an implementation of the Factory [9, 11] design pattern, which is a simplification of the Abstract Factory pattern [9, 11]. Concretely, it consists of a data structure mapping class names to allocators. Therefore, it is possible to access allocators that are able to create objects using only the name of their class. This principle is at the basis of the parametrization by reading configuration files, which is explained in the next section.

5. SIMULATOR CONFIGURATION

In order to properly present the mechanism of configuration by files, some key elements of a simulation must be reviewed. As seen in Sec. 3.1., the simulation is represented by processes modifying individual variables, thus simulating the occurrence of events. A process can observe the clock or any individual or environment variables, which means that when the observed value changes, all its observers are notified and updated automatically, following the Observer design pattern [9]. The simulation execution is based upon that functionality since a scenario is the first process executed at time zero, followed by the triggering of pushed processes at clock ticks. Domino effects happen when an observed variable is modified, leading to the execution of its observers, which may also modify variables and trigger their observers, and so on.

5.1. Process modelling

The concept of configuration using parameter files relies primarily on the organization of information in tree structures. As seen in Fig. 5(a), a tree is made of a root node and all its descendants. A branch represents the relation between a node and its child and a node without any child is called a leaf. Information propagates in a tree from the bottom to the top i.e. from leaves to the root. Any Boolean, mathematical or logical operation can be represented as a tree and executed by going through children sequentially from left to right, computing the arguments of the operation. In such representation, nodes correspond to operators and leaves, to arguments. Fig. 5(b) presents an example of tree modelling of the following operation: $Z*Y+6/(X+1)$.

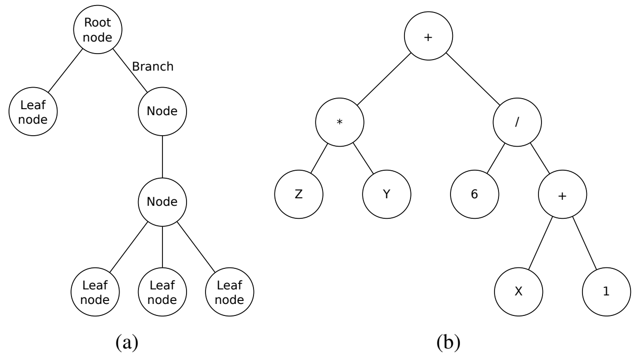

Figure 5. Modelling as trees: (a) general tree architecture; (b) $Z*Y+6/(X+1)$ as a tree.

The simulation framework includes objects of type `Primitive` to implement Boolean, mathematical and logical operations or any specific functions. Tab. 1 presents the fundamental functions, grouped by type of operations. These primitives are designed to be organized in trees, allowing a graphical modelling of any combination of operations, up to the representation of complex computer routines or programs.

5.2. Plugins

The set of available primitives can be extended to provide users with primitives more specific to different domains using the `Plugin` object. A plugin is an external dynamic loading library (*shared object* on Unixes or *DLL* on Microsoft Windows) containing classes that define new primitives. A library is not included in the main program, it is developed

Table 1. Examples of some basic primitives available in the simulator.

Operation type	Primitive
Boolean	OR
	AND
Mathematical	Add
	Subtract
	Multiply
	Divide
	Power
	Modulo
	Absolute value
	Is less (<)
	Is equal (=)
Logical	If then else

outside and the program calls for it when needed. There is one main framework, the same for every user, but each user can add their own libraries to it in order to use homemade primitives. As an example, a screening library was developed to provide health care decision makers with pre-built primitives to compute screening tests, treatments and events. The concept of plugins supports the modular and extensible characteristics of the framework and tends toward the Black-Box pattern typical of the evolution of a software framework, as described by Roberts and Johnson [14]. Once a plugin is developed and tested, its components (primitives) can be reused without worrying about how they work, only knowing what they do: they become black-boxed.

5.3. XML Configuration Files

Because of its natural tree-based organization, the XML (eXtensible Markup Language) [13] language is ideal for modelling our processes, represented by primitive trees. Therefore, it has been chosen as the language for the simulator configuration files. Every objects in the framework implement methods for reading and writing themselves to XML, using the factory to instantiate components from their name. To standardize the parameters file, a XSD (XML Schema Definition) [16] is used to indicate the proper form of the configuration file. Each plugin must also provide its own XSD to specify how its primitives might be used. An advantage of proceeding in this way is that mastery of a programming language is not required to design a simulation.

5.4. Graphical User Interface

However, since the modelling of a very complex simulation can produce heavy XML files that are difficult to understand, a graphical user interface was developed to facilitate the capture of the simulation parameters. It is a generic interface that uses the XSD files of the simulator to configure itself. Thus it can be used for any possible set of primitives and can be extended to support any plugin by reading their XSD. Process modelling is done through visual programming, with each configuration aspect developed in a tab of the interface. Information is presented to users in a coherent manner, avoiding the use of XML tags, providing a wizard to facilitate the task by modularizing the configuration process, and supporting users with fully graphical tools (e.g. entering decision trees by drag-and-drop).

6. CASE STUDY: OSTEOPOROSIS

This section presents a case study to show the capabilities of the simulator for providing public health decision makers with crucial information in a specific field. The simulation process involves several phases represented in Fig. 6. First, the real-world problem corresponds to the need for simulation. The conceptual model is a representation of the real world problem, made by domain specialists and usually represented as a tree. Then, this model is transformed so it can be processed by the simulator becoming the computer model. Finally, the simulation is run and analyzed, usually by the same people that built the conceptual model so they can validate that the execution was representative of the initial model.

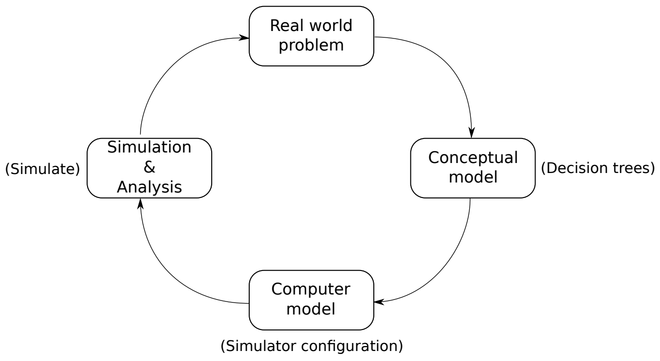

Figure 6. The simulation process: from a real world problem to its simulation. [15]

6.1. Real-World Problem Presentation

The problem considered here is about the possible benefits of deploying prevention strategies for osteoporosis. Osteoporosis is a common disease that affects mainly post-menopausal women, reducing their bone mineral density (BMD) thus increasing their risk of fracture. Women can prevent its occurrence by adapting their lifestyle and through some medication, but up to now, there has been a lack of information regarding which option is the most cost/effective, hence should be considered for a general implementation into the population. Decision makers want to simulate the evolution of a population of post-menopausal women and the occurrence of fractures according to different prevention strategies in order to compare them together and with the case of no prevention at all. To simplify the problem, only the most frequent fractures where considered: hip, wrist and vertebral.

6.2. Conceptual Model

The conceptual model includes both the population targeted by the simulation and the processes that compose the scenarios.

Population The population is defined according to information provided by the Institute of Statistics of Quebec, Canada (ISQ) to resemble the real Quebec population in 2008: $2,018,819$ women of 40 years old and up.

Scenarios For the current case, three scenarios are considered: without prevention or using two different approaches. Fig. 7 shows the conceptual model for the no-prevention scenario. The two others are very similar except that they include a node of prevention before the node of fracture. These scenarios represent all events (including clinical interventions) and effects that may happen to an individual, in a year, concerning osteoporosis. They are applied to the lifespan of simulated individuals.

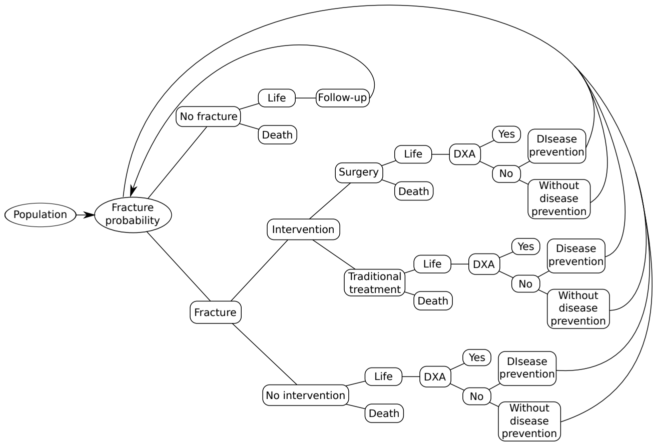

Figure 7. Conceptual model for fracture occurrence and treatment without any prevention strategies.

6.3. Computer Model

Population The state of individuals composing the population is made of variables that reflect important characteristics of post-menopausal women. Tab. 2(a) presents a typical state with several possible variables values.

Because the simulation must stop only when all individuals are dead, there is a variable that keeps track of the number of individuals still alive. It is included in the environment state, which is shown in Tab. 2(b).

Table 2. Individuals and environment state modelling.

(a) Some variables composing the individual state.

Variable	Type	Possible Values
Age	Integer	$\{40,41,\ldots\}$
Death	Boolean	True/False
Fracture probability	Double	$[0,1]$
Death rate	Double	$[0,1]$
Medical costs	Double	$[0,\infty)$

(b) Variables composing the environment state.

Variable	Type	Possible values
Individuals alive	Integer	$\{0,1,\ldots,\text{Pop. size}\}$

Scenarios Scenarios are a selection of recurrent processes. Here, they are defined separately to facilitate the comprehension of each tree. Fig. 8 shows the computer model for the scenario without prevention and some of its referenced processes detailing simulated operations.

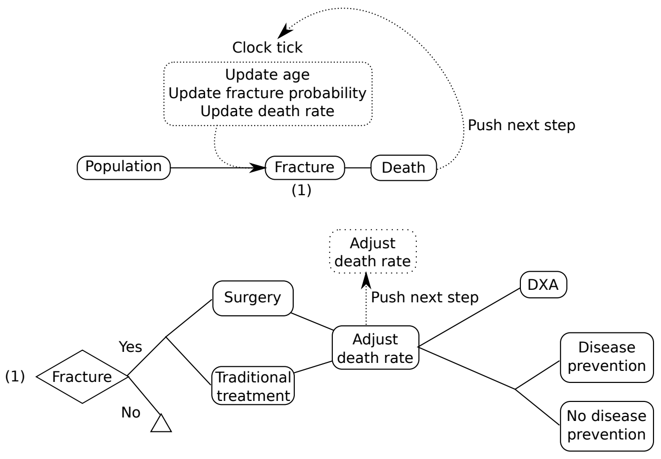

Figure 8. Computer model for fracture occurrence and treatment without any prevention strategies.

6.4. Simulation and Analysis

The simulation produces a text file containing one individual per line, with all its variables separated by a tabulation. Statistics for analysis can be computed on this output using most database tool. The results of osteoporosis simulations are currently being analyzed and validated by health care specialists in order to help decision makers develop an efficient prevention policy.

7. CONCLUSION

SCHNAPS intends to provide clinical researchers with a generic and convenient simulator, targeting the domain of health care services as first objective. By developing SCHNAPS in close collaboration with health care specialists and policy makers, we ensure that our simulator will be usable by these people. It has been designed to handle the complexity of their problems while trying to keep as generic as possible, since it is meant to be extended to other applications outside the domain of health care. It is currently developed under GNU General Public License and the source code is available on Google Code[1].

Its impact is being shown by its application to osteoporosis prevention modelling, presented as a case study, which was an example taken from current work.

Its particular architecture is designed to integrate, in a near future, optimization modules that will propose better solutions in addition of simulating current issues. This will be achieved by mixing planning in Partially Observable Markov Decision Process (POMDP) with data mining of results.

Acknowledgements

The authors acknowledge financial support from FQRSC (Québec), CIHR Institute of Genetics (Canada), CIHR Institute of Health Services Research (Canada), NSERC (Canada) and APOGEE-Net/CanGèneTest Research and Knowledge Network on Genetic Health Services and Policy[2]. The authors are grateful to Annette Schwerdtfeger and Matthew Walker for proofreading this manuscript.

REFERENCES

[1] Balci, O., 1988, "The implementation of four conceptual frameworks for simulation modeling in high-level languages." In *WSC '88: Proceedings of the 20th conference on Winter simulation*, ACM, New York, NY, USA, 287–295.

[2] Brailsford, S. C., 2007, "Tutorial: Advances and challenges in healthcare simulation modeling." In *WSC '07: Proceedings of the 39th conference on Winter simulation*, IEEE Press, 1436–1448.

[3] Cassandras, C. G. and Lafortune, S., 2006, *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc.

[4] Cooper, K.; Brailsford, S. C.; and Davies, R., 2007, "Choice of modelling technique for evaluating health care interventions." *Journal of the Operational Research Society*, 58(2), 168–176.

[5] Davies, R.; Brailsford, S.; Roderick, P.; Canning, C.; and Crabbe, D., 2000, "Using Simulation Modelling for Evaluating Screening Services for Diabetic Retinopathy." *The Journal of the Operational Research Society*, 51(4), 476 – 484.

[6] Eldabi, T.; Paul, R. J.; and Young, T., 2007, "Simulation modelling in healthcare: reviewing legacies and investigating futures." *Journal of the Operational Research Society*, 58(2), 262–270.

[7] Gagné, C. and Parizeau, M., 2006, "Genericity in Evolutionary Computation software tools: principles and case-study." *International Journal on Artificial Intelligence Tools*, 15:2, 173–194.

[8] Gagné, G.; Reinharz, D.; Laflamme, N.; Adams, P. C.; and Rousseau, F., 2007, "Hereditary hemochromatosis screening: effect of mutation penetrance and prevalence on cost-effectiveness of testing algorithms." *Clinical genetics*, 71(1), 46–58.

[9] Gamma, E.; Helm, R.; Johnson, R. E.; and Vlissides, J., 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[10] Gekas, J.; Gagné, G.; Bujold, E.; Douillard, D.; Forest, J.-C.; Reinharz, D.; and Rousseau, F., 2009, "Comparison of different strategies in prenatal screening for Down's syndrome: cost effectiveness analysis of computer simulation." *BMJ*, 338(feb13 1), b138–b138.

[11] Larman, C., 2004, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, 3rd ed.

[12] Mauskopf, J., 1998, "Prevalence-based economic evaluation." *Value in health : the journal of the International Society for Pharmacoeconomics and Outcomes Research*, 1(4), 251–259.

[13] Ray, E. T., 2003, *Learning XML*. O'Reilly Media, 2nd ed.

[14] Roberts, D. and Johnson, R., 1996, "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks." In Addison-Wesley, ed., *Proceedings of the Third Conference on Pattern Languages and Programming*.

[15] Robinson, S., 2007, *Simulation: The Practice of Model Development and Use*. Wiley.

[16] van Der Vlist, E., 2002, *XML Schema The W3C's Object-Oriented Descriptions for XML*. O'reilly Media, 1st ed.

[1] `http://code.google.com/p/lsdsimulation/`

[2] `http://www.cangenetest.org/`