A Framework for Simulation of Dynamic Systems

Marek TOMŠA, Richard VESELÝ, Oto VOZÁR∗

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovak Republic

Abstract. We present in this paper our implementation of framework that provides development environment for basic as well as an advanced modeling of systems from several branches of the artificial intelligence field of research with the emphasis on population dynamics, computational genetics, multi-agent systems and fuzzy logic. Framework implementation is, technologically, based upon .NET Framework, software development platform created by Microsoft. The project has two fundamental layers - the framework itself and a presentation layer dedicated to graphically depict intricate inner workings of the underlying architecture. Framework itself is, hierarchically, divided into several namespaces, each grouping classes into distinct logical units. Framework implementation features clean object-oriented code and makes heavy use of novel technologies like runtime code compilation.

1 Introduction

The lack of simple and unified modeling environment that would simplify the process of development of the evolutionary algorithms and system dynamics caused us to aim our focus on envisioning and designing such a platform. The purpose of our project was to make implementation of those, usually quite cumbersome problems easier and to fill the gap by creating and offering a framework that would meet the aforementioned criteria.

∗ Supervisor: doc. Ing. Mária Bieliková, PhD., Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU Bratislava
2 Framework overview

Overview of the frameworks fundamental building blocks:

- Core - contains main classes used to simulate dynamic processes. Provides the class World, which encapsulates the simulation loop, WorldSystem acting as a junction between simulated system and the interface. Core namespace also contains many base classes utilizable in the process of dynamic system modeling.

- Core.Genetics – provides classes for genetics and genetic process modeling. DNA class provides mechanism for information inheritance as well as methods for their actual recombination. AllelePair class is an abstraction forming the basic means of genetic feature bearing.

- Core.FuzzyLogic – contains the implementation of the fuzzy sets, which can be used for simulation purposes or one might extend them to suit needs of the specific application based on the framework.

- CoreCompilation – contains classes providing runtime compilation of the .NET assemblies and dynamic activation of types contained within, used primarily during runtime for behavior phenotype generation. Potential uses include runtime entity generation.

3 Genetics

Genetics (from the Greek genno γεννώ - give birth) is the science of genes, heredity, and the variation of organisms. The word genetics was first applied to describe the study of inheritance and the science of variation by English scientist William Bateson in a letter to Adam Sedgewick, dated April 18, 1905.

Virtual simulation of complex genetic processes, which was previously impossible, quickly became viable as we entered age of computers more than one century later.

3.1 Implementation of genetics

- DNA class – implements operator that provides the ability to recombine two different DNAs of parents to form the resulting offspring DNA.

- AllelePair class – class encapsulating two Allele instances, forming pair containing genetic information, also a mechanism to choose prevailing allele which determines phenotype is provided.

- Allele class – abstract class able to hold any type of information (making it inheritable)
- IMutable interface – interface enabling specific feature mutations when implemented in Alleles. DNA implements IMutable interface so it enables to mutate all its IMutable Alleles

4 Evolutionary algorithms

An evolutionary algorithm is a generic term used to indicate any population-based optimization algorithm that uses mechanisms inspired by biological evolution, such as reproduction, mutation and recombination. Candidate solutions to the optimization problem play the role of individuals in a population, and the cost function determines the environment within which the solutions live. Evolution of the population then takes place after the repeated application of the above operators.

4.1 Possible implementations of evolutionary algorithms

Framework provides two possible types of implementation, each giving different approach to obtain desired result set.

a) Exogenous algorithm control – each entity has access to global system queues, which force them to perform actions specified by the system, control flow is centralized (traditional model), Entities act in this case inherently just as data structures.

b) Modeling of behavior of the initial population – this approach includes implementing a mechanism for behavior inheritance utilizing the Behavior class and implementing behavioral Patterns (discussed later).

In this case, simulation flow is decentralized, with each entity able to act autonomously.

Behavior inheritance and recombination supported by the framework makes it possible to model uncommon evolutionary algorithms and forms good basis for modeling complex system dynamics.

5 System dynamics

System dynamics is one approach to modeling the dynamics of population, ecological and economic systems, which usually interact strongly with each other.

5.1 Population dynamics

Population dynamics is the study of marginal and long-term changes in the numbers, individual weights and age composition of individuals in one or several populations, and biological and environmental processes influencing those changes.

Population dynamics is also one of the possible applications of the VirtualWorld framework. Framework enables one to simulate various effects
observable in common population, including but not limited to, inheritance of quantitative and qualitative features, patterns of behavior affecting offspring by getting distinct behavioral patterns from both parents, mutations caused by either internal or external stimuli as well as needs that individual entities necessarily need to satisfy in order to stay alive while struggling between feral instincts and sexual desire, dictated by parental heritage in form of their behavioral pattern mixture. While only few could hope to ever simulate real population precisely, we tried to design complex yet easily comprehensible code model while maintaining high level of abstraction with support for future extension.

6 Framework application - RabbitWorld

We have provided RabbitWorld as one particular implementation of simple closed system. It served the purpose of testing major part of the framework.

Notable simulated entities:

- *Rabbit* – generalization of simple movable organic entity
- *RabbitFood* – class serving as a virtual food for each Rabbit instance, which has to be sought by the individuals in order to sustain their internal life-supporting states
- *RadioactiveCloud* – mutational factor

Monitored genetic features include:

- *RabbitFur* – rabbit fur color, implements IMutable interface
- *RabbitBehavior* – model of rabbit behavior

6.1 Modeling of behavior

Framework allows to model complex behavior of individual entities that is later, during the cross-breeding process, inherited along other genetic features. These steps have to be followed, in order to model behavior:

RabbitBehavior class, representing behavioral phenotype is designed first, closely followed by the design of an abstract RabbitPattern class, representing genotype template of the behavior model.

```java
public abstract class RabbitPattern : Pattern
{
    [ThinkingBehavior]
    public abstract void Think(Rabbit e);
    [IdlingBehavior]
    public abstract void Idle(Rabbit e);
    [PerceptionBehavior]
    public abstract Entity Percieve(Rabbit e);
    [HuntingBehavior]
}```
public abstract void Hunt(Rabbit e, Entity victim);
[MatingBehavior]
public abstract void Mate(Rabbit e, Entity partner);
}

Implementation of individual methods representing species-specific behavior follows. For testing purposes, we have developed two basic behavioral patterns: LazyRabbitPattern and DilligentRabbitPattern.

AllelePairs containing newly implemented behavior patterns have to be created next. BehaviorAllele, which aggregates MethodImplementation class instance that provides atomic behavioral element abstraction, serves this very purpose.

<table>
<thead>
<tr>
<th>matingBehavior : BehaviorAllele</th>
</tr>
</thead>
<tbody>
<tr>
<td>method : MethodImplementation</td>
</tr>
</tbody>
</table>

[MatingBehavior]
public override void Mate(Rabbit e, Entity partner)
{
    Rabbit matingPartner = partner as Rabbit;
    e.Copulate(matingPartner);
}

dominance : enum Dominance

{Recessive, Dominant}

Fig. 1. An example of BehaviorAllele instance bearing Mate method definition from LazyRabbitPatternClass

AllelePair has to be created for each method that represents modeled behavior, which enables every behavioral element to be inherited independently. Resulting AllelePairs are inserted into DNA from which the individuals of the initial population are constructed. During entity instantiation (rabbit, in this case) behavior phenotype construction occurs. Behavior class is a phenotype abstraction, serving in our case as a base for the derived RabbitBehavior class.

RabbitBehavior class constructor takes DNA class instance as a parameter, which is used in the process of AllelePair abstraction, with each containing Alleles responsible for the behavior itself that are used to generate resulting class source code from. This class represents the behavior of the new individual. Source code is dynamically compiled afterwards, using the methods contained within the EntityFactory class. Resulting RabbitBehavior class instance is then assigned to the
instantiated individual in form of an attribute and its methods are called during the Refresh method cycle.

This mechanism provides behavior inheritance while retaining clear object-oriented style. It solves the runtime binding of method calls responsible for the behavior of the modeled entity.

Individual BehaviorAlleles, as well as other AllelePairs in the DNA are recombined in the process of cross-breeding and during the instantiation of the new entity the process of the behavior phenotype initialization is repeated, as described earlier. Alleles may, naturally, remain in the heterozygous form, which results in dominant behavior component manifestation and an individual carrying complementary recessive component.

7 Conclusions

We have modeled simple system which we used to monitor basic population dynamics principles and verified the framework functionality. We have shown unique framework features and presented elementary example to demonstrate principle of the behavior inheritance model functioning.

Related works

G – System - both a framework for simulation as well as a virtual reality itself which tries to simulate world in which evolution of life takes place. Project is still in its early phase. http://www.g-system.at

Simile - a software tool for computer simulation of complex dynamic systems in the earth, environmental and life sciences. Simile uses logic-based declarative modeling technology to represent the interactions in these systems in a structured way. In Simile, models are developed diagrammatically, while our framework uses different approach - C# programming language is used for modeling of objects as well as their behavior. http://www.simulistics.com/documents/Simile.pdf

References