Discrete Event Modeling and Simulation of Large Markov Decision Process : Application to the Leverage Effects in Financial Asset Optimization Processes

Emanuele Barbieri

To cite this version:

Emanuele Barbieri. Discrete Event Modeling and Simulation of Large Markov Decision Process : Application to the Leverage Effects in Financial Asset Optimization Processes. Performance [cs.PF]. Université Pascal Paoli, 2023. English. NNT : 2023CORT0001 . tel-04103673

HAL Id: tel-04103673
https://theses.hal.science/tel-04103673

Submitted on 23 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE DE CORSE-PASCAL PAOLI
ECOLE DOCTORALE ENVIRONNEMENT ET SOCIETE
SPE UMR CNRS 6134

Thèse présentée pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE

Soutenue publiquement par

Emanuele Barbieri

Le 17 mars 2023

Discrete Event Modeling and Simulation of Large Markov Decision Process: Application to the Leverage Effects in Financial Asset Optimization Processes

---

Directeurs :
M. Capocchi Laurent, MCF HDR, Université de Corse
M. Santucci Jean-François, PRU, Université de Corse

Rapporteurs :
Mr Zeigler Bernard, PR Émérite, Université d'Arizona
Mme. Frydman Claudia, PRU, Université de Aix Marseille

Jury :
Mr Zeigler Bernard, PR Émérite, Université d'Arizona
Mme. Frydman Claudia, PRU, Université de Aix Marseille
M. Prévot Frédéric, PR associé HDR, KEDGE Business School
M. Santucci Jean-François, PRU, Université de Corse
M. Capocchi Laurent, PRU, Université de Corse

Declaration

These doctoral studies were conducted under the supervision of Laurent CAPOCCHI Assistant-Professor with the accreditation to direct research (HDR). The work submitted in this thesis is a result of original research carried out by myself, in collaboration with others, while enrolled as a PhD student in the Department "Sciences pour l'Environment" (SPE) laboratory UMR CNRS 6134 at the University of Corsica, Pasquale Paoli. It has not been submitted for any other degree or award.

March 2023

Acknowledgements

I thank my supervisor, Assistant Professor Laurent Capocchi, and my previous supervisor, Professor Jean-François Santucci, for guiding me through this research. Thank you to all the staff and students of the Sciences for the Environment (SPE) laboratory, past and present, for their constant support and friendship. My family gave me unflagging support throughout the preparation of this thesis. Finally, I thank my children, Amedeo, Maria-Vittoria, and Leonardo, for everything. I dedicate this thesis to the people of Corsica who never stop exploring.

Abstract

Markov Decision Process (MDP) models are widely used to model decision-making problems in many research fields. MDPs can be readily designed through modeling and simulation (M&S) using the Discrete Event System Specification formalism (DEVS) due to its modular and hierarchical aspects, which improve the explainability of the models. In particular, the separation between the agent and the environment components involved in the traditional reinforcement learning (RL) algorithm, such as Q-Learning, is clearly formalized to enhance observability and envision the integration of AI components in the decision-making process. Our proposed DEVS model also improves the trust of decision makers by mitigating the risk of delegation to machines in decision-making processes. The main focus of this work is to provide the possibility of designing a Markovian system with a modeling and simulation formalism to optimize a decision-making process with greater explainability through simulation. Furthermore, the work involves an investigation based on financial process management, its specification as an MDP-based RL system, and its M&S with DEVS formalism. The DEVSimPy Python M&S environment is used to implement the Agent-Environment RL system as event-based interactions between an Agent and Environment atomic models stored in a new DEVS-RL library. The research work proposed in this thesis focused on a concrete case of portfolio management of stock market indices. Our DEVS-RL model allows for a leverage effect three times higher than some of the most important naive market indexes in the world over a thirty-year period and may contribute to addressing the modern portfolio theory with a novel approach. The results of the DEVS-RL model are compared in terms of compatibility and combined with popular optimization algorithms such as efficient frontier semivariance and neural network models like LSTM. This combination is used to address a decision-making management policy for complex systems evolving in highly volatile environments in which, the state evolution depends entirely on the occurrence of discrete asynchronous events over time.

Keywords: Discrete-event system, Reinforcement learning, Markov processes, Decision making, Financial management, Artificial Intelligence, Explainability, Supervised learning, Stock market Volatility.

Résumé

Les modèles de processus de décision de Markov (MDP) sont largement utilisés dans de nombreux domaines de recherche pour modéliser les problèmes de prise de décision. Les MDP peuvent être facilement conçus par modélisation et simulation (M&S) à travers le formalisme de spécification de système à événements discrets (DEVS) grâce à ses aspects modulaires et hiérarchiques qui améliorent entre autre l'explicabilité des modèles. En particulier, la séparation entre l'agent et les composants de l'environnement impliqués dans l'algorithme d'apprentissage par renforcement (RL) traditionnel, tel que Q-Learning, est clairement formalisé pour améliorer l'observabilité et envisager l'intégration des composants de l'IA dans le processus de prise de décision. Notre modèle DEVS renforce également la confiance des décideurs en atténuant le risque de délégation aux machines dans les processus de prise de décision.

A cet effet, l'objectif principal de ce travail est de fournir la possibilité de concevoir avec une plus grande explicabilité un système Markovien à l'aide d'un formalisme de M&S pour optimiser, par simulation, un processus de prise de décision. En outre, le travail implique une étude de cas basée sur la gestion des processus financiers, sa spécification en tant que système RL basé sur MDP, et sa M&S avec le formalisme DEVS. L'environnement de M&S DEVSimPy est utilisé pour implémenter le système Agent-Environnement RL en tant que librairie DEVS-RL composée de modèles DEVS intéragissant par événements discrets pour mettre en oeuvre l'apprentissage. Le travail de recherche proposé dans cette thèse porte sur un cas concret de gestion de portefeuille d'indices boursiers. Notre modèle DEVS-RL permet de produire un effet de levier trois fois supérieur à certains des indices de marché naïfs parmi les plus importants au monde sur une période de trente ans et peut contribuer à aborder la théorie moderne du portefeuille avec une approche novatrice.

Les résultats du modèle DEVS-RL sont confrontés en termes de compatibilité et combinés avec les algorithmes d'optimisation les plus populaires tels que Efficient Frontier Semivariance et les modèles basés sur les réseaux de neurones tels que LSTM.

Mots clés : Système à événement discret, Apprentissage par renforcement, Processus de Markov, Prise de décision, Management financier, Intelligence artificiel, Explicabilité, Apprentissage supervisé, Volatilité des indices boursier.

Table of contents

Abstract **vii**

Résumé **ix**

List of figures **xv**

List of tables **xxi**

Nomenclature **xxiv**

1 Introduction **1**

1.1 Research Topics and Objectives 1

1.2 Outline 7

2 State of the Art **9**

2.1 Decision-Making Relevant to Finance 9

2.1.1 Introduction 9

2.1.2 Volatility in a Stock Market 10

2.1.3 Portfolio Management Decision-Making Methods 15

2.1.3.1 Generalized Autoregressive Conditional Heteroscedasticity Process 15

2.1.3.2 Long Short-Term Memory 17

2.1.3.3 Efficient Frontiers Methods 19

2.1.4 Conclusion 24

2.2 Markov Decision Process 26

2.2.1 Introduction 26

2.2.2 Markov Chains 26

2.2.2.1 Markov Reward Processes 29

2.2.3 Markov Decision Processes Formulation 31

2.2.4 Conclusion . . . 33

2.3 Reinforcement Learning . . . 34

2.3.1 Introduction . . . 34

2.3.2 Artificial Intelligence Approaches . . . 34

2.3.2.1 Model-Based and Model-Free Methods . . . 35

2.3.2.2 Temporal-Difference Methods . . . 36

2.3.2.3 Learning Agent Algorithm . . . 37

2.3.3 Bellman Equation . . . 39

2.3.4 Q-Learning Algorithm - The Brain of the Agent . . . 39

2.3.5 Explainable Artificial Intelligence in Reinforcement Learning . . . 43

2.3.6 Conclusion . . . 46

2.4 Discrete-Event System Specification Formalism . . . 47

2.4.1 DEVS Atomic Model . . . 49

2.4.2 DEVS Coupled Model . . . 53

2.4.3 DEVS Simulator . . . 55

2.4.4 DEVSimPy Environment . . . 55

2.4.5 Conclusion . . . 57

2.5 Conclusion . . . 57

3 DEVS-Based Reinforcement Learning System Modeling and Simulation 59

3.1 Introduction . . . 59

3.2 Discrete Event Modeling and Simulation for Machine Learning . . . 61

3.2.1 Discrete Event System Specification and Reinforcement Learning . 63

3.3 DEVS-Based RL Architectural Pattern . . . 65

3.3.0.1 Atomic-based RL Modeling Approach . . . 66

3.3.0.2 Coupled-based RL Modeling Approach . . . 67

3.3.0.3 Multi-Agent-based RL Modeling Approach . . . 69

3.3.1 What About Observability and Explainability? . . . 71

3.3.2 Explicit Time in Q-Learning . . . 73

3.4 DEVSimPy Modeling and Simulation of RL . . . 74

3.4.1 The Pursuit-Evasion Case Study . . . 74

3.5 Conclusion . . . 80

4 Case Study: Leverage Effects in Asset Management Optimization Processes 83

4.1 Introduction . . . 83

4.1.1 First Experiment . . . 86

4.1.1.1 DEVSimPy Modeling . . . 87

4.1.1.2 DEVSimPy Simulation 89
4.1.1.2.1 Single Episode Case 89
4.1.1.2.2 Multiple Episode Case 90
4.1.1.2.3 Discussion 94
4.1.2 Compatibility and Combination 95
4.1.2.1 Efficient Frontier vs. DEVS 95
4.1.2.2 LSTM Model Combinated with DEVS-RL 100
4.1.2.3 ESV, LSTM and DEVS-RL Combination 104
4.2 Conclusion 105

5 Conclusions and Future Works 107

6 List of Publications 111

References 113

7 Agent Atomic DEVS Model Specification 127

8 Environment Atomic DEVS Model Specification 131

9 Efficient Frontier Semi Variance Implementation 137

10 LSTM Implementation 139

List of figures

1.1 An example of a leverage effect based on financial leverage. 2

1.2 In the generic framework, DEVS-RL is a general methodological approach based on the combination of DEVS M&S and RL algorithm. The study case allows us to validate the quality of results in terms of financial gains. Confrontation and combination allow us to highlight the similarity and complementary between DEVS-RL, EF, and LSTM techniques. 6

2.1 LSTM architecture with forget, input, and output gates. **X** is the pointwise multiplication, **+** is pointwise addition, sigmoid activated NN, Tanh (yellow) pointwise Tanh not NN, Tanh (green) activated NN 18

2.2 LSTM model for predicting stock prices [89]. 19

2.3 Actual vs. predicted stock price by an LSTM model (period: Jan. 1, 2021, to June 1, 2021) [163]. 20

2.4 Efficient Frontier Process from historical stock price data to diversified portfolio. 20

2.5 Markowitz Efficient Frontier [88]. 21

2.6 Efficient Frontier example. 22

2.7 Scalper day-life routine Markov chain. 28

2.8 MRP scalper daily routine MRP. 29

2.9 Three types of Machine Learning: supervised/unsupervised/reinforcement learning with their application domain. 34

2.10 Interaction between the Agent and the Environment in RL. The Environment sends rewards and new states in response to actions from its Agent. Adapted from Sutton [178]. 38

2.11 Q-Learning algorithm from [177]. 42

2.12 XAI toxonomy in RL (adapted from [2]). 45

2.13 DEVS experimental frame with the model and the simulator interacting through the modeling and the simulation relationships. 48

2.14 Classic DEVS Atomic Model in action. 50

2.15 Finite-state machine of the EXEC DEVS atomic model. 52

2.16 State trajectory of the EXEC DEVS atomic model after simulation. 53

2.17 An example of a DEVS coupled model. 54

2.18 DEVSimPy general interface. The left panel shows the library panel with all atomic or coupled DEVS models that are instantiated in the right panel to create a diagram of a simulation model. The dialogue window on the bottom-right allows one to simulate the current diagram by assigning a simulation time. 56

3.1 M&S for Machine Learning. Modeling part highlight important aspects related to discrete event M&S that brings benefits as temporal aspect for delayed reward notion in RL for example. Simulation part point out that it can be used to improve the output analysis or to facilitate the Monte Carlo process realization. 62

3.2 Traditional RL workflow and the corresponding phases in DEVS. 64

3.3 Learning by reinforcement with the DEVS Agent and Environment models. The Decision logic is embedded in the agent that reacts to a new state and reward couple by sending an action that will be evaluated by the Environment model. The Environment model can be a discrete event simulation model executed in a specific experimental frame. 65

3.4 Atomic-based modeling approach with the DEVS atomic models of the RL Agent and RL Environment DEVS atomic models. The atomic model Observers can be inserted after the RL Agent to observe the mean of the Q matrix, which can be used to see its convergence. *N* Generator models are used to consider external events in the behavior of the Environment model. . 66

3.5 UML sequence diagram of the User-Environment-Agent interactions in the Q-Learning algorithm framework. The user starts the simulation by invoking the initial (init) phase of the Environment model that sends an event with the state/action map used to initialize the Agent model. The Agent then sends an action depending on its initial state in return that activates the δ_{ext} function of the Environment model that immediately updates the state, the reward, and the done flag (which informs if the end state has been reached) to return to the Agent. The agent then updates its Q matrix according to the action received. This cycle (episode) is repeated until the end state is reached (the *done* flag is true). When the Q matrix is stable, the final policy can be outputted by the Agent model and the Environment model can print the simulation trace before becoming inactive. 68

3.6 Coupled-Based Modeling Approach with the Agent and Environment DEVS Coupled Models which improve the explainability of the AI embedded. Each observer is an atomic model that turns the signal into an interpretable feature. 69

3.7 Multi-agent DEVS reinforcement learning model with a supervisor. Each agent explores a subset of possible states and gives its optimal decision plot based on interactions with its own environment. The supervisor then proceeds to the final decision-making logic. 70

3.8 RL library into the DEVSimPy software. The Properties panel of the Agent and Environment models allows one to configure these models. The *algo* property of the agent allows one to select the learning algorithm between Q-Leaning or SARSA. Properties γ, ε, and α are related to the Bellman equation (see Section 2.3.3). Concerning the Env model, the option *goal_reward* allows us to define the reward as a goal (when $goal\_reward = True$ is checked) or as a penalty (when *goal_reward* is uncheked) (see Section 2.3.2.3) 75

3.9 Mouse-cat M&S into the DEVSimPy framework. 77

3.10 Mean Q-value per episode with fear factor. 78

3.11 Mean Q-value per episode without fear factor. 78

4.1 Approaches driven by humans and AI. This process is theoretically carried out taking into account macroeconomic indicators, technical data, stock trends, market risk indicators, and the current composition of the trader's portfolio. In other words, to address the issue of risk aversion, traders trade a small portion of their portfolio at a regular pace, such as quarterly, monthly, or weekly. The supervised AI agent has a different reward system to reduce human biases and to produce a leverage effect (make more money) with a more sustainable long-term strategy. 84
4.2 DEVS-RL driven proposed approach. 85
4.3 DEVSimPy model that puts into action the DEVS-RL library (Section 3.4) with the three generator atomic models CAC40, DJI, and IXIC. 88
4.4 Mean of the Q matrix in the Agent model for the single-episode simulation case after 35000 episodes. 90
4.5 Stock market indexes from 1991-01-02 to 2018-07-05. 90
4.6 Size of the episode (left part) and number of steps (right part) during the period and ordered by the initial cash scenario (from top to bottom: 0$, 8000$, 16000$, 24000$). 91
4.7 Index multiplicity (IXIC on the left, CAC40 on the middle, and DJI on the right) during the simulation for the four scenarios depending on the initial cash (from top to bottom: 0$, 8000$, 16000$, 24000$). 92
4.8 Total assets (stock + cash) during simulations according to the period 1991-01-02 to 2018-07-05. 92
4.9 Residual cash during simulations for the four scenarios. 93
4.10 Cash investment time (with initial cash from top to bottom: 0$, 8000$, 16000$, 24000$). 94
4.11 Efficient Frontier for the IXIC, DJI, GSPC, and RUT indexes with no short sale in the period from January 1, 2019 to January 1, 2020. The optimal solution (red cross) obtained for an expected annual return equal to 30.0%. . 96
4.12 IXIC, DJI, GSPC, and RUT index values for each day of the period from January 1, 2019 to January 1, 2020. 97
4.13 DEVSimPy simulation model with DEVS-RL agent and environment atomic models and the four DEVS generators (IXIC, DJI, GSPC, and RUT) that send the index values for each day of the period from January 1, 2019 to January 1, 2020. 98
4.14 DEVSimPy properties of the atomic model Agent with a configuration defined for the compatibility experiment. 98

4.15 DEVSimPy properties of the Env atomic model with a configuration defined for the compatibility experiment. 99

4.16 Leverage effect after DEVS-RL simulation from 100,000$ with N equal to 4,8 and 14 from January 1, 2019 to January 1, 2020. The leverage effect for $N = 4$ is 34.72% and the discrete allocation is (3 IXIC, 3 RUT, 3 DJI, 3 GSPC) that corresponds to one of the efficient frontier expected return points. For $N = 4$ at the end of 2019, the current value of the portfolio is 153 188.002 for an initial capital investment of 100,000$. The initial capital represents 65.28% of the current value. The leverage effect for 2019 is 34.72% with a return ratio of 53.18% which represents the gain for $N = 4$ in 2019. 100

4.17 Family of paths obtained after simulations with different values of ε (noted on each arc). Depending on the selected ε, each path traced the evolution of the indexes. If ε is (resp. far from) 1.0, the path is obtained in a minimal (resp. maximal) time due to the exploitation (resp. exploration) policy executed by the agent (ε-greedy algorithm). 101

4.18 LSTM index predictions from September 2012 to November 2022. The blue, green, and orange lines represent, respectively, the real values during the training period, the real values during the prediction period, and the predicted values. Visually, the orange and green lines seem to be superimposed. . . . 102

4.19 LSTM and DEVS-RL combination. LSTM coupled model contains the four generator model (for the four indexes) based on the predicted data extracted from the LSTM algorithm. 103

4.20 The real and LSTM predicted leverage effects after the DEVS-RL simulation from 100,000$ with $N = 4$ from January 1, 2019 to January 1, 2020. 104

4.21 Our combined architecture from input data to optimal portfolio. The prediction model use distinctively or combined LSTM and GARCH to build a portfolio based on risk model volatility. Next, the portfolio optimization is based on DEVS-RL and the results are evaluated by Efficient Semivariance. 105

7.1 The DEVS agent atomic model with its two input ports In_0 and In_1 used to receive the initialization message (for the Q matrix) and the set (s,r,d) from the DEVS environment model. The two output ports Out_0 (resp. Out_1) are used to send the action a (resp. Q matrix) to be evaluated by the environment model. 127

8.1 The DEVS environment atomic model with its two input ports In_0 and In_1 used to receive the action a from the Agent DEVS model and the message $[v]$ from the generators. The two output ports Out_0 (resp. Out_1) are used to send the initial message (resp. (s,r,d)) to the Agent model. 131

List of tables

3.1 Comparison using XAI taxonomy. 72

4.1 LSTM models prediction performance. The RMSE values confirm that our LSTM models compute predictions on a one-year horizon with a high degree of precision given the respective 2019 average price (AP). 102

4.2 LSTM models prediction performance with three different epoch settings. With 1 epoch, the computational time of the model was about 72 seconds, and every next epoch needs 71 seconds more. The results in terms of prediction do not vary significantly. 103

Nomenclature

Acronyms / Abbreviations

ACER	Actor-Critic with Experienced Replay
ACW	Average Actual Weight
API	Application Programming Interface
IXIC	NASDAQ Composite
CNN	Convolutional Neural Networks
CPS	Cyber-Physical Systems
CR	Capital Requirements
DDPG	Deep Deterministic Policy Gradient
DEVS	Discrete Event system Specification
DEVSimPy	DEVS Simulator in Python language
DJI	Dow Jones Industrial Average
DQN	Deep Q-learning
EF	Efficient Frontier
ESV	Efficient Frontier Semi-Variance
GARCH	Generalized Autoregressive Conditional Heteroscedasticity Process
GDPR	General Data Protection Regulation
GSPC	S&P 500

IF	Internal Financing
LSTM	Long Short-Term Memory
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MPT	Modern Portfolio Theory
MRP	Markov Reward Process
MVO	Mean-Variance
ORS	Ordinary Least Squares
OTC	Over-The-Vounter
PDEVS	Parallel Discrete Event system Specification
RMSE	Root Mean Squared
RNN	Recurrent Neural Networks
RUT	Russell 2000
SES	System Entity Structure
TD	Temporal Difference
TM&S	Theory of Modeling and Simulation
VaR	Value-at-Risk
VIX	Chicago Board Options Exchange Volatility Index
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

The core work presented in this thesis deals with the resolution of Markov decision processes (MDP) using a discrete event modeling and simulation (M&S) approach to optimize a decision-making process, such as leverage effects in asset management to improve its understanding. The relationship between MDP and (M&S) plays an important role in the literature in a wide range of fields, i.e. clinical decision-making [17], sequential decision-making under uncertainty [5], aviation [133], job-shop scheduling [205], cybersecurity [206], real-time energy management of photovoltaic-assisted electric vehicles [194] and many more. The main focus of this work will be to provide the possibility of designing a Markovian system/algorithm with a M&S formalism with greater explainability, in order to optimize a decision-making process through simulation. Furthermore, the work involves an investigation based on financial process management, its specification as a MDP-based reinforcement learning (RL) system, and its M&S with the discrete event system specification system (DEVS) formalism. The results of the DEVS model are confronted and combined with the most popular optimization algorithms such as Efficient Frontier Semivariance and RNN models such as LSTM [189]. The research topics, objectives, methodological design, and outline of the document are discussed in this general introduction.

1.1 Research Topics and Objectives

Let us introduce the Research Topic section with a brief explanation of what a financial leverage effect is, as shown in Figure 1.1. Imagine that we invest 100€ of our savings in a first company, that is, we sign corporate bonds for one year. After a year we receive 120€ , 100€ is the reimbursement of our capital invested and 20€ is the dividend or gain (gray bubble in Figure1.1). We may say that we have a gain or a return of 20%. Our invested capital 100 represents 83% of the current capital 120€ and 20€ (the gain) represents 17% of our current capital that what we consider our leverage effect without borrowed capital. To calculate the leverage effect of the investment without borrowed capital, we use the following formula:

$$Leverage \quad effect = 1 - \frac{Initial \quad value}{Current \quad value} * 100$$

Now, imagine that we also invest 100€ in a second company. To buy the second bond, we split the money invested in two parts: 10€ from our savings and 90€ from a loan at the bank. The loan has a 1% interest and we have to reimburse the capital at the end of the year. We receive from this second company 120€ (black bubble in Figure1.1), 90€ as a reimbursement of the capital invested, 10€ as a reimbursement of our savings and 20€ of dividends.We have to pay the bank 90€ plus 0.9€ of interest, and the rest are our gains, that is 19.1€ . If we compare the 10€ invested from our saving to the 19.1€ of gain, we may say that we have a leverage effect with a borrowed capital of 52% from this second investment. This is an example of **leverage effect** that is usually based on **financial leverage**, i.e. the use of borrowed money from banks. To calculate the leverage effect of the investment with borrowed capital, we use the following formula:

$$Leverage \quad effect = 1 - \frac{Initial \quad savings \quad value}{Current \quad gain \quad value} * 100$$

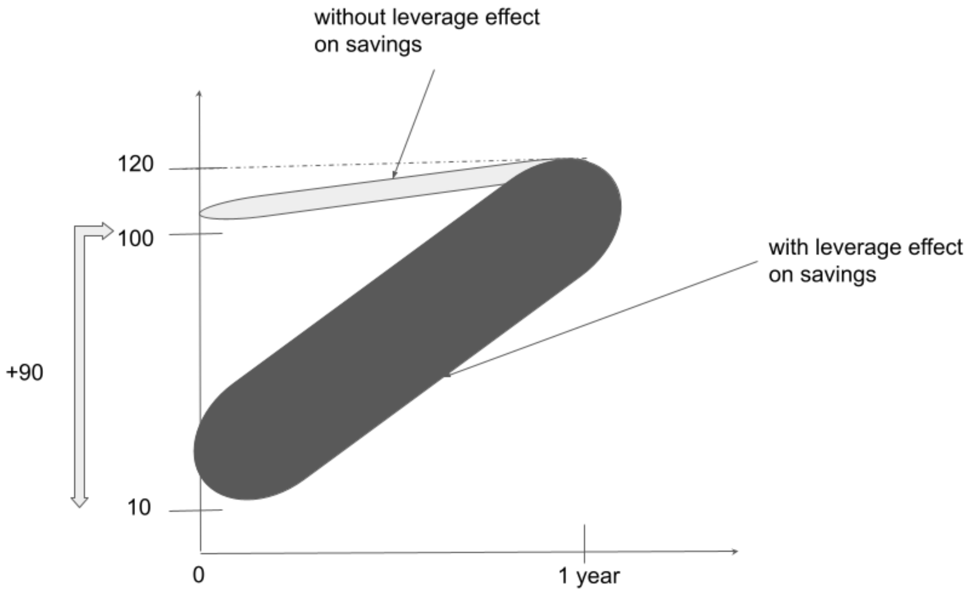


Fig. 1.1 An example of a leverage effect based on financial leverage.

Our asset management MDP decision-making approach would play a role in reducing the risk of budget degradation by offering the opportunity to invest with our tool, stakeholder private party financing [192] during, for example, the pre-trial phase of a EU funds application process that can last 8 to 12 months. The risk of budget degradation is particularly high in Corsica, which is one of the poorest regions of Europe [43]. So, in terms of regional interest, our research effort, focused in the world of finance, specifically on how borrowed capital (backed by company internal finance) can produce a leverage effect on stock markets and offer an alternative solution to reduce the economic effect of budget degradation. Portfolio optimization using AI was a juvenile field in 2017, the year of the beginning of this research. Since 2017, the scientific literature on AI has offered more and more work on portfolio optimization, in particular using efficient frontier (EF) models [39], long-short-term memory (LSTM) [197], and generalized autoregressive conditional heteroskedasticity process (GARCH) [95] models. In particular, EF and LSTM, two of the most explored methods in the literature, will be used in this research to verify the possibility of compatibility and combination with the results of the DEVS model. Our final goal is to offer a novel approach that combines discrete event simulation with artificial intelligence. To achieve this goal, we propose a Markovian approach to create value by taking advantage of the volatility of some of the world's stock market indices. The proposal aims to model and simulate an optimized portfolio policy [113] in terms of complete allocation and indices trends by combining MDP and discrete-event M&S domains.

A **Markov chain** [98] is a stochastic model used to describe a sequence of potential events under a condition where the probability of each event is only limited to the state established in the previous case. In probability theory, a Markov process (named after the Russian mathematician A. Markov) is a stochastic system that satisfies the Markov property, i.e. if one can make predictions for the future of the process based only on its current state. Markov chains are commonly defined as discrete or continuous Markov processes with countable state space (thus, independent of the nature of time), but it is also common to define a Markov chain as having discrete time in a countable or continuous state space. For example, Markov chains have many uses as statistical models of real processes such as speed control systems in automobiles, queues or lines of customers arriving at airports, rates of currency exchange, storage systems such as dams, and population growth of certain animal species. The algorithm known as PageRank [29], which was originally proposed for the Google search engine on the Internet, is based on a Markov process. Markov processes are the basis of general stochastic simulation, which are used to simulate sampling from complex probability distributions and have found wide application in Bayesian statistics as introduced in 2.2.2.

Most leverage-effect financial instruments are future-oriented. **The today leverage effect is the result of the decision-making policies of the past**. But new decisions are taken following the environment market inputs on a quarter-on-a-day basis, but not on a seven- or ten-year basis. Stock price volatility is a highly complex non-linear dynamic system. The volume of trade affects the self-correlation and inertial effect of the stock, and the adjustment of the stock does not advance with a homogeneous time process, which has its own independent time to promote the process [115]. If the present state "the today" is Markov, the future will depend only on the present state. Therefore, an optimization process of financial assets responds to the Markov chain property, which considers that the next state of a system depends only on the present state. In other words, the history of past transitions does not influence the determination of the future state.

Markov decision processes (MDPs) are generally defined as a controlled stochastic process that fulfills Markov's properties and rewards state transitions; they are defined as controlling processes [21, 148]. MDPs make it possible to model the dynamics of the state of a system subject to the control of an agent within a stochastic environment. The agent follows a procedure to automatically choose at a given moment the action to be performed on and execute the action. A Markov decision-making problem is to search among a family of policies for those that optimize a given performance criterion for the Markov decision-making process under consideration. This criterion aims to characterize the policies ticks that will generate the largest possible reward sequences. In formal terms, this always amounts to evaluating a policy on the basis of a measure of the expected accumulation of instantaneous rewards along a trajectory. This choice of expected accumulation is, of course, important because it makes it possible to establish that the sub-policies of the optimal policy are optimal sub-policies [195] (**Bellman** principle of optimality). This principle is at the base of many dynamic programming algorithms that allow us to solve MDPs efficiently.

The approach of modeling in the form of an MDP to represent a decision-making process applied to asset management appears to be most appropriate to the very volatile nature of financial markets. Because of the large number of states and actions to be considered in the Markov-based management of financial processes to achieve a possible leverage effect, their conduct often involves simulation, which allows one to resolve large state-space system in an iterative way. Furthermore, the simulation-based Markovian approach [64] corresponds to our simulation context in which decision making must be conducted with a high degree of uncertainty without the possibility of defining a state transition frame a priori.

The definition and the simulation of an MDP model requires a formal framework to represent the interaction (actions/rewards) between the environment and the agent of an MDP. The **DEVS formalism** (Discrete EVent System Specification) [203], is the appropriate mathematical framework for specifically specifying an MDP. In addition, DEVS offers the automatic simulation of these models and thus the simulation of artificial intelligence (AI) as RL algorithms (e.g. Q-Learning) [111] applied to MDPs in order to obtain the optimal solution in the framework of a decision-making problem with a high degree of uncertainty. Therefore, we point out that DEVS make an AI system more explicable by using its modular and hierarchical modeling specific approach and its building blocks and architectural patterns capabilities [199]. AI applications are booming in the world of today, and the weight of decisions is delegated to AI. Considering this enormous volume and also the scrutiny decisions that need to go through, having clarity on why a certain decision was made has become paramount. Explainability is the next data science superpower, in particular for AI based solutions and DEVS is a good candidate to improve the explainability aspect of AI systems.

To mitigate the risk of delegation to machines in decision-making processes, DEVS offers the opportunity to remain under observation, interactions, and separation between the agent and the environment involved in the traditional RL algorithm, such as Q-Learning. In RL, an agent seeks an optimal control policy for a sequential decision problem. Unlike in supervised learning, the agent never sees examples of correct or incorrect behavior. Instead, it receives only positive and negative rewards for the actions it tries. Since many practical real-world problems (such as robot control, game play, and system optimization) fall into this category, developing effective RL algorithms is important to the progress of AI. The Q-Learning algorithm needs to be formally separated in a DEVS Agent component that reacts with a dynamic DEVS Environment component. This modular capability of DEVS permits a total control of the Q-Learning loop in order to drive the algorithm convergence. Basically, an agent and an environment communicate in order to converge the agent towards a best possible policy. Due to the modular and hierarchical aspect of DEVS, the separation between the agent and the environment within the RL algorithms is improved. Our contribution is to propose a formalized way to drive RL algorithms into a discrete event system. A methodological approach is implemented to design **DEVS-RL model**. The process is shown in the following Figure 1.2.

The objective of this work is to propose a decision-making management policy for complex systems evolving in highly dynamic environments using a combination of the DEVS formalism and the RL technique. This work raises a number of questions relating to fundamental research issues, such as the following.

- What are the advantages of coupling DEVS simulation and AI techniques, particularly reinforcement learning, in the decision-making process of a complex system?

- How can the DEVS formalism improve understanding of AI algorithms?

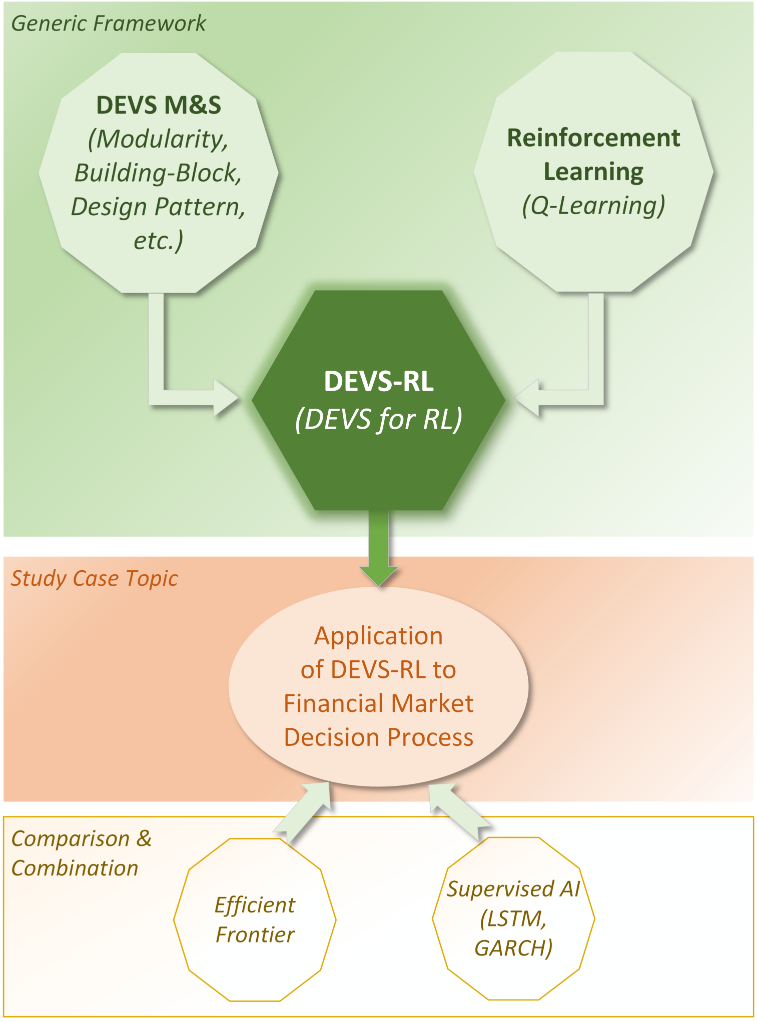


Fig. 1.2 In the generic framework, DEVS-RL is a general methodological approach based on the combination of DEVS M&S and RL algorithm. The study case allows us to validate the quality of results in terms of financial gains. Confrontation and combination allow us to highlight the similarity and complementary between DEVS-RL, EF, and LSTM techniques.

- How does the association of RL algorithms with DEVS simulation improve the management of leverage effects in financial management processes?

The research work proposed in this thesis contributes to answering these questions relying on a concrete case of optimizing the management of a portfolio of stocks using discrete event simulation. This thesis presents the benefits of DEVS formalism aspects to assist in the realization of ML models with a special focus of RL. The DEVS formalism is an ideal solution for implementing an RL algorithm such as Q-learning because it makes it possible to represent and carry out the learning of the system by simulation in a formal, modular, and hierarchical framework. Basically, in the RL model, an agent and an environment communicate to converge the agent towards the best possible policy. Due to the modular and hierarchical aspects of DEVS, the interaction between the agent and the environment within the RL algorithms in experimental frames can be improved in terms of explainability and observability. A new generic DEVS modeling of the Q-Learning algorithm based on two DEVS models has been proposed: the DEVS models Agent and Environment. An implementation in Python language with an object-oriented approach and a confrontation are presented to show the benefits of the proposed approach. The DEVSimPy [33] M&S environment is used to implement the Q-Learning algorithm as event-based interactions between an Agent and Environment DEVS atomic models and facilitate the M&S of the proposed case study.

A validation has been proposed in a dynamic and uncertain environment that corresponds to one of the best application domains of the Markov chain property. Firstly, we decided to present our approach using the historical data of stock exchange markets because the stock option management perfectly fits to a very volatile environment where the next state of a system depends only on the present state. The Markov property is clearly identified by the fact that past financial gains are no guarantee of future gains. Second, we compare the DEVS model to validate by confrontation the similarity of the results with an algorithm commonly used to optimize portfolio such as the EF model. Third, due to the modularity of DEVS, we combine with RNN models largely used for portfolio prediction, such as LSTM or GARCH, to propose a novel approach combining AI and DEVS simulation.

1.2 Outline

This thesis is organized into five chapters.

Chapter 2: *State of the Art*. The analysis and literature review of portfolio management decision-making methods, MDP, RL, and DEVS, are discussed in this chapter. Portfolio decision-making methods are introduced through volatility analysis and risk and return analysis in financial decision-making. In this section, three types of methods are reviewed, GARCH, LSTM, and EF. MDP is analyzed through the concatenation with the Markov chain and the Markov reward process to introduce the different functions related to MDP. The main approaches to RL are discussed, RL methods are detailed, and the combination with MDP is analyzed. DEVS formalism with the DEVSimPy collaborative framework is introduced.

Chapter 3: *DEVS-Based Reinforcement Learning System Modeling and Simulation*. This chapter is dedicated to the presentation of the approach developed in this thesis, which consists of proposing discrete event-oriented modeling of RL system using the DEVS formalism. The benefits of hierarchical and specification aspects of the DEVS formalism are presented to assist in the realization of RL models. More specifically, the DEVS formalism makes it possible by specifying the Q-Learning algorithm using a set of interconnected atomic or coupled models reacting by its external transition function. We present the atomic and coupled-based approaches in detail with an additional approach that considers a multi-agent version of an RL system. Observability and explainability aspects are considered to show how, when applied using the DEVS formalism, they improve the understanding of RL algorithms and outputs. Finally, a DEVS modeling of the Agent-Environment RL model is proposed with its implementation in the DEVSimPy environment, and a complete case study is presented.

Chapter 4: *Case Study: Leverage Effects in Financial Asset Optimization Processes*. This chapter presents the case study of the M&S with DEVS-RL of leverage effects in financial asset optimization processes. This case study highlights the interest of our approach based on the combination of DEVS and RL. The consideration of a very volatile environment, because it is composed of rapidly changing stock market indices, makes it possible to show the effectiveness of our M&S approach to improve observability, explainability, and decision-making in this complex dynamic system. A section dedicated to compatibility with the EF technique is presented, complete by a study of the combination of our approach with the LSTM approach.

Chapter 5: *Conclusion and Perspectives*. In this chapter, a summary of different contributions is proposed, and a discussion of the result obtained is presented. Finally, some interesting perspectives for future work are given.

Chapter 2

State of the Art

The combination between the DEVS formalism and the Markov decision process environment is, in general, a very juvenile matter in the scientific literature [14, 45, 99]. Some authors have exploited the DEVS formalism in decision making, for example, in agriculture [4], in the smart grid issue or, more recently, in building a model to understand the spread of Covid 19 [35]. However, the contribution of the scientific community to this combination is still limited. We are probably pioneering in exploring the potential of the combination between the DEVS formalism and the Markov decision process environment applied to financial issues decision making based on market volatility. Therefore, our research does not aim to improve the results of work done by other authors or even try to prove by comparison that our research effort achieves better computational or financial results compared to other methods. We aim to prove that our research effort is valid and valuable and may become the benchmark for future studies that will combine the DEVS formalism and Markov decision processes environment to build solutions for the decision makers of stock market asset management. In the state-of-the-art, we detail, through literature analysis, the main concepts we effectively used or explored to get to our decision-making process based on the market volatility in a Markovian environment with a DEVS formalism. In this chapter, we will detail a review of the three domains exposed in this document.

2.1 Decision-Making Relevant to Finance

2.1.1 Introduction

First of all, we briefly introduce some key topics of our study case, such as volatility in decision-making, the Stock Market, and in particular, we detail Market Volatility a little more deeply, which plays a main role in the study case's process. To make those financial topics as accessible as possible to readers who are not particularly familiar with stock market issues, we detail market volatility by giving samples and literature on how and when traders take it into account. Second, we introduce two close topics related to volatility and decision making i.e. **risk and returns** [10] that will allow the reader to better understand why earning money (returns) as maximum as possible will be the **reward** of our agent. We give details of a parameter that at the end we do not take into consideration in our study case, that is, transaction costs, i.e. the sum we pay every time we trade on a trading platform. Taking into account the transition cost may be a future improvement of our model to make it more generic, and we give an example of the authors that deal with that parameter.

Third, we detail some of the most commonly used models and algorithms used by stock market decision-makers to predict or optimize an asset or a portfolio. We first introduce the GARCH model, even if we do not use it in the study case or in the comparison. Honestly, the literature gives evidence that GARCH is one of the most sophisticated models to characterize the volatility of the markets; we do not use it by a lack of know-how, but the GARCH DEVS combination would be one of the possible future developments of our research, in particular, if we want to take more into consideration the "time" in our decision-making path.

Next, we detail two groups of algorithms that use volatility to predict or optimize asset positions or portfolios such as LSTM and EF with the semivariance. Predictions and optimization results from LSTM and EF will be detailed in the comparison chapter to give evidence of the value of the decision-making driven by an agent in a Markovian environment with a DEVS formalism.

2.1.2 Volatility in a Stock Market

Volatility is a close representation of the concept of risk and returns and plays an important role in decision-making policies in different fields. The role of volatility is illustrated in the literature by the relationship between income volatility and health care decision making [3]. Self-adaptive systems, i.e. systems with the ability to adjust their behavior in response to their perception of the environment (a central part of our research effort) are also related in the literature to the concept of volatility [140]. In general, volatility plays a role in growth analysis [83], as well as in development policies [107]. Volatility is also an important analysis factor in correlation forecasting [8]. Volatility may also be used to better understand Entrepreneur's decision-making ability [143]. Since the 1990s, it has also been a fundamental parameter in studying the market [170].

The term **Stock Market** refers to various exchanges where shares of listed companies are bought and sold. These financial activities are conducted through formal trading and over-the-counter (OTC) markets that operate within a defined set of regulations. The stock market allows buyers and sellers of securities to meet, interact, and negotiate. Markets provide information on company stock prices and act as a barometer of the overall economy. Buyers and sellers are assured of a fair price, a high degree of liquidity, and transparency as market participants compete on the open market [26]. The first stock exchanges issued and traded physical paper stock certificates. Stock markets today operate electronically.

Market volatility is the frequency and magnitude of price movements, up or down [184]. The larger and more frequent price fluctuations, the more volatile the market is said to be [69]. Market volatility is measured by determining the deviation of the price change over a period of time [53]. The statistical concept of standard deviation makes it possible to see how much something differs from an average. Traders calculate standard deviations from market [147] values based on end-of-day trading values, change in values during trading sessions, intraday volatility, or potential future changes in values. Casual market observers are probably more familiar with the latter method, used by the Chicago Board Options Exchange Volatility Index, commonly referred to as VIX [150]. VIX, also called the 'fear index', is the most well-known indicator of volatility in the stock market [48]. Measures investors' expectations [110] of where stock prices will move in the next 30 days based on S&P 500 options trading. VIX indicates how much traders expect stock prices to S&P 500 change, up or down, over the next month [37]. Typically, the higher the VIX, the more expensive the options [57]. The increase in the value of these put options becomes a warning sign of the expected decline in the market and, hence, volatility, as the stock market generally experiences more dramatic upward or downward value changes. Historically, normal VIX levels have been around 20, which means that S&P 500 is very rarely less than 20% above the average growth [52]. Most days, the stock market is fairly calm, with short periods of above-average volatility interrupted by short periods of calm. These times tend to make the average volatility higher than it would be on most days [49]. In general, bull markets (bullish trend) tend to have low volatility, but bear markets (bearish trend) are typically accompanied by unpredictable price movements that tend to be to the downside [71].

The main question for an investor or a portfolio manager is how to manage market volatility? There are numerous ways to react to fluctuations in a portfolio of stocks. For example, under Basel capital agreements, capital requirements (CR) for exposure to market risk of banks are value-at-risk (VaR) [180]. VaR is defined as the loss associated with the low percentile of the return distribution. Basel II Capital Agreements codifies VaR as the industry standard de facto for banking and insurance companies alike [62]. Risk exposure due to volatility is one of the main reasons why an investor tends to optimize its portfolio. One of the traditional ways to optimize is to balance the weight of each asset allocated in the portfolio, and one of the common ways to take into account the exposure i.e. the VaR is the optimization by the mean-variance (MVO) to determine a set of portfolios characterized by the optimal trade-off between the expected return and VaR.

More generally, experts caution against panic sales after a sharp drop in the market. According to analysts at the Schwab Center for Financial Research [46] since 1970, when stocks fell 20% or more, they have made the greatest gains in the first 12 months of recovery. Hence, if due to panic we sold our portfolio at once during the crash and waited for return, our investment would have lost significant gains and may never have recovered the value they had lost. Instead, when market volatility causes investors to be nervous, they can try one of these approaches:

1. Don't forget a long-term plan. Investing is a long-term game and it is true that a well-balanced and diversified portfolio was constructed in times like these with this approach in mind [162]. If we need your funds in the near future, they should not be in the market, where volatility can affect our ability to get them out quickly. But for long-term goals, volatility is part of the race for meaningful growth [12]. This can help mentally manage market volatility by thinking about how many stocks we can buy while the market is in a bearish state [166] [165]. During the 2020 bear market [119], for example, we could have bought shares in an S&P 500 index fund at approximately a third of the price a month earlier, after more than a decade of steady growth. At the end of the year, our investment would have increased by approximately 65% from the minimum and 14% since the beginning of the year.

2. Maintain a healthy emergency fund. Market volatility is not an issue, unless we need to liquidate an investment [80], as we may be forced to sell assets in a bear market. For this reason, it is particularly important for investors to have an emergency fund equal to three to six months of out-of-pocket expenses. If you are close to retirement, you should recommend an even bigger safety net, up to two years of non-market-related activity [31]. This includes bonds, cash, life insurance cash values, home equity lines of credit, and home equity conversion mortgages.

3. Rebalance the portfolio if necessary. As market volatility can cause large swings in the value of investments, your asset allocation can deviate from the desired fractions after periods of intense swings back and forth [97]. During these periods, we must rebalance our portfolio to meet our investment goals and meet the level of risk you want [13]. When rebalancing, we sell part of the asset class that has moved to a larger part of our portfolio than we want and use the proceeds to buy more of the asset class that has become too small. It is a good idea to rebalance when the allocation deviates by 5% or more from the original target mix [130]. We may also want to rebalance if

> we see a gap greater than 20% in an asset class. For example, if we want emerging market equities to make up 10% of our portfolio and, after a severe market downturn, we find that emerging markets are more like 8% or 12% of our portfolio, we may want to change our resources.

Approaches (1), (2) and (3) have been used to influence the behavior of our agent in the case study.

Risk versus Returns in portfolio management is related to MDP, which will be detailed in the next section. Risk aversion in MDPs is a topic that has been addressed by many authors [131] and leverage aversion [90] can have a large effect on the choice of a portfolio of an investor [91].

As described by the authors in [67]: *Actually, trading is a decision-making activity that has a dynamic relationship with three fundamental parameters: risk, returns and volatility*. Risk is the main determinant of investment return, which is why an equilibrium approach to risk and returns is generally applied to decision-making policies [193]. To illustrate an equilibrium approach, imagine a simple case of a portfolio composed of two assets (securities). The portfolio return can be calculated as follows.

$$E_{(rp)} = WA.E_{(ra)} + WB.E_{(rb)},$$

where $E_{(rp)}$ expected return of portfolio P, $E_{(ra)}$ expected return of security A, $E_{(rb)}$ expected return of security B, WA proportion of portfolio invested in security A, WB proportion of portfolio invested in security B.

More generally, the relationship between risk and return plays an important role in the decision-making literature [127]. The balance of risk and policy return is taken into account in many industrial fields, i.e. the effective deployment of photovoltaics in Mediterranean countries [117]. The analysis of risk and return is also crucial in public policy investment [19] or in sustainability analysis [175].

The expected return [124] ($E_{(rp)}$) is often the objective function of choice in planning problems, where the results depend not only on the actor's decisions, but also on random events. Expectations are often there, a natural choice, because the law of large numbers [145] ensures that the average returns to many races will converge in waiting. In addition, the linearity of expectations can often be exploited to obtain efficient algorithms. Some experiences, however, can only take place once, either because they take a lot of time (invest for the withdrawal), because we do not have the possibility to try again if we lose (parachuting, crossing the street), or because experience versions are not available like in the stock market. In this context, we cannot longer use the law of large numbers to ensure that the return is close to its expectations with high probability [157], so the expected return may not be the best target to optimize. If we were pessimistic, we could assume that anything that can go wrong will go wrong and try to minimize losses based on this assumption. A general approach would include *minmax* optimization and expectation optimization [70], corresponding absolute risk aversion and risk ignorance, respectively [102], but would also allow for a range of policies between these extremes.

Standard theory suggests that investors must be compensated for the risk that they take, so we can attribute to each asset an expected compensation (i.e., a prior estimate of returns). This is quantified by the market-implied risk premium, which is the excess return of the market $R - R_f$ divided by its variance σ:

$$\delta = \frac{R - R_f}{\sigma^2}.$$

To calculate the market-implied returns, we then use the following formula:

$$\prod = \delta \sum \omega_{mkt},$$

where ω_{mkt} denotes the market cap weights. This formula is used to calculate the total amount of risk contributed by an asset and multiply it by the market price of the risk, resulting in the market-implied return vector $\prod$.

Most traders can quickly define the required win rate, expected average return, risk levels, and position sizes that are necessary for their success [66]. These metrics often vary from trader to trader, but should be as accurate as possible over the long term to ensure success. Slight deviations in these numbers can mean the difference between success and failure. Performance truly depends on a fine balance. If a trader executes his orders incorrectly, he can find the source of his shortcomings and remedy them by changing his approach, strategy, frequency, etc.

On the other hand, an inevitable problem will always remain embedded **transaction costs** [6], such as fees and the cost of the spread. Of these two frequently encountered problems, one is a little more difficult to solve: fees. An example of such an unavoidable situation is when a person trades CFDs or contracts for difference. The exchange may not advertise fees, but that is only because they are already built into the spread. A trader might pay upwards of 1.5% to move a position back and forth. This means that before even considering the possibility of a favorable or unfavorable development of a trade, there is already a loss to enter and exit. This may seem like a small value, but now consider the cumulative effect this can have, especially on an active trader. Most people are already sensitive to trading costs in some way or another, but the largest traders are hit the hardest [18]. Consider a trader who scalps momentums on short time frames [128]. For this example, let us say that this trader benefits from a trading approach with positive expectations. Without even talking about the returns or losses of his strategy, let us imagine that he is trading with an account of 10,000. As a trader on short timeframes, it is likely that he will move back and forth on his positions a lot. To simplify to the extreme, our trader risks 1% of his account in each transaction and trades with the total value of his account. Scalers can execute between 50 and 300 trades per week [181]. For the purposes of this experiment, let us take a low random number and say that our scalper executes 75 trades per week [105], or about 10.71 trades per day. This represents approximately 37 back-and-forth, that is, opening a position to buy and then selling this position to close, and vice versa. In this case, our broker is operating on an extremely popular exchange that charges a maker fee of 0.10%, or 10 basis points. In this example, this trader lost 722 fees alone for the week. Of course, once again we assume that all his trades were closed at breakeven. Now imagine how this could be incorporated into a system that would additionally suffer losses and how it could exacerbate any negative combination that already includes transaction costs by default. You are rightly assuming that, especially in the case of traders who operate at higher frequencies, fees can completely eat away at any return.

In general, a transaction cost is known in advance and some authors take it into account as a simple deterministic and controllable state transition [136]. We have chosen not to take transaction costs into consideration in our case study, for the simple reason that our beta tester, BCC Felsinea Bank (Italy), suggested that we not apply any transition cost. Actually, to widen the windows of possible final users, it would have been preferable to take into account the transition costs.

2.1.3 Portfolio Management Decision-Making Methods

The scientific literature on AI offers more and more works on portfolio optimization. The section introduces three of the most widely exploited in the scientific literature, such as generalized autoregressive conditional heteroskedasticity process (GARCH) [95] models, efficient frontier (EF) [39], and long-short-term memory (LSTM) [197].

2.1.3.1 Generalized Autoregressive Conditional Heteroscedasticity Process

Generalized autoregressive conditional heteroscedasticity process (GARCH) is a sophisticated econometric model that has been developed to characterize the volatility of financial markets. Financial institutions use the model to estimate the volatility of returns from investment vehicles [76]. These models assume that the conditional volatility of future returns depends on shocks in the current volatility or other state variables, whereas the unconditional volatility is constant over time. In other words, we can say that GARCH shares with Markov decision processes the property that the future depends only on the present. More in detail, heteroskedasticity describes the irregular pattern of variation of an error term or variable in a statistical model. Basically, when there is heteroskedasticity, the observations do not conform to a linear pattern. Instead, they tend to cluster together. The result is that the conclusions and predictive value drawn from the model will not be reliable. GARCH is a statistical model that can be used to analyze different types of financial data, such as macroeconomic data. As we have already highlighted, financial institutions typically use this model to estimate the volatility of stock, bond, and stock index returns. They use the resulting information to determine prices and judge which assets will potentially offer higher returns, and predict current investment returns to aid in asset allocation, hedging, risk management, and optimization decisions.

The general process of a GARCH model has three steps. The first is to estimate a more suitable autoregressive model. The second consists in calculating the autocorrelations of the error term. The third step is to verify the meaning. Two other widely used approaches to estimate and predict financial volatility are the classic historical volatility method (VolSD) and the exponentially weighted moving average volatility method (VolEWMA).

The GARCH model was introduced by Engle in 1982 [7]. Particularly popular is its simplest version of the GARCH(1,1) model:

$$r_t = \mu \ + \varepsilon_t,$$

where r_t is the return of an asset at time t, μ is the average return, and ε_t are the residual returns, defined as $\varepsilon_t = \sigma_t z_t$.

GARCH processes differ from homoskedastic models, which assume constant volatility and are used in basic ordinary least squares (OLS) analysis. The objective of OLS is to minimize deviations between data points and a regression line to fit those points. With asset returns, volatility appears to vary during certain periods and depends on past variance, making a homoskedastic model suboptimal [25].

GARCH processes, because they are autoregressive, depend on past squared observations and past variances to model the current variance. GARCH processes are widely used in finance due to their effectiveness in modeling asset returns and inflation. GARCH aims to minimize forecast errors by taking into account previous forecast errors and improving the accuracy of current forecasts [25]. To forecast the volatility of the stock price index, GARCH-type models are usually integrated with a recurrent neural network such as LSTM or models such as EF or semiVariance [101].

2.1.3.2 Long Short-Term Memory

Long short-term memory (LSTM) networks are one of the most widely used deep learning methods, especially for time series analysis [176]. Since this pioneering work, LSTMs have been modified and popularized by many researchers [68].

Due to the limited capacity of the single LSTM cell in handling engineering problems, LSTM cells have to be organized into a specific network architecture when processing practical data. LSTM networks can store the state information in memory cells or specially designed gates. Information available at the current time slot is received at the input gates. Using the results of aggregation at the forget gates and the input gates, the networks predict the next value of the target variable. The predicted value is available at the output gates. As a better version of recurrent neural networks (RNN) [23], LSTM has been used in many fields. In particular, in the prediction of the stock price, LSTM had been proposed and applied by some notable researchers, such as Murtaza et al. [156] and Nelson et al. [134]. In general, the LSTM architecture is designed to accurately forecast future stock prices. After the elapsed period of time since portfolio construction, the actual returns yielded by the portfolios and those predicted by the LSTM models are computed. The actual and predicted returns are compared to evaluate the accuracy of the LSTM model [163].

The root mean squared (RMSE) was used to measure the difference between the predicted and practical data. RMSE has been calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \dot{(}y_i - y_i^{'})^2}{n}},$$

where $Y = (y_1, y_2, \ldots, y_n)$ is a vector of actual observations, $Y^{'} = (y_1^{'}, y_2^{'}, \ldots, y_n^{'})$ is a vector of predicted values, and n is the number of observations.

LSTM is an advanced soft computing method [84] introduced for the first time to address the limitation found in the conventional RNN method, especially to solve problems with the long-term dependency issue [78]. The output of an LSTM at a particular point in time depends on three things:

- The current long-term memory of the network (the *Cell state* noted C_{t-1})
- The output at the previous point in time (the previous *Hidden state* noted H_{t-1})
- The input data at the current time step.

LSTM uses a series of gates that control how information in a sequence of data enters, how it is stored, and how it leaves the network. As shown in 2.1, there are three gates in an

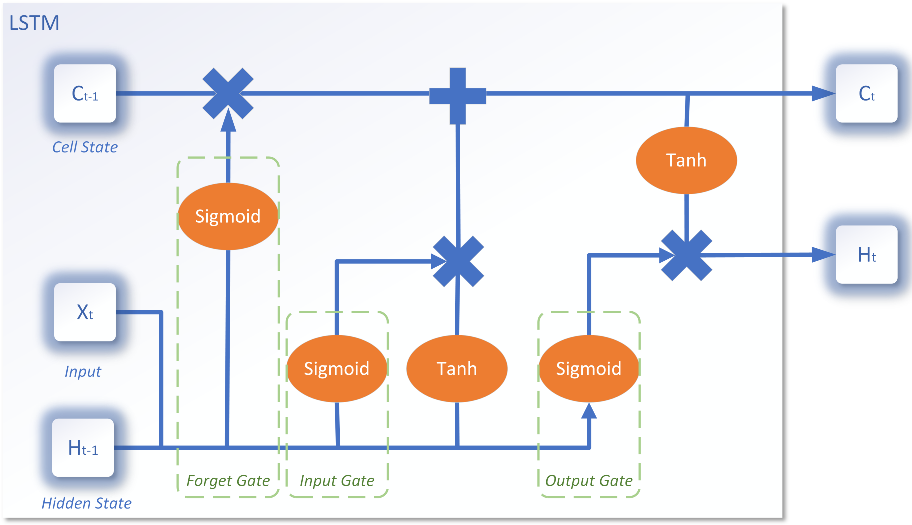


Fig. 2.1 LSTM architecture with forget, input, and output gates. **X** is the pointwise multiplication, **+** is pointwise addition, sigmoid activated NN, Tanh (yellow) pointwise Tanh not NN, Tanh (green) activated NN

LSTM: forget gate, input gate, and output gate. These gates can be thought of as filters and each have their own neural network. A LSTM network consists of several LSTM cells that are self-connected and are used to store the temporal state of the networks using the three gates. What happens in an LSTM cell can be summarized with the following steps. Readers are invited to look at figure 2.1 moving from left to right.

- The first step in the process is *forget gate*. Based on the previous hidden state and the new input data, it is decided which bits of the cell state are important for the long-term memory of the network. That is why the new input and the previous hidden state data are fed into a neural network. The forget gate decides which pieces of long-term memory have less weight and should now be forgotten.

- Given the previous hidden state and the new input data, the second step corresponds to the *input gate* step that determines what new information should be added to the cell state. The sigmoid activated network is the input gate, which first checks if the new input data is even worth remembering and then acts as a filter, identifying which components of the new memory vector are worth retaining. The sigmoid function is useful to filter because it can only output values between 0 and 1, so the knowledge that is no longer needed is forgotten, as it is multiplied by 0 and, consequently, it is dropped out. *Tanh* activated neural network is a new memory network which has learned how to combine the previous hidden state and new input data to generate a new memory update vector. This vector essentially contains information from the new input data given the context of the previous hidden state.

- The third step. First, apply the tanh function to the current cell state pointwise to obtain the squished cell state, which now lies in $[-1,1]$. Second, pass the previous hidden state and current input data through the sigmoid activated neural network to obtain the filter vector. Third, apply this filter vector to the squished cell state by point-wise multiplication. Finally, output the new hidden state.
- The steps above are repeated many times. For example, if we want to predict the value of an asset based on the previous 150 values, the steps will be repeated 150 times. In other words, the model have produced iteratively 150 hidden states to predict the value of tomorrow or to any given horizon.

For future values of stock price prediction, we present in figure 2.2 a classic example of a schematic design of an LSTM model. The model uses daily close prices of the stock of the last 150 days as input. The input data for 150 days with a single characteristic (that is, close values) have a shape (150, 1). The input layer receives the data and transmits them to the first LSTM layer of 256 nodes, which is made up of the input layer. The LSTM layer yields a shape of (150, 256) at its output. This implies that 256 features are extracted by each node of the LSTM layer from every record in the input data. A dropout layer is used after the first LSTM layer that randomly switches off the output of 30% of the nodes in the 256 LSTM to avoid model overfitting. Another LSTM layer with the same architecture as the previous one receives the output of the first and applies a dropout rate of 30%. A dense layer with 256 nodes receives the output of the second LSTM layer.

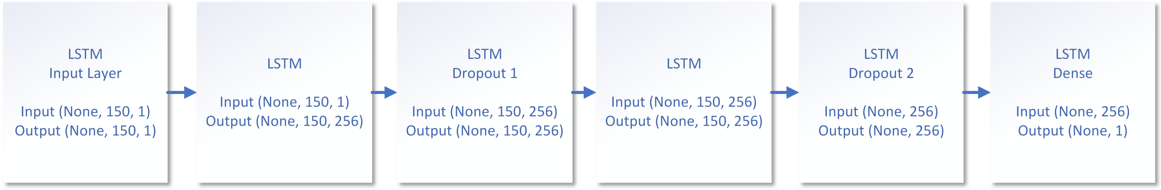


Fig. 2.2 LSTM model for predicting stock prices [89].

Figure 2.3 shows the output cell of an LSTM model as modeled in Figure 2.2.

2.1.3.3 Efficient Frontiers Methods

In modern portfolio theory (MPT), **efficient frontier** (EF) (or portfolio frontier) [121] is an investment portfolio that occupies the "efficient" parts of the risk–return spectrum. Formally,

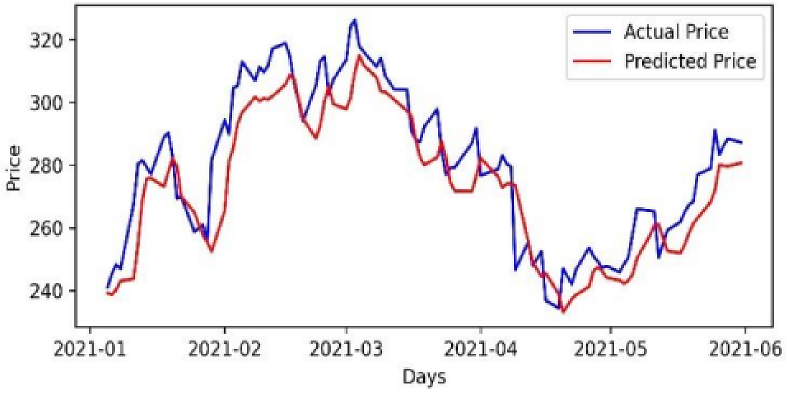


Fig. 2.3 Actual vs. predicted stock price by an LSTM model (period: Jan. 1, 2021, to June 1, 2021) [163].

it is the set of portfolios which satisfy the condition that no other portfolio exists with a higher expected return but with the same standard deviation of return (i.e. the risk). For instance, a portfolio manager has a set of assets to which he would like to allocate a certain sum of money, and to take his decision, he has estimates the expected returns of each asset and the covariance relationships amongst the assets.

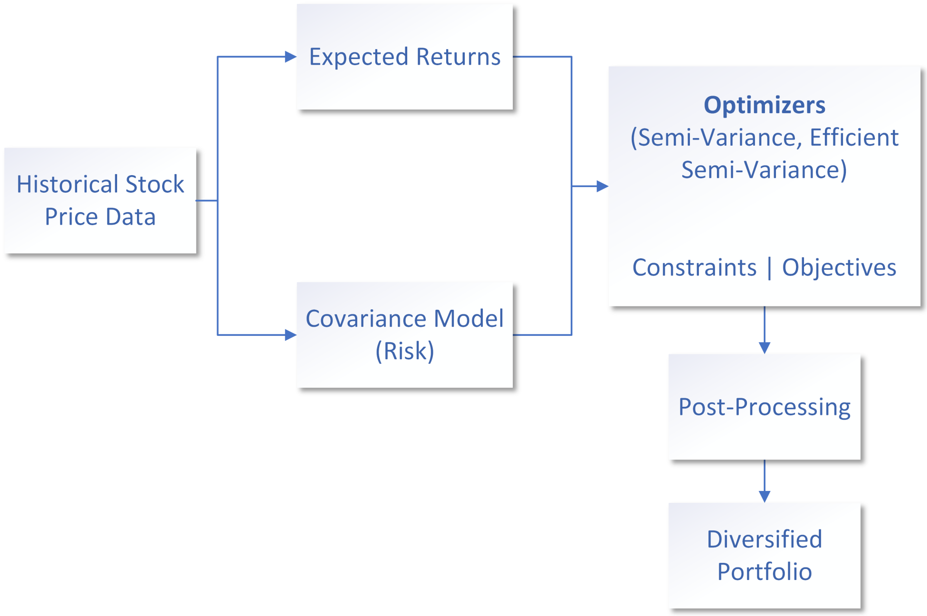


Fig. 2.4 Efficient Frontier Process from historical stock price data to diversified portfolio.

As shown in Figure 2.4 EF methods depend on two key intuitions [108]: The first is that investors seek returns, but are risk averse. The risk is measured by the volatility of the portfolio of assets. The second is that a rational investor who receives two portfolios exhibiting the same expected return would choose a portfolio that is less volatile. However, we are subject to the following constraints:

- Capital must be allocated in full. This is an equality constraint.
- A minimum portion of our capital must be allocated to each asset. This is an inequality constraints.
- The maximum allocation must not exceed one asset. This is an inequality constraint.

This kind of inequality constraint has a strong disadvantage for portfolio managers. In fact, they cannot short the poor performing assets and use the proceeds to leverage their positions in the high performing assets [22].

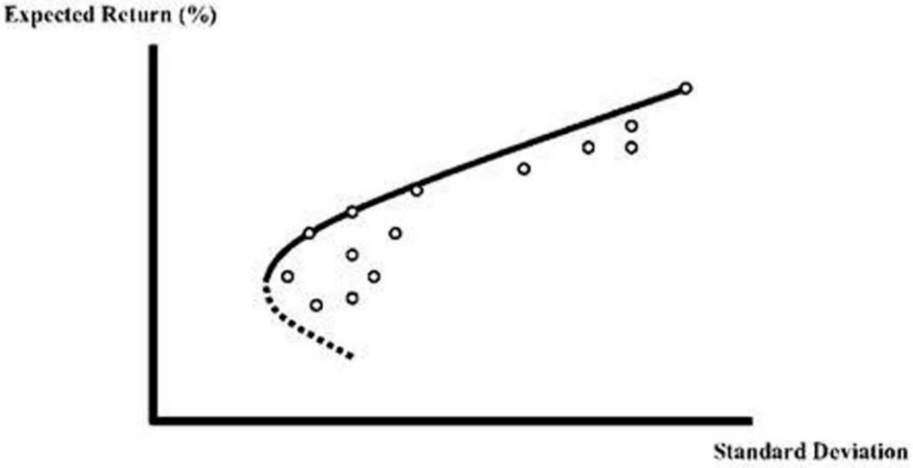


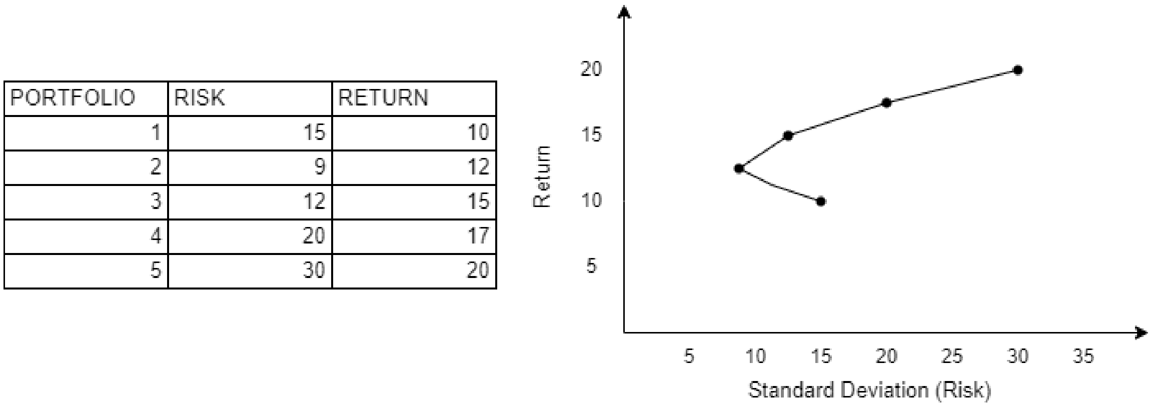
Fig. 2.5 Markowitz Efficient Frontier [88].

.

Figure 2.5 shows the complete set of investment opportunities, which represents the set of all possible combinations of risk and return offered by portfolios made up of assets in different proportions. The combination of specific risky assets in the portfolio plotted on an efficient frontier represents the lowest possible risk of the portfolio for the desired level of expected return for an acceptable risk level. The line along the upper edge of this region is known as the efficient frontier, sometimes called the ’Markowitz bullet’ [154]. Combinations along this line represent portfolios (explicitly excluding the risk-free alternative) for which there is the lowest risk for a given level of return. On the contrary, for a given amount of risk, the portfolio on the efficient frontier represents the combination that offers the best possible return. Mathematically, the efficient frontier is the intersection of the set of portfolios with minimum risk and the set of portfolios with maximum return [79].

In other words, the efficient frontier in Figure 2.4 for a given sector represented the contour of a large number of portfolios in which the returns and the risks are plotted along the y *axis* and x *axis*, respectively. Points on an efficient frontier have the property that they are the portfolios that yield the maximum return on a given risk. They may introduce the minimum risk on a given return. The point at the leftmost point on the efficient frontier depicts the minimum-risk portfolio.

To concretely detail Figure 2.5 with a simple example, suppose that a portfolio consists of five stocks with, respectively, an evaluation of the risk and the return.



PORTFOLIO	RISK	RETURN
1	15	10
2	9	12
3	12	15
4	20	17
5	30	20

Fig. 2.6 Efficient Frontier example.

MPT is an investment theory developed by Harry Markowitz and published under the title "Portfolio Selection" in the Journal of Finance in 1952 [122]. There are a few underlying concepts that can help anyone understand MPT. An acronym in data science and optimization research is known as "TANSTAAFL". It is a famous acronym for "There Ain't No Such Thing As A Free Lunch". This concept is also closely related to the 'risk-return trade-off' in finance [75].

MPT is a practical investment selection technique to maximize the overall return with an appropriate level of risk [89]. Investment portfolio theories guide the way a portfolio decision maker allocates money and other capital assets within an investment portfolio. An investment portfolio has long-term objectives independent of daily market fluctuations. Due to these goals, investment portfolio theories aim to help portfolio decision makers with tools to estimate the expected risk and return associated with investments [89]. Passive portfolio theories, on the one hand, combine an portfolio decision maker's goals and temperament with financial actions. Passive theories propose minimal input from the investor; instead, passive strategies rely on diversification, buying many stocks in the same industry or market, to match the performance of a market index. Passive theories use market data and other available information to forecast investment performance [59]. Active Portfolio Theories come in three varieties. Active portfolios can be patient, aggressive, or conservative. Patient portfolios invest in established, stable companies that pay dividends and earn income despite economic conditions. Aggressive portfolios buy riskier stocks, those that are growing, in an attempt to maximize returns; because of the volatility to which this type of portfolio is exposed, it has a high turnover rate. As the name implies, conservative portfolios invest with an eye on long-term yield and stability [74]. Some authors propose to bridge the gap between portfolio theory and machine learning [39].

One of the EF techniques is **mean-variance optimization** which represents an important approach in financial decision making, especially for static (one-stage) problems [120]. In general, the optimization procedure is robust and provides strong mathematical guarantees with the correct inputs. On the other hand, optimization of mean variance requires knowledge of expected returns [198]. In practice, these are rather difficult to know with any certainty. Therefore, the best we can do is to make estimates, e.g. by extrapolating historical data. This is the main flaw in mean-variance optimization and is also the reason many authors have attempted to cross mean-variance and MDPs [16].

When we use mean-variance optimization, we assume that portfolio optimization problems are convex. This is not true in many cases [9]. However, the convex optimization problem is a well-understood class of problems that are useful in finance. A convex problem has the following form:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimise}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0, i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0, i = 1, \ldots, p \end{aligned}$$

This notation describes the problem of finding $\mathbf{x} \in \mathbb{R}^n$ that minimizes $f(\mathbf{x})$ among all $\mathbf{x}$ satisfying $g_i(\mathbf{x}) \leq 0$, $i = 1, ..., m$ and $h_i(\mathbf{x}) = 0$, $i = 1, ..., p$. The function f is the objective function of the problem, and the functions g_i and h_i are the inequality and equality constraint functions.

There are basically two things that need to be clarified: optimization constraints and optimization objective. For example, the classic problem of portfolio optimization is to minimize risk on a return basis (i.e., the portfolio must return more than a certain amount). From the implementation point of view [125], there are few differences between the objectives and the constraints. The role of risk and return can be changed. One of the main advantages of the mean-variance is that it has a simple and clear interpretation in terms of individual portfolio choice and utility optimization, although some of its drawbacks are nowadays well known. Li and Ng [207] introduced a technique to tackle the multi-period mean–variance problem, with market uncertainties reproduced by stochastic models, in which the key parameters, expected return and volatility, are deterministic. An EF object contains multiple optimization methods that can be called with various parameters. For example, it may be used to optimize for minimum volatility or to maximize the return for a given target risk or, on the contrary, to minimize the risk for a given target return [94]. The mean-variance optimization methods described previously can be used whenever we have a vector of expected returns and a covariance matrix. The objective and constraints will be some combination of portfolio return and portfolio volatility.

However, we may want to construct the EF for an entirely different type of risk model, that is, one that does not depend on covariance matrices, or optimize an objective not related to portfolio return, for example, tracking error [24]. The one we have tested is efficient semivariance.

Efficient semivariance is characterized by the fact that, instead of penalizing volatility, mean semivariance optimization seeks to only penalize downside volatility, since upside volatility may be desirable. The mean semivariance optimization can be written as a convex problem [54], which can be solved to give an exact solution. For example, to maximize the return for a target semivariance s^* (long only), we would solve the following problem [123]:

$$\begin{aligned}
\underset{w}{\text{maximise}} \quad & w^T \mu \\
\text{subject to} \quad & n^T n \leq s^* \\
& Bw - p + n = 0 \\
& w^T \mathbf{1} = 1 \\
& n \geq 0 \\
& p \geq 0
\end{aligned}$$

where, w is the weight of each stock, μ, is the expected return, n and p are the decision variables in the optimization problem, that is, n the total number of observations below the mean and p is the mean portfolio return. T is the estimation window, that is, the realizations of the returns of stocks n. B is the normalized return (the benchmark), and here B it is the $T \times N$ (scaled) matrix of excess returns TxN (scaled), $B = \frac{(returns - benchmark)}{\sqrt{(T)}}$. Additional linear equality constraints and convex inequality constraints can be added.

2.1.4 Conclusion

In this section dedicated to decision making relevant to finance, we briefly introduced some key topics of our study case, such as market volatility and risk and returns, which plays a major role in the study case's process. Portfolio return and GARCH (1,1) return, average and residual returns, were concepts that inspire the way we define the reward and the agent behavior in our study case. We detail some of the most commonly used models and algorithms used by stock market decision-makers to predict or optimize an asset or portfolio. We introduced the GARCH model and two other asset portfolio optimizers, LSTM and EF, that we implemented to make a comparison with the model we present in the case study.

2.2 Markov Decision Process

2.2.1 Introduction

Markov decision processes (MDPs) are an extension of Markov chains [85]. In the literature, many authors have already addressed the question of MDPs and the role of MDPs in reinforcement learning [179]. In this paragraph, we try to point out the way we got into MDP and the process we carried out to understand this environment step by step, starting from the Markov chain to the MDP through the Markov reward process (MRP). We want to present the reader with our learning/exploitation decision-making path. At the same time, we will also address the question of reward, discount factor, value of a state, value of an action, and use of the Bellman equation to calculate the state-action function. We will use some simple and hand-crafted examples to illustrate the state-of-the-art.

2.2.2 Markov Chains

The applications of Markov chains span a wide range of fields in which models have been designed and implemented to simulate random processes. Many aspects of life are characterized by events that occur randomly. It seems that the world does not work as perfectly as we hope it would. In an effort to help quantify, model and forecast the randomness of our world, the theory of probability and stochastic processes has been developed and may help answer some questions about how the world works [58]. The focus of this paragraph is on one type of stochastic process known as Markov chains.

Markov chain theory was created in the early 20th century by the Russian mathematician Andrei Andreyevich Markov. Markov's interest in the Law of Large Numbers and its extensions eventually led to the development of what is now called the theory of Markov chains, named after Andrei Markov himself [38, 174].

To simply illustrate a Markov chain, let us consider a stochastic process.

$$\{X^n\}, n \in \mathbb{R}^+_{0,\infty},$$

that takes on a *finite* or *countable* set M.

Let X^n be the close price of a financial asset on the nth day which can be

$$M = \{10, 11, 9, 8\}.$$

One may have the following realization:

$$X^{(0)} = 10, X^{(1)} = 11, X^{(2)} = 10, X^{(3)} = 9, X^{(4)} = 8, \ldots$$

An element in M is called a *state* of the process. Suppose there is a fixed probability P_{ij} independent of time such that

$$P(X^{n+1} = i | X^n = j, X^{n-1} = i_{n-1}, \ldots, X^{(0)} = i_0) = P_{ij},\ n \geq 0$$

where i, j, i_0, i_1,...,$i_{n-1} \in M$. Then this is called a Markov chain process.

In other words, a Markov chain is simply a sequence of random variables that evolve over time i.e. a sequence of random states $S_1, S_2, \ldots$ with the Markov property. Markov chains are stochastic processes that are characterized by their memoryless property, where the probability of the process being in the next state of the system depends only on the current state and not on any of the previous states. This property is known as the Markov property: *"The future is independent of the past given the present"* [58] and, in other words, it means that the present state captures all relevant information from the history completely characterising the process and once the present state is known, the history can be thrown away, and that means that the present state is a sufficient statistic of the future.

A state S_t is Markov if and only if:

$$\mathbb{P}\,[S_{t+1}|S_t] = \mathbb{P}\,[S_{t+1}|S_1, \ldots, S_t]$$

A state space is the set of all values which a random process can take [182]. Furthermore, the elements in a state space are known as states and are a main component in the construction of Markov chain models. With these three pieces, along with the Markov property, a Markov chain can be created and can model how a random process will evolve over time. The changes between states of the system are known as transitions, and probabilities associated with various state changes are known as **transition probabilities** [168]. A Markov chain is characterized by three pieces of information: a state space, a transition matrix with the entries being transition probabilities between states, and an initial state or initial distribution across the state space. For a Markov state s and successor state s', the state transition probability is defined by:

$$P_{ss'} = \mathbb{P}\,[S_{t+1} = s' | S_t = s]$$

The following State transition matrix P defines transition probabilities from all states s to all successor states s':

$$P = \begin{bmatrix} P_{11} & \dots & P_{1n} \\ P_{n1} & \dots & P_{nn} \end{bmatrix}$$

This is a simple example of a Markov Chain related to a scalper daily routine which have three states: (i) $S1$ to buy stocks (i) $S2$ to sell stocks and (iii) $S3$ to wait. If we are in $S3$, figure 2.7 shows that the probability to move for example from $S3$ to $S2$ only depends on $S3$, and it does not depend on the way how you previously get into $S3$ i.e. the probability to move from $S3$ to $S2$ are not different if you get into $S3$ from $S1$ or from $S2$.

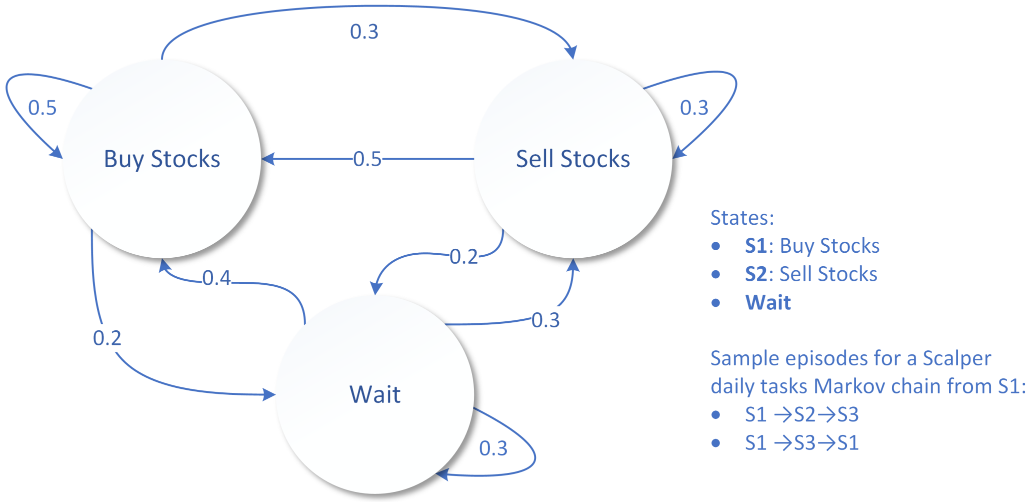


Fig. 2.7 Scalper day-life routine Markov chain.

There are many interesting applications of Markov chains in academic disciplines and industrial fields [50]. For example, Markov chains have been used in Mendelian genetics to model and predict what future generations of a gene will look like. Another example of where Markov chains have been applied is the popular children's board game Chutes and ladders. For example, in Chutes and Ladders, at each turn, a player is residing in a state in the state space (one square on the board), and from there the player has transition probabilities of moving to any other state in the state space. In fact, the transition probabilities are fixed since they are determined by the roll of a fair die. Nevertheless, the probability of moving to the next state is determined only by the current state and not by how the player arrived there and is therefore capable of being modeled as a Markov chain. Markov chains have been applied to areas as disparate as chemistry, statistics, operations research, economics, music, and, of course, finance. Markov chain is the first necessary step to get to the MDP. To build the link between them, we need to briefly introduce the Markov reward processes (MRP) and the reward function.

2.2.2.1 Markov Reward Processes

Let us start with the same scalper daily life example, but with the introduction of the rewards. As we can see in Figure 2.8, moving or staying in a state is associated with a value R. In the following paragraphs, we will introduce the value of a state (state value) and the value of an action (action value) presented at the end of this section. Now, let us first introduce MRP.

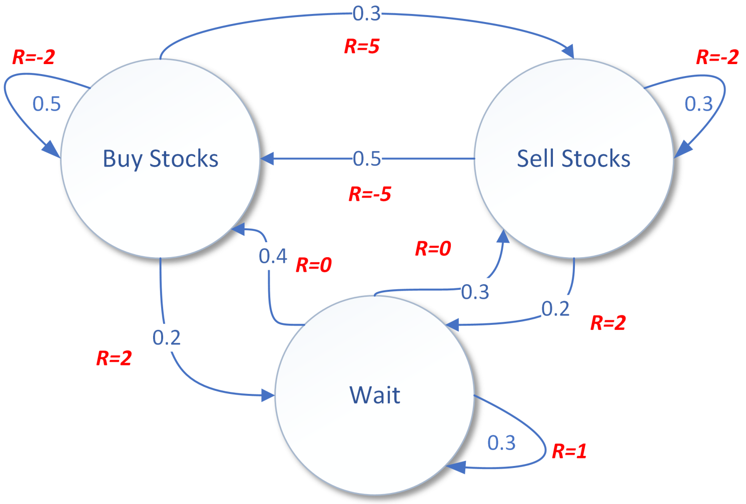


Fig. 2.8 MRP scalper daily routine MRP.

MRP is a tuple (S, P, R, γ) with two more values than the Markov chain (S, P): R is a reward function and γ is a discount factor $\gamma \in [0, 1]$. The rewards and discount factor allow us to calculate the return G_t by going deeper into the topic of risk and the returns introduced at the end of subsection 2.1.2. We can deal with a short-horizon also called a "myopic" evaluation or with a long-horizon also called a "farsighted" evaluation. Return G_t is the total discounted reward from step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=o}^{\infty} \gamma^k R_{t+k+1}$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards.
- The value of receiving reward R after $k+1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward:
  - Close to 0 leads to ”myopic” evaluation.
  - Close to 1 leads to ”far-sighted” evaluation.

Now, let's detail why Markov reward processes are discounted [1]. First of all, human behaviour shows preference for immediate reward. Secondly, the discount is useful to avoid infinite returns in cyclic Markov processes. Thirdly, it reduce the risk that uncertainty about the future may not be fully represented. Fourthly, in our study case where the reward is financial, immediate rewards may earn more interest than delayed rewards. Fifth it is mathematically convenient to discount rewards. Last but not least, it's still possible to use undiscounted Markov reward processes i.e. $\gamma = 1$.

Let's introduce two samples of returns applied to the Scalper daily routine MRP: Starting from $S1$ = Buy stocks with $\gamma = \frac{1}{2}$ with two different scalpers behaviours:

- A lazy scalper: $S1 \rightarrow S2 \rightarrow S3$ and $G_1 = 5 + 2 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} = 6.25$
- An active scalper: $S1 \rightarrow S2 \rightarrow S1 \rightarrow S2 \rightarrow S1 \rightarrow S2 \rightarrow S1 \rightarrow S3$ and $G_1 = 5 + 5 \cdot \frac{1}{2} + 5 \cdot \frac{1}{4} + 5 \cdot \frac{1}{8} + 5 \cdot \frac{1}{16} + 5 \cdot \frac{1}{32} + 2 \cdot \frac{1}{64} + 1 \cdot \frac{1}{128} = 9.88$

As we can see, entry in or exit from $S1$ or $S2$ do not produce the same exact reward overtime. The issue is to understand the long-term value of each state, this is called the value function $v(s)$:

$$v(s) = \mathbb{E}\left[G_t | S_t = s\right].$$

In figure 2.8 and with $\gamma = 1$, $v(s)$ the value function in state $S1$ is $v(S1) = 5 + 0.3 \cdot -2 + 2 + 1 \cdot 0.3 = 6.7$.

On our example this value will evolve overtime at every state transition untill the end of the day, that is the reason why we calculate the state-value function by using the Bellman equation (introduced below).

The value function can be decomposed into two parts:

- immediate reward R_{t+1},
- discounted value of successor state $\gamma . v(S_{t+1})$,

and the value function can be formulated as:

$$v(s) = \mathbb{E}\left[G_t | S_t = s\right] = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) | S_t = s\right].$$

Using matrices the Bellman equation can be expressed concisely as:

$$v = R + \gamma P v,$$

where v is a column vector with one entry per state.

The Bellman linear equation can be solved directly and it is a direct solution only for small MRP. For large MRP we may use other iterative methods such as Temporal-Difference learning or Dynamic Programming [177].

2.2.3 Markov Decision Processes Formulation

To go straight to the point, we detail in few items the MDP by comparing with an MRP.

First, a MDP is a deterministic system, satisfying the Markov property [169], when for each state and action we specify a new state and it is a controlled stochastic processes when for each state and action we specify a probability distribution over next states and we assign rewards to state transitions [20, 149].

Second, it is an environment or a framework in which all states are Markov. Third, A MDP is a tuple (S,A,P,R,γ) that is an MRP (S,P,R,γ) with decisions (which are actions) with:

- S is a finite set of states or a state space in which the process evolves.
- A is a finite set of actions that controls the dynamics of the state.
- P is a state transition probability matrix.
- $P^a_{ss'} = \mathbb{P}\ [S_{t+1} = s' | S_t = s, A_t = a]$.
- R is a reward function on transitions between states, $R^a_s = \mathbb{E}\ [R_{t+1} | S_t = s, A_t = a]$.
- γ is a discount factor $\gamma \in [0,1]$.

A distribution over actions given states is a **policy** π. The policy plays an important role because it fully defines the behaviour of the agent. Obviously, MDP policies depend on the current state not on the past (Markov property). Policies are time-independent (stationary),

$$\pi(a|s) = \mathbb{P}\ [A_t = a | S_t = s].$$

So, given an MDP, $M = \langle\ S, A, P, R, \gamma\rangle$ and a policy π, the state sequence $S_1, S_2, \cdots$ is a Markov chain $\langle S, P^\pi\rangle$ and the state reward sequence is an MRP (S,P^π,R^π,γ) where:

$$P^\pi_{s,s'} = \sum_{a \in A} \pi(a|s) P^a_{ss'},$$

$$R^\pi_s = \sum_{a \in A} \pi(a|s) R^a_s,$$

As same as in MRP, we try to understand what is the value of a state in MDP. The difference between MDP and MRP is that in MRP the state-value function $v_\pi\pi(s)$ is the expected return starting from state s, following policy π:

$$v_\pi\pi(s) = \mathbb{E}_\pi\left[G_t|S_t = s\right].$$

In MDP, the action-value function $q\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π:

$$q_\pi\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right].$$

As same as for MRP we can use the Bellman equation. The state-value function can again be decomposed into immediate reward plus discounted value of successor state.

$$v_\pi\pi(s) = \mathbb{E}\ \left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s\right].$$

Another difference between MRP and MDP is the action-value function. This function can similarly be decomposed:

$$q_\pi\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}|S_t = s, A_t = a\right]$$

In the case of a large-scale simulation-based MDP, the Bellman equation [196] allows us to determine the best possible policy without generating a probability transition matrix, as is the case in the traditional approach.

Find the "best" solution for MDP is to find the "best path" through the Markov chain. The optimal state-value function $v(s)$ is the maximum value function over all policies $v(s) = \max_\pi v\pi(s)$. This is in not necessarily the best policy, but it is the way to get the maximum reward from the system.

The optimal action-value function $q(s,a)$ is the maximum action-value function over all policies [161]. Namely, the maximum quantity of reward starting at state s and taking the action a considering all the reward taking the action a and onward. This means that the optimal value function specifies the best possible performance in the MDP:

$$q(s,a) = \max_\pi q\pi(s,a).$$

If we can determine q, we have found the optimal way to behave the MDP. Namely, an MDP is “solved” when we know the optimal value function. But even if an MDP is solved the Agent does not know yet how to behave in the system. The optimal value functions are recursively related by the Bellman optimality equations:

$$v(s) = \max_a q(s,a).$$

The bellman equation may be solved with a Temporal-Difference methods such as the Q-learning algorithm (introduced below).

2.2.4 Conclusion

This paragraph allowed to present the Markov chain, the Markov reward processes, the Markov decision processes and the concatenation between the three to address the question of the reward, the discounted factor the value of a state/action and the use of the Bellman equation to perform the state-action function. After this brief introduction of the Markov decision processes we will introduce RL, that is the way how our agent learns in our study case. Actually, MDP formally describes an environment for RL where the environment is fully observable. MDP is a key formalism for RL, where it allows us to model the way the agent learns from an environment. So in the next subsection, we introduce RL, our artificial intelligence learning approach aimed at learning policies that maximize the expected cumulative discounted reward in MDP.

2.3 Reinforcement Learning

2.3.1 Introduction

As same as for MDP, in this paragraph, we will present the reader the way we got into reinforcement learning and all the concepts we exploit in the case study. We want to present the reader our learning/exploitation decision-making path. In this paragraph we will introduce methods, functions and algorithm that help readers to understand the core of our work. We will briefly address the following methods: temporal-difference, Model-based, Model-free, On-policy, Off-policy. We will also address some functions: the reward, optimal value, optimal action-value. We will introduce more in detail the Q-Learning algorithm and the the very simple idea behind Bellman equation and its role in dynamic programming.

2.3.2 Artificial Intelligence Approaches

In artificial intelligence, there are three approaches to solve a decision-making problem depending on the availability of learning data as presented in figure 2.9.

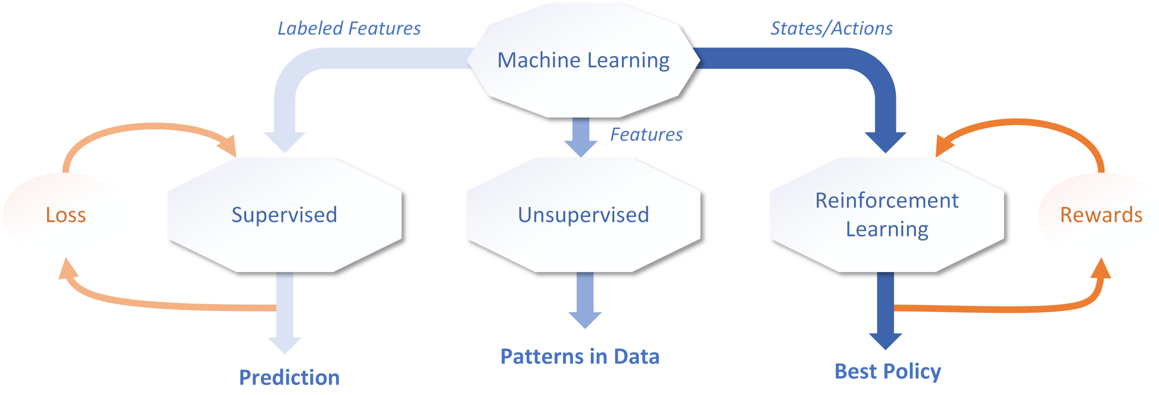


Fig. 2.9 Three types of Machine Learning: supervised/unsupervised/reinforcement learning with their application domain.

The **supervised** approach used labeled features designed to train algorithms to classify data or predict results accurately for classification and regression. Classification and regression are two loss functions that are used to evaluate the degree to which the specific algorithm models the given labeled features. The **unsupervised** approach is based on algorithms to analyze and group unlabeled data sets (features) for clustering, association, and reduction of dimensionality, which are patterns in the data. The latter is **reinforcement learning** (RL), which is similar to how humans learn to maximize the expected cumulative discounted reward to find the best policies. In fact, many of the RL algorithms are inspired by biological learning systems. RL focuses on learning how good it is for an agent to be in a state over the long run, called a value of state (such as in MRP and MDP), or how good it is to take an action in a given state over the long run, called a value of action.

In RL, there are two main approaches to estimate state-action values: model-based and model-free. Let us briefly introduce the model-based and model-free methods. First, we specify what a model is. The model signifies the transition function and the reward function in an MDP. The model simulates the dynamics of the environment and allows inference of how the environment will behave. When the sequential decision problem is modeled as a Markov decision process (MDP) [15], the agent's policy can be represented as a mapping of each state it can encounter to a probability distribution of available actions. In some cases, the agent can use its experience in interacting with the environment to estimate an MDP model and then compute an optimal policy using off-line planning techniques such as dynamic programming [77]. When learning a model is not feasible, the agent can still learn an optimal policy using **temporal-difference (TD) methods**, such as Q-Learning [176] (more detailed below), one of the widely used model-free methods. Q-learning was developed by Christopher John Cornish and Hellaby Watkins [191]. According to Watkins, 'it provides agents with the ability to learn to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build domain maps.'

2.3.2.1 Model-Based and Model-Free Methods

As said just before, **Model-based methods**, require a model of the environment, such as dynamic programming, and learn the transition and reward models from interaction with the environment and then use the learned model to calculate the optimal policy by value iteration. The model of the environment is learned from experience, and the value functions are updated by value iteration on the learned model [47]. By learning a model of the environment, an agent can then use it to predict how the environment will respond to its actions, i.e. predict the next state and the next reward given a state and an action. If an agent learns an accurate model, it can obtain an optimal policy based on the model without additional experience in the environment. Model-based methods are more sample-efficient than model-free methods, but extensive exploration is often necessary to learn a perfect model of the environment [82]. Since a model mimics the behavior of the environment, it allows us to estimate how the environments will change in response to what the agent does. However, learning a model allows the agent to perform a targeted exploration. If some states are not visited enough or are uncertain enough to learn a model correctly, this insufficiency of information drives the agent to explore more of those states. Thus, optimistic value initialization is commonly used for the exploration method. If the agent takes an action a in state s, the action value $Q(s,a)$ is updated using the Bellman equation.

Model-free methods improve the value function directly from observed experience and do not rely on transition and reward models. Value functions are learned through trial and error. These methods are simple and can have advantages when a problem is complex, making it difficult to learn an accurate model. However, model-free methods learn without a model, such as temporal difference (detailed below) and require more samples than model-based methods to learn value functions [86]. In particular in model-free methods, where the improvement of the value function comes directly from observed data and through interactions with the environment by the agent itself, the quality of the data collected is crucial. More crucial, for example, is supervised learning, where the data set is fixed. This dependence can lead to a vicious circle. If the agent collects poor-quality data. For example, referring to our study case, if the agent had collected portfolios with no associated rewards, then it would not improve and it would continue to accumulate bad portfolios.

2.3.2.2 Temporal-Difference Methods

Just before detailing the temporal difference (TD), we need to briefly introduce the on- and off-policy methods that we will use in the TD presentation. In **on-policy methods**, the agent learns the best policy and uses it to make decisions. In **off-policy methods** separate it into two policies. That is, the agent learns a policy different from what is currently generating behavior. The policy about which we are learning is called the target policy, and the policy used to generate behavior is called the behavior policy. Since learning is from experience 'off' the target policy.

TD learning is an **unsupervised technique** to predict a variable's expected value in a sequence of states. TD uses a mathematical trick to replace complex reasoning about the future with a simple learning procedure that can produce the same results. Instead of calculating the total future reward, TD tries to predict the combination of immediate reward and its own prediction of rewards at the next moment in time. Then, when the next moment arrives with new information, the new prediction is compared to what it was expected to be. If they are different, the algorithm calculates how different they are and uses this 'temporal difference' to adjust the old prediction to the new prediction. By always striving to bring these numbers closer together at every moment in time, matching expectations with reality, the entire chain of predictions gradually becomes more accurate. TD methods are guaranteed to converge in the limit to **optimal action-value function**, from which an optimal policy can be easily derived.

An example of a temporal difference method is the off-policy methods such as **Q-learning** [190], the behavior policy, used to control the agent during learning, is different from the estimation policy, whose value is being learned. The advantage of this approach is that the agent can employ an exploratory behavior policy to ensure that it gathers sufficiently diverse data while still learning how to behave once exploration is no longer necessary. However, an on-policy approach, in which the behavior and estimation policies are identical, also has important advantages. In particular, it has stronger convergence guarantees when combined with the function approximation, since off-policy approaches can diverge in that case [11, 27, 73] and it has a potential advantage over off-policy methods in its online performance, since the estimation policy, which is iteratively improved, is also the policy that is used to control its behavior. By annealing exploration over time, on-policy methods can discover the same policies in the limit as off-policy approaches.

2.3.2.3 Learning Agent Algorithm

In RL, an agent learns from the ongoing interaction with an environment to achieve an explicit goal. Such an interaction produces a lot of information on the consequences of the behavior, which helps to improve its performance. Whenever the learning agent takes an action, the environment responds to its action by giving a reward and presenting a new state. The objective of the agent is to maximize the total amount of reward that it receives. Through experience in its environment, it discovers which actions stochastically produce the greatest reward and uses this experience to improve its performance for subsequent trials. In other words, the agent learns how to behave to achieve the goals [138]. As we already mentioned, RL focuses on learning how good it is for the agent to be in a state over the long run, called a value of state, or how good it is to take an action in a given state over the long run, called a value of action. As shown in Figure 2.10, RL has different parts: Agent, Agent's action, Environment within which an agent takes actions, State or observation and rewards that an agent obtains as a consequence of an action it takes.

A reward is given immediately by the 'critic' in the environment as a response to the action of the agent, and a learning agent uses the reward to evaluate the value of a state or action. The best action is selected by the values of the states or actions because the highest value brings about the greatest amount of reward in the long run. Then the learning agent can maximize the cumulative reward it receives. A model represents the dynamics of the environment. A learning agent learns value functions with (Model-based) or without (Model-free) models. These value functions can be represented using tabular forms, but, in large and complicated problems, tabular forms cannot efficiently store all value functions. In this case,

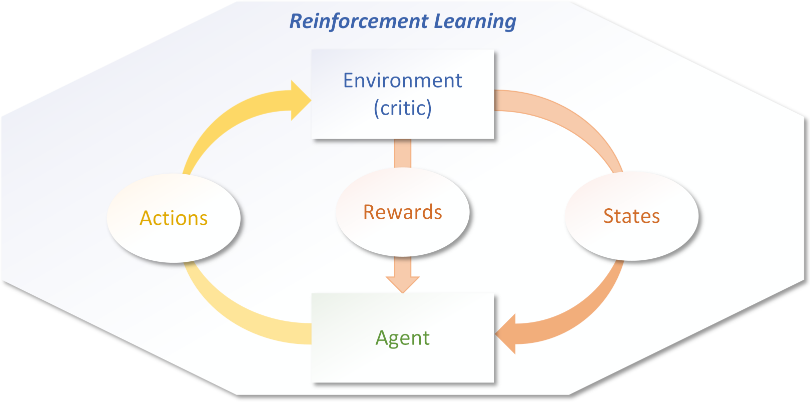


Fig. 2.10 Interaction between the Agent and the Environment in RL. The Environment sends rewards and new states in response to actions from its Agent. Adapted from Sutton [178].

the functions must be approximated by using a parameterized function representation for large problems.

The state action value function **the reward function** is also crucial. In the literature, there is a question about the importance of expert knowledge in defining the reward function. In fact, to achieve the desired behavior, expert knowledge is often required to design an adequate reward function, even if expert knowledge is incomplete and sometimes biased [187]. At this point, we may briefly distinguish between reward engineering and feature engineering. If we take into account the distinction between RL and supervised learning, where we know from the start what we want to optimize, it may be considered true that we do not know a priori the best solution to an RL problem. Thus, defining a reward function by expert knowledge of the study case field can bias the agent toward what and reduce the possibility of developing a solution as general as possible like in a black box [96]. However, the definition of a reward function should not be compared with feature engineering in supervised learning [100]. Instead, a change in the reward function is more similar to a change in the objective function [188]. This does not mean that feature engineering is not used in RL [204]. In RL, feature engineering is about how you represent state and action spaces.

There are at least two ways to define the notion of reward in an RL problem, the representation by goal or by penalty. In the representation **goal-reward**, the agent is rewarded only when it reaches a final state (goal state). This representation was used by [103]: $r(s,a) = 1$ if succ(s,a) $\in G$, 0 otherwise. In the penalty representation, the agent is penalized for each action it takes $r(s,a) = -1$. This representation has a denser reward structure than the one based on the reward per goal (the agent receives more zero reward in the latter case) if the goals are numerous.

In the next section, we introduce the Bellman equation, which is probably one of the most widely used in RL and dynamic programming.

2.3.3 Bellman Equation

The idea behind **Bellman equation** is very simple; we take the sequence of rewards from the first time step we consider, and we can break it into two parts. Namely, the value function can be decomposed into two parts: Immediate rewards R_{t+1} Discounted value of the successor state $\gamma(S_{t+1})$ at the end, all these rewards are added. The Bellman equation (Equation 2.1) states that the expected long-term reward for a given action is equal to the immediate reward of the current action combined with the expected reward of the best future action taken in the next state. This means that the value Q for one or more states and actions (a) should represent the current reward (r) plus the expected future maximum reward (γ) expected for the next state (s'). The discount factor γ makes it possible to estimate at more ($\gamma = 1$) or in the medium term ($\gamma < 1$) future values of Q. According to the Temporal Difference [177] learning technique, the matrix Q can be updated as follows:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma(\max_{a'} Q(s',a')) - Q(s,a)] \tag{2.1}$$

with $s' \in S$, $a' \in A$, $\gamma \in [0,1]$ the discount factor, and $\alpha \in [0,1]$ the learning rate. The learning rate determines how much new calculated information will outperform the old one. The discount factor γ determines the importance of future rewards. 0 would make the myopic agent consider only the current rewards, while a factor close to 1 would also involve the more distant rewards.

The variable Q allows us to decide how much future rewards can be compared to the current reward. In fact, it is a matrix composed of a number of rows equal to the number of states and a number of columns equal to the number of actions considered. The Q-Learning algorithm is used to determine, by iteration, the optimal value of the variable Q to find the best possible policy. Maximization allows us to select only the action a from all possible actions for which $Q(s,a)$ has the highest value.

2.3.4 Q-Learning Algorithm - The Brain of the Agent

There are a multitude of algorithms to resolve the Bellman equation that can be classified according to the criteria presented above. The Sutton and Barto book [178] is probably one of the most cited reference sources for accessing these algorithms. SARSA and Q-learning are two of the most widely used temporal-difference algorithms.

Why do we decide to embark on a Q-learning algorithm in our agent instead of a state-action-reward-state-action (SARSA) algorithm? The difference may be considered subtle, but first we need to introduce and briefly stress the difference between two main approaches to learning action values: **on-policy** and **off-policy**.

This question still usually represents an important challenge of RL, the exploration / exploitation dilemma. The agent has to learn the optimal policy while not acting optimally, that is, by exploring all actions. On the other hand, off-policy methods separate it into two policies. That is, the agent learns a policy different from what is currently generating behavior. The policy about which we are learning is called the target policy, and the policy used to generate behavior is called the behavior policy [109]. Since learning is from experience 'off' the target policy.

The difference is very subtle: For Q-learning, which is the algorithm of the off-policy, when passing the reward from the next state (s, a) to the current state, it takes the maximum possible reward from the new state (s) and ignores whatever policy we are using. For SARSA, which is on-policy, we still follow the policy (e-greedy), compute the next state (a), and pass the reward corresponding to that exact a back to the previous step. To reiterate, Q-learning considers the best possible case if you get to the next state, while SARSA considers the reward if we follow the current policy at the next state. Therefore, if our policy is **greedy**, SARSA and Q-learning will be the same. But we are using **e-greedy** here, so there is a slight difference. The greedy policy is characterized by an agent always choosing an action with the maximum expected return. The e-greedy policy is characterized by an agent that takes actions using the greedy policy with a probability of $1-\varepsilon$ and a random action with a probability of ε. This approach ensures that all of the action space is explored.

The Q-learning update rule [15] is just a special case of the expected SARSA update rule [176] for the case where the estimation policy is greedy.

Another good/bad reason may be that SARSA only looks up the next policy value, while Q-learning looks up the next maximum policy value. But from an algorithmic standpoint, it can be a mean, max, or best action depending on how we choose to implement it.

The two methods have advantages. We decided to use Q-Learning, which is outside the policy, because we don't need the advantages of the on-policy method, but we need the algorithm to converge to optimal Q-values as long as exploration occurs. Furthermore, in [185] the authors state that SARSA and Q-learning are expected to outperform SARSA.

At this point, we need to briefly introduce how Q-learning is implemented [44]:

1. **Initialize Q-Table** The Q-Table is initialized as a mxn matrix with all its values set to zero and where m is the size of the state space and n is the size of the action space.

2. **Define ε-Greedy Policy**. Depending on the selected epsilon parameter, the ε-greedy policy selects the action with the highest Q-Value for the given state or a random action. An epsilon of 0.50 means that 50% of the time an action will be chosen randomly, while an epsilon of 1 means that the action will always be chosen randomly (100% of the time). Theoretically, the value ε can vary during the training, exploration, and exploitation phases. However, for our study case, training is carried out with a constant epsilon value.

3. **Define the execution of an Episode**. For each episode, the agent will complete as many time steps as necessary to reach a final state. After executing it, the agent will observe the new state reached and the reward obtained, information that will be used to update the Q values of its Q table. This process is repeated for each time step until the optimal Q values are obtained. The Q-value is updated by applying the Bellman equation.

4. **Training the Agent**. At this point, only the algorithm hyperparameters must be defined, which are: learning rate α, discount factor γ, and ε. Additionally, the number of episodes that the agent must perform is specified to consider the completed training. Training execution will consist of running an execute_episode() function for each training episode. Each episode updates the Q-Table until optimal values are reached.

5. **Evaluate the Agent**. To assess the agent, rewards for each training episode, visual representations of the trained agent performing an episode, and metrics from several training events are used for the trained agent. The reward obtained during training shows the convergence of the rewards with optimal values. In our study case, as the rewards of each timestep are 0 except for the goal state.

As briefly introduced above, the mind of the agent in Q-learning is a table with the rows as the state or observation of the agent from the environment and the columns as the actions to take. Each of the cells of the table will be filled with a value called the Q-value, which is the value that an action brings considering the state in which it is in. This table is called the Q-table. The Q-table is actually the brain of the agent. What is the Q-value then? Basically, the Q-value is the reward obtained from the current state plus the maximum Q-value from the next state. The agent has to get the reward and the next state from its experience in the memory and add the reward to the highest Q-value derived from the row of the next state in the Q-table, and the result will go into the row of the current state and the column of the action, both obtained from the experience in the memory. In our study case, all states except one bring to the agent a reward equal to 0.

Figure 2.11 shows the Q-Learning algorithm of [177]. In line 3, the start state is defined before the loop (line 4) in charge of trying to reach the goal state (line 9). For each step, the Q matrix is updated according to Equation 2.1 by considering a new action a, a reward r and state s'. When the goal state is obtained, a new episode occurs.

```
1    Initialize Q(s,a)
2    Repeat (for each episode):
3       Initialize s
4       Repeat (for each step of episode):
5           Choose a from s using policy derived from Q
6           Take action a, observe r and s'
7           Update Q (according to the equation 2)
8           s <- s'
9       until s is goal state
```

Fig. 2.11 Q-Learning algorithm from [177].

Regarding the convergence of the two algorithms:

- SARSA will learn the optimal ε-greedy policy, i.e., the Q-value function will converge to an optimal Q-value function but only in the space of the ε-greedy policy (as long as each action pair will be visited indefinitely). We expect that in the limit of ε decreasing to 0, SARSA will converge to the optimal global policy. In [178] by Sutton&Barto: The convergence properties of the SARSA algorithm depend on the nature of the dependence of the policy on Q. For example, we could use ε-greedy or ε-soft policies. According to Satinder Singh (personal communication), SARSA converges with probability 1 to an optimal policy and an action value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to greedy policy (which can be arranged, for example, with ε-greedy policies by setting $\varepsilon = \frac{1}{T}$), but this result has not yet been published in the literature.

- Basically, and as pointed out in [56], the Q-learning algorithm converges in polynomial time depending on the value of the learning rates. However, if the discount factor is close to or equal to 1, the value of Q may diverge [159].

So, for the reasons described above, we use Q-learning, one of the algorithms tailored for this domain that has been used to solve many MDP problems. To welcome our decision, let us briefly present a simple example of a Q-learning problem in an MDP environment.

Our study case is Discrete Actions - Single Process Automate. Actually, in these cases, Q-learning or its RN-version deep Q-learning (DQN) and extensions or actor-critic with experienced replay (ACER) are the recommended algorithms [178].

DQN are usually slower to train (with respect to wall clock time), but are the most sample-efficient (because of its replay buffer). In addition, our environment is a reward environment for the goal; the choice of the algorithm depends on our action space. The difference between Q-learning and DQN is the brain of the agent. The agent's brain in Q learning is the Q table, but in DQN, the agent's brain is a deep neural network. We implement our research with the Q-learning algorithm, but our approach is not restrictive and can be implemented with DQN.

Two other techniques are also essential for the formation of the DQN:

- Experiment Replay: Since the training samples in a typical RL setup are highly correlated and less data-efficient, this will make convergence more difficult for the network. One way to solve the sample distribution problem is to adopt experience proofreading. Essentially, the sample transitions are stored, which will then be randomly selected from the 'transition pool' to update the knowledge.

- Separate Target Network: The Q target network has the same structure as the value estimator. Each step of C, according to the above pseudocode, the target network is reset to another. Therefore, the fluctuation becomes less severe, resulting in more stable formations.

Although DQN has achieved enormous success in higher-dimensional problems (such as the Atari game), the action space remains low-key. However, in many tasks of interest, especially physical control tasks, the action space is continuous. If the action space is discretized too finely, the action space becomes too large. For example, suppose that the degree of free random system is 10. For each degree, divide the space into 4 parts. You end up having $4^{10} = 1048576$ shares. It is also extremely difficult to converge for such an action space. The deep deterministic policy gradient (DDPG) [172] borrows the ideas of replay of the experience and separates the target network from the DQN. A problem with DDPG is that it rarely performs action exploration. A solution to this is to add noise on the parameter space or on the action space.

2.3.5 Explainable Artificial Intelligence in Reinforcement Learning

AI applications are blossoming in today's world, and the weight of decision-making decisions delegated to AI is an all-time high. Given this large amount and the scrutiny decisions that must be made, it is extremely important to be clear on why a particular decision was made. explainable artificial intelligence (XAI), that is, the development of more transparent and interpretable AI models, has gained greater traction over the past few years.

Why is explainability so crucial? First, explainability is critical to building confidence in disruptive technologies for decision makers, academics, and the public and has been identified as a key component in both increases in user participation. So, there is one psychology-related reason: "if users do not trust a model or a prediction, they will not use it" [155]. Explainability is key for decision makers interacting with AI to understand AI's conclusions and recommendations. **Trust** is an essential prerequisite for the use of a model or system [87]. There are situations where decision-makers may not have full access to the decision-making process that an AI may undergo. To be clear, we use an example financial investment algorithm. AI offers recommendations for portfolio management. These recommendations need to be transparent, and decisions need to be justifiable. Decision makers should be able to adequately explain why these recommendations are made and what data are used to make them, as well as the reasoning behind the recommendation, to provide sufficient explanations for them. **Transparency** also justifies the decisions of the system and makes them fair and ethical. And even confidence does not mean that the decision is based on reasonably learned data. Therefore, decision-making processes must be reviewable, especially if AI is working with highly uncertain data. In terms of reviewability, there is also a legal element that needs to be taken into account in terms of reviewability; the EU general data protection regulation (GDPR) [55], which came into effect in May 2018, aims to ensure a 'right to explanation' [72] regarding automated decision making and profiling. It states that "[...] such processing should be subject to suitable safeguards, which should include [...] the right to obtain human intervention [...] and an explanation of the decision reached after such assessment" [55]. Furthermore, the European Commission established an AI strategy with transparency and accountability as important principles to be respected [40], and in its Guidelines on trustworthy AI [41], they state seven key requirements, with transparency and accountability as two of them.

Second block of reasons, AI technologies have become an important part in almost cyber-physical systems (CPSs) domains. CPSs are systems in which all behaviors originate from machine learning, including RL. In this second block of reason, we can include the aim for increased efficiency and the need to accommodate volatile parts of today's critical business such as porfolio management or Smart Grid such as a high share of energy sources. Over time, AI technologies expanded from being an added input to an otherwise thoroughly defined control system to increase the state of awareness of CPSs. AlphaGo is probably the most widely known representative of the last category [173]. Moreover, despite the increasing efficiency and versatility of AI, its incomprehensibility reduces its usefulness, since "incomprehensible decision making can still be effective, but its effectiveness does not mean that it cannot be faulty" [112] We also have to bear in mind and consider the fact that, nowadays, AI can act increasingly autonomous, explaining and justifying the decisions is now more crucial than ever, especially in the domain of RL where an agent learns by itself, without human interaction.

XAI methods can be categorized based on two factors; first, based on when the information is extracted, the method can be intrinsic or post hoc, and second, the scope can be global or local. Global and local interpretability are meant to refer to the scope of an explanation; global models explain the entire behavior of the model, and local models explain specific decisions. Global models try to explain the entire logic of a model by inspecting the structure of the model [135]. Local explanations attempt to answer the question: 'Why did the model make a certain prediction/decision for an instance/for a group of instances?' [2]. They also try to identify the contributions of each characteristic in the input towards a specific output [51]. Furthermore, global interpretability techniques lead to users trusting a model, while local techniques lead to trusting a prediction [51]. Post hoc versus Intrinsic Interpretability depend on the time when the explanation is extracted/generated; An intrinsic model is an ML model that is designed to be inherently interpretable or self-explanatory at the time of training by constraining the model complexity, for example, decision trees [51].

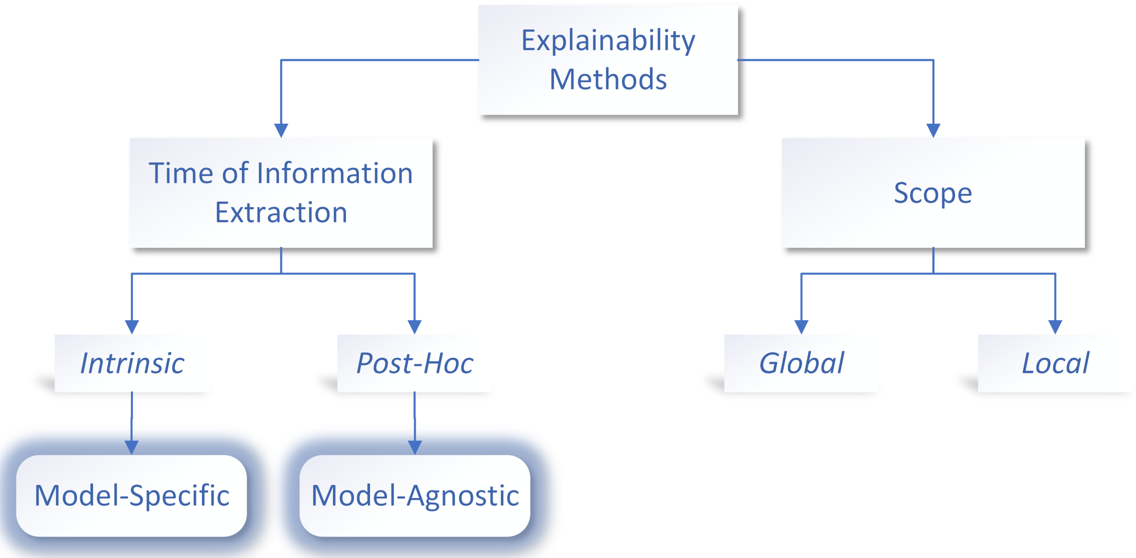


Fig. 2.12 XAI toxonomy in RL (adapted from [2]).

In contrast, post hoc interpretability is achieved by analyzing the model after training by creating a second, simpler model, to provide explanations for the original model [51], such as surrogate models or saliency maps [2].

The two most representative post hoc models in the literature are LIME [155] and SHAP [116], and they are based on two completely different mechanisms. While LIME uses feature perturbations to build a linear surrogate model out of it (such as a decision tree), SHAP is based on game theory, where predictions are explained by assuming that each feature value of the instance is a 'player' in a game where prediction is the payment. SHAP uses Shapley values, which is a method of fairly distributing 'payment' between different characteristics [132].

Just like the models themselves, these interpretability models also suffer from a transparency-accuracy-trade-off; intrinsic models usually offer accurate explanations, but, due to their simplicity, their prediction performance suffers. In contrast, post hoc interpretability models usually keep the accuracy of the original model intact, but are harder to derive satisfying and simple explanations from [51].

Another distinction, which usually coincides with the classification into intrinsic and post hoc interpretability, is the model-specific or model-agnostic classification. Techniques are model-specific if they are limited to a specific model or class of models [132], and are model-agnostic if they can be used on any model [132]. As you can also see in Figure 2.12, intrinsic models are model-specific, while post hoc interpretability models are usually model-agnostic.

2.3.6 Conclusion

This paragraph allowed us to address the question of methods such as temporal difference, model-based, model-free, on-policy, and off-policy, which will be necessary for our study case and for future work. We also addressed some crucial functions such as reward, reward function, optimal value, and optimal action value that we will exploit in our study case. As well, we introduced in detail Q-Learning and the decision why we preferred Q-Learning to SARSA algorithm as well as the very simple idea behind Bellman equation and its role in dynamic programming. Finally, we detail explainable artificial intelligence in RL, which will be a useful concept for comparing DEVS decision-making models with the efficient semivariance model and the combination between DEVS and LSTM.

In the next section, let us introduce our decision-making formalism, which is the discrete event system specification (DEVS).

2.4 Discrete-Event System Specification Formalism

When one wants to represent a complex system, it is customary to define a model. The model makes it possible to formalize the behavior of a system by specifying a set of rules, most often using mathematics. The modeling of a system leads to making it manipulable by the human being because he then has a model with which he can experiment ad infinitum. The advantage of owning a model is that it can be simulated. These simulations make it possible to put the system under experimental conditions in order to observe its behavior. A valid model is a model which, when simulated, perfectly reproduces the behavior of the system it represents.

Modeling a system requires the use of a description formalism. The choice of formalism is conditioned by the representation of space, time, and the states of a system. During the modeling process, it is important to know whether the time and states representing the system are considered discrete or continuous. In the same way, it is important to know if the concept of space is to be taken into consideration in the evolution of the system and if so if it is modeled in a discrete or continuous way. When a system is described by considering its states and time in a continuous way, it can be modeled using a formalism such as classical differential equations. When the notion of space must also be taken into account, it is preferable to use partial differential equations. When the change of state of a system is discrete and time is considered continuous, the formalism used can be that of discrete events regardless of how the space is considered (discrete, continuous, or absent).

The choice of formalism (and, therefore, the way in which states, time and space are considered) is often conditioned by the scientific culture of the modeller. Traditionally, a complex system is modeled by considering its evolution continuously in time (or in space), and its behavior is described from differential equations which are solved analytically or numerically depending on the complexity of the model. In this type of approach, the evolution of the system (of these states) is modeled independently of the time scale, and it can be said that the resolution of the model is done with a continuous simulation approach. This modeling method applies very well to systems whose states evolve continuously in time (or space) and whose states must be observable at any instant. However, some systems evolve in time only at specific instants and it is not necessary to know precisely their behavior between these instants. They evolve in a discrete way, and only their observations matter at these precise moments when they change state. In this case, discrete event modeling and simulation can be used to study these systems. It is important to note that the choice of formalism also depends on the discrete or continuous nature of the system that one wishes to model. Continuous simulation will be preferred in the case of the study of a model of population dynamics in an ecosystem due to the presence of continuous state variables such as the speed of movement of individuals. On the other hand, discrete event simulation will be preferred for modeling an industrial production chain system, in which it is important to know the evolution of a product by stages.

The theory of modeling and simulation (TM&S) and the DEVS formalism were introduced by Zeigler in the 1970s [202] to model discrete event systems in a hierarchical and modular way. Combining a formal approach and general theory of systems, it had a wide echo in the 1990s and 2000s in the French community. Today, there are around 10 French-speaking laboratories that develop tools and applications based on the (TM&S) formalization, namely the DEVS formalism and its extensions. This formalism is based on the general theory of systems [186]. It is common for the DEVS formalism, in its original form Zeigler [201], to be adapted and extended to be replaced in more specific contexts of an application domain. This is, for example, the case when it comes to modeling differential equations [104].

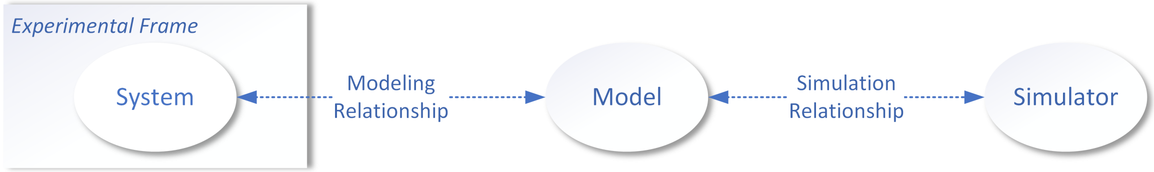


Fig. 2.13 DEVS experimental frame with the model and the simulator interacting through the modeling and the simulation relationships.

Professor Zeigler has proposed a conceptual architecture for modeling and the simulation of systems, particularly suited to the DEVS formalism. As shown in Figure 2.13, this architecture has three entities:

- **The system**: it is the phenomenon observed in a given environment. The environment provides the specifications of the conditions under which the system is being operated and allows for its experimentation and validation.
- **The model**: it is the representation of the system generally based on the set of definitions of instructions, rules, equations, and constraints that allow the generation of a behavior after simulation. The model defines the behavior and structure of a system that evolves in a given environment.
- **The simulator**: it is an entity that is responsible for interpreting the model (executing these instructions) to generate its behavior.

These entities are linked by two relationships:

- **The modeling relationship**: it is made up of construction rules and model validation.

- **The simulation relationship**: it is made up of the model execution rules that ensure that the simulator generates the expected behavior of the system from the model.

The explicit separation between the entities allows one to benefit from several advantages such as simulating a model with different types of simulators or different types of environment. The enthusiasm for object-oriented programming in the early 1980s led Professor Zeigler to use an object approach to define his formalism. DEVS formalizes what a model is, what it must contain and what it does not contain (experimentation and simulation control parameters are not contained in the model). Moreover, DEVS is universal and unique for discrete-event system models. Any system that accepts events as input over time and generates events as output over time is equivalent to a DEVS model. DEVS allows for automatic simulation on multiple different execution platforms, including those on desktops (for development) and those on high-performance platforms (such as multicore processors). With DEVS, a large system model can be decomposed into smaller component models with couplings between them. DEVS formalism defines two kinds of model: (i) atomic models that represent the basic models providing specifications for the dynamics of a subsystem using function transitions and (ii) coupled models that describe how to couple several component models (which can be atomic or coupled models) together to form a new model. This hierarchy inherent to the DEVS formalism can be called a description hierarchy, allowing the definition of a model using hierarchical decomposition. It should be pointed out that this kind of hierarchy does not involve any abstraction-level definition since the behaviors of all implied models are defined at the same level of abstraction. However, as a hierarchy of description, the hierarchical decomposition in DEVS may still be regarded as a kind of abstraction: In top-down design, modelers can consider couplings and interfaces between models without considering details of internal components.

2.4.1 DEVS Atomic Model

A classic DEVS AM atomic model (Figure 2.14) with the behavior is represented by the following structure:

$$AM = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

where:

- $X : \{(p,v)|(p \in inputports,\ v \in X_p^h)\}$ is the set of input ports and values.
- $Y : \{(p,v)|(p \in outputports,\ v \in Y_p^h)\}$ is the set of output ports and values.
- S: is the set of states.

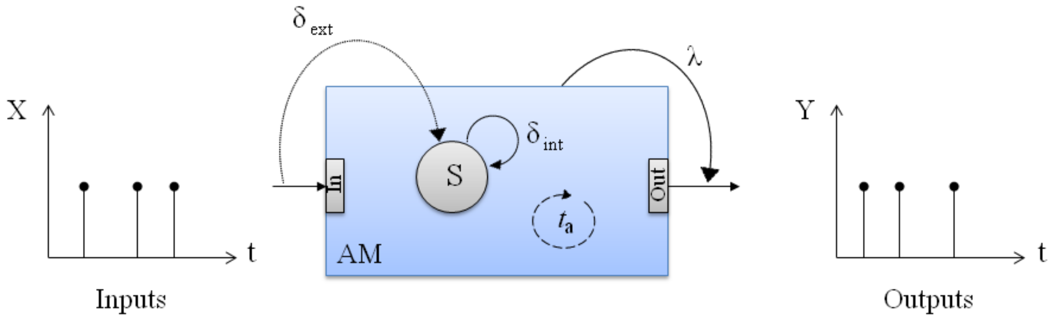


Fig. 2.14 Classic DEVS Atomic Model in action.

- $\delta_{int} : S \rightarrow S$ is the internal transition function that will move the system to the next state after the time returned by the time advance function.
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function that will schedule changes in the states in reaction to an external input event.
- $\lambda : S \rightarrow Y$ is the output function that will generate external events just before the internal transition takes places.
- $t_a : S \rightarrow \mathbb{R}^+_\infty$ is the time advance function that will give the life time of the current state.

An atomic DEVS model can be considered as an automaton with a set of states and transition functions that allow the state to change when an event occurs or not. When no events occur, the state of the atomic model can be changed by an internal transition function, as noted δ_{int}. When an external event occurs, the atomic model can intercept it and change its state by applying an external transition function as noted δ_{ext}. The lifetime of a state is determined by a time advance function called t_a. Each state change can produce an output message via an output function called λ.

The dynamic interpretation is the following.

- $Q = \{(s,e) | s \in S^h, 0 < e < t_a(s)\}$ is the total state set.
- e is the elapsed time since the last transition and s the partial set of states for the duration of $t_a(s)$ if no external event occur.
- δ_{int} : the model being in a state s at t_i, it will go into $s' = \delta_{int}(s)$ if no external events occur before $t_i + t_a(s)$.
- δ_{ext} : when an external event occurs, the model being in state s since the elapsed time e enters s'. The next state depends on the elapsed time in the present state. At every state change e is reset to 0.

- λ : the output function is executed before an internal transition before emitting an output event, and the model remains in a transient state.
- A state with infinite lifetime is a passive state *(steady state)*, otherwise it is an active state *(transient state)*. If the state s is passive, the model can evolve only with the occurrence of an input event.

We will give the DEVS specifications of an atomic model named EXEC which represents a system that becomes 'active' (s_1) when it receives input x_i on an input port p_1, processes these data for a time 'proc', and then generates outputs y_i on an output port p_2. If an external event occurs while the system is in the active state, its lifetime will be reduced by the time elapsed since the last change in state (e). If no event occurs, the system remains in a passive state (s_2). The system is in an initial 'passive' state.

- $X = \{(p_1, x_i) | i \in \mathbb{R}^+\}$
- $Y = \{(p_2, y_i) | i \in \mathbb{R}^+\}$
- $S = \{s_1, s_2\}$
- $\delta_{int}(S)$:

```
If s is $s_1$ then
    $s \leftarrow s_2$ during $t_a(s_2) \leftarrow \infty$
else pass
```

- $\delta_{ext}(QxS)$:

```
If s is $s_2$ then
    $s \leftarrow s_1$ during $t_a(s_1) \leftarrow proc$
else
    $t_a(s_1) \leftarrow t_a(s_1) - e$
```

- $\lambda(S)$:

```
If s is $s_1$ then
    send $(p_2, y_i)$
```

- $t_a(S)$:

```
if s is $s_1$ then
    return $proc$
```

```
12. else
13.     return $\infty$
```

It often happens that the time advance function ($t_a(S)$) is replaced by the manipulation of an attribute of the atomic model called sigma (σ). The description above is then written in a more condensed way for δ_{int}, δ_{ext}, and t_a:

- $\delta_{int}(S)$:

```
1. if s is $s_1$ then
2.     $s \leftarrow s_2$ during $\sigma \leftarrow \infty$
3. else pass
```

- $\delta_{ext}(QxS)$:

```
4. if s is $s_2$ then
5.     $s \leftarrow s_1$ during $\sigma \leftarrow proc$
6. else
7.     $t_a(s_1) \leftarrow t_a(s_1) - e$
```

- $t_a(S)$: return σ

It is quite common to represent the behavior description using an automaton (Figure 2.15). The automaton starts in state s_2 with an infinite lifetime. When an event arrives at the input port, the state changes in s_1 for a time *proc*. If an external event occurs during this time, the state does not change, but the lifetime is updated by considering the time that has passed since the last change in state (e).

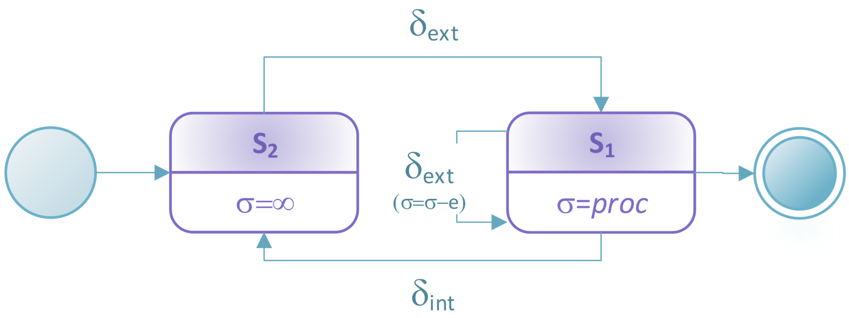


Fig. 2.15 Finite-state machine of the EXEC DEVS atomic model.

The state trajectory allows us to trace the changes in state according to the input events (Figure 2.16). When an event occurs at time t_1, the system goes from the passive state s_2 to the active state s1 for a time *proc*. When time $t_1 + proc$ elapses, the model generates an exit event. The input event that occurs at time t_3 does not change the state of the system (s_1), but the lifetime of the latter is updated according to e.

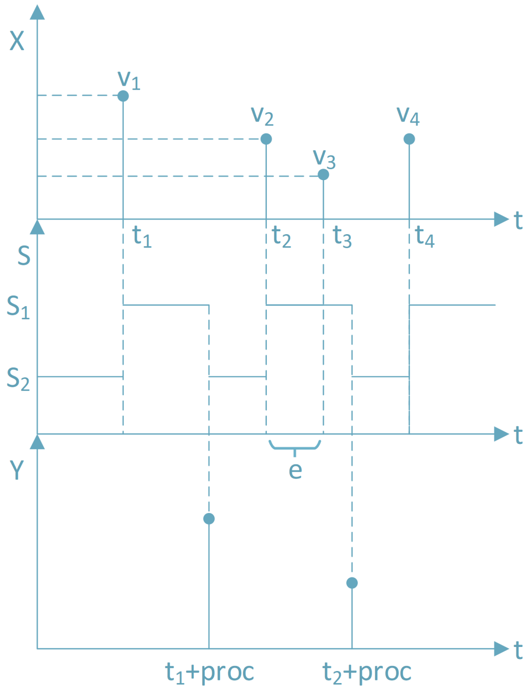


Fig. 2.16 State trajectory of the EXEC DEVS atomic model after simulation.

2.4.2 DEVS Coupled Model

The DEVS coupled model CM is a structure:

$$CM = < X, Y, D, \{M_d \in D\}, EIC, EOC, IC >$$

where:

- X is the set of input ports for the reception of external events.
- Y is the set of output ports for the emission of external events.
- D is the set of components (coupled or basic models).
- M_d is the DEVS model for each $d \in D$.
- EIC is the set of input links that connects the input of the coupled model to one or more of the inputs of the components that it contains.

- EOC is the set of output links that connect the output of one or more of the contained components to the output of the coupled model.
- IC is the set of internal links that connect the output ports of the components to the input ports of the components in coupled models.

In a coupled model, an output port from a model $M_d \in D$ can be connected to the input of another $M_d \in D$, but cannot be connected directly to itself.

Consider a coupled model CM_1 composed of three atomic models AM_1, AM_2 and AM_3 and another coupled model CM_2 composed of two atomic models AM_4 and AM_5. To simplify the example, we assign a single pe_1 input port and a single ps_1 output port to each of the models. Figure 2.17 shows the coupled model CM_1.

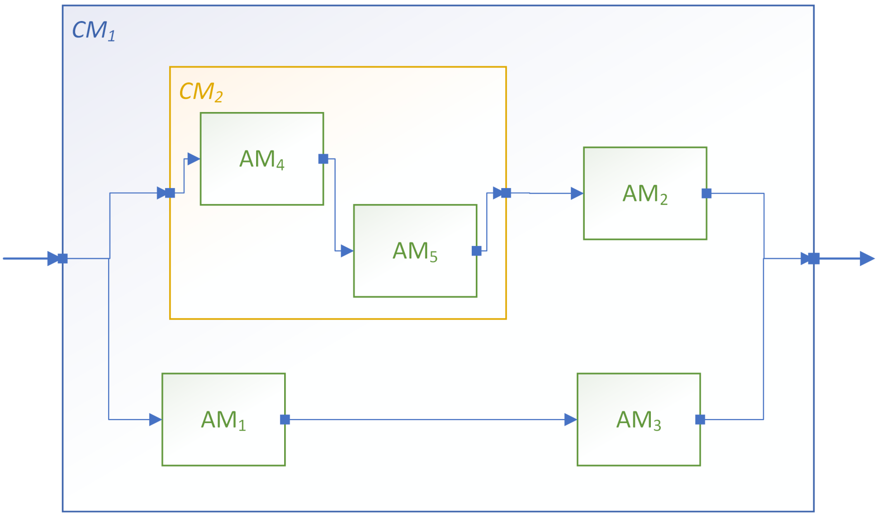


Fig. 2.17 An example of a DEVS coupled model.

The DEVS specification of the coupled model of figure 2.17 is as follows:

- $X = \{(p_1, x_i) | i \in \mathbb{R}^+\}$
- $Y = \{(p_2, y_i) | i \in \mathbb{R}^+\}$
- $D = \{AM_1, MA_2, MA_3, AM_4, AM_5, CM_2\}$
- $EOC = \{((AM_2, ps_1), (CM_1, ps_1)), ((AM_3, ps_1), (CM_1, ps_1))\}$
- $IC = \{((AM_1, ps_1), (AM_3, pe_1)), ((CM_2, ps_1), (AM_2, pe_1))\}$
- $EIC = \{((CM_1, pe_1), (CM_2, pe_1)), ((CM_1, pe_1), (AM_1, pe_1))\}$
- $select = (CM_2, AM_1, AM_2, AM_3)$

In the example above, during a possible execution conflict, the coupled model CM_2 has priority over AM_1 which has priority over AM_2 which has priority over AM_3. This order of precedence is implemented in the $select$ function.

The DEVS formalism ensures that any coupled model is equivalent to a single-atomic model. In fact, an atomic model can be decomposed into several atomic submodels organized into several coupled models. The modeler organizes these models hierarchically as he wishes in models coupled with the desired level of description.

2.4.3 DEVS Simulator

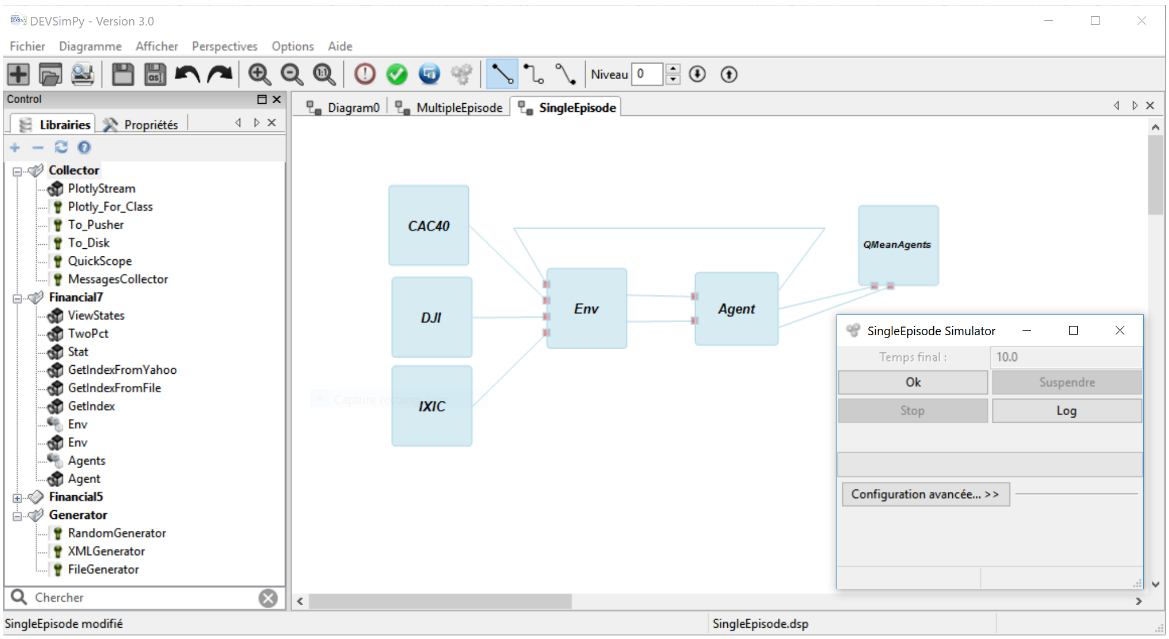
A simulator is associated with the DEVS formalism to exercise the instructions of coupled models to actually generate its behavior. The architecture of a DEVS simulation system is derived from the abstract simulator concepts associated with the hierarchical and modular DEVS formalism. Parallel DEVS (PDEVS) essentially extends classic DEVS by allowing bags of inputs to the external transition function. Bags can collect inputs that are built on the same date and process their effects on the outputs that will result in new bags. This formalism offers a solution to manage simultaneous events that could not be easily managed with classic DEVS. In PDEVS, the notion of event bag has been added. This notion integrates the fact that several events that occur at the same time can be grouped together in a bag. The set is denoted X^b. Similarly, an atomic model can output multiple events at the same time. The output set is then denoted by Y^b. The function $\delta_{conv}(SX^b)$ is introduced to resolve the runtime conflict between the inner and outer transition functions of an atomic model with the particular case where $\delta_{conv}(S) = \delta_{int}(S)$. The association of these two notions (bag and δ_{conv}) makes it possible to manage collisions between the internal and external transition functions and at the same time process several events arriving at the same time on an atomic model. In the classical DEVS formalism, each time an event arrives, the external transition function is invoked. Therefore, there were as many invocations as there were simultaneous messages on the ports of an atomic model. With this extension, simultaneous events are available in the single invocation of the transition function.

2.4.4 DEVSimPy Environment

DEVSimPy [34] (DEVS Simulator in Python language) is an open source project (under GPL V.3 license) supported by the SPE team of the university of Corsica Pasquale Paoli. This objective is to provide a GUI for modeling and simulation of PyDEVS [114] models. PyDEVS is an application programming interface (API) that allows the implementation of the DEVS formalism in Python language. Python is known as an interpreted, very high-level, object-oriented programming language widely used to quickly implement algorithms without focusing on code debugging [146].

The DEVSimPy environment has been developed in Python with the wxPython [153] graphical library without strong dependence on the scientific Python libraries than the scientific Python libraries Scipy [93] and Numpy [139] scientific Python libraries. The basic idea behind DEVSimPy is to wrap the PyDEVS API with a GUI allowing significant simplification of handling PyDEVS models (like the coupling between models or their storage). Figure 2.18 shows the general interface of the DEVSimPy environment. A panel on the left (left part of Figure 2.18) shows the libraries of DEVSimPy models.

The user can instantiate the models using drag-and-drop functionality. The right part of Figure 2.18 shows the modeling part based on a canvas with the interconnection of instantiated models. This canvas is a diagram of atomic or coupled DEVS models waiting to be simulated.



Temps final : 10.0

Fig. 2.18 DEVSimPy general interface. The left panel shows the library panel with all atomic or coupled DEVS models that are instantiated in the right panel to create a diagram of a simulation model. The dialogue window on the bottom-right allows one to simulate the current diagram by assigning a simulation time.

A DEVSimPy model can be stored locally on the hard disk or in cloud through the Web in the form of a compressed file including the behavior and the graphical view of the model separately. The behavior of the model can be extended using specific plug-ins embedded in the DEVSimPy compressed file.

DEVSimPy capitalizes on the intrinsic qualities of the DEVS formalism to automatically simulate models. The simulation is carried out by pressing a simple button, which invokes an error checker before the building of the simulation tree. The simulation algorithm can be selected among hierarchical simulators (default with the DEVS formalism) or direct coupling simulators (most efficient when the model is composed of DEVS coupled models).

Simpy **??** is an open source process-based DES package available in Python widely used for M&S. Another example of an open source Python-based library is OR-Gym **??**, which provides a Python-based RL environment consisting of old-fashioned operations and optimization optimization problems. Commercial software packages have embraced modeling features with RL in recent years but there is a lack of M&S software for the RL system. Our work also contributes to the literature by demonstrating how to connect DEVSimPy, to a RL model. This work will create a way of research to model and simulate a RL agent-environment in a discrete event manner with DEVS.

2.4.5 Conclusion

We assert that, in terms of explainability and modularity, the DEVS formalism offers a real advantage. It allows us to describe the system from an unambiguous semantics of modeling (model) and simulation (simulator). In contrast, modeling a heterogeneous complex system may require the use of several differential or algebraic equations, an automate or Petri network, continuous and discrete evolution over time.

Thanks to its properties of building models by composition, its hierarchy of description and abstraction, and its explicit separation between a model and simulator, which is generated automatically, DEVS allows an M&S of systems with an effective formal and generic approach approved for more than 40 years in several fields of application.

DEVS allows an approach to building its models that can be based on design patterns (set of DEVS atomic or coupled models defined to model/simulate systems) and building blocks (well-known notion of directly reusable coupled models and containing a set of interconnected submodels). These two notions (in addition to the management of state duration specific to DEVS) are highlighted in [200] and make DEVS an excellent language for Internet of Things (IoT) systems, for example.

2.5 Conclusion

This paragraph underlined how our research field, which combines the DEVS formalism and Markov decision processes, is juvenile in the scientific literature. It also pointed out that the DEVS formalism has been underexploited in the decision-making literature. It also outlined the hypothesis that, in the relevant finance decision-making literature, our research may represent a new benchmark. Furthermore, this paragraph sought to review the relevance of volatility in asset management. The paragraph also contributes to the detailed concepts, methods, and algorithms related to financial decision making when volatility plays a relevant role. This paragraph also stressed some topics from the literature that would be useful to explore in depth in future research.

Chapter 3

DEVS-Based Reinforcement Learning System Modeling and Simulation

3.1 Introduction

M&S, and AI, are two domains that can complement each other. For example, AI can help the "simulationist" in modeling complex systems that are impossible to represent mathematically [137]. On the other hand, M&S can help AI models that did not handle complex systems due to the lack of simple or unworkable heuristics.

Systems that already use AI, such as digital supply chains, "smart factories", and other industrial processes in Industry 4.0, will inevitably need to include AI in their simulation models [61]. For example, with simulation analysis systems, AI components can be directly integrated into the simulation model to enable testing and forecasting. In [129] the authors use a recursive learning algorithm (Q-Learning) to combine a dynamic load balancing algorithm and a bounded-window algorithm for discrete-event simulation of VLSI circuits (Very Large Scale Integration) at the gate level.

Machine learning is a type of AI that uses three types of algorithms (Supervised Learning, Unsupervised Learning, Reinforcement Learning) in order to build models that can get an input set to predict an output set using statistical analysis. Although simulation and ML help to model and provide optimal solutions in various problems, they never work together. The main reason is that simulation is process-centric, while ML is data-centric. To build a simulation model, knowledge of the process (system behavior) is essential, and it is also necessary to get closer to the people in charge of the process because only they know how it works. It is also necessary to acquire the data of the system and observe its evolution to be modeled. To build an ML model, only obtaining the data is essential. Often, data is analyzed to find a correlation before applying ML algorithms. Very little (if any) situational awareness is required from the modeler.

Therefore, building simulation models and ML models will provide different skills from scientists. Why combine ML and simulation? The two successful techniques will probably solve problems that have been impossible to solve separately so far. Several ideas can be proposed:

- A factory does not have documented rules and policies to run its operations. Everyone tells you otherwise. It is impossible to model heuristics with the traditional explicit simulation approach. Use a decision tree model to indicate the rules by looking at historical data.

- People or process docs describe how they behave, but they do not tell you their actual behavior. Again, an ML model can be used to determine historical heuristics.

- Path finding algorithms are not good enough to navigate in a simulation environment. Reinforcement learning helps to find optimal paths. Reinforcement learning is an ML training method based on rewarding desired behaviors and/or punishing undesired ones.

- It is difficult to match the output of the simulation model with real performance indicators. An ML model can "steer" the simulation to match reality.

- The real system implements advanced ML algorithms to make decisions. To simulate such a system, you must be able to reproduce the ML algorithms in the simulation itself.

There are probably many more possibilities, and as with most innovations, some will only reveal themselves when we start experimenting with them.

Basically, three ways to integrate ML techniques inside simulation models can be observed:

- **ML before the simulation**: ML algorithms can be used as input data for the simulation model. The most obvious approach would be to develop data-driven decision heuristics that agents can apply.

- **ML inside the simulation**: Make the simulation learn certain aspects and apply this learning directly. This coupled approach makes sense when training ML algorithms on specifics of the simulation model itself, such as pedestrians trying to find a path in the environment of the simulation model. There could be three ways to further split:

(1) train the ML algorithm as part of the simulation, (2) reuse previously trained ML models, or (3) train the ML model as the simulation progresses (i.e., for Reinforcement Learning).

- **ML after the simulation**: Taking the output of the simulation and feeding it into an ML algorithm. There is little use of this on small scales currently. However, this can be used in very advanced AI algorithm training, such as autonomous driving. The models are not only driven with real cars, but the AI can "drive" through the simulation cities. This takes the concept of simulation models as testbeds of algorithms.

In the other direction, it is possible to imagine the benefit of simulation for ML and, more specifically, for RL. In the case of RL, the discrete event approach has its place in an architecture defined to determine an optimal policy in a communication scheme by event between an agent and an environment. The DEVS formalism is an ideal solution for implementing an RL algorithm such as Q-learning because it makes it possible to represent and carry out the learning of the system by simulation in a formal, modular, and hierarchical framework. This thesis presents the benefits of DEVS formalism aspects to assist in the realization of ML models with a special focus of RL.

3.2 Discrete Event Modeling and Simulation for Machine Learning

Figure 3.1 shows some possible integration of M&S aspects into ML framework.

For the simulation part, Monte Carlo simulation is often used to solve ML problems using a "trial-and-error" approach. Monte Carlo simulation can also be used to generate random outcomes from a model estimated by some ML technique. ML for optimization is another opportunity for integration into simulation modeling. Agent-based systems often have many hyperparameters and require significant execution times to explore all their permutations to find the best configuration of models. ML can speed up this configuration phase and provide more efficient optimization. In return, simulation can also speed up the learning and configuration process of AI algorithms. Simulation can also improve the experimental replay generation in RL problems where the agent's experiences are stored during the learning phase. RL components can be developed to replace rule-based models. This is possible when considering human behavior and decision-making. For example, in [60] the authors consider that by observing a desired behavior, in the form of outputs produced in response to inputs, an equivalent behavioral model can be constructed (Output Analysis in Figure 3.1). These learning components can be used in simulation models to reflect the actual system or to train

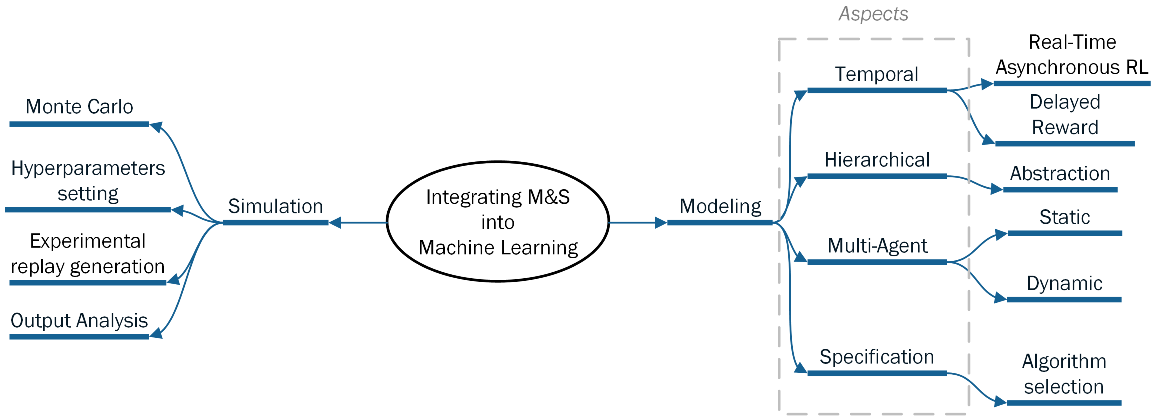


Fig. 3.1 M&S for Machine Learning. Modeling part highlight important aspects related to discrete event M&S that brings benefits as temporal aspect for delayed reward notion in RL for example. Simulation part point out that it can be used to improve the output analysis or to facilitate the Monte Carlo process realization.

ML components. By generating the data sets needed to learn neural networks, simulation models can be a powerful tool for deploying recursive learning algorithms.

Concerning the modeling part, many modeling aspects can be highlighted to help ML implementation:

- **Temporal**: Basically, the temporal aspect is implicit in the RL models. For instance, MDPs consider the notion of discrete and continuous time in order to consider the life time of a state. In RL models, the introduction of time in the awarding of rewards allows the modeling of a non-immediate response of a system. The use of the notion of time in the RL models also makes it possible to perform asynchronous simulations from real data.

- **Hierarchical**: The abstraction hierarchy allows the modeling of RL systems with different levels of detail. This makes it possible to determine optimal policies by levels of abstraction, and therefore to have more or less precise policies depending on the level chosen by the user.

- **Multi-Agent**: Multi-agent modeling can be used as part of RL where the optimal policy is based on a communication between an environment and one or more agents that are instantiated in a static or dynamic way.

- **Specification**: The assembly of all the ML pieces needed to solve problems can be a daunting task. There are many ML algorithms to choose from, and deciding where to start can be discouraging. Using specifications based on model libraries and system input analysis, the choice of the appropriate algorithm becomes simpler.

This thesis presents the benefits of hierarchical and specification aspects of the DEVS formalism to assist in the realization of RL models. Let us now see how the DEVS formalism makes it possible to represent and simulate in a generic manner an RL system.

3.2.1 Discrete Event System Specification and Reinforcement Learning

AI learning techniques have already been used in a DEVS simulation context. In fact, in [160] the authors propose the integration of some predictive algorithms for automatic learning into the DEVS simulator to considerably reduce simulation execution times for many applications without compromising accuracy. In [183], the comparative and concurrent DEVS simulation is used to test all possible configurations of the hyperparameters (momentum, learning rate, etc.) of a neural network. In [164], the authors present the formal concepts underlying the DEVS Markov models and how they are implemented in MS4Me software [203]. Markov concepts of states and state transitions are fully compatible with the DEVS characterization of discrete event systems. In [151], temporal aspects have been considered in generalized semi-MDPs with observable time and a new simulation-based RL method has been proposed.

More generally, the DEVS formalism can be used to facilitate the development of the three traditional phases involved in a learning process by strengthening a given system (Figure 3.2):

- The Data Analysis phase consists of an exploratory analysis of the data which will make it possible to determine the type of learning algorithm (supervised, unsupervised, reinforcement) to deal with a given decision problem. In addition, this phase also allows us to determine the state variables of the future learning model. This phase is one of the most important phases in the modeling process. As noted in Figure 3.2, the system entity structure (SES) [203] can be used to define a family of models of the RL algorithm (DQN, DDQN, A3C, Q-Learning, SARSA, etc.) [177] based on the results of data analysis that use both statistical tools and the nature of the RL model (model free / model-based and on-policy / off-policy; see Chapter 2.3).

- The Simulation-based learning of an agent consists of simulating entry sets of a learning model to calibrate it by avoiding over-learning (learning phase). The DEVS formalism makes this possible by simulating the environment as an atomic model. However, the environment that interacts with the agent as part of a traditional RL scheme can also be considered as a coupled model composed of several interconnected atomic models.

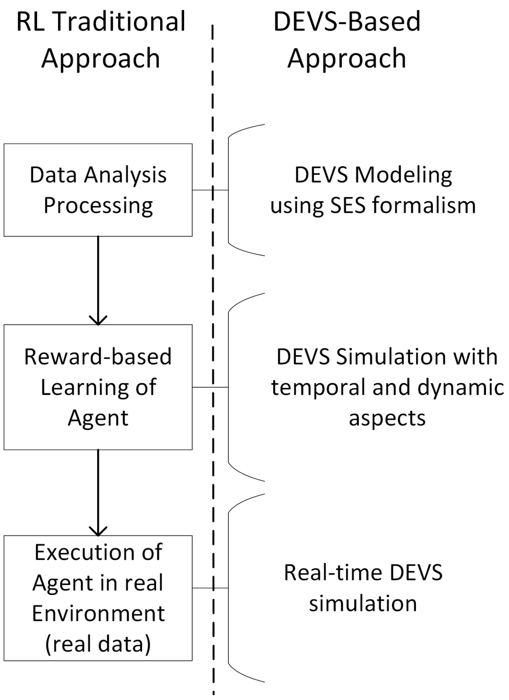

RL Traditional Approach
DEVS-Based Approach
Data Analysis Processing
DEVS Modeling using SES formalism
Reward-based Learning of Agent
DEVS Simulation with temporal and dynamic aspects
Execution of Agent in real Environment (real data)
Real-time DEVS simulation

Fig. 3.2 Traditional RL workflow and the corresponding phases in DEVS.

It is in this context that the environment is considered as a dynamic multi-agent DEVS model in which the number of agents can vary over time.

- The real-time simulation phase consists of submission of the model to the actual input data (test phase). The DEVS formalism and its experimental frame is an excellent candidate for simulating in real time the decision policies from real simulation data.

The DEVS formalism allows one to formally model a complex system with a discrete-event approach. The diagram of a RL system suggests that the Agent and Environment components can be represented by DEVS models (atomic or coupled depending on the modeling approach). The coupling of RL and DEVS can be done in two ways:

- DEVS can integrate the RL algorithms into its modeling specifications (transition functions, SES, modeling part) or within its simulation algorithms (sequential, parallel, or distributed, simulation part) in order to benefit from an AI, for example, improving the pruning phase in the specification process using SES or improving simulation performance by introducing a neural network-based architecture into the simulation engine.

- RL algorithms can benefit from the DEVS formalism to improve the explanability, observability, convergence, or search for optimal hyper-parameters, for instance. Moreover, the multi-agent RL algorithm can be modeled by DEVS due to its modular and hierarchical modeling possibilities. By separating Generators, Observers, and Transducers, the DEVS experimental frame allows a good framework to implement RL algorithms based on Agent-Environment interactions.

The thesis focused on this last point and relies on the formal framework proposed by the DEVS formalism in order to model and simulate the Q-Learning RL algorithm. The DEVS formalism makes it possible by specifying the Q-Learning algorithm using a set of interconnected components that react by its external transition function.

3.3 DEVS-Based RL Architectural Pattern

Basically, in the RL model, an agent and an environment communicate to converge the agent towards the best possible policy (Figure 3.3). Due to the modular and hierarchical aspect of DEVS, the separation/interaction between the agent and the environment within the RL algorithms is improved. A new generic DEVS modeling of the Q-Learning algorithm based on two DEVS models has been proposed: the DEVS models Agent and Environment.

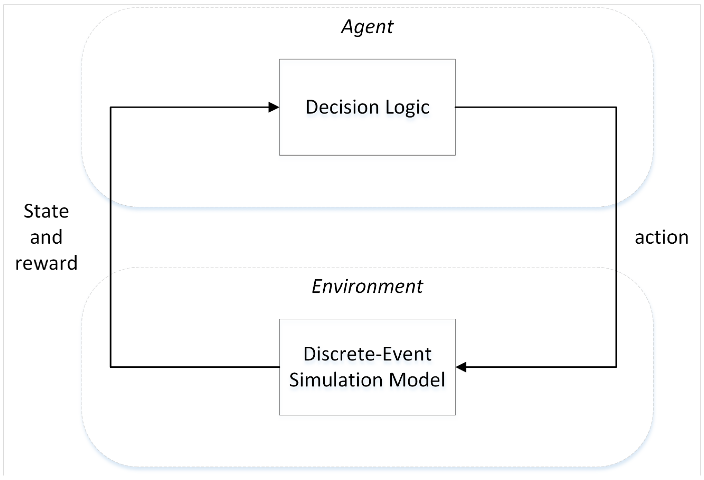


Fig. 3.3 Learning by reinforcement with the DEVS Agent and Environment models. The Decision logic is embedded in the agent that reacts to a new state and reward couple by sending an action that will be evaluated by the Environment model. The Environment model can be a discrete event simulation model executed in a specific experimental frame.

There are several ways to implement DEVS modeling of RL algorithms, and several DEVS modeling schemes are possible.

1. **Atomic-based Modeling Approach**: The agent and the environment are two atomic DEVS models with an update of the matrix Q in the environment or in the agent. The first approach is quick to set up, but does not respect the consistency of communication between the two components. The environment does not need to know the optimal policy determined by the agent and therefore to calculate the matrix Q in the Q-Learning algorithm. The second approach seems to be more correct from a behavioral point of view.

2. **Coupled-based Modeling Approach**: The agent and the environment are two coupled DEVS models with an update of the matrix Q in the agent (highlighted in the first case). This approach allows us to refine the decomposition of the parts of the RL algorithm of the agent and the environment separately. Moreover, it makes it possible to consider a multi-agent approach in a coupled Agent model, as detailed in Section 3.3.0.3. Finally, a hierarchy of abstraction and a parallel simulation are also possible with the use of the coupled modeling approach.

In the following subsections, we present the two approaches in detail with an additional approach that considers a multi-agent version of an RL system.

3.3.0.1 Atomic-based RL Modeling Approach

In Figure 3.4, the agent and the environment are two interconnected atomic models that communicate and perform their transition functions in a repetitive cycle until the agent determines the best policy with respect to the rewards it has received from its environment in response to its actions.

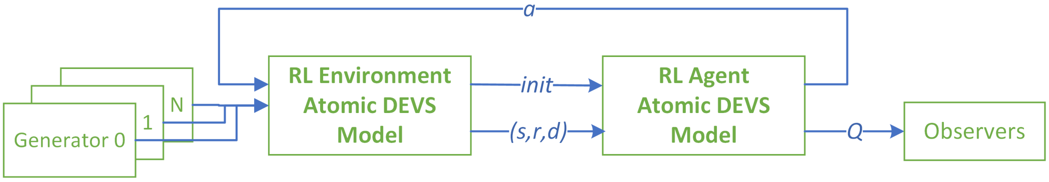


Fig. 3.4 Atomic-based modeling approach with the DEVS atomic models of the RL Agent and RL Environment DEVS atomic models. The atomic model Observers can be inserted after the RL Agent to observe the mean of the Q matrix, which can be used to see its convergence. *N* Generator models are used to consider external events in the behavior of the Environment model.

Figure 3.5 shows the UML sequence diagram of the User-Environment-Agent interactions in the Q-Learning algorithm framework. After an initialization phase, the Agent and Environment models begin a series of episodes in which the environment sends a state/reward couple in response to an action (chosen according to the ε-greedy policy - see 2.11 in Section 2.3.3) by the agent. At each end of the episode, the Q matrix is updated, and it is only when it no longer evolves that the learning is finished and that the agent's policy is determined.

The atomic model Agent is intended to respond to events that occur from the Environment model. When it receives a new tuple (s,r,d), it will return an action ($a \in A$) following a policy depending on its current state $s \in S$ and the implemented algorithm (ε-greedy for example). The model has a state variable Q which is an SxA dimension matrix for implementing the learning algorithm (Q-Learning or SARSA, for example). When convergence is reached (depending on the values of Q), the model becomes passive and no longer responds to the environment. The update of the Q attribute is done in δ_{ext} after receiving the tuple (s,r,d). The internal transition function makes the model passive. The output function is activated when the external function is executed. The DEVS specification of the RL Agent Atomic model is presented in Appendix 7.

The atomic model Environment will respond to the requests of the agent atomic model by assigning it a new state s, a reward r according to an action it has received. In addition, the model will inform whether it reaches the final state due to a variable Boolean d. Therefore, this communication will take place through an external transition function. The λ DEVS function will be activated immediately after the δ_{ext} function to send the pair (s,r,d) to the Agent model. The δ_{int} function will update the state variables of the model without generating an output. Finally, the initial state will be to determine the list of possible states S and actions A and the reward matrix R. The model is responsible for the generation of episodes (when a final state is reached). The DEVS specification of the RL Environment Atomic model is presented in Appendix 8.

3.3.0.2 Coupled-based RL Modeling Approach

This approach makes it possible to decompose the two previous atomic models Agent and Environment into an interconnection of specific atomic models. For example, the Agent model can be decomposed into four atomic models as shown in Figure 3.6.

The atomic model *Define State-Action-Map* of the Agents coupled model in Figure 3.6 could be dynamically instantiated only when a message arrives at its input port 0. The chain *Get State-Reward-Done/Get Action/Update Q Matrix* is activated as soon as a message arrives at input port 0. With this splitting, it is also possible to implement a deferred reward over time. The *Update Q Matrix* model implements the RL algorithm (Q-Learning, SARSA in our case). The *Get Action* model makes it possible to centralize the choice of a search by exploring / exploitation of the action, for example, with an e-greedy method. In the same

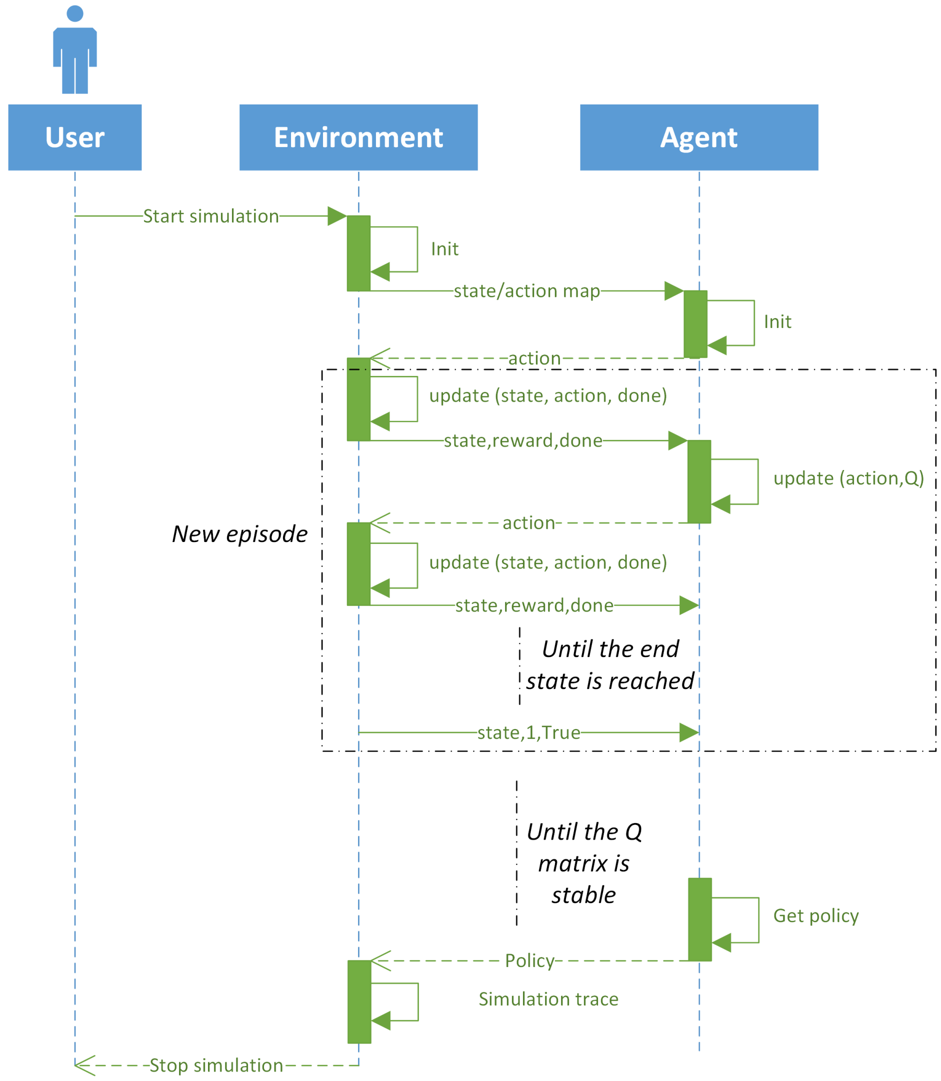


Fig. 3.5 UML sequence diagram of the User-Environment-Agent interactions in the Q-Learning algorithm framework. The user starts the simulation by invoking the initial (init) phase of the Environment model that sends an event with the state/action map used to initialize the Agent model. The Agent then sends an action depending on its initial state in return that activates the δ_{ext} function of the Environment model that immediately updates the state, the reward, and the done flag (which informs if the end state has been reached) to return to the Agent. The agent then updates its Q matrix according to the action received. This cycle (episode) is repeated until the end state is reached (the *done* flag is true). When the Q matrix is stable, the final policy can be outputted by the Agent model and the Environment model can print the simulation trace before becoming inactive.

way, we can imagine a split for the atomic model Environment in which the management of the acquisition of the input data (Generators in Figure 3.6) responsible for the construction

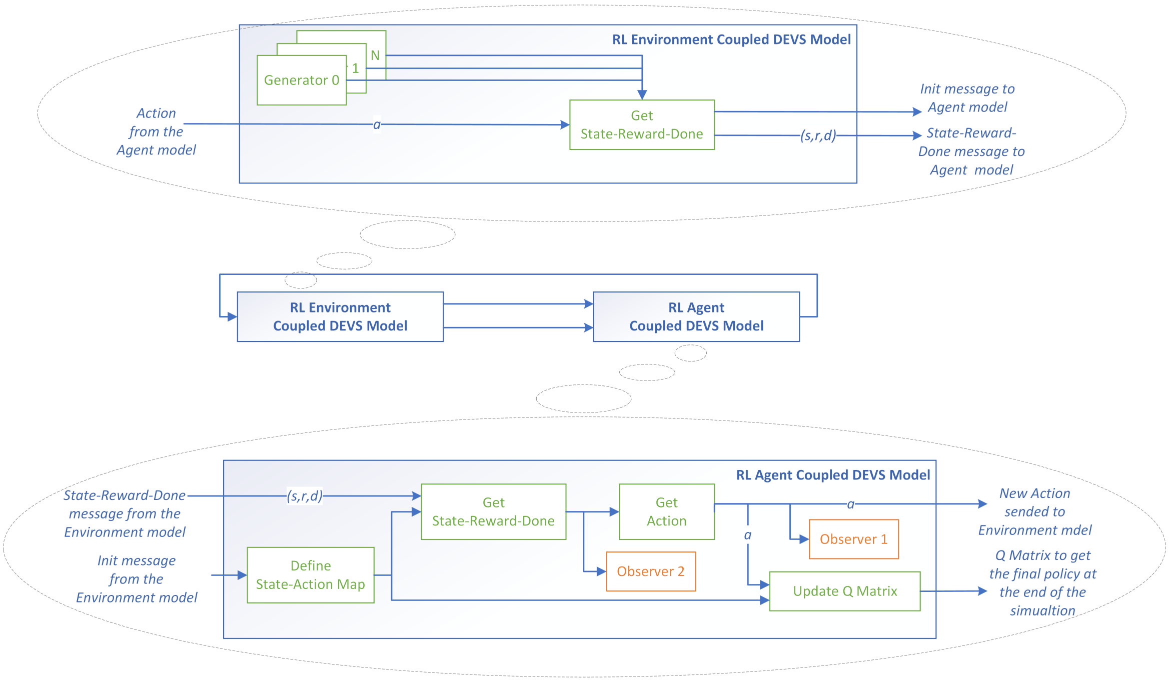


Fig. 3.6 Coupled-Based Modeling Approach with the Agent and Environment DEVS Coupled Models which improve the explainability of the AI embedded. Each observer is an atomic model that turns the signal into an interpretable feature.

of the states and the possible actions (in the *Get State-Reward-Done* of the Environment coupled model) would be decoupled from the management of the attribution of the new pairs (s,r,d) in response to each new action from an agent. This assignment depends on the type of RL algorithm chosen.

Among the advantages attributed by DEVS, composability makes it possible to introduce observers and thus increase the explainability of the model. Indeed, observers 2 and 1 make it possible to trace, during the simulation, the state-reward pairs in order to follow the episodes and the actions chosen by the agent for its learning. In this sense, we can say that DEVS makes it easier to control the progress of the learning algorithm. In addition, it is possible to position in parallel with its observers models capable of modifying the actions or the rewards during the learning loop without modifying the original algorithm.

3.3.0.3 Multi-Agent-based RL Modeling Approach

In human society, learning is an essential component of intelligent behavior. However, each individual agent needs to learn everything from scratch by its own discovery. Instead, the students exchanged information and knowledge with each other and learned from their peers or teachers. When a state space is too large for a single agent to handle, multiple agents may cooperate to take into account a larger amount of information to optimize a decision-policy process. Traditionally, RL is used to study intelligent agents. Each RL agent can incrementally learn an efficient decision policy over a state space by trial-and-error, where the inputs from an environment are next states and a delayed scalar reward. It seems important to remember that in RL there are no predefined data and that the whole RL process itself is a training and testing phase.

Although most of the work on RL has focused exclusively on a single agent that interacts with a component of the environment [178]. However, the single-agent approach is limited by the size of the state space. That is the reason why, to overcome this kind of limit, neuron network approaches are used to approximate the state matrix [63, 141]. Furthermore, with a neural network technique, the multi-agent reinforcement learning (MARL) [32, 65] approach can be applied [42, 92, 142] in order to divide a large state space by distributing the state space set from a single agent to multiple agents that interact and learn independently. This approach is close to modular RL in the sense that the state space is broken down into a subset of states, but the collaborative aspect, although learning is independent, specifies the approach proposed as MARL. DEVS is used to straightforwardly extend RL to multiple independent agents. Together, they will outperform any single agent due to the fact that they have twice as many indices to invest in and, therefore, a better chance of receiving rewards. However, the goal is to study the added value of implementing MARL in the DEVS formalism and to compare the performance of an independent agent with the one of a multi-agent agent and to identify their trade-offs.

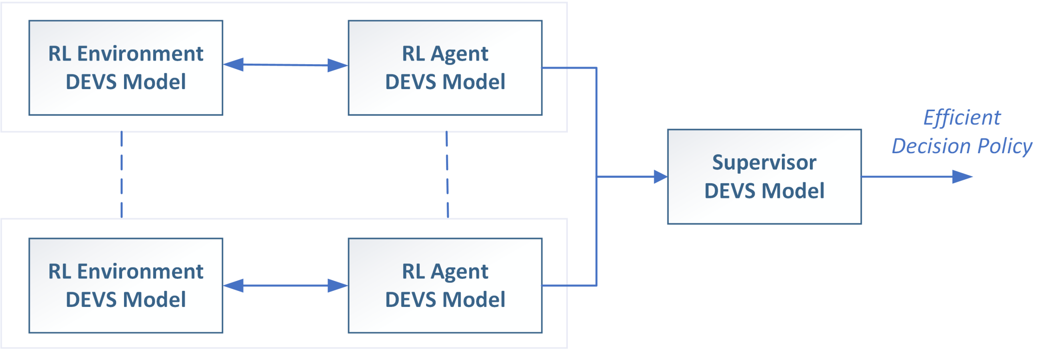

RL Environment DEVS Model
RL Agent DEVS Model
RL Environment DEVS Model
RL Agent DEVS Model
Supervisor DEVS Model
Efficient Decision Policy

Fig. 3.7 Multi-agent DEVS reinforcement learning model with a supervisor. Each agent explores a subset of possible states and gives its optimal decision plot based on interactions with its own environment. The supervisor then proceeds to the final decision-making logic.

The proposed approach aims to answer a problem due to the necessity of dealing with a large number of state frameworks. Instead of letting one single agent iterate in a single large environment, the task is divided into multiple agents that explore a part of the number of states. As depicted in Figure 3.7, each agent gives an optimal policy resulting in the execution of the Q-Learning algorithm that depends on the interaction with its environment. Once the supervisor has received all agent policies at each simulation time step, it determines the best combined policy and communicates to the single agent the action to be taken. The construction of the best-combined policy is the result of a process of trial-and-error by reward return. If the environment changes, the process must be restarted. A policy is applicable to a given environment.

DEVS is used for its modular and hierarchical power, which makes it possible to describe a system by interconnections of atomic models (agent and environment) in order to describe a multi-agent model. In the case of multi-agent models, DEVS makes it possible to solve the problems of synchronization of events generally present in this type of system due to its time advance function specification [32]. The DEVS formalism makes it possible to simulate the environment as an atomic model. However, the environment that interacts with the agent in a traditional learning-by-reinforcement scheme may also be considered as a coupled model composed of several interconnected atomic models. In that context, we may consider the system as a dynamic structure DEVS model with a static structure of the coupled DEVS model in which the number of agents/environment can vary over time and can be executed in parallel. The supervisor model builds its policy by integrating all policies and experiences already learned by each agent. This is the age-old divide-and-rule policy.

3.3.1 What About Observability and Explainability?

According to Section 2.3.5, two advantages of using the coupled model-oriented architecture that improve observability and explainability are: the increase in understanding of the model (explainability) and the possibility of making observable internal signals belonging to the learning algorithm (observability). Learning models are called black boxes because it is often difficult to understand the mechanisms that lead to correct but difficult to explain results. The use of coupled models allows for a greater hierarchy of description and, therefore, a functional breakdown of the learning algorithm. The workflow *State-Reward-Done/Get Action/Update Q* makes it possible to isolate the basic functions of the learning algorithm and to have access to observable signals such as the actions, states, and rewards involved in learning. The increase in observability allows a better understanding of the mechanism that leads the agent to converge towards the best decision-making policy. The observability and the explainability of the approach by coupled models is also achievable with the approach by atomic models, but it is less modular. With the coupled model approach, the developer does not need to insert debugging instructions inside the code to observe a property of a system.

You do not need to implement a new function (or method, in the case of an object-oriented approach) to introduce or change the behavior of a system.

. In recent years, many XAI tools (IntepretML, LIME, SHAP, Seldon Alibi, etc.) have been developed and integrated in ML workflows: These tools focus on making ML clear, at least visually, and how much particular features influenced the prediction. The development of explainable AI (XAI) tools is still juvenile.

1. How do we incorporate explainability into our experiences without detracting from the user experience or distracting from the task at hand?
2. Do certain processes or specific information need to be hidden from users, and how do we explain them to users?
3. Which segments of our AI decision-making process can be easily digestible and explainable to users?

In general, answering those questions gives us valuable insights in terms of explainability for carrying out projects in the five most important applications in Computer Science: debugging, informing feature engineering, directing future data collection, informing human decision-making, building trust, regulatory compliance, and high-risk applications.

In terms of building trust, let us first stress the main difference between the three models, according to the taxonomy described in the paragraph dedicated to XAI of the State of the Art.

Scope Time	Local	Global
Intrinsic Model-Specific	LSTM ESV DEVS	LSTM ESV DEVS
Post-Hoc Model-agnostic	DEVS	DEVS

Table 3.1 Comparison using XAI taxonomy.

As synthesized in Table 3.1, DEVS is global and post hoc. It is global because it leads the user to trust the model, not just the prediction, and it is post hoc, in the sense that it will turn a model-free portfolio optimization based on modeling and simulation into a simpler model-base optimization model. A determined weighted portfolio with specific stock volatility leads to a determined portfolio reallocation. LSTM is intrinsic or model-specific and local. Our LSTM model leads the user to trust the prediction for a specific stock because we can compare the prediction results with the real values of the stock. Our LSTM is also local because it offers an explanation of the specific prediction and answers the following question: How do we evaluate the precision of the prediction made by the model?

3.3.2 Explicit Time in Q-Learning

In MDPs, the time spent in any transition is the same. However, this assumption is not the case for many real-world problems.

In [28], the authors extend the classical RL algorithms developed for MDP and semi-Markov decision processes. Semi-MDPs (SMDPs) extend MDPs by allowing transitions to have different durations (t). In [144], the authors considered the problem of learning optimal policies in the time-limited and time-unlimited domains using time-limited interactions (limited number of steps k) between agents and environment models. The notion of time is explicitly considered, but only in terms of the limited time T considered to maximize the total reward assigned to the agent who tries to maximize the discounted sum of future rewards:

$$G_{t:T} = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k}$$

In [118], the authors introduce a new model-free RL algorithm to solve SMDP problems under the average-reward model. In addition, in [179], the authors introduce the theory of options to bridge the gap between MDPs and SMDPs. In SMDPs, temporally extended actions or state transitions are considered as indivisible units; therefore, there is no way to examine or improve the structures inside extended actions or state transitions. The option theory introduces temporally extended actions or state transitions, called options, as temporal abstractions of an underlying MDP. It allows to represent components at multiple levels of temporal abstractions and the possibility of modifying options and changing the course of temporally extended actions or state transitions. In [152], the authors explore several approaches to model a continuous dependence on time in the MDP framework, leading to the definition of Temporal Markov Decision Problems. They then propose a formalism called Generalized Semi-MDP (GSMDP) in order to deal with an explicit event modeling approach. They establish a link between the Discrete Event Systems Specification (DEVS) theory and the GSMDP formalism, thus allowing the definition of coherent simulators.

In the case of explicit time, the time of a transition has to be taken into account. The time between actions is an explicit variable (which may be stochastic) and depends on the state and the action. This transition time is known as the notion of sojourn [158]. DEVS Markov models [164] are capable of explicitly separate probabilities specified in transitions and defined in times / rates. Furthermore, the dynamic properties involved in the DEVS formalism allow one to dynamically modify these specifications during model simulation. Transition probabilities are classically associated with a Markov chain, while transition times / rate probabilities are associated with the sojourn notion [30]. This modeling feature associated with Markov chains offers the possibility of explicitly and independently defining transition probabilities and transition times.

The following section presents the implementation of the Agent-Environment model in the DEVSimPy environment.

3.4 DEVSimPy Modeling and Simulation of RL

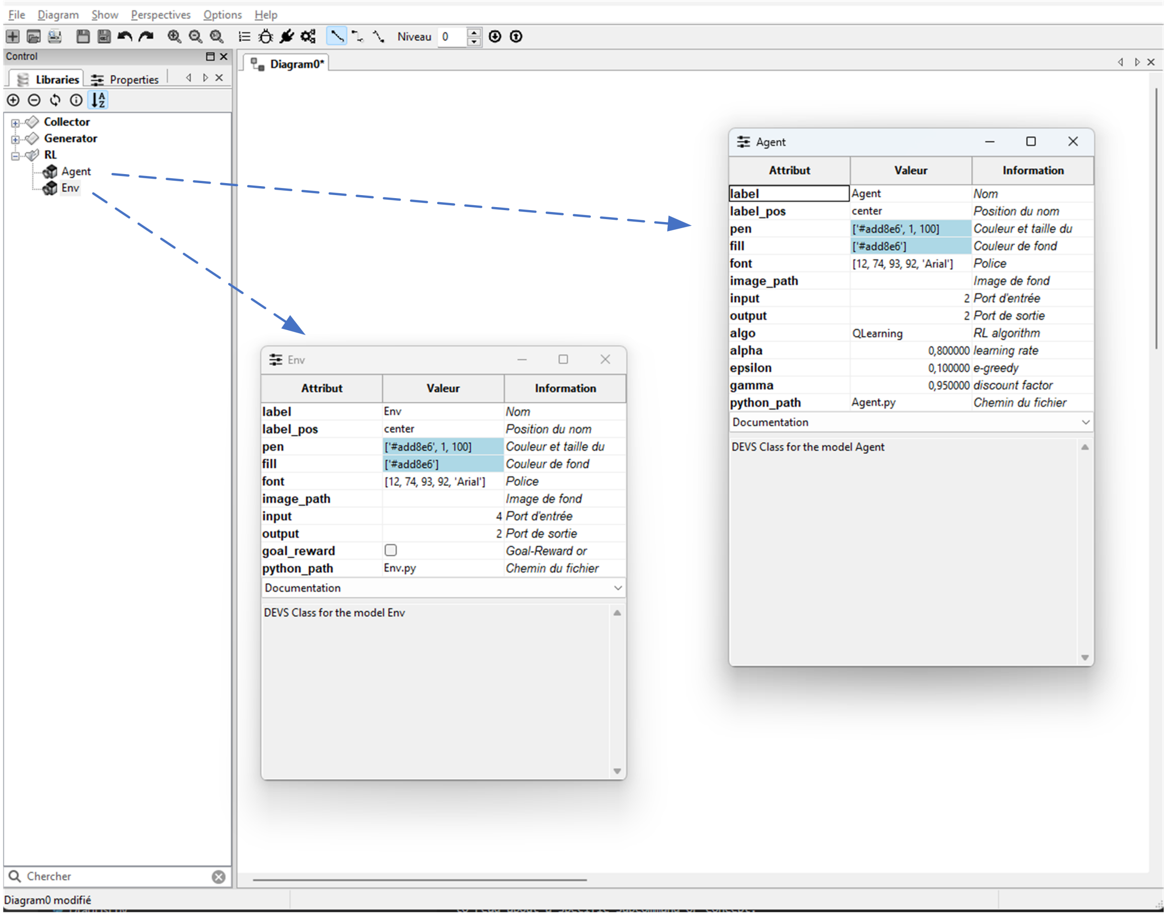
DEVSimPy is a graphical environment that allows the M&S in Python language of DEVS models [33]. We have implemented the Agent-Environment model as atomic models in a library called "RL" (left part of Figure 3.8). The Agent.amd and Env.amd files implement the agent model and environment models according to the Q-learning or SARSA algorithm.

Implementing an RL model in DEVSimPy consists of dragging and dropping one or more Agent and Environment models in the right panel, interconnecting them, and then configuring them. The configuration goes through the definition of the behavioral properties mentioned above, but it is also necessary to define the methods specific to the field studied. It is the environment model code that needs to be adapted. As mentioned in 8, the methods $GetInitState(inputs1N)$, $GetEndState(inputs1N)$, and $GetStateActionMap(inputs1N)$ must be implemented based on the input of model 1 through N ($inputs1N$). If the environment is a coupled model, it will also be necessary to define other DEVS models which will participate in the definition of the state action map.

3.4.1 The Pursuit-Evasion Case Study

In order to illustrate the use of DEVSimPy to implement a RL framework (involving multi-agents), we propose to use a case study known as Pursuit-Evasion [81]. Thanks to this example, we show that DEVS makes it possible to build RL models by combining different Agents with learning logics. Indeed, in this example, an agent will use a "Best Escape algorithm" to choose his action instead of respecting the e-greedy algorithm, for example.

One or more pursuing agents, here called the "cat", must have one or more fugitive agents, called the "mouse". This type of problem distinguishes Search and Rescue problems in the sense that the fugitive agent tries at all costs to avoid pursuers. The mouse-cat problem is played one by one: Each agent performs an action and must wait for all other agents to



Agent

Attribut	Valeur	Information
label	Agent	Nom
label_pos	center	Position du nom
pen	['#add8e6', 1, 100]	Couleur et taille du
fill	['#add8e6']	Couleur de fond
font	[12, 74, 93, 92, 'Arial']	Police
image_path		Image de fond
input	2	Port d'entrée
output	2	Port de sortie
algo	QLearning	RL algorithm
alpha	0,800000	learning rate
epsilon	0,100000	e-greedy
gamma	0,950000	discount factor
python_path	Agent.py	Chemin du fichier

Documentation

DEVS Class for the model Agent

Env

Attribut	Valeur	Information
label	Env	Nom
label_pos	center	Position du nom
pen	['#add8e6', 1, 100]	Couleur et taille du
fill	['#add8e6']	Couleur de fond
font	[12, 74, 93, 92, 'Arial']	Police
image_path		Image de fond
input	4	Port d'entrée
output	2	Port de sortie
goal_reward	☐	Goal-Reward or
python_path	Env.py	Chemin du fichier

Documentation

DEVS Class for the model Env

Fig. 3.8 RL library into the DEVSimPy software. The Properties panel of the Agent and Environment models allows one to configure these models. The *algo* property of the agent allows one to select the learning algorithm between Q-Leaning or SARSA. Properties γ, ε, and α are related to the Bellman equation (see Section 2.3.3). Concerning the Env model, the option *goal_reward* allows us to define the reward as a goal (when $goal\_reward = True$ is checked) or as a penalty (when *goal_reward* is uncheked) (see Section 2.3.2.3)

perform an action before executing the next one. In this example, we consider only two agents: one mouse and one cat. We also consider a discrete time: each action corresponds to a unit of time $t \in \tau$ with $\tau = \{1,2,\ldots,T\}$. Reflection and observations are considered to be performed in zero time.

Like time, the space of the problem is discrete. Therefore, the space in which the part unfolds is a finite and discrete set of cells $x = \{1,2,\ldots,X\}$ arranged in two dimensions. All movements are therefore discrete and concern the displacement of an integer number of squares, defined by the unit by default. We will formally define the distance between Agents 1 and 2 as the number of actions necessary for Agent 1 to occupy the place of Agent 2, this one not being deployed. This amounts to calculating the Manhattan distance if agents are not allowed to use diagonals.

Two primordial notions of the environment in RL are distance and identification of a state. The distance is used as a reward function since it allows one to measure the distance between the cat and the mouse. The identification of a state corresponds to the position of an agent (cat or mouse) in the matrix. For example, in a 9x9 matrix, there are 81 states, each corresponding to a given position.

A cat is an agent whose goal is to catch a mouse agent. Clearly, this means performing a series of actions that would lead to a final state where the cat-mouse distance would be zero. At the level of perception, a cat has perfect acuity: At any time, he knows the position of all agents in the space. Therefore, information on the state of the environment is complete and correct. A cat can perform 5 actions, which are listed below:

- Action 0: Stay still.
- Action 1: Up, move one box north.
- Action 2: Right, move one space to the East.
- Action 3: Bottom, advance one box to the South.
- Action 4: Move one space to the left.

Note that our cat cannot move diagonally. The mouse agent enjoys exactly the same properties as the cat: global vision of the space and the same five possible actions.

Our mouse aims to escape the cats. We have decided to develop a custom algorithm called best-escape in order to perform the way the mouse is moving. The principle of the best escape algorithm is very simple. It consists first of all of refusing any movement which would throw the mouse into the claws of a cat or any movement which would lead the mouse into the immediate neighborhood of a cat (we mean the adjacent box). These two actions would, in fact, be synonymous with certain death. For each of the remaining actions, we compute the escape value, the sum of the squares of the distances from the arrival point of the action with each of the mice present in the game. The goal is obviously to maximize this value. Then we choose the action that gives the highest escape value possible. If several actions have the same escape value, we take the first one from the list. Therefore, the choice is completely deterministic and can be predicted for each given situation. We introduced a fear factor as a variable in the mouse agent. This very simple parameter actually introduces the fact that the mouse will not perform any action at a given moment, and this without regard to the rest of the environment. It can also be seen as the fact that the mouse is frozen in fear.

The main difference between DEVS multi-agent learning and a single agent is that the learning agent is connected to a DEVS coupled model instead of an atomic model (Figure 3.9).

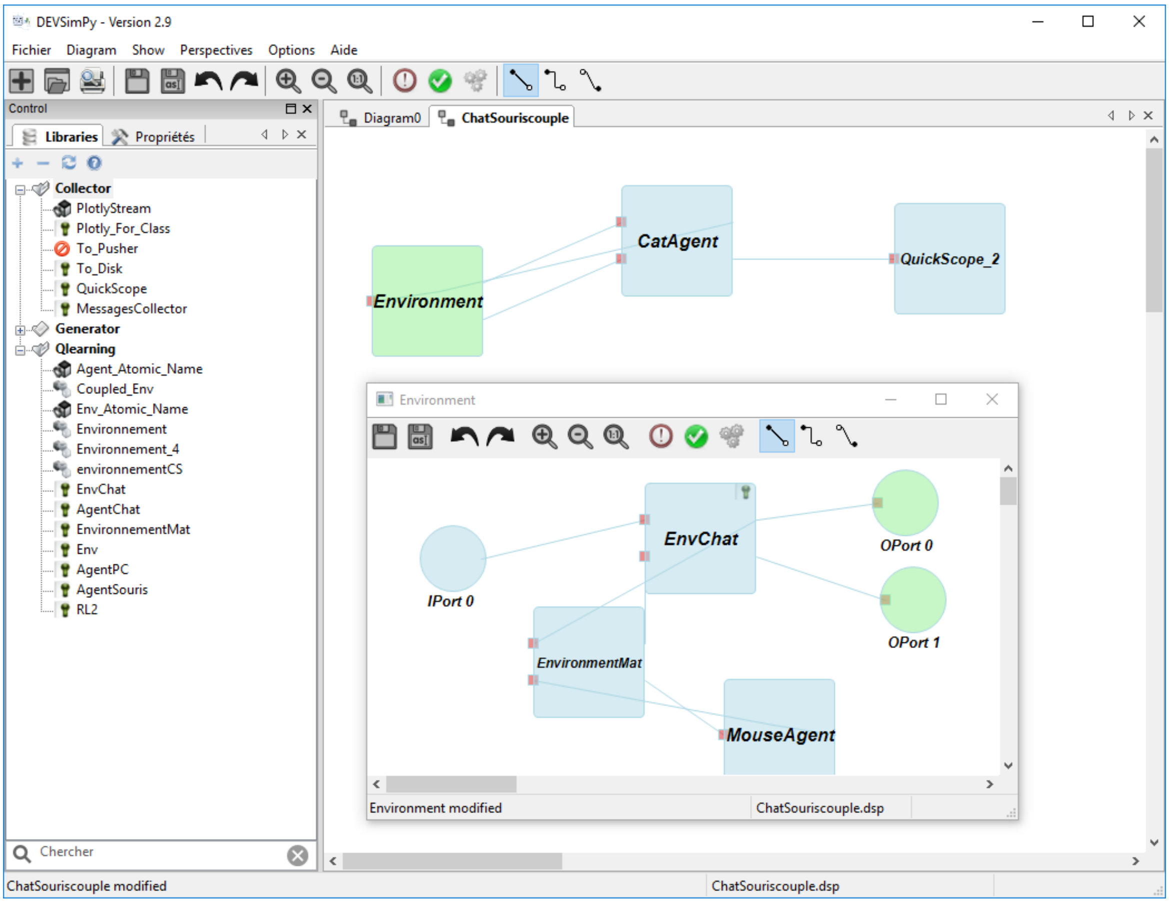


Fig. 3.9 Mouse-cat M&S into the DEVSimPy framework.

This coupled model embeds other agents, the global environment vision, and the specific environment associated with a learning agent. The learning agent called *CatAgent*

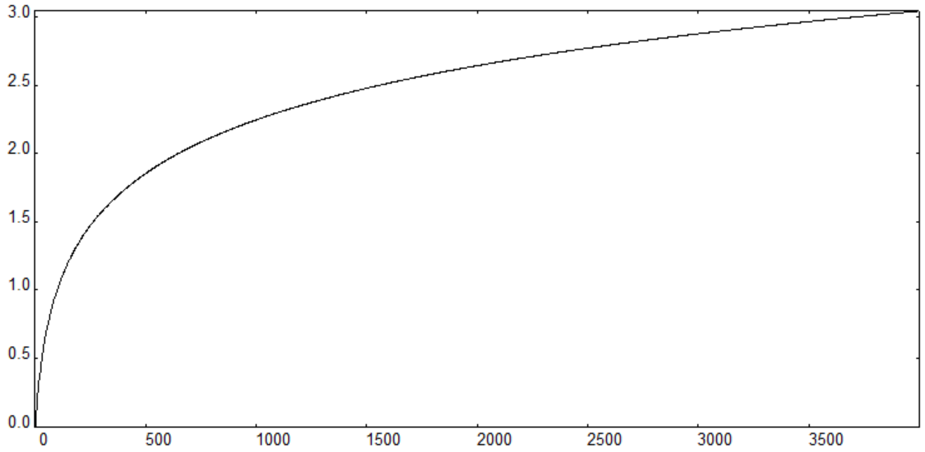


Fig. 3.10 Mean Q-value per episode with fear factor.

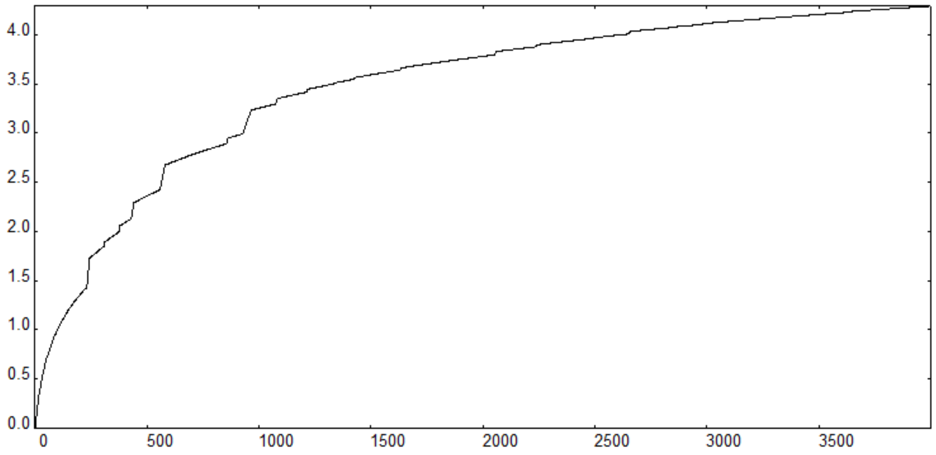


Fig. 3.11 Mean Q-value per episode without fear factor.

in Figure 3.9 is connected to the coupled DEVS model (called *Environment*) embedding other agents (as *MouseAgent*), the global environment vision (as *EnvironmentMat*) and the specific environment (as *CatEnv*) associated with the learning agent (as *CatAgent*).

The global environment (*EnvironmentMat*) defines the space in which the agents will evolve. In our implementation, this space is a simple matrix of integers. 0 means an empty box. Each other integer means an agent (1 for the cat and 2 for the mouse). It is quite possible to imagine extending this environment to, for example, search in a space with 3 dimensions or more. An action is denoted by an integer and corresponds to the movement of the agent in question. If the number of possible actions is not problematic, it is also necessary to assign to each action the corresponding movements of the concerned agent. The chosen implementation consisted of a simple dictionary that assigned, for each action, the corresponding movements.

The heart of the Q-Learning algorithm [177] is implemented in the agent (*CatAgent*) and is associated with the atomic model of the specific environment (*CatEnv*). From this observation, we will compute the reward to associate with the action. Both the new state and the computed reward are sent to the input of the *CatAgent* atomic model. The Q-Learning algorithm assumes that *CatAgent* has received a reward and a new state after performing an action.

The mouse agent does not use an RL technique. To choose its action, the mouse uses the Best Escape algorithm: the mouse tries to minimize the sum of the squares of the distances between itself and the cats, and, if no solution is possible, the mouse remains on the spot.

The simulation has been performed using the fear factor parameter for the mouse agent: (value 0 with fear factor/value 1 without fear factor). Figures 3.10 and 3.11 show the evolution of the mean Q value during the learning phase with and without the fear factor between a mouse agent and a cat agent. Notice that since the mouse cannot move, the Figure 3.10 is a regular curve. While the mouse tries to escape when the fear factor is equal to 0, the Figure 3.11 is not regular. These irregularities of the curve fit the fact that the mouse is escaping.

In this example, only one cat agent and one mouse agent are involved. However, when considering multiple cats, one of the questions is to decide whether the agents will perform the learning phase collectively as a team or individually. Furthermore, when considering multiple mice, we have to decide whether the goal is finally to catch a mouse or all the mice. These problems are general problems inherent to multi-agent learning, since research in this domain focuses on studying software agents that learn and adapt to the behavior of other software agents [171]. In addition, the presence of other learning agents complicates learning, since the environment may become non-stationary (a situation of learning a moving target similar with the fear factor of the mouse agent). This will be a problem we will return to in the case study of this thesis.

Of course, the reward function will also be harder to define since it depends on both the learning agents and the multiple and moving targets. An RL agent learns by interacting with its dynamic environment. At each time step, the agent perceives the state of the environment and takes an action, which causes the environment to transit into a new state. A scalar reward signal evaluates the quality of each transition, and the agent must maximize the cumulative reward throughout the course of interaction. Well-understood, provably convergent algorithms are available for solving the single-agent RL task. Together with the simplicity and generality of the setting, this makes RL attractive also for multi-agent learning in an environment.

3.5 Conclusion

The discrete event-oriented modeling of the Agent-Environment system by the DEVS formalism allows us to explore the Q-Learning and SARSA algorithm in a more behavioral way. In general, the Q-Learning and SARSA algorithms are based on the nesting of two loops: a repetitive loop on a number of episodes, which includes a conditional loop on actions. The DEVS discrete event approach makes it possible to dissociate these two loops through the communication of two models (atomic or coupled) Agent and Env. This makes it easier to interact with the two learning algorithms in order to implement specific stopping conditions, for example. In addition, the modularity provided by the DEVS formalism makes it possible to divide the algorithms into an interconnection of atomic models, thus improving the experimentation of new calculation methods to update the variable Q or the method of allocating new actions by the agent.

The coupled models approach has the advantage of facilitating the implementation of a multi-agent model for RL. Indeed, the Agents coupled model can be composed of an interconnection of Agent atomic models that communicate both with each other and with the environment. The DEVS formalism is a good candidate for setting up a multi-agent model. In addition, the possibility of simulating coupled models in parallel thanks to PDEVS is an interesting avenue when one wants to set up multi-agent models which can lead to significant simulation times. We do not present this aspect in this thesis.

DEVS is based on a formal representation of time in finite-state automata. This property is not highlighted in this thesis. However, when we talk about delayed reward, we can think of a different simulation-time exploitation than the one implemented in the atomic models presented in this thesis. It would then be interesting to see what the consequences would be for the final policy.

Chapter 4

Case Study: Leverage Effects in Asset Management Optimization Processes

4.1 Introduction

The case study deals with a decision-making process carried out by traders during the process of optimizing asset management, which may lead to a leverage effect. The first thing to consider is how a human trader would perceive their market environment. What observations would they make before deciding to make a trade? A trader would most likely look at some charts of the price action of a stock, possibly overlaid with a few technical and macroeconomic indicators. From there, they would combine this visual information with their prior knowledge of similar price action to make an informed decision on the likely direction of the stock.

Figure 4.1 shows the human-driven trading process. Typically, agents, whether humans or AI, perceive the market environment by observing characteristics of stocks such as the open price, high, low, close price, daily volume, etc. for a certain number of days, as well as other data points such as their account balance, current stock positions, and current profit. Then, **human traders** considers the price action leading up to the current price, as well as the status of their own portfolio, in order to make an informed decision about their next action. Once a trader has perceived their market environment, they need to take action. The range of actions available to human traders consists of three possibilities: buy a stock, sell a stock, or do nothing (wait).

Here, we come to the most interesting part of the topic of human decision-making. In most **AI agents** (the AI-driven approach in Figure 4.1) developed to date, the AI agent behaves like a traditional human trader. In other words, to solve a given problem, it is

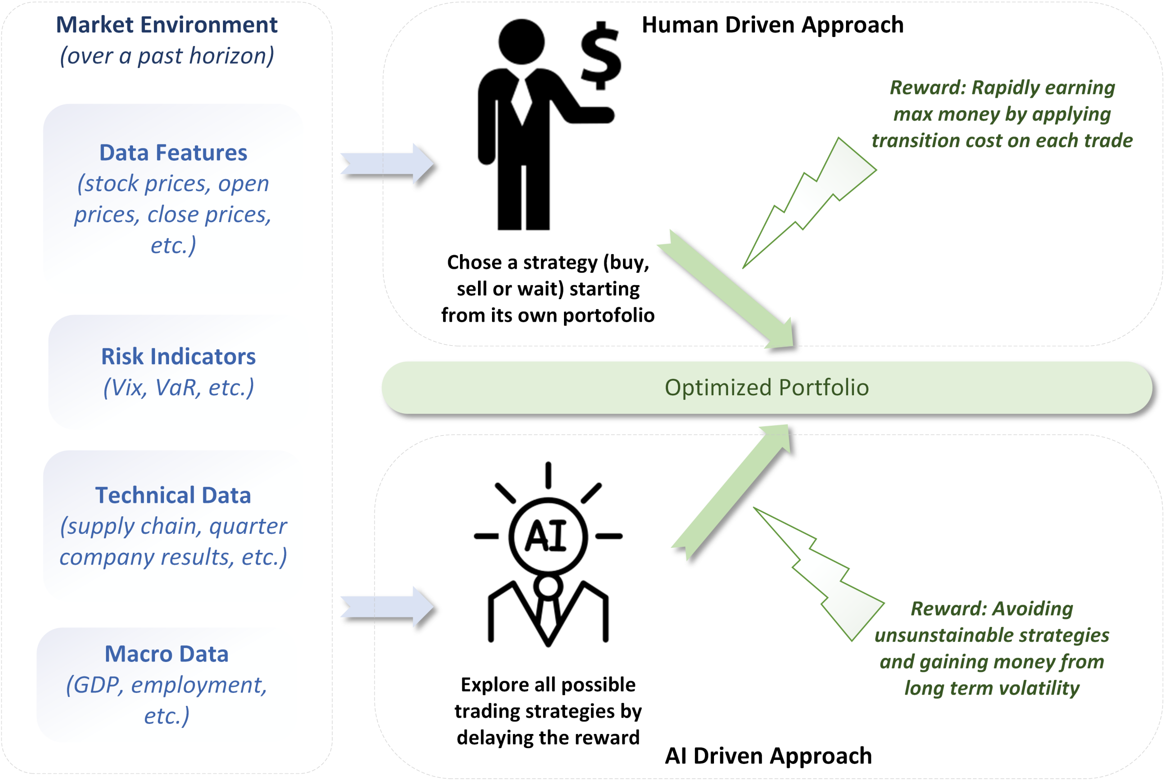

Market Environment
(over a past horizon)
Data Features
(stock prices, open prices, close prices, etc.)
Risk Indicators
(Vix, VaR, etc.)
Technical Data
(supply chain, quarter company results, etc.)
Macro Data
(GDP, employment, etc.)
Human Driven Approach
Reward: Rapidly earning max money by applying transition cost on each trade
Chose a strategy (buy, sell or wait) starting from its own portofolio
Optimized Portfolio
AI
Explore all possible trading strategies by delaying the reward
Reward: Avoiding unsunstainable strategies and gaining money from long term volatility
AI Driven Approach

Fig. 4.1 Approaches driven by humans and AI. This process is theoretically carried out taking into account macroeconomic indicators, technical data, stock trends, market risk indicators, and the current composition of the trader's portfolio. In other words, to address the issue of risk aversion, traders trade a small portion of their portfolio at a regular pace, such as quarterly, monthly, or weekly. The supervised AI agent has a different reward system to reduce human biases and to produce a leverage effect (make more money) with a more sustainable long-term strategy.

necessary to understand how current actions result in future rewards. However, the only proposed approach to understanding the market is to buy or sell an AAA-rated stock in small amounts every month or period of time to maximize the probability of buying at a lower price and selling at a higher price. This simple routine is basically their portfolio optimization decision process to make them earn money (get their rewards) from the transaction fees charged to their customers. AAA is the highest rating assigned to any debt issuer; we interviewed dozens of bankers and all applied the same strategy. To predict the total future reward that will result from an action, it is often necessary to take many steps into the future by trying strategies that may depend only on the present and not on a consolidated past.

The question of constructing the **reward** becomes central. In fact, the following rule is usually applied: Classically, banks want to incentivize profits that are sustained over long periods of time. At each step, they will set the reward as the balance of the account multiplied by some fraction of the number of time steps so far. When traders use AI agents, the purpose of this is to delay rewarding the agent too quickly in the early stages and to allow it to explore sufficiently before optimizing a single strategy too deeply. It will also reward agents who maintain a higher balance for longer, rather than those who rapidly gain money using unsustainable strategies. But why? Rapidly gaining money usually means using unsustainable strategies by default.

As introduced in Figure 4.2, our research effort aims to provide samples and results to prove the contrary and provide some evidence that this is probably the best policy for large human trading institutions, such as banks, which make money taking a percentage of the transaction cost and not based on the pure financial performance of the portfolio, but not for scalpers, as in our case study.

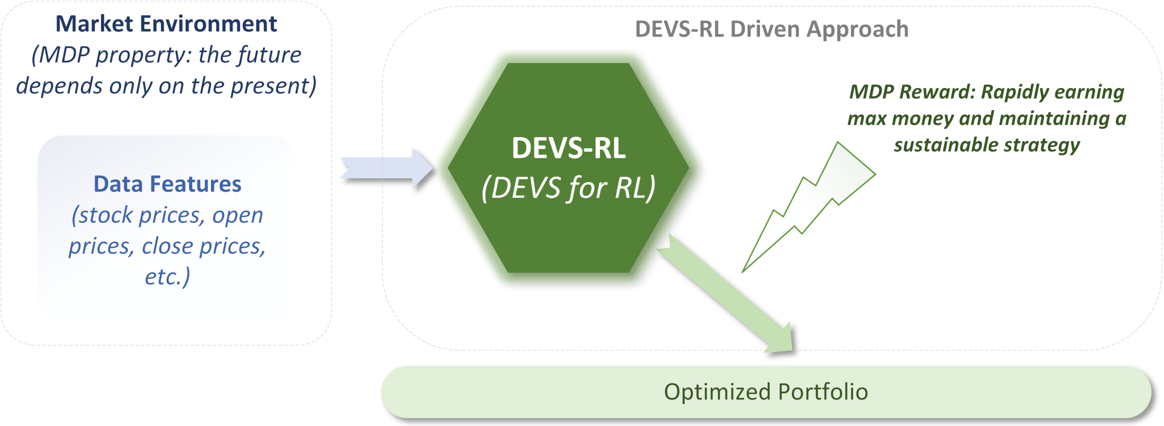

Market Environment
(MDP property: the future depends only on the present)
Data Features
(stock prices, open prices, close prices, etc.)
DEVS-RL Driven Approach
DEVS-RL
(DEVS for RL)
MDP Reward: Rapidly earning max money and maintaining a sustainable strategy
Optimized Portfolio

Fig. 4.2 DEVS-RL driven proposed approach.

In this section, we demonstrate how the DEVS (Discrete Event System Specification) formalism can be used to create a formal framework for an RL (reinforcement learning) system, resulting in improved modularity and clearer explanations of the AI models. A DEVS-RL (DEVS with reinforcement learning) simulation model will be developed to study the effects of leverage on financial assets and to compare the effectiveness of the proposed approach with other commonly used AI tools. The modularity of the approach will also be highlighted, as it can be combined with other AI algorithms to create decision and prediction models through simulation. The validity of the approach will be tested through the implementation and analysis of a simulation model in a first experiment. In addition, compatibility and combination studies will be conducted with other AI algorithms to further assess the effectiveness of the approach and examine its compatibility with other intelligent learning models.

4.1.1 First Experiment

In our study case, let us consider the possibility of raising funds by borrowing capital. Our Agent gains or losses are driven by a machine learning algorithm. The **leverage effect** will be estimated by calculating the difference between the evolution of the value of the initial portfolio (basket of indexes bought with the borrowed capital) without sell and buy and the total value of the portfolio driven by the machine learning algorithm embedded in the Agent.

We propose four scenarios. In simulation number 1, named initial cash=0, the internal financing (IF) is 705$ and leverage effect borrowed capital in 10 times the IF, that is 7053$ that are invested in an investment portfolio (equivalent to a state) of 1 to 3 stock indexes ($index_a, index_b, index_c$) among the main world indexes (CAC40, DJI (Dow Jones Industrial Average) and IXIC (Nasdaq Composite)). In the three other simulations, the Agent has the same amount in stocks as in simulation number 1 and the possibility to invest 3 different extra cash values. The goal is to have a portfolio of a multiplicity N of 0 to N-1 of each of these indexes that allows one to obtain at any moment the maximum possible value by adding together all the values of the N indexes (see equation 4.1).

$$max \sum_{i=a,b,c} \{0;N-1\} * index_i \tag{4.1}$$

Investing IF in stock market indexes rather than in company-specific shares makes it possible to take into account the evolution of the environment of financial markets. Indeed, the volatility of the indexes that represent the trend of the best (or the average of all) shares of the market reflects the behavior of the major agents who influence market trends and its environment (correction, bullish, bearish, etc.). It is important to note how much the environment (volatility of the indexes or new cash inflow to buy additional indexes) can have an impact on our simulation. In fact, the change in the value of an index or the availability of new cash will influence, for example, the number of states or their length of life. Our simulation example will deal with the policy to be followed in the case of a change in environment related to the increase of the Agent cash availability and indexes volatility.

In an RL system, the agent learns from the rules. In our case, the rules are as follows:

- The whole agent state is finite and is calculated from equation 4.1,
- The Agent can take one action at a time, among 3 actions (buy, sell, wait) that are chosen on the basis of the indexes volatility,
- The Agent uses a goal-reward representation [177] and gets a reward different from 0 only when it reaches the goal state corresponding to the maximum value of the investment portfolio.

- The Agent uses the single-goal approach that considers only one goal state of policy research.

Taking into account our case study, an MDP can be formalized as follows.

- The finite set of states $S = \{(s_0,\ldots,s_k)|k \in \mathbb{N}, s \in [0,N-1]\}$ and the size of the state space are $|S| = N^k$ with N the multiplicity of the indexes. If $N = 8$, 512 possible states can be considered. Each state can be reached from every other state.

- The nonempty set of goal states $G \subseteq S$.

- The finite set of actions $A = \{(a_0,\ldots,a_k)|k \in \mathbb{N}, a \in [-1,0,1]\}$ (-1 for selling, 1 for buying, and 0 for waiting) and the total number of actions is $\sum_{s \in S} |A(s)| = 7$. All actions are deterministic, and an ε-greedy algorithm is used to choose them with an exploitation-exploration approach. One action at a time is possible and is controlled by a parameter $M = 1$.

- According to the goal-reward representation [177] where the agent is compensated for entering a goal state, but is not compensated or penalized otherwise, the set of reward $r : S \times A \longrightarrow \mathbb{R}_0^1$

$$r(s,a) = \begin{cases} 1 & \text{if s is a goal state} \\ 0 & otherwise \end{cases}$$

In our case and due to the goal-reward approach, the convergence of the Q-Learning algorithm can be obtained by following the Q matrix until a stable value is obtained [136]. Line 2 of algorithm 2.11 in Section 2.3.3 could be replaced by "Repeat until Q converges".

The goal is to implement the case study of an IF invested in 3 stock indexes, CAC40, DJI and IXIC, using the DEVS-RL library and the DEVSimPy M&S environment [34]. A DEVSimPy modeling is proposed using the Agent-Environment RL approach combined with a discrete event Q-learning algorithm. Simulations have been done to obtain optimal policies that are able to manage an IF invested in stock market indexes.

4.1.1.1 DEVSimPy Modeling

Figure 4.3 details the DEVSimPy model of the case study presented in this section. The model includes five atomic DEVS models:

- The CAC40, DJI, and IXIC DEVS models: They are generator models that send on their output the index values collected during a period (stored in a csv file) or the real index values from the real stock market (using an API REST request). When a control message is received on the input port, the new index value is obtained and triggered on the output port.

- The Env atomic DEVS model: It is an environment component according to the Q-learning algorithm. When all the inputs are received, the possible states/action tuples are computed, and the reward is defined to 1 on the goal state. The model interacts with the Agent model through its port 0 to send the new tuple state/reward (line 4 in algorithm 2.11) and on its port 1 to activate the Agent model for a new episode (line 2 in algorithm 2.11).
- The Agent atomic DEVS model: It is an agent component according to the Q-learning algorithm. When a message is received on port 0, the agent updates the Bellman equation through its external transition function and sends to the Env model an action depending on the exploration/exploitation configuration defined in the model. When all steps are calculated and the goal is reached (line 9 of algorithm 2.11, the agent gives the best possible policy and sends a message on port 1 to the index generator models.

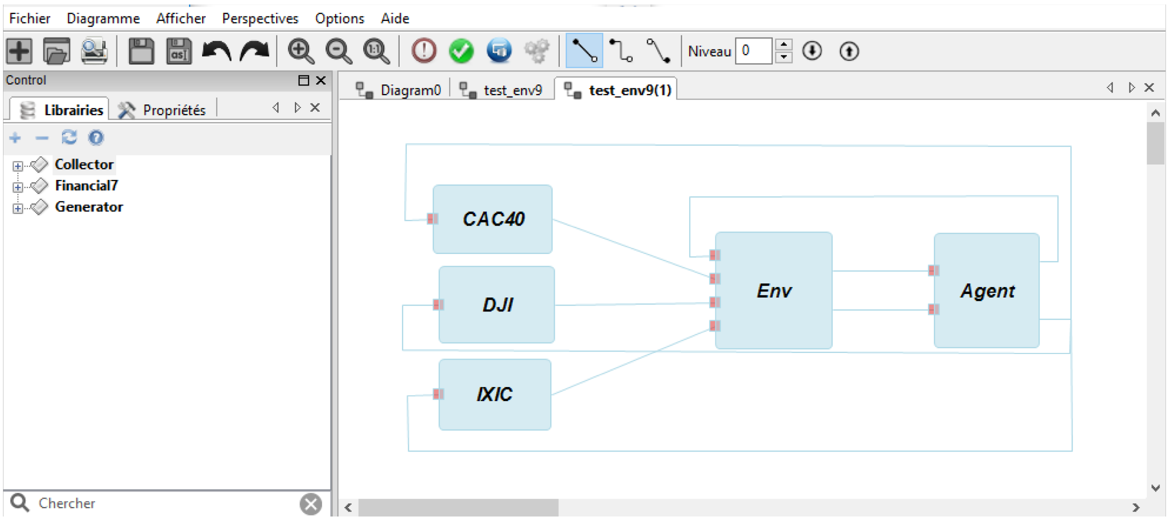


Fig. 4.3 DEVSimPy model that puts into action the DEVS-RL library (Section 3.4) with the three generator atomic models CAC40, DJI, and IXIC.

As depicted in Figure 4.3, the DEVS model presents two loops. The first is the traditional communication between the agent according to the Q-Learning algorithm (line 4 of the algorithm 2.11). The less traditional second allows us to consider the evolution of the indexes in a real stock market (line 2 of the algorithm 2.11). DEVS generator models (CAC40, DJI and IXIC in Figure 4.3) are connected to the output port 1 of the Agent model to run a new simulation (new episode) when the Env model generates a policy that solves a current simulation.

The next section is dedicated to the Q-Learning DEVSimPy simulations of this case study model. Two cases are considered. Single-episode case where the previous Q-Learning algorithm is executed only once and multiple-episode case where the Q-Learning is updated depending on the evolution of the indexes controlled by generators.

4.1.1.2 DEVSimPy Simulation

This section presents two types of simulation schemes. In the single episode case, only one set of indexes is simulated, and only one best possible policy is obtained at the end of the simulation. In the multiple-episode case, the indexes from 1991-01-02 to 2018-07-05 are simulated, and all optimal policies are stored and analyzed to validate our approach.

4.1.1.2.1 Single Episode Case : In this section, the model of Figure 4.3 has been simulated with the following settings:

- IXIC model index/volatility values: 360.200012/-0.01906318.
- CAC40 model index/volatility values: 1508.0/-0.025210084.
- DJI model index/volatility values: 2522.77002/-0.016881741.
- For the Env model: cash = 10000\$; $M = 1$; $N = 8$; init state = (1 IXIC, 1 CAC40, 2 DJI). These properties have been added to the initial Env class of the DEVS-RL library presented in Section 3.4.
- For the Agent model: $\alpha = 0.8$, $\varepsilon = 1$, and discount factor $\gamma = 0.95$. These properties were already available in the initial Agent class presented in Section 3.4.

The end of the simulation is obtained with the convergence of matrix Q. The simulation results give the path of state/action to reach the optimal goal state (0 IXIC, 0 CAC40, 6 DJI) among 250 possible states and 7 possible actions:

$$(1,1,2) \xrightarrow{\textit{[0,-1,0]}} (1,0,2) \xrightarrow{\textit{[0,0,1]}} (1,0,3) \xrightarrow{\textit{[0,0,1]}} (1,0,4) \xrightarrow{\textit{[0,0,1]}} (1,0,5) \xrightarrow{\textit{[-1,0,0]}} (0,0,5) \xrightarrow{\textit{[0,0,1]}} (0,0,6) \rightarrow \textit{wait}$$

Due to the Q-Learning algorithm, the proposed path can reach the goal state with a minimal number of transitions ($\varepsilon = 1$) and without cash loss.

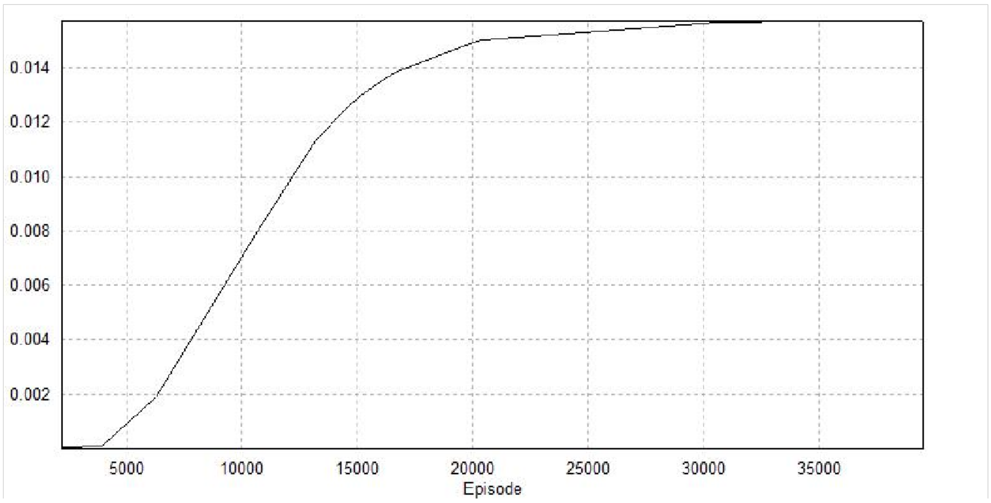


Fig. 4.4 Mean of the Q matrix in the Agent model for the single-episode simulation case after 35000 episodes.

4.1.1.2.2 Multiple Episode Case : In this section, the model of Figure 4.3 has been simulated with index/volatility values from the IXIC, CAC40, DJI models from 1991-01-02 to 2018-07-05 (Figure 4.5) and without changing the settings of the Agent and Environment models.

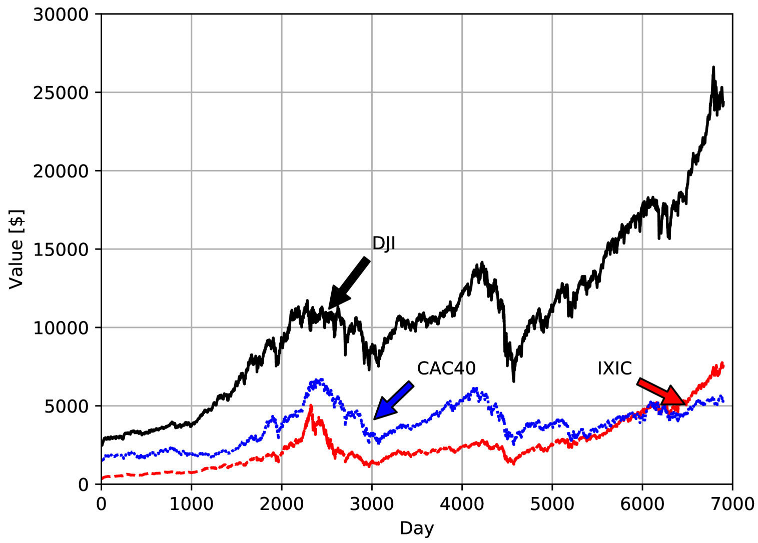


Fig. 4.5 Stock market indexes from 1991-01-02 to 2018-07-05.

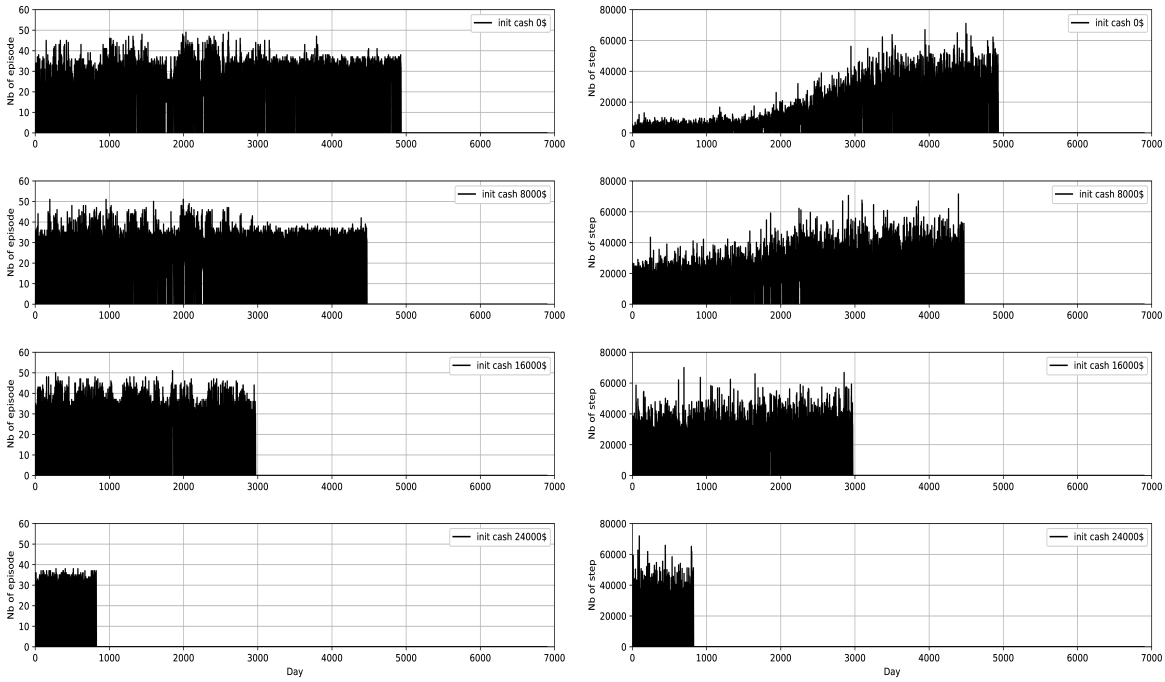


Fig. 4.6 Size of the episode (left part) and number of steps (right part) during the period and ordered by the initial cash scenario (from top to bottom: 0$, 8000$, 16000$, 24000$).

For all values of each index, the agent gives the best possible policy to obtain a leverage effect depending on the initial cash and the initial state. Scenarios with different initial cash values have been simulated and analyzed with a new algorithm that consists of determining, for all tuples' state/action, the maximum of the Q value during the evolution of the indexes.

The next scenarios are defined with an initial cash equal to 0$, 8000$, 16000$ and 24000$ added to the previous general setting.

Figure 4.6 shows the size of the episode and the number of steps during the simulations. One step is equivalent to a message between the Agent and the Environment. An episode consists of steps to find the best possible policy. As shown in Figure 4.6, the size of the episode (and the steps) is short, since the initial amount of cash is higher. It seems to be correct that the state (7,7,7) is reached earlier with a higher amount of initial cash. The sizes of the episode are ordered following the cash amount availability. Between 0 to 2000, concerning the number of steps, its seems obvious that there is a correspondence between the number of steps and the number of states generated by the amount of cash. In a specific case of initial cash 0, the growth of the number of steps at the same level of the other proves that there is a real growth in value of the state from day 4000 to last day of the simulation.

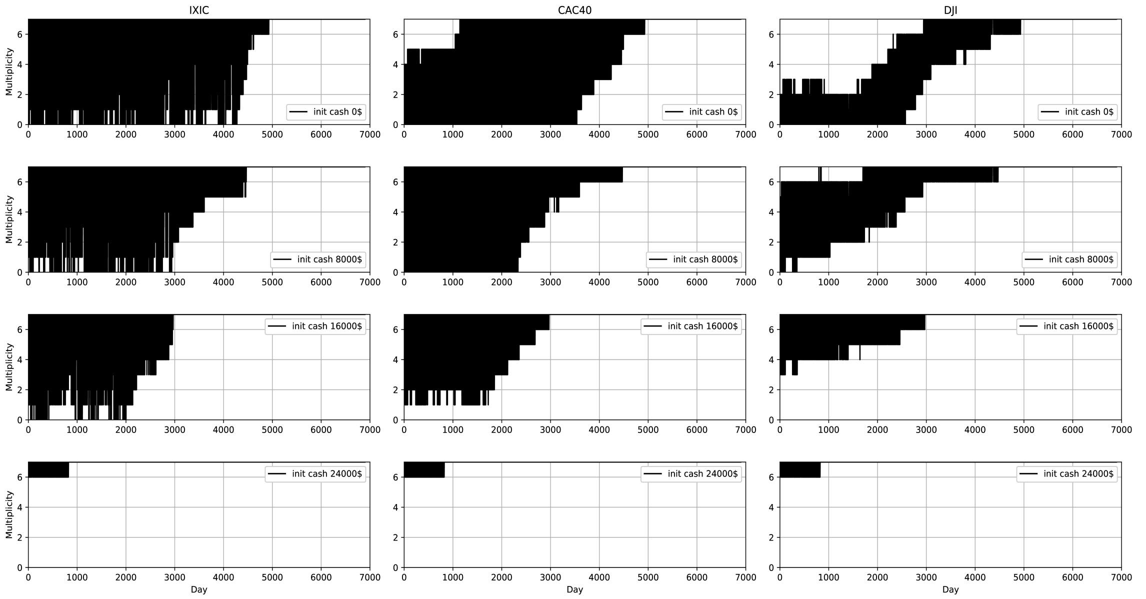


Fig. 4.7 Index multiplicity (IXIC on the left, CAC40 on the middle, and DJI on the right) during the simulation for the four scenarios depending on the initial cash (from top to bottom: 0$, 8000$, 16000$, 24000$).

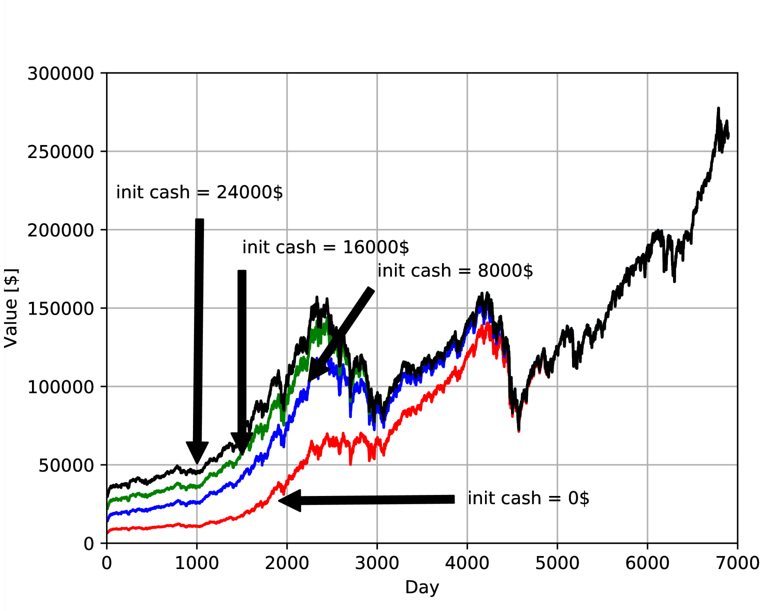


Fig. 4.8 Total assets (stock + cash) during simulations according to the period 1991-01-02 to 2018-07-05.

Figure 4.7 shows that the index multiplicity is reached more rapidly in the init cash 24000 scenario, which confirms that the Agent is acting to reach the best policy in the shorter "time" and the three other scenarios respect the rule. Figure 4.6 validates that the agent behaves in order to maximize the value in the portfolio; indeed, during the simulation period, it is only the best time to take an action for one specific state.

Figure 4.8 shows the variations of the total assets (stock + cash) during the simulations. Simulations validate that our Agent behaves correctly following the volatility and values of the indexes as shown in Figure 4.9, which also validate that the Agent respects the learning rules, indeed even when the values of all indexes are dropping, the Agent continues to try to reach the best possible state and at the same time to invest the maximum amount of cash available.

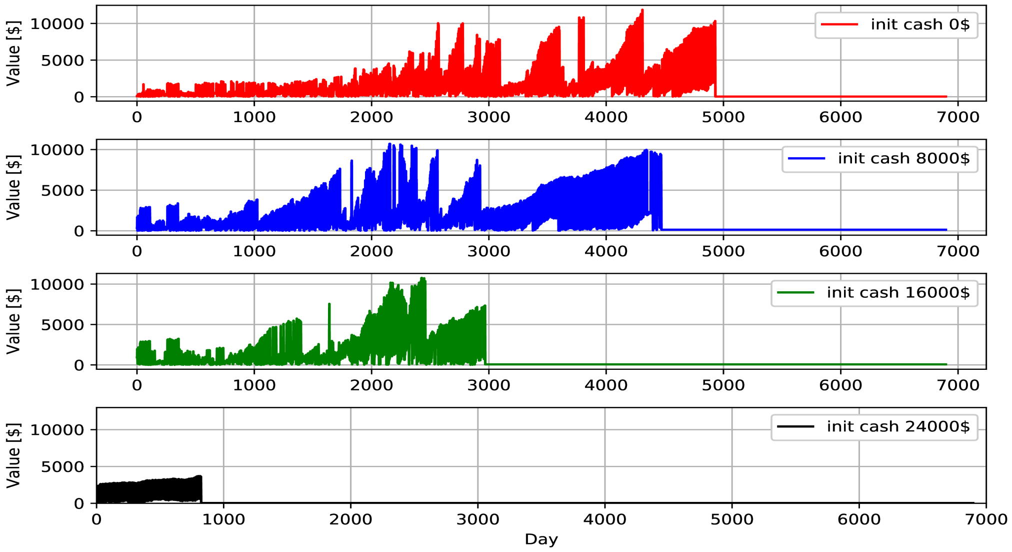


Fig. 4.9 Residual cash during simulations for the four scenarios.

Figure 4.9 also validates that cash is not a limiting factor in obtaining the best investment policy.

Figure 4.10, clearly validates that the Agent correctly invests the cash to try to reach the best investment policy as soon as it can; indeed, the time to have a residual cash close to 0 is faster with a higher initial cash, and the three other scenarios respect the same order. Figure 4.10 also validates that the agent has a real leverage effect even with the lowest initial cash; In fact, if we compare the evolution in the period of the value of the initial investment portfolio (1 IXIC,1 CAC 40,2 DJI) that reaches at 2018-07-05 the value of 63 thousand and the value of the same initial portfolio driven by the Agent. In this second case, the final value of the portfolio is 252 thousand, i.e., a leverage effect 4 times higher than the same portfolio

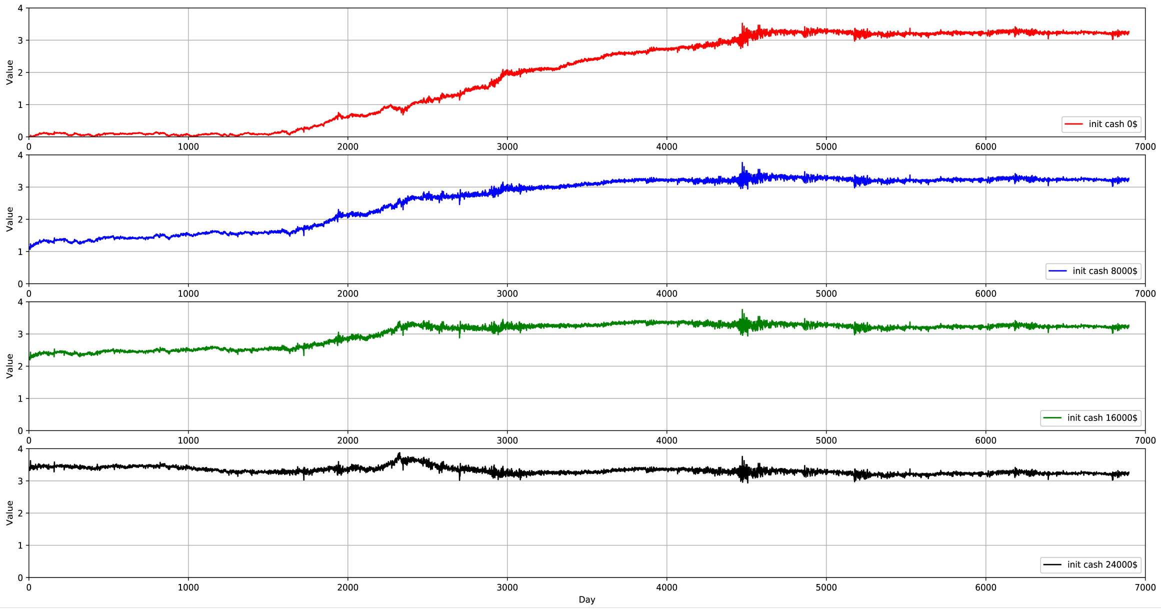


Fig. 4.10 Cash investment time (with initial cash from top to bottom: 0$, 8000$, 16000$, 24000$).

with no trades (buy and sell), and the four scenarios arrive at this value correctly following the residual cash order.

4.1.1.2.3 Discussion : This research effort aimed to provide samples and showed that DEVS can promote trust in algorithm-driven decision-making. Generally speaking, modeling is the process of representing a model which includes its construction and working. One of the main steps in modeling is to create a model that represents a system, including its properties. To achieve a model, it is fundamental to establish an experimental frame, i.e. the specification of the conditions under which the system is observed or experimented with. Most of the time, the system is a mathematical object, but mathematical representation may be challenged by AI to model some real-world applications, in particular those with a high degree of uncertainty. The specification of the conditions needs a fairness approach. This fairness approach plays an important role in AI explainability AI. Machine learning Fairness can be considered as a subdomain of machine learning interpretability that focuses solely on the social and ethical impact of machine learning algorithms by evaluating them in terms of impartiality and discrimination. To gain insight into the behavior of the system or to support a decision-making process, simulation helps to strengthen modeling processes.

One of the reasons we decided to use DEVS is because it helps us alter the model construction process and is a way to ensure fairness in machine learning models. DEVS formalism will allow us to put into action a fundamental strategy to guarantee fairness in explainability. The first strategy, which is called suppression, detects the features that correlate the most, according to some threshold with any sensitive features, such as stock market influencers opinions or economic results claims. To reduce the impact of sensitive features on model decisions, sensitive features are removed, along with their most correlated features, prior to training. This forces the model to learn from and, therefore, to base its decisions on other attributes, thus not being biased against certain market trends.

The first experiment shows how the observability made possible thanks to DEVS makes it possible to visualize the step and episode signals, thus improving the explainability of the RL model and also strengthening confidence in the RL model. In addition, access to the cash signal, which is specific to the application, also makes it possible to validate the model by simply interception of an event by an observer atomic model in the DEVS modular framework.

4.1.2 Compatibility and Combination

We decide to carry out compatibility of our DEVS-RL simulation model with the EF and a combination with the LSTM model. In this compatibility, we focus on building trust applications.

4.1.2.1 Efficient Frontier vs. DEVS

ESV Model : Our efficient frontier semivariance (ESV) portfolio consists of N stocks with $S_0 = (s_1^0, \dots, s_N^0)$ being the set of initial values for each stock in the portfolio. The number of each stock in the portfolio is denoted as $X = (x_1, \dots, x_N)$. The initial value of the portfolio V_0 is calculated as follows:

$$V_0 = \sum_{i=1}^{N} x_i s_i^0.$$

The decision on the number of shares in each asset is expressed as the weights $W = (w_1, \dots, w_N)$ with the constraint $\sum_{i=1}^{N} w_i = 1$, defined by $w_i = \frac{x_i s_i^0}{V_0}$ with $i = \{1, \dots, N\}$. At the end of the period t, the values of the stocks change $S_t = (s_1^t, \dots, s_N^t)$, which gives the final value of the portfolio V_t as a random variable:

$$V_t = \sum_{i=1}^{N} x_i s_i^t.$$

The actual return of a portfolio $R_p = (r_1, \dots, r_N)$ is the set of random returns for each stock in the portfolio, and the vector of expected return of $\mu = (\mu_1, \mu_2, \dots, \mu_N)$ with $\mu_i = E(r_i)$ for $i = 1, 2, \dots, N$. The actual return on multiple asset portfolios over a specified period of time is easily calculated as follows.

$$R_p = w_1 r_1 + w_2 r_2, \dots, w_n r_n.$$

The expected portfolio return is a weighted average for each asset in our portfolio. The weight assigned to the expected return of each asset is the percentage of the asset's market value to the total market value of the portfolio. Therefore, the expected return $E(R_p) = \mu_p$ of the portfolio at the end of the period t is calculated as follows:

$$E(R_p) = \sum_{i=1}^{N} w_i \mu_i.$$

The ESV has been implemented in the Google Colaboratory Python Environment and is detailed in Appendix 9. The four main indexes of the US market are Nasdaq Composite (IXIC), Dow Jones Industrial Average (DJI), S&P 500 (GSPC), and Russell 2000 (RUT) have been chosen. For this model, the total value of the portfolio is equal to 100,000$ with a weight distribution of 25% per asset.

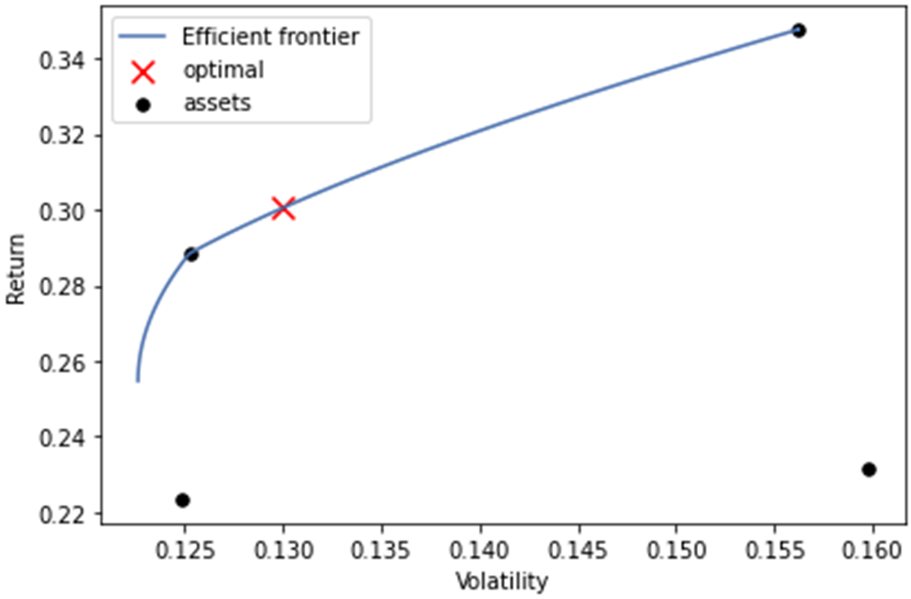


Fig. 4.11 Efficient Frontier for the IXIC, DJI, GSPC, and RUT indexes with no short sale in the period from January 1, 2019 to January 1, 2020. The optimal solution (red cross) obtained for an expected annual return equal to 30.0%.

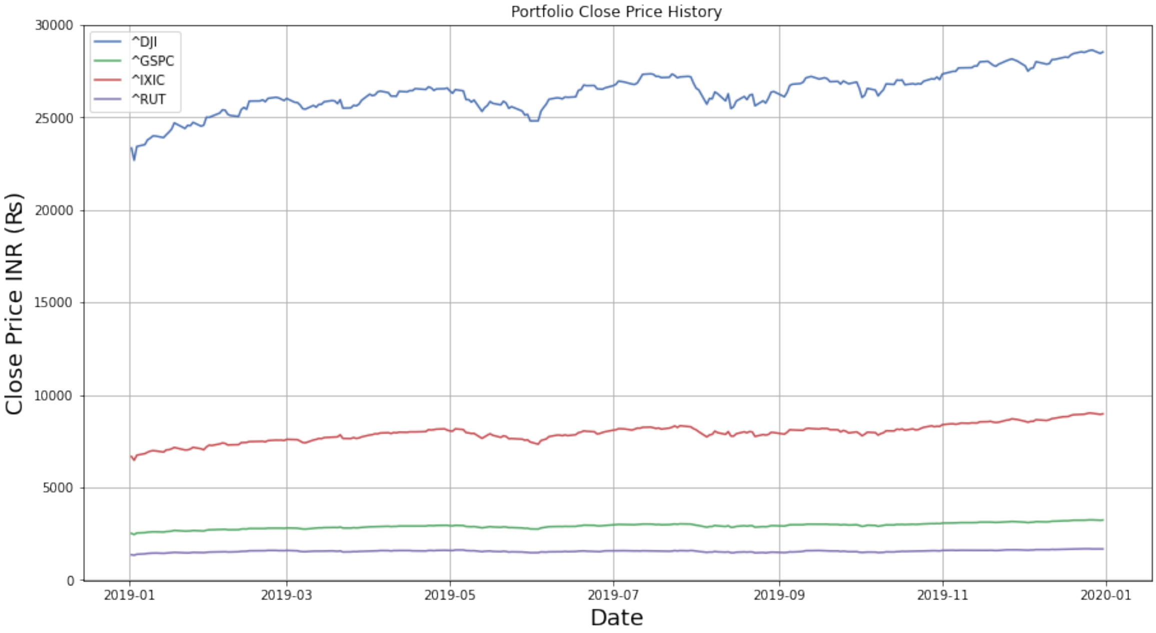


Fig. 4.12 IXIC, DJI, GSPC, and RUT index values for each day of the period from January 1, 2019 to January 1, 2020.

Figure 4.12 shows the values of the IXIC, DJI, GSPC, and RUT indexes each day of the period from January 1, 2019 to January 1, 2020. Figure 4.11 shows the efficient frontier with the associated expected annual return value and the volatility (risk) of optimizing the ESV portfolio over a period from January 1, 2019 to January 1, 2020. The frontier is obtained in a few seconds and gives portfolio optimization allocations within a range of expected annual return from 25% up to 35%. The optimal one is (50 GSPC, 4 IXIC) with an expected annual return equal to 30%. The ESV's efficient frontier would be compared to the expected annual return obtained with DEVS-RL with two different N.

DEVS-RL Model : The DEVSimPy simulation model has been implemented as shown in Figure 4.13.

The DEVS agent atomic model has been configured with $\alpha = 0.95$, $\gamma = 0.2$, and $\varepsilon = 1.0$ (ε-greedy) as shown in the Properties panel in Figure 4.14. The DEVS Env atomic model has been configured with $M = 1$ (action multiplicity), $cash = 0$ (initial cash) and with the goal-reward activated as shown in Figure 4.15. The parameter N is manually fixed depending on the initial value of our portfolio. N is chosen with two constraints. The first is to avoid the risk that the agent will invest all its initial cash just in one asset (the one with the highest volatility) by limiting the number of stocks per asset (not all the eggs in the same basket). The second constraint is to invest at least all its initial cash. Among all possible values of N

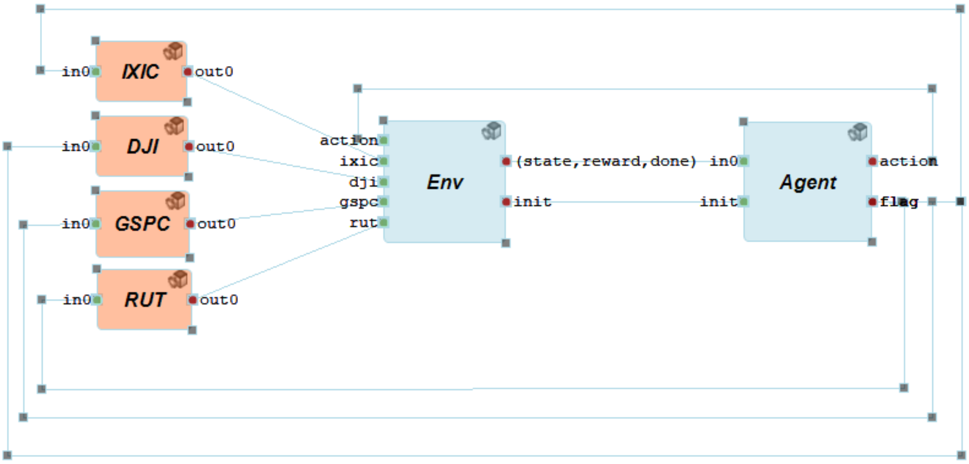
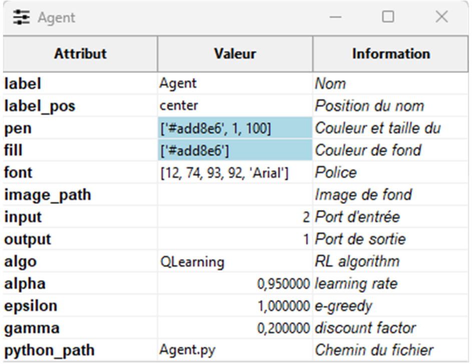


Fig. 4.13 DEVSimPy simulation model with DEVS-RL agent and environment atomic models and the four DEVS generators (IXIC, DJI, GSPC, and RUT) that send the index values for each day of the period from January 1, 2019 to January 1, 2020.

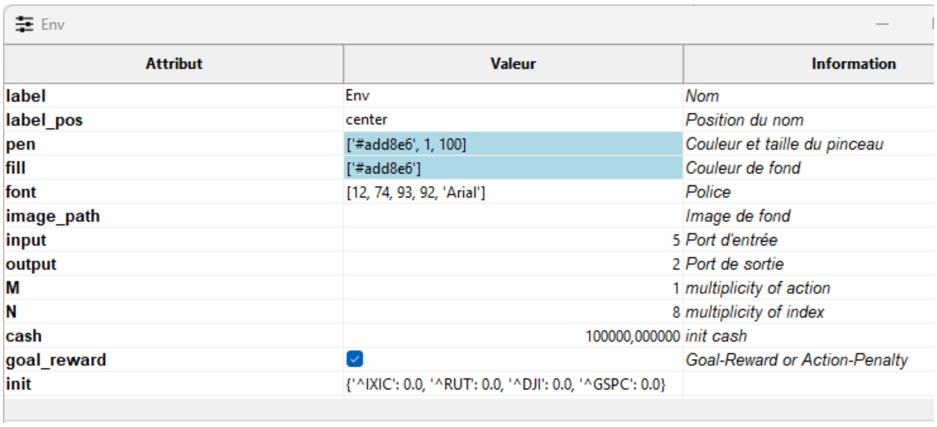
that satisfy the constraint, the final value that will be chosen is the one that will result in an expected annual performance close to the ESV benchmark, which is the optimal allocation of ESV or at least one of the allocations on the efficient frontier of ESV. Note that the higher the value of *N*, the larger the number of states to consider and the longer the search for the optimal policy. The initial state (the first state on the path formed by [IXIC, RUT, DJI, GSPC]) is defined to have the same total portfolio value as the ESV (100,000$).



Attribut	Valeur	Information
label	Agent	Nom
label_pos	center	Position du nom
pen	['#add8e6', 1, 100]	Couleur et taille du
fill	['#add8e6']	Couleur de fond
font	[12, 74, 93, 92, 'Arial']	Police
image_path		Image de fond
input	2	Port d'entrée
output	1	Port de sortie
algo	QLearning	RL algorithm
alpha	0,950000	learning rate
epsilon	1,000000	e-greedy
gamma	0,200000	discount factor
python_path	Agent.py	Chemin du fichier

Fig. 4.14 DEVSimPy properties of the atomic model Agent with a configuration defined for the compatibility experiment.

DEVSimPy simulations have been performed with DEVSimPy Version 4.0 and the following software/hardware characteristics: Windows 11; Python 3.9.13; 12th Gen Intel(R)



Attribut	Valeur	Information
label	Env	*Nom*
label_pos	center	*Position du nom*
pen	['#add8e6', 1, 100]	*Couleur et taille du pinceau*
fill	['#add8e6']	*Couleur de fond*
font	[12, 74, 93, 92, 'Arial']	*Police*
image_path		*Image de fond*
input	5	*Port d'entrée*
output	2	*Port de sortie*
M	1	*multiplicity of action*
N	8	*multiplicity of index*
cash	100000,000000	*init cash*
goal_reward	☑	*Goal-Reward or Action-Penalty*
init	{'^IXIC': 0.0, '^RUT': 0.0, '^DJI': 0.0, '^GSPC': 0.0}	

Fig. 4.15 DEVSimPy properties of the Env atomic model with a configuration defined for the compatibility experiment.

Core(TM) i9-12900H 2.50 GHz and 64GB of RAM. The results obtained with ESV are compared with those obtained by simulation with DEVS-RL shown in Figures 4.16.

Now, let us experiment with the effect of search-by-exploration on the part of the agent. It is necessary to vary the parameter epsilon (for example, with the values (0.97, 0.95, 0.9, 0.8, 0.7, 0.6, 0.5)) in each simulation and observe the policy obtained at the end of the simulations. Figure 4.17 shows the tree obtained when we merge the paths of each simulation for the date September 1, 2022. From the initial state (2,2,2,2), the final state is reached by taking different paths depending on the value of ε noted on the arc. For example, if $\varepsilon = 0.97$ the end state is reached by going through 9 states ((2,2,1,2), (3,2,1,2), (4,2,1,2), (4,2,0,2), (5,2,0,2), (6,2,0,2), (7,2,0,2), (7,2,0,1), (7,2,0,0)) for more than 4 minutes of simulation time and the path is (2,2,2,2) $\rightarrow$ (2,2,1,2) $\rightarrow$ (3,2,1,2) $\rightarrow$ (4,2,1,2) $\rightarrow$ (4,2,0,2) $\rightarrow$ (5,2,0,2) $\rightarrow$ (6,2,0,2) $\rightarrow$ (7,2,0,2) $\rightarrow$ (7,2,0,1) $\rightarrow$ (7,2,0,0) $\rightarrow$ (8,2,0,0) $\rightarrow$ wait. If $\varepsilon = 0.5$, the number of intermediate states increases (13) and the simulation time is longer than 2 hours.

Discussion : With N equal to 8 and 14, the allocation of the DEVS-RL optimization portfolio offers higher expected returns than $N = 4$, but does not match the efficient frontier. In contrast, with $N = 4$ the DEVS-RL optimized portfolio became one of the solutions of the efficient frontier with an expected annual return equal to 34.72%. The compatibility with ESV definitely suggests that our novel DEVS-RL portfolio optimization model based on the Markov property is valid and offers at least one of the possible solutions of an ESV model. Moreover, the same results in terms of optimization are achieved without the need to take historical returns into account, turning the optimization process into a simpler interpretable model due to the benefits of the RL algorithm easily implemented with DEVS. This verdict

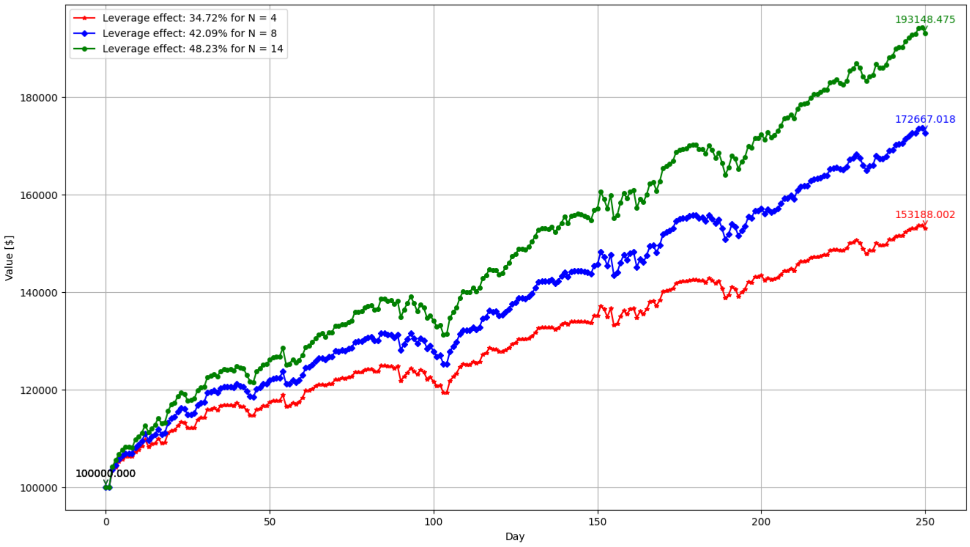


Fig. 4.16 Leverage effect after DEVS-RL simulation from 100,000$ with N equal to 4,8 and 14 from January 1, 2019 to January 1, 2020. The leverage effect for $N = 4$ is 34.72% and the discrete allocation is (3 IXIC, 3 RUT, 3 DJI, 3 GSPC) that corresponds to one of the efficient frontier expected return points. For $N = 4$ at the end of 2019, the current value of the portfolio is 153 188.002 for an initial capital investment of 100,000$. The initial capital represents 65.28% of the current value. The leverage effect for 2019 is 34.72% with a return ratio of 53.18% which represents the gain for $N = 4$ in 2019.

also suggests that it would be challenging to further investigate the relationship between volatility, return, and volatility return and to offer a novel approach to reconsider the market risk exposure as defined by VaR, i.e., the standard for banking and insurance companies. Figure 4.16 shows that around day 100 all DEVS-RL portfolio values with N equal to 4,8, and 14 decrease by approximately 7%. The lost value is recovered after 20 days. On day 100 all portfolio indexes lost values by about 4% to 6%. It seems important to anticipate discrete events such as the Bear market using a supervised neuronal network prediction model such as LSTM or GARCH. In the next section, we introduce a combination of the LSTM and the DEVS-RL model to improve portfolio optimization using an index trend prediction model.

4.1.2.2 LSTM Model Combinated with DEVS-RL

Unlikely or in combination, GARCH and LSTM are commonly used to predict volatility of commodity market returns and stock index [106, 208]. Recent work highlights that

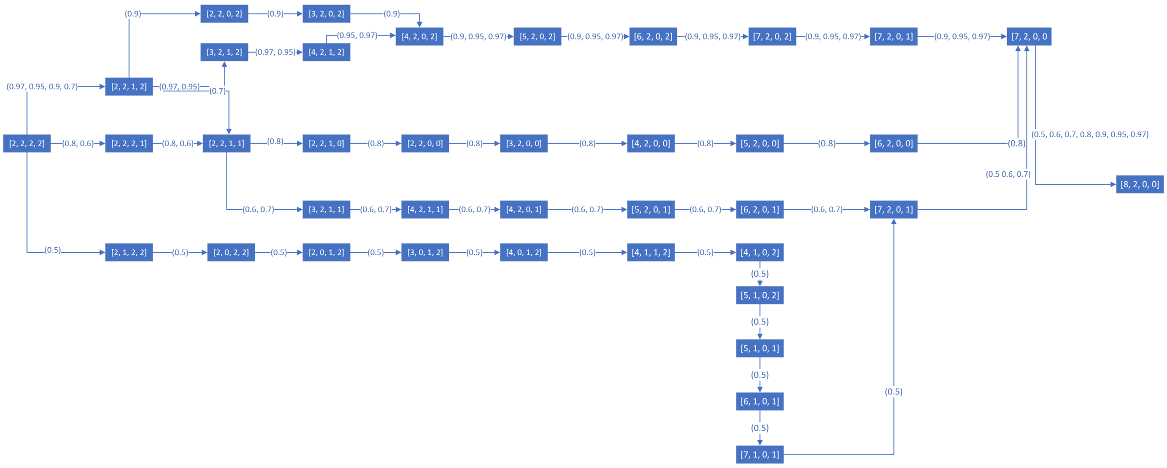


Fig. 4.17 Family of paths obtained after simulations with different values of ε (noted on each arc). Depending on the selected ε, each path traced the evolution of the indexes. If ε is (resp. far from) 1.0, the path is obtained in a minimal (resp. maximal) time due to the exploitation (resp. exploration) policy executed by the agent (ε-greedy algorithm).

combining the information of the GARCH types into an LSTM model leads to a superior volatility forecasting capability [95]. As many other authors, our combined architecture, as shown in Figure 4.21 uses GARCH and LSTM distinctively, as well as together, to lead to portfolio construction based on volatility of the risk model. The novelty of our approach is to propose a prediction model using distinctively or combined GARCH and LSTM and an optimization model based on DEVS-RL with the possibility of benchmarking with ESV the evaluation of the results of portfolio optimization.

Our LSTM models have been coded under the Google Colaboratory Python Environment and the Google GPU accelerator (see Appendix 10).

Figure 4.18, shows the four plots of our predictions for the index data from September 2012 to November 2022. Visually, the level of one-year-horizon prediction seems pretty good.

To evaluate the prediction error rates and performance of the LSTM model, the root mean squared (RMSE) was used to measure the difference between the predicted and practical data. RMSE has been calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - y_i^{'})^2}{n}},$$

where $Y = (y_1, y_2, \ldots, y_n)$ is a vector of actual observations, $Y^{'} = (y_1^{'}, y_2^{'}, \ldots, y_n^{'})$ is a vector of predicted values, and n is the number of observations.

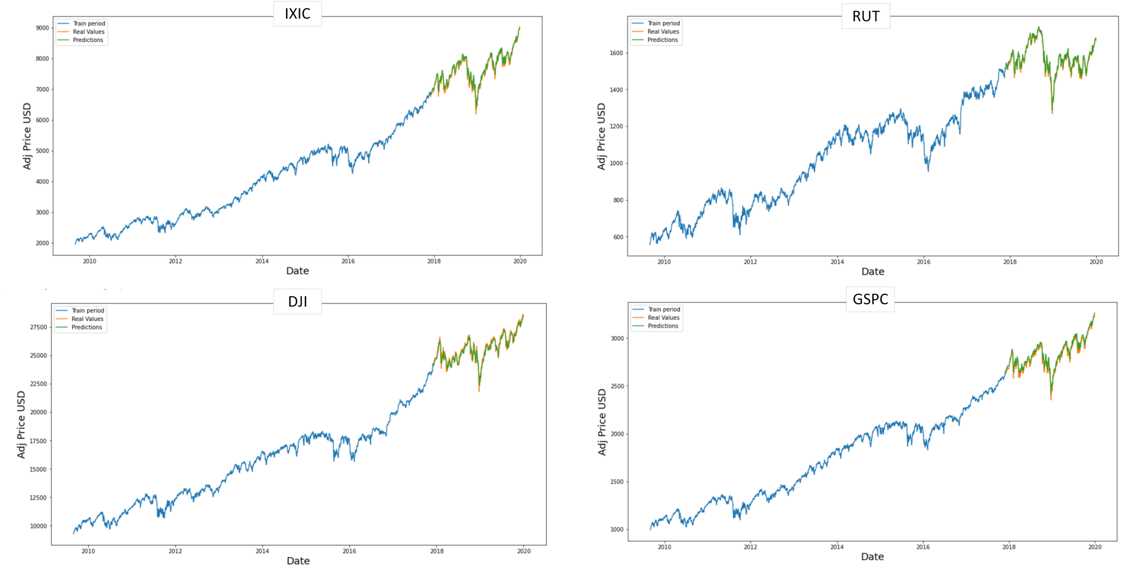


Fig. 4.18 LSTM index predictions from September 2012 to November 2022. The blue, green, and orange lines represent, respectively, the real values during the training period, the real values during the prediction period, and the predicted values. Visually, the orange and green lines seem to be superimposed.

The RMSE, that is, the weighted average error between the predictions and the actuals, is shown in Table 4.1.

Index	RMSE	AP	RMSE/AP [%]
IXIC	107.78	7965.39	1.35
RUT	18.11	1551.52	1.17
DJI	303.20	26232.78	1.15
GSPC	37.46	2933.76	1.27

Table 4.1 LSTM models prediction performance. The RMSE values confirm that our LSTM models compute predictions on a one-year horizon with a high degree of precision given the respective 2019 average price (AP).

We also tried the same LSTM architecture with 10 to 100 epochs, but the prediction was actually close.

Figure 4.19 shows the integration of the LSTM DEVSimPy coupled model into our previous DEVS-RL library. The LSTM model contains the four generators that implement the Python code presented in Appendix 10 to have the index predictions.

Figure 4.20 shows the real and LSTM predicted leverage effects after the DEVS-RL simulation from 100,000$ with $N = 4$ from January 1, 2019 to January 1, 2020. $N = 4$ is one

Index	RMSE/AP 1 epochs [%]	RMSE/AP 10 epochs [%]	RMSE/AP 100 epochs [%]
IXIC	1.35	1.34	1.34
RUT	1.17	1.17	1.16
DJI	1.15	1.14	1.13
GSPC	1.27	1.27	1.25

Table 4.2 LSTM models prediction performance with three different epoch settings. With 1 epoch, the computational time of the model was about 72 seconds, and every next epoch needs 71 seconds more. The results in terms of prediction do not vary significantly.

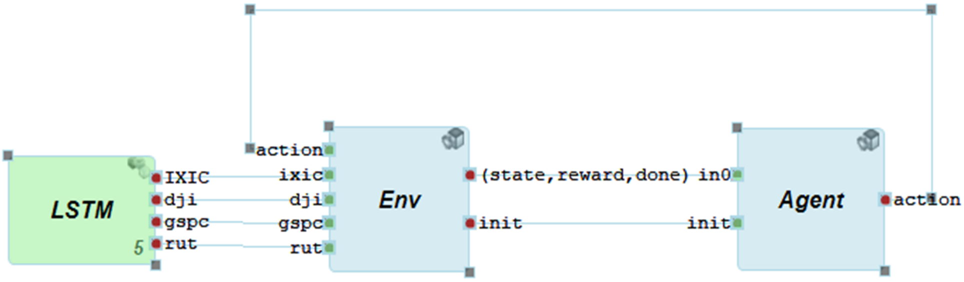


Fig. 4.19 LSTM and DEVS-RL combination. LSTM coupled model contains the four generator model (for the four indexes) based on the predicted data extracted from the LSTM algorithm.

of the solutions of the ESV efficient frontier. The expected return lines for real and predicted data are following the same trend, which means that respectively the lowest points and the highest points of both lines are simultaneous. Actually, the leverage effect for predicted data is 10.82 lower than the real data one, this is probably due to the cumulative effects of RMSE. But this signal is still an important signal to exploit for portfolio optimization because it indicates when it is time to sell (highest points) and when it is time to buy (lowest points) no matter what is the price.

Discussion Our LSTM models would probably need to be perfected to obtain RMSE values closer to zero. However, the actual predictions of the LSTM model will be useful for predicting the bull and bear markets. The bull market is characterized by a rise of stock prices. On the contrary, a bear market is a period characterized by the beginning of a decline in stock prices. In reality, a bear market is defined as a period with a decline of 20% of the main index. In our case study, we consider any period with more than a week of price decline as a time signal from the bear market. LSTM prediction will send a signal to DEVS-RL

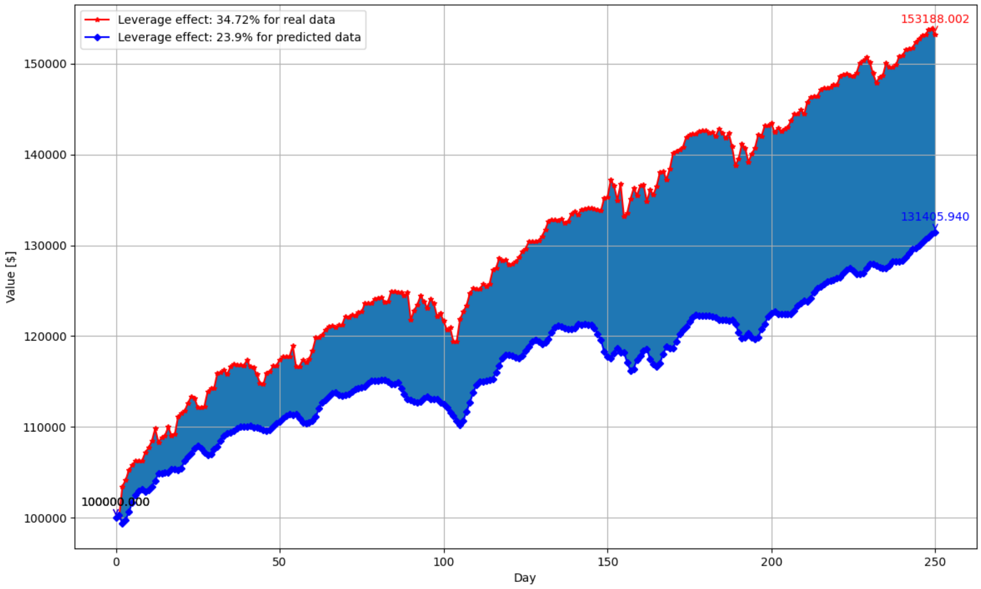


Fig. 4.20 The real and LSTM predicted leverage effects after the DEVS-RL simulation from 100,000$ with $N = 4$ from January 1, 2019 to January 1, 2020.

to seek or not seek an optimized portfolio. During the Bull market, predicted by LSTM, DEVS-RL will launch the simulation to trade the portfolio in order to get to the optimal one. In contrast, with a LSTM predicted bear market signal, DEVS-RL will not launch the simulation to avoid trading in a period of declining stock prices with the certainty of losing portfolio value. The Bull market and the Bear market are discrete events, and the DEVS formalism perfectly fits with this constraint.

4.1.2.3 ESV, LSTM and DEVS-RL Combination

The portfolio presented in Figure 4.18 was constructed by prediction the Bear and Bull Market. DEVS-RL simulation may use or not the signal of the Bear Market to avoid simulation until the Bull market is predicted by the LSTM. Then, the optimal allocation of stocks for the constructed portfolio was evaluated by simulation and optimization modeling, where DEVS-RL and ESV were used to evaluate the optimal allocation weights of stocks. In Figure 4.21, the combination can result in a model based on forecasting of price returns through supervised neural networks, followed by a decision policy driven by an MDP model implemented with the DEVS formalism. In this combination section, we tried to route the

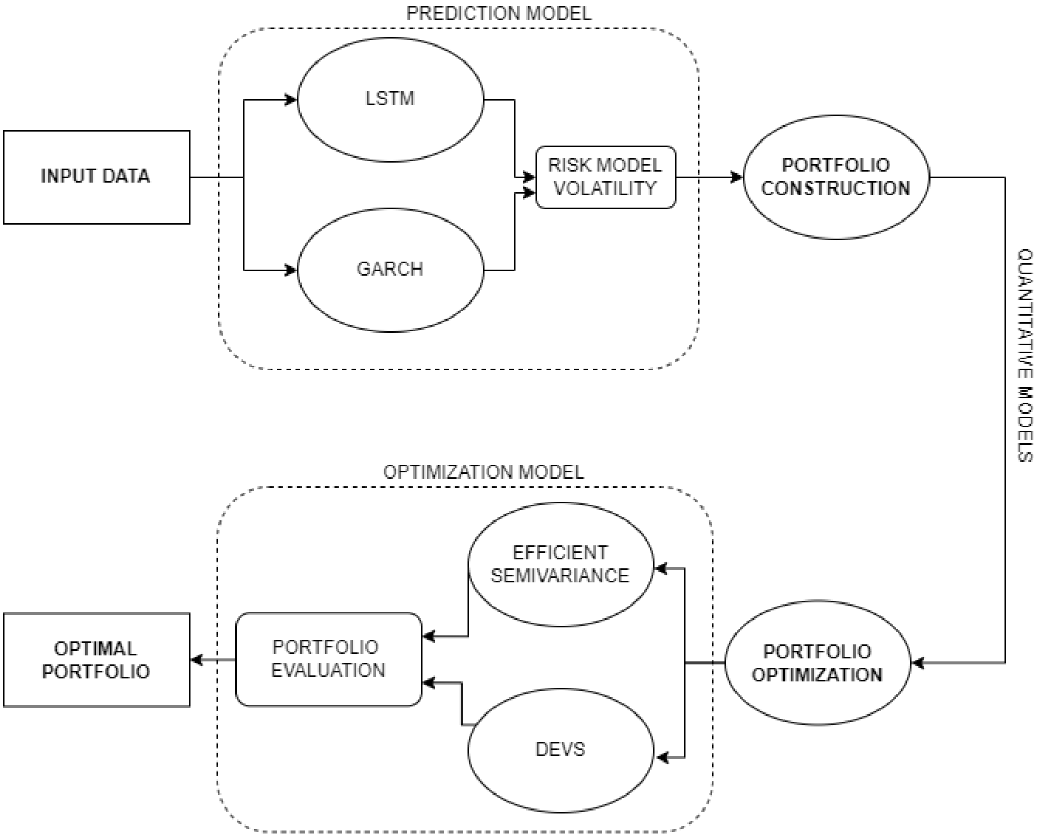


Fig. 4.21 Our combined architecture from input data to optimal portfolio. The prediction model use distinctively or combined LSTM and GARCH to build a portfolio based on risk model volatility. Next, the portfolio optimization is based on DEVS-RL and the results are evaluated by Efficient Semivariance.

way to formulate the hypothesis that, while preserving computational efficiency, it is possible to improve the financial performance of the forecast-based approach by a better optimization of the trading agent's financial performance. In particular, our aim is to propose lines of research for future work that will focus on more adequate ways of designing the training patterns from those price data features that change over time; new trading rules based on different forecast horizons; and the use of adaptation rules able to cope with transaction costs.

4.2 Conclusion

The case study deals with a decision-making process carried out by traders during a process of optimizing asset management that leads to a possible leverage effect.

This section detailed a first experiment in which simulation models were used to prove that our DEVS-RL combines short-term financial leverage with a sustainable strategy over a 30-year period. The validity of asset management is highlighted by earnings that reach three times the value of three of the main world stock markets managed naively.

Furthermore, this section shows how the DEVS formalism makes it possible to implement an RL system in a formal framework, allowing one to benefit from a better modularity and explanation of the AI models. We also highlighted the effectiveness of the proposed approach in terms of complementarity and combinability with other AI tools such as ESV and LSTM. We also provided evidence of how the modular aspect of DEVS-RL allowed one to build decision and prediction models through simulation combined with other AI algorithms.

Chapter 5

Conclusions and Future Works

The objective of this thesis is to propose a decision-making management policy for complex systems evolving in highly dynamic environments, combining the DEVS formalism and the RL technique. This work raises a number of questions mentioned in the Introduction, which can now be answered.

What are the advantages of coupling DEVS simulation and AI techniques, particularly reinforcement learning, in the decision-making process of a complex system?

In the simulation, ML makes the simulation learn some aspects and directly apply these lessons. This coupled approach is useful when teaching ML algorithms in specific parts of the simulation model itself, such as the agent trying to find paths within the simulation model environment. To mitigate the risk of delegation to machines in decision-making processes, DEVS offers the opportunity to remain under observation, interactions, and separation between the agent and the environment involved in the traditional RL algorithm, such as Q-Learning. In RL, an agent seeks an optimal control policy for a sequential decision problem. Unlike in supervised learning, the agent never sees examples of correct or incorrect behavior. Instead, it receives only positive and negative rewards for the actions it tries. Since many practical real-world problems (such as robot control, game play, and system optimization) fall into this category, developing effective RL algorithms is important to the progress of AI. The Q-Learning algorithm needs to be formally separated in a DEVS Agent component that reacts with a dynamic DEVS Environment component. This modular capability of DEVS permits a total control of the Q-Learning loop in order to drive the algorithm convergence.

How the DEVS formalism is able to improve understanding of AI algorithms?

Simulation models are built 'process-centric', while ML models are built 'data-centric'. The discrete event-oriented modeling of the Agent-Environment RL system by the DEVS formalism allows us to explore the Q-Learning and SARSA algorithms in a more behavioral way. In general, the Q-Learning and SARSA algorithms are based on the nesting of two loops: a repetitive loop on a number of episodes, which includes a conditional loop on actions. The DEVS discrete event approach makes it possible to dissociate these two loops through the communication of two models (atomic or coupled) Agent and Environment. This makes it easier to interact with the two learning algorithms in order to implement specific stopping conditions, for example. In addition, the modularity provided by the DEVS formalism makes it possible to divide the algorithms into an interconnection of atomic models, thus improving the experimentation of new calculation methods to update the variable Q or the method of allocating new actions by the agent.

The coupled models approach has the advantage of facilitating the implementation of a multi-agent model for RL. Indeed, the Agents coupled model can be composed of an interconnection of Agent atomic models that communicate both with each other and with the environment. The DEVS formalism is a good candidate for setting up a multi-agent model. In addition, the possibility of simulating coupled models in parallel thanks to PDEVS is an interesting avenue when one wants to set up multi-agent models which can lead to significant simulation times. We do not present this aspect in this thesis.

How does the association of RL algorithms with DEVS simulation improve the management of leverage effects in financial management processes? The research work proposed in this thesis contributes to answering these questions using a concrete case of optimizing the management of a portfolio of stocks using discrete event simulation. The DEVS-RL models results clearly validate that the Agent correctly invests the cash to try to reach the best investment policy. This research effort provided samples and showed that DEVS can promote trust in algorithm-driven decision making. We highlighted several advantages brought by the coupling of DEVS simulation and reinforcement learning in the decision-making process of a complex system by offering the opportunity to keep the Q-Learning loop under observation including all the interactions, and due to the modular and hierarchical aspect of DEVS, the separation between the agent and the environment involved in the traditional RL algorithm, such as Q-Learning, is improved. We also gave samples of how the DEVS formalism is capable of improving the understanding and deployment of AI algorithms by mitigating the risk of delegation to machines in decision-making processes. We also provided evidence of how the association of RL algorithms with DEVS simulation improved the management of leverage effects in asset management by offering complementarity and combinability with the most explored algorithms and supervised IA in the data finance literature. DEVS-RL model allowed to produce leverage effects three times higher than some of the most important market places in the world over a thirty-year period and may contribute to addressing the Active portfolio theory with a novel approach that question the need to distinguish between patient, aggressive, and conservative behavior. Bridging portfolio theory and RL, DEVS-RL combined all three in one by maximizing returns as an aggressive approach and having a positive impact (thirty years) on long-term yield and stability as a conservative approach with income earning despite economic conditions as wished by a patient portfolio investor.

In perspective, the next stages of our research will be to add to Q-Learning the estimation of the Bellman equation by a neural network (Deep Q-Networks). In fact, the Q-Learning algorithm based on table management reaches these limits when the number of states increases. The use of neural networks makes it possible to have an approximator of the variable Q and makes it possible to consider a much larger number of possible states. This development should allow us to avoid having to estimate the rewards upstream and to respond to the increase in the number of states generated by considering new constituent factors of the environment.

Another solution to deal with a large number of state frameworks is to consider multi-agent approach. Instead of letting one single agent iterate in a single large environment, the task is divided into multiple agents that explore a part of the number of states. Each agent gives an optimal policy that results in the execution of the Q-Learning algorithm, depending on the interaction with its environment. Once the supervisor has received all agent policies at each simulation time step, it determines the best combined policy and communicates to the single agent the action to be taken. The best-combined policy is determined as follows: sell the indexes with the lowest volatility or a negative one and buy the ones with the highest volatility. These two actions are limited by the amount of cash available to be added to the indexes prices. The agents interact through the supervisor. Agent policies communicated to the supervisor will represent an opportunity or a constraint for the other agents. The supervisor synchronizes the interaction between agents. Learning takes place inside the agents. The construction of the best-combined policy is the result of a process of trial-and-error by reward return. If the environment changes, the process must be restarted. A policy is applicable for a given environment. In that context, we may consider the system as a dynamic structure DEVS model with a static structure of the coupled DEVS model in which the number of agents/environment can vary over time and can be executed in parallel. The supervisor model builds its policy by integrating all the policies and experiences that each agent has already learned.

To broaden the scope of application, the compatibility between the DEVS-RL portfolio optimization results and the ESV model results also suggests that it would be interesting by integrating GARCH to further investigate the relationship between volatility, return, and volatility return and to offer a novel approach to reconsider market risk exposure as defined by the VaR, the standard for banking and insurance companies.

Chapter 6

List of Publications

1. E. Barbieri, L. Capocchi, J.F. Santucci, "DEVS Modeling AND Simulation Based on Markov Decision Process Applied to Financial Portfolio Management in a Volatile Environment", JDF 2021 - Les Journées DEVS Francophones - Théorie et Applications, Nov. 2-5, 2021, Cargese, France.

2. E. Barbieri, L. Capocchi, J.F. Santucci, "Multi-agent Discrete-Event Simulation Based Reinforcement Learning Algorithms Applied to Financial Leverage Effect", JDF 2020 - Les Journées DEVS Francophones - Théorie et Applications, Nov. 2-6, 2020, Cargese, France.

3. E. Barbieri, L. Capocchi, J.F. Santucci, Discrete-Event Simulation-Based Q-Learning Algorithm Applied to Financial Leverage Effect, J.F. SN Computer Science, 1: 50, Jan 2020. doi: https://doi.org/10.1007/s42979-019-0051-7

4. E. Barbieri, L. Capocchi, J.F. Santucci, "DEVS Modeling and Simulation of Financial Leverage Effect Based on Markov Decision Process", 4th IEEE International Conference on Universal Village, Boston, MA, USA, October 21-24, 2018, pp. 1-5, DO-10.1109/UV.2018.8642121UR.

5. E. Barbieri, L. Capocchi, J.F. Santucci, "DEVS Modeling and Simulation Based on Markov Decision Process of Financial Leverage Effect in the EU Developpment Programs", JDF 2018 - Les Journées DEVS Francophones - Théorie et Applications, 31 Avril - 4 Mai, 2018, Cargese, France.

6. E. Barbieri, L. Capocchi, J.F. Santucci, DEVS Modeling and Simulation Based on Markov Decision Process of Financial Leverage Effect in the EU Development Pro-

grams, in Proc. of Winter Simulation Conference (WinterSim'17), Dec. 3-6, 2017, Las Vegas, NV, USA, pp. 4558-4559. (Code Australia rank: B)

7. E. Barbieri, L. Capocchi, J.F. Santucci, DEVS Modeling AND Simulation Based on Markov Decision Process of Financial Leverage Effect in the EU Development Programs, poster à la Journée Des Doctorants (JDD), Université de Corse, 15 juin 2017.

References

[1] Abel, D., Dabney, W., Harutyunyan, A., Ho, M. K., Littman, M., Precup, D., and Singh, S. (2021). On the expressivity of markov reward. *Advances in Neural Information Processing Systems*, 34:7799–7812.

[2] Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: a survey on explainable artificial intelligence (xai). ieee access 6: 52138–52160.

[3] Addo, M. and Servon, L. (2021). Income volatility and health care decision-making. *Economic Studies at Brookings*.

[4] Akplogan, M., Quesnel, G., Garcia, F., Joannon, A., and Martin-Clouaire, R. (2010). Towards a deliberative agent system based on devs formalism for application in agriculture. In *Proceedings of the 2010 Summer Computer Simulation Conference*, pages 250–257.

[5] Alagoz, O., Hsu, H., Schaefer, A. J., and Roberts, M. S. (2010). Markov decision processes: a tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483.

[6] Allen, D. W. (1991). What are transaction costs? *Rsch. in L. & Econ.*, 14:1.

[7] Andersen, T., Bollerslev, T., and Hadi, A. (2014). *ARCH and GARCH models*. John Wiley & Sons.

[8] Andersen, T. G., Bollerslev, T., Christoffersen, P. F., and Diebold, F. X. (2006). Volatility and correlation forecasting. *Handbook of economic forecasting*, 1:777–878.

[9] Ardia, D., Boudt, K., Carl, P., Mullen, K., and Peterson, B. G. (2011). Differential evolution with deoptim: an application to non-convex portfolio optimization. *The R Journal*, 3(1):27–34.

[10] Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. (1999). Coherent measures of risk. *Mathematical finance*, 9(3):203–228.

[11] Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In Prieditis, A. and Russell, S., editors, *Machine Learning Proceedings 1995*, pages 30–37. Morgan Kaufmann, San Francisco (CA).

[12] Barton, D. (2011). Capitalism for the long term. *Harvard business review*, 89(3):84–91.

[13] Beach, S. L. and Rose, C. C. (2005). Does portfolio rebalancing help investors avoid common mistakes? *JOURNAL OF FINANCIAL PLANNING-DENVER-*, 18(5):56.

[14] Beccaria, E., Bogado, V., and Palombarini, J. A. (2021). A devs based methodological framework for reinforcement learning agent training. *IEEE Latin America Transactions*, 19(4):679–687.

[15] Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

[16] Benhamou, E., Saltiel, D., Ungari, S., and Mukhopadhyay, A. (2020). Bridging the gap between markowitz planning and deep reinforcement learning. *arXiv preprint arXiv:2010.09108*.

[17] Bennett, C. C. and Hauser, K. (2013). Artificial intelligence framework for simulating clinical decision-making: A markov decision process approach. *Artificial intelligence in medicine*, 57(1):9–19.

[18] Berkowitz, S. A., Logue, D. E., and Noser Jr, E. A. (1988). The total cost of transactions on the nyse. *The Journal of Finance*, 43(1):97–112.

[19] Bertelli, A. M. and John, P. (2013). Public policy investment: Risk and return in british politics. *British Journal of Political Science*, 43(4):741–773.

[20] Bertsekas, D. P. (1987a). *Dynamic Programming: Determinist. and Stochast. Models*. Prentice-Hall.

[21] Bertsekas, D. P. (1987b). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[22] Best, M. J. and Hlouskova, J. (2000). The efficient frontier for bounded assets. *Mathematical methods of operations research*, 52(2):195–212.

[23] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 edition.

[24] Boasson, V., Boasson, E., and Zhou, Z. (2011). Portfolio optimization in a mean-semivariance framework. *Investment Management and Financial Innovations*, 8:58–68.

[25] Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327.

[26] Bollerslev, T., Li, S. Z., and Zhao, B. (2020). Good volatility, bad volatility, and the cross section of stock returns. *Journal of Financial and Quantitative Analysis*, 55(3):751–781.

[27] Boyan, J. and Moore, A. (1994). Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, 7.

[28] Bradtke, S. and Duff, M. (1994). Reinforcement learning methods for continuous-time markov decision problems.

[29] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117. Proceedings of the Seventh International World Wide Web Conference.

[30] Cammarota, V. and Lachal, A. (2015). Entrance and sojourn times for markov chains. application to (l, r)-random walks. *arXiv preprint arXiv:1503.08632*.

[31] Campbell, A. L. (2014). *Trapped in America's Safety Net: One Family's Struggle*. University of Chicago Press.

[32] Camus, B., Bourjot, C., and Chevrier, V. (2015). Combining devs with multi-agent concepts to design and simulate multi-models of complex systems (wip). In *Proceedings of the Symposium on Theory of Modeling Simulation: DEVS Integrative MS Symposium*, DEVS '15, page 85–90, San Diego, CA, USA. Society for Computer Simulation International.

[33] Capocchi, L. (2012). DEVSimPy software.

[34] Capocchi, L., Santucci, J. F., Poggi, B., and Nicolai, C. (2011). DEVSimPy: A collaborative python software for modeling and simulation of DEVS systems. In *Proc. of 20th IEEE International Workshops on Enabling Technologies*, pages 170–175.

[35] Cárdenas, R., Henares, K., Ruiz-Martín, C., and Wainer, G. (2020). Cell-devs models for the spread of covid-19. In *International Conference on Cellular Automata for Research and Industry*, pages 239–249. Springer.

[36] Carneiro, T., Da Nóbrega, R. V. M., Nepomuceno, T., Bian, G.-B., De Albuquerque, V. H. C., and Reboucas Filho, P. P. (2018). Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685.

[37] Cheuathonghua, M., Padungsaksawasdi, C., Boonchoo, P., and Tongurai, J. (2019). Extreme spillovers of vix fear index to international equity markets. *Financial Markets and Portfolio Management*, 33(1):1–38.

[38] Ching, W.-K. and Ng, M. K. (2006). Markov chains. *Models, algorithms and applications*.

[39] Clark, B., Feinstein, Z., and Simaan, M. (2020). A machine learning efficient frontier. *Operations Research Letters*, 48(5):630–634.

[40] Commission, T. E. (2018a). The european commission: Communication from the commission to the european parliament, the european council, the council, the european economic and social committee and the committee of the regions. the european commission (2018). *The European Commission*, https://ec.europa.eu/digital-single-market/en/news/communication-artificial-intelligence-europe.

[41] Commission, T. E. (2018b). The european commission: Independent high-level expert group on artificial intelligence set up by the european commission. the european commission (2018). *The European Commission*, https://ec.europa.eu/digital-single-market/en/news/communication-artificial-intelligence-europe).

[42] Daavarani Asl, Z., Derhami, V., and Yazdian-Dehkordi, M. (2017). A new approach on multi-agent multi-objective reinforcement learning based on agents' preferences. In *2017 Artificial Intelligence and Signal Processing Conference (AISP)*, pages 75–79.

[43] Dall'Erba, S. (2005). Distribution of regional income and regional funds in europe 1989–1999: an exploratory spatial data analysis. *The Annals of Regional Science*, 39(1):121–148.

[44] Dann, C., Mansour, Y., Mohri, M., Sekhari, A., and Sridharan, K. (2022). Guarantees for epsilon-greedy reinforcement learning with function approximation. In *International Conference on Machine Learning*, pages 4666–4689. PMLR.

[45] David, I. and Syriani, E. (2022). Devs model construction as a reinforcement learning problem. In *2022 Annual Modeling and Simulation Conference (ANNSIM)*, pages 30–41.

[46] Davidow, A. and Peterson, J. (2014). A modern approach to asset allocation and portfolio construction. *Schwab Center for Financial Research.*

[47] Dewanto, V., Dunn, G., Eshragh, A., Gallagher, M., and Roosta, F. (2020). Average-reward model-free reinforcement learning: a systematic review and literature mapping. *arXiv preprint arXiv:2010.08920.*

[48] Dhaene, J., Dony, J., Forys, M., Linders, D., and Schoutens, W. (2012). Fix: the fear index—measuring market fear. In *Topics in Numerical Methods for Finance*, pages 37–55. Springer.

[49] Doran, J. S., Peterson, D. R., and Tarrant, B. C. (2007). Is there information in the volatility skew? *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 27(10):921–959.

[50] Douc, R., Moulines, E., Priouret, P., and Soulier, P. (2018). *Markov chains.* Springer.

[51] Du, M., Liu, N., and Hu, X. (2019). Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77.

[52] Dunn, J., Fitzgibbons, S., and Pomorski, L. (2018). Assessing risk through environmental, social and governance exposures. *Journal of Investment Management*, 16(1):4–17.

[53] Ederington, L. H. and Guan, W. (2006). Measuring historical volatility. *Journal of Applied Finance*, 16(1).

[54] Estrada, J. (2008). Mean-semivariance optimization: A heuristic approach. *Journal of Applied Finance (Formerly Financial Practice and Education)*, 18(1).

[55] European Commission, P. (2016). European commission, parliament: Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). oj l 119, 1–88 (2016). *European Commission.*

[56] Even-Dar, E. and Mansour, Y. (2004). Learning rates for q-learning. *J. Mach. Learn. Res.*, 5:1–25.

[57] Fassas, A. P. and Hourvouliades, N. (2019). Vix futures as a market timing indicator. *Journal of Risk and Financial Management*, 12(3):113.

[58] Feldman, R. M. and Valdez-Flores, C. (2010). *Applied probability and stochastic processes.* Springer.

[59] Fernholz, R. and Shay, B. (1982). Stochastic portfolio theory and stock market equilibrium. *The Journal of Finance*, 37(2):615–624.

[60] Floyd, M. W. and Wainer, G. A. (2010). Creation of devs models using imitation learning. In *Proceedings of the 2010 Summer Computer Simulation Conference*, SCSC '10, pages 334–341, San Diego, CA, USA. Society for Computer Simulation International.

[61] Foo, N. Y. and Peppas, P. (2004). Systems theory: Melding the AI and simulation perspectives. In *Artificial Intelligence and Simulation, 13th International Conference on AI, Simulation, and Planning in High Autonomy Systems, AIS 2004, Jeju Island, Korea, October 4-6, 2004, Revised Selected Papers*, pages 14–23.

[62] Framework, I. (2009). Revisions to the basel ii market risk framework. *Julio, www. bis. org*.

[63] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). *An Introduction to Deep Reinforcement Learning*. now.

[64] Fu, M. C. (2014). *Handbook of Simulation Optimization*. Springer Publishing Company, Incorporated.

[65] Fudenberg, D. and Levine, D. K. (2007). An economist's perspective on multi-agent learning. *Artificial Intelligence*, 171(7):378 – 381. Foundations of Multi-Agent Learning.

[66] Gârleanu, N. and Pedersen, L. H. (2013). Dynamic trading with predictable returns and transaction costs. *The Journal of Finance*, 68(6):2309–2340.

[67] Gębka, B. (2012). The dynamic relation between returns, trading volume, and volatility: Lessons from spillovers between asia and the united states. *Bulletin of Economic Research*, 64(1):65–90.

[68] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

[69] Gillemot, L., Farmer, J. D., and Lillo, F. (2006). There's more to volatility than volume. *Quantitative finance*, 6(5):371–384.

[70] Goll, T. and Rüschendorf, L. (2001). Minimax and minimal distance martingale measures and their relationship to portfolio optimization. *Finance and Stochastics*, 5(4):557–581.

[71] Gonzalez, L., Powell, J. G., Shi, J., and Wilson, A. (2005). Two centuries of bull and bear market cycles. *International Review of Economics & Finance*, 14(4):469–486.

[72] Goodman, B. and Flaxman, S. (2017). European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57.

[73] Gordon, G. J. (1995). Stable function approximation in dynamic programming. In Prieditis, A. and Russell, S. J., editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 261–268, San Francisco, CA, USA. Morgan Kauffman.

[74] Goto, S. (2015). Academic theories meet the practice of active portfolio management.

[75] Grossmann, V., Steger, T. M., and Trimborn, T. (2013). The macroeconomics of tanstaafl. *Journal of Macroeconomics*, 38:76–85.

[76] Hamid, K. and Hasan, A. (2016). Volatility modeling and asset pricing: Extension of garch model with macro economic variables, value-at-risk and semi-variance for kse. *Pakistan Journal of Commerce and Social Sciences*, 10(3):569–587.

[77] Hammer, P. C. (1958). Dynamic programming. *Science*, 127(3304):976–976.

[78] Hansun, S. and Young, J. C. (2021). Predicting lq45 financial sector indices using rnn-lstm. *Journal of Big Data*, 8(1):1–13.

[79] Haugen, R. A. and Baker, N. L. (1990). Dedicated stock portfolios. *The Journal of Portfolio Management*, 16(4):17–22.

[80] Haugen, R. A. and Baker, N. L. (1991). The efficient market inefficiency of capitalization–weighted stock portfolios. *The journal of portfolio management*, 17(3):35–40.

[81] Hespanha, J. P., Kim, H. J., and Sastry, S. (1999). Multiple-agent probabilistic pursuit-evasion games. In *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, volume 3, pages 2432–2437 vol.3.

[82] Hester, T. and Stone, P. (2009). Generalized model learning for reinforcement learning in factored domains. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, page 717–724, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[83] Hnatkovska, V. (2004). *Volatility and growth*, volume 3184. World Bank Publications.

[84] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[85] Hölzl, J. (2017). Markov chains and markov decision processes in isabelle/hol. *Journal of Automated Reasoning*, 59(3):345–387.

[86] Huang, Q. (2020). Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221. IEEE.

[87] Israelsen, B. W. and Ahmed, N. R. (2019). "dave... i can assure you... that it's going to be all right..." a definition, case for, and survey of algorithmic assurances in human-autonomy trust relationships. *ACM Computing Surveys (CSUR)*, 51(6):1–37.

[88] Ivanova, M. and Dospatliev, L. (2017). Application of markowitz portfolio optimization on bulgarian stock market from 2013 to 2016. *International Journal of Pure and Applied Mathematics*, 117(2):291–307.

[89] Iyiola, O., Munirat, Y., and Nwufo, C. (2012). The modern portfolio theory as an investment decision tool. *Journal of Accounting and Taxation*, 4(2):19–28.

[90] Jacobs, B. I. and Levy, K. N. (2012). Leverage aversion and portfolio optimality. *Financial Analysts Journal*, 68(5):89–94.

[91] Jacobs, B. I. and Levy, K. N. (2013). Leverage aversion, efficient frontiers, and the efficient region. *The Journal of Portfolio Management*, 39(3):54–64.

[92] Jang, B., Kim, M., Harerimana, G., and Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667.

[93] Jones, E., Oliphant, T., and Peterson, P. (2001). Scipy: Open source scientific tools for python. http://www.scipy.org/.

[94] Kaczmarek, T. and Perez, K. (2022). Building portfolios based on machine learning predictions. *Economic Research-Ekonomska Istraživanja*, 35(1):19–37.

[95] Kakade, K., Mishra, A. K., Ghate, K., and Gupta, S. (2022). Forecasting commodity market returns volatility: A hybrid ensemble learning garch-lstm based approach. *Intelligent Systems in Accounting, Finance and Management*, 29(2):103–117.

[96] Kanade, V., McMahan, H. B., and Bryan, B. (2009). Sleeping experts and bandits with stochastic action availability and adversarial rewards. In *Artificial Intelligence and Statistics*, pages 272–279. PMLR.

[97] Kashyap, R. (2016). The circle of investment: Connecting the dots of the portfolio management cycle... *arXiv preprint arXiv:1603.06047*.

[98] Kemeny, J. G. and Snell, J. L. (1976). *Finite Markov Chains*. Springer.

[99] Kessler, C., Capocchi, L., Santucci, J.-F., and Zeigler, B. (2017). Hierarchical markov decision process based on devs formalism. In *2017 Winter Simulation Conference (WSC)*, pages 1001–1012. IEEE.

[100] Khurana, U., Samulowitz, H., and Turaga, D. (2018). Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[101] Kim, H. Y. and Won, C. H. (2018). Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models. *Expert Systems with Applications*, 103:25–37.

[102] Kinney, D. and Bright, L. K. (2021). Risk aversion and elite-group ignorance. *Philosophy and Phenomenological Research*.

[103] Koenig, S. and Simmons, R. G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Mach. Learn.*, 22(1-3):227–250.

[104] Kofman, E. (2000). Discrete event based simulation and control of continuous systems. Master's thesis.

[105] Kondrin, D. and Novoselova, V. (2013). What is your trading style? . . , , (12):147–148.

[106] Koo, E. and Kim, G. (2022). A hybrid prediction model integrating garch models with a distribution manipulation strategy based on lstm networks for stock market volatility. *IEEE Access*, 10:34743–34754.

[107] Koren, M. and Tenreyro, S. (2007). Volatility and development. *The Quarterly Journal of Economics*, 122(1):243–287.

[108] Krokhmal, P., Palmquist, J., and Uryasev, S. (2002). Portfolio optimization with conditional value-at-risk objective and constraints. *Journal of risk*, 4:43–68.

[109] Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191.

[110] Lahmiri, S. and Bekiros, S. (2020). Renyi entropy and mutual information measurement of market expectations and investor fear during the covid-19 pandemic. *Chaos, Solitons & Fractals*, 139:110084.

[111] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2016). Building machines that learn and think like people. *CoRR*, abs/1604.00289.

[112] Lee, J. H. (2019). Complementary reinforcement learning towards explainable agents. *arXiv preprint arXiv:1901.00188.*

[113] Lee, J. W., Park, J., O, J., Lee, J., and Hong, E. (2007). A multiagent approach to q-learning for daily stock trading. *Trans. Sys. Man Cyber. Part A*, 37(6):864–877.

[114] Li, X., Vangheluwe, H., Lei, Y., Song, H., and Wang, W. (2011). A testing framework for devs formalism implementations. In *Proceedings on the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 183–188, San Diego, CA, USA. Society for Computer Simulation International.

[115] Liu, S., Liao, G., and Ding, Y. (2018). Stock transaction prediction modeling and analysis based on lstm. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 2787–2790. IEEE.

[116] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

[117] Lüthi, S. (2010). Effective deployment of photovoltaics in the mediterranean countries: Balancing policy risk and return. *Solar Energy*, 84(6):1059–1071.

[118] Mahadevan, S., Marchalleck, N., Das, T., and Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning.

[119] Maheu, J. M., McCurdy, T. H., and Song, Y. (2021). Bull and bear markets during the covid-19 pandemic. *Finance research letters*, 42:102091.

[120] Mannor, S. and Tsitsiklis, J. (2011). Mean-variance optimization in markov decision processes. *arXiv preprint arXiv:1104.5601.*

[121] Markowitz, H. M. (1999). The early history of portfolio theory: 1600–1960. *Financial analysts journal*, 55(4):5–16.

[122] Markowitz, H. M. (2010). Portfolio theory: as i still see it. *Annu. Rev. Financ. Econ.*, 2(1):1–23.

[123] Markowitz, H. M., Starer, D., Fram, H., and Gerber, S. (2020). Avoiding the downside: A practical review of the critical line algorithm for mean–semivariance portfolio optimization. *Handbook of Applied Investment Research*, pages 369–415.

[124] Martin, I. W. and Wagner, C. (2019). What is the expected return on a stock? *The Journal of Finance*, 74(4):1887–1929.

[125] Martin, R. A. (2021a). Pyportfolioopt: portfolio optimization in python. *Journal of Open Source Software*, 6(61):3066.

[126] Martin, R. A. (2021b). Pyportfolioopt: portfolio optimization in python. *Journal of Open Source Software*, 6(61):3066.

[127] McNamara, G. and Bromiley, P. (1999). Risk and return in organizational decision making. *Academy of Management Journal*, 42(3):330–339.

[128] Men, U. (2021). Design and backtesting of a trading algortihm with scalping day trading strategy for xau/usd fx market for individual traders. Master's thesis, Başkent Üniversitesi Sosyal Bilimler Enstitüsü.

[129] Meraji, S. and Tropper, C. (2010). A machine learning approach for optimizing parallel logic simulation. In *2010 39th International Conference on Parallel Processing*, pages 545–554.

[130] Michaud, R. O. and Michaud, R. O. (2008). *Efficient asset management: a practical guide to stock portfolio optimization and asset allocation*. Oxford University Press.

[131] Moldovan, T. and Abbeel, P. (2012). Risk aversion in markov decision processes via near optimal chernoff bounds. *Advances in neural information processing systems*, 25.

[132] Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

[133] Mueller, E. R. and Kochenderfer, M. (2016). Multi-rotor aircraft collision avoidance using partially observable markov decision processes. In *AIAA Modeling and Simulation Technologies Conference*, page 3673.

[134] Nelson, D. M., Pereira, A. C., and De Oliveira, R. A. (2017). Stock market's price movement prediction with lstm neural networks. In *2017 International joint conference on neural networks (IJCNN)*, pages 1419–1426. Ieee.

[135] Neto, M. P. and Paulovich, F. V. (2020). Explainable matrix-visualization for global and local interpretability of random forest classification ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1427–1437.

[136] Neuneier, R. (1997). Enhancing q-learning for optimal asset allocation. *Advances in neural information processing systems*, 10.

[137] Nielsen, N. R. (1991). *Application of Artificial Intelligence Techniques to Simulation*, pages 1–19. Springer New York, New York, NY.

[138] Odell, J. J., Parunak, H., Fleischer, M., and Brueckner, S. (2002). Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer.

[139] Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science and Engineering*, 9:10–20.

[140] Palmerino, J., Yu, Q., Desell, T., and Krutz, D. (2019). Improving the decision-making process of self-adaptive systems by accounting for tactic volatility. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 949–961. IEEE.

[141] Pan, J., Wang, X., Cheng, Y., and Yu, Q. (2018). Multisource transfer double dqn based on actor learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2227–2238.

[142] Pan, Y., Jiang, H., Yang, H., and Zhang, J. (2019). A novel method for improving the training efficiency of deep multi-agent reinforcement learning. *IEEE Access*, 7:137992–137999.

[143] Pan, Y., Wang, T. Y., and Weisbach, M. S. (2015). Learning about ceo ability and stock return volatility. *The review of financial studies*, 28(6):1623–1666.

[144] Pardo, F., Tavakoli, A., Levdik, V., and Kormushev, P. (2018). Time limits in reinforcement learning.

[145] Peng, S. (2007). Law of large numbers and central limit theorem under nonlinear expectations. *arXiv preprint math/0702358*.

[146] Perez, F., Granger, B. E., and Hunter, J. D. (2011). Python: An ecosystem for scientific computing. *Computing in Science and Engineering*, 13(2):13–21.

[147] Preis, T., Moat, H. S., and Stanley, H. E. (2013). Quantifying trading behavior in financial markets using google trends. *Scientific reports*, 3(1):1–6.

[148] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[149] Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[150] Qadan, M. and Cohen, G. (2011). Is it profitable to invest according to the vix fear index. *Journal of Modern Accounting and Auditing*, 7(1):86–90.

[151] Rachelson, E., Quesnel, G., Garcia, F., and Fabiani, P. (2008a). A simulation-based approach for solving generalized semi-markov decision processes. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 583–587, Amsterdam, The Netherlands, The Netherlands. IOS Press.

[152] Rachelson, E., Quesnel, G., Garcia, F., and Fabiani, P. (2008b). A simulation-based approach for solving generalized semi-markov decision processes. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 583–587, Amsterdam, The Netherlands, The Netherlands. IOS Press.

[153] Rappin, N. and Dunn, R. (2006). *WxPython in action*. Manning.

[154] Read, C. (2012). The theory of an efficient portfolio. In *The Portfolio Theorists*, pages 184–192. Springer.

[155] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

[156] Roondiwala, M., Patel, H., and Varma, S. (2017). Predicting stock prices using lstm. *International Journal of Science and Research (IJSR)*, 6(4):1754–1756.

[157] Ross, S. A. (2013). The arbitrage theory of capital asset pricing. In *Handbook of the fundamentals of financial decision making: Part I*, pages 11–30. World Scientific.

[158] Rubino, G. and Sericola, B. (1989). Sojourn times in finite markov processes. *Journal of Applied Probability*, 26(4):744–756.

[159] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[160] Saadawi, H., Wainer, G., and Pliego, G. (2016). Devs execution acceleration with machine learning. In *2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS)*, pages 1–6.

[161] Sato, Y. (2019). Model-free reinforcement learning for financial portfolios: a brief survey. *arXiv preprint arXiv:1904.04973*.

[162] Schmidt, D. (2003). Private equity-, stock-and mixed asset-portfolios: A bootstrap approach to determine performance characteristics, diversification benefits and optimal portfolio allocations. Technical report, CFS Working Paper.

[163] Sen, J., Dutta, A., and Mehtab, S. (2021). Stock portfolio optimization using a deep learning lstm model. In *2021 IEEE Mysore Sub Section International Conference (MysuruCon)*, pages 263–271. IEEE.

[164] Seo, C., Zeigler, B. P., and Kim, D. (2018). Devs markov modeling and simulation: Formal definition and implementation. In *Proceedings of the Theory of Modeling and Simulation Symposium*, TMS '18, pages 1:1–1:12, San Diego, CA, USA. Society for Computer Simulation International.

[165] Shahzad, S. J. H., Bouri, E., Roubaud, D., and Kristoufek, L. (2020). Safe haven, hedge and diversification for g7 stock markets: Gold versus bitcoin. *Economic Modelling*, 87:212–224.

[166] Shahzad, S. J. H., Raza, N., Shahbaz, M., and Ali, A. (2017). Dependence of stock markets with gold and bonds under bullish and bearish market states. *Resources Policy*, 52:308–319.

[167] Sharpe, W. F., Chen, P., Pinto, J. E., and McLeavey, D. W. (1990). Asset allocation. *Managing investment portfolios: A dynamic process*, 2.

[168] Sherlaw-Johnson, C., Gallivan, S., and Burridge, J. (1995). Estimating a markov transition matrix from observational data. *Journal of the Operational Research Society*, 46(3):405–410.

[169] Shi, C., Wan, R., Song, R., Lu, W., and Leng, L. (2020). Does the markov decision process fit the data: Testing for the markov property in sequential decision making. In *International Conference on Machine Learning*, pages 8807–8817. PMLR.

[170] Shiller, R. J. (1992). *Market volatility*. MIT press.

[171] Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent reinforcement learning: a critical survey. Technical report.

[172] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org.

[173] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[174] Sinai, Y. (2003). Andrei andreyevich markov. In *Russian Mathematicians In The 20th Century*, pages 599–621. World Scientific.

[175] Sudha, S. (2015). Risk-return and volatility analysis of sustainability index in india. *Environment, development and sustainability*, 17(6):1329–1342.

[176] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

[177] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[178] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

[179] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

[180] Szegö, G. (2005). Measures of risk. *European Journal of Operational Research*, 163(1):5–19.

[181] Thompson, S. R. and Waller, M. L. (1987). The execution cost of trading in commodity futures markets. *Food Research Institute Studies*, 20(1387-2016-116196):141–163.

[182] Tierney, L. (1996). Introduction to general state-space markov chain theory. *Markov chain Monte Carlo in practice*, pages 59–74.

[183] Toma, S. (2014). *Detection and identication methodology for multiple faults in complex systems using discrete-events and neural networks: applied to the wind turbines diagnosis*. Theses, University of Corsica.

[184] Tsang, E. P., Tao, R., Serguieva, A., and Ma, S. (2017). Profiling high-frequency equity price movements in directional changes. *Quantitative finance*, 17(2):217–225.

[185] Van Seijen, H., Van Hasselt, H., Whiteson, S., and Wiering, M. (2009). A theoretical and empirical analysis of expected sarsa. In *2009 ieee symposium on adaptive dynamic programming and reinforcement learning*, pages 177–184. IEEE.

[186] von Bertalanffy, L. (1968). *General system theory*. George Braziller, New York.

[187] Wang, D., Ding, B., and Feng, D. (2020a). Meta reinforcement learning with generative adversarial reward from expert knowledge. In *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pages 1–7. IEEE.

[188] Wang, R., Zhong, P., Du, S. S., Salakhutdinov, R. R., and Yang, L. (2020b). Planning with general objective functions: Going beyond total rewards. *Advances in Neural Information Processing Systems*, 33:14486–14497.

[189] Wang, Y., Liu, P., Zhu, K., Liu, L., Zhang, Y., and Xu, G. (2022). A garlic-price-prediction approach based on combined lstm and garch-family model. *Applied Sciences*, 12(22):11366.

[190] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.

[191] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

[192] Węgrzyn, J. and Wojewnik-Filipkowska, A. (2022). Stakeholder analysis and their attitude towards ppp success. *Sustainability*, 14(3):1570.

[193] Whitelaw, R. F. (2000). Stock market risk and return: An equilibrium approach. *The Review of Financial Studies*, 13(3):521–547.

[194] Wu, Y., Zhang, J., Ravey, A., Chrenko, D., and Miraoui, A. (2020). Real-time energy management of photovoltaic-assisted electric vehicle charging station by markov decision process. *Journal of Power Sources*, 476:228504.

[195] Yu, H., Mahmood, A. R., and Sutton, R. S. (2017). On generalized bellman equations and temporal-difference learning. In *Advances in Artificial Intelligence - 30th Canadian Conference on Artificial Intelligence, Canadian AI 2017, Edmonton, AB, Canada, May 16-19, 2017, Proceedings*, pages 3–14.

[196] Yu, H., Mahmood, A. R., and Sutton, R. S. (2018). On generalized bellman equations and temporal-difference learning. *The Journal of Machine Learning Research*, 19(1):1864–1912.

[197] Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270.

[198] Yudin, A. (2021). Essential financial tasks done with python. In *Basic Python for Data Management, Finance, and Marketing*, pages 231–275. Springer.

[199] Zeigler, B. (2021a). Devs-based building blocks and architectural patterns for intelligent hybrid cyberphysical system design. *Information*, 12(12).

[200] Zeigler, B. (2021b). Devs-based building blocks and architectural patterns for intelligent hybrid cyberphysical system design. *Information*, 12(12).

[201] Zeigler, B. P. (1976). *Theory of Modeling and Simulation*. Academic Press.

[202] Zeigler, B. P., Muzy, A., and Kofman, E. (2019). *Theory of Modeling and Simulation (Third Edition)*. Academic Press, third edition edition.

[203] Zeigler, B. P. and Sarjoughian, H. S. (2013). System entity structure basics. In *Guide to Modeling and Simulation of Systems of Systems*, Simulation Foundations, Methods and Applications, pages 27–37. Springer London.

[204] Zhang, J., Hao, J., Fogelman-Soulié, F., and Wang, Z. (2019). Automatic feature engineering by deep reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2312–2314.

[205] Zhang, T., Xie, S., and Rose, O. (2017). Real-time job shop scheduling based on simulation and markov decision processes. In *2017 Winter Simulation Conference (WSC)*, pages 3899–3907. IEEE.

[206] Zheng, J. and Siami Namin, A. (2018). A markov decision process to determine optimal policies in moving target. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 2321–2323.

[207] Zhou, X. Y. and Li, D. (2000). Continuous-time mean-variance portfolio selection: A stochastic lq framework. *Applied Mathematics and Optimization*, 42(1):19–33.

[208] Zolfaghari, M. and Gholami, S. (2021). A hybrid approach of adaptive wavelet transform, long short-term memory and arima-garch family models for the stock index prediction. *Expert Systems with Applications*, 182:115149.

Chapter 7

Agent Atomic DEVS Model Specification

An agent entity is modeled as a DEVS atomic model with two input ports and on the output port as (Figure 7.1):

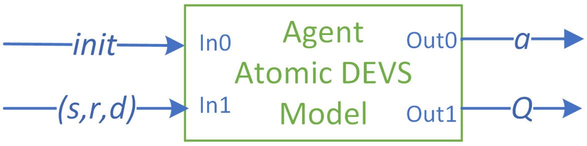


Fig. 7.1 The DEVS agent atomic model with its two input ports In_0 and In_1 used to receive the initialization message (for the Q matrix) and the set (s,r,d) from the DEVS environment model. The two output ports Out_0 (resp. Out_1) are used to send the action a (resp. Q matrix) to be evaluated by the environment model.

$$Agent =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

where:

- $X : \{(p,v)|p \in [In_0, In_1], v \in V\}$ is the set of input ports with: (i) In_0 to receive the new state, reward and done flag from the Environment model, and (ii) In_1 to receive the state-action dict object from the Environment model. The set V includes all of this input with different types such as string, dict object, and float.
- $Y : \{(p,v)|p \in [Out_0, Out_1], v \in V\}$ is the set of the two output ports p that allow us to transmit the action to the Environment model (Out_0), and to trace the mean of the Q matrix (Out_1).
- $sigma \in \mathbb{R}^{+}_{0,\infty}$ is the variable introduced to manage the time advance function.

- $state \in \{'IDLE','UPDATE\_STATE\_ACTION\_MAP','SEND\_ACTION', 'SEND\_QMEAN'\}$ is the state space.
- $Q = S^M xA$ is the Q matrix with a number of rows equal to the number of Markov states S^M and a number of columns equal to the number of actions A.
- $state\_action\_map$ is the Markov states - actions dictionary sent by the Environment model.
- $stop$ is a flag to inform if the
- $\varepsilon \in \mathbb{R}^+_{0,1}$ is the epsilon Q-Learning parameter.
- $\gamma \in \mathbb{R}^+_{0,1}$ is the gamma Q-Learning parameter.
- $\alpha \in \mathbb{R}^+_{0,1}$ is the alpha Q-Learning parameter.
- s is the Markov state received from the Environment model.
- r is the reward received from the Environment model.
- d is the flag received from the Environment model.
- a is the current action chosen by the Agent model.
- $S : state \times sigma$ is the set of sequential states.
- $\delta_{int}(S \to S)$:

```
1. if $state$ is 'SEND_ACTION' then
2.     $stop \leftarrow new\_QMEAN$ is $old\_QMEAN$ and $\varepsilon$ is 1.0.
3. $state \leftarrow ('IDLE', \infty)$
```

- $\delta_{ext}(Q \times X \to S)$:

```
1. (p,v) $\leftarrow$ peek(X)
2. if p is $Out\_1$ then
3.     $state\_action\_map \leftarrow$ v
4.     $Q \leftarrow \emptyset$
5.     $stop \leftarrow$ False
6.     $new\_QMEAN \leftarrow old\_QMEAN \leftarrow$ 0.0
7.     $state \leftarrow ('UPDATE\_STATE\_ACTION\_MAP', 0.00001)$
8. else
9.     $s, r, d \leftarrow$ v
```

```
        call $updateQ(s,a,r)$
        if $d$ is True then
            $state \leftarrow ('SEND\_QMEAN', 0.00001)$
        else
            $state \leftarrow ('SEND\_ACTION', 0.00001)$
```

- $\lambda(S \rightarrow Y)$:

```
  if $state$ in $['SEND\_ACTION','UPDATE\_STATE\_ACTION\_MAP']$ then
      $a \leftarrow ChooseAction(s,\varepsilon)$
      call $send(Out_0, a)$
  if $state$ is $'SEND\_QMEAN'$ then
      call $send(Out_1, Mean(Q))$
```

- $t_a(S \rightarrow \mathbb{R}^{+}_{0,\infty}) \leftarrow sigma$

The functions $UpdateQ$ and $ChooseAction$ are specified as follows.

The $UpdateQ$ function depends on the choice of the learning algorithm : Q-Learning or SARSA.

```
function $UpdateQ(s0,s1,a0,r)$
    if Q-Learning then
        ga ← max(Q[s1,:]) // greedy action and temporal difference
        td ← r + $\gamma$*ga - Q[s0,a0]
        Q[s0,a0] +← $\alpha$*td
    else
        $a1 \leftarrow ChooseAction(s1,\varepsilon)$
        $ga \leftarrow Q[s1,a1]$
        td ← r + $\gamma$*ga - Q[s0,a0]
        Q[s0,a0] +← $\alpha$*td
        $a \leftarrow a1$ // update action
    // Q mean update
    $old\_QMEAN \leftarrow new\_QMEAN$
    $new\_QMEAN \leftarrow Q.mean()$

function $ChooseAction(s, \varepsilon)$
    $values \leftarrow Q[s, :]$ // values of Q for s
    $e \leftarrow random()$ // e is a random value
    // select the action with highest cummulative reward
    if $e < \varepsilon$ or $Sum(values)$ is 0 then
        $a \leftarrow randint(0, len(state\_action\_map[s]))$
        return state_action_map[s][a]['action']
    else
        $a \leftarrow argmax(v)$
        loop for d $\in$ state_action_map[s]
            if d['action'] is $a$ then
                return d['action']
```

Chapter 8

Environment Atomic DEVS Model Specification

An environment entity is modeled as a DEVS atomic model with N input ports and two output ports as (Figure 8.1):

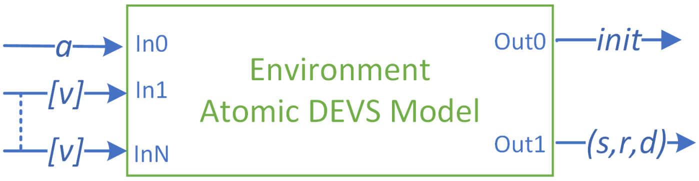


Fig. 8.1 The DEVS environment atomic model with its two input ports In_0 and In_1 used to receive the action a from the Agent DEVS model and the message $[v]$ from the generators. The two output ports Out_0 (resp. Out_1) are used to send the initial message (resp. (s,r,d)) to the Agent model.

$$Environment = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a >$$

where:

- $X : \{(p,v) | p \in [In_0, \ldots, In_N], v \in V\}$ is the set of the N input ports of the Agent model. The first port In_0 is used to receive the action a that should be evaluated. The last input ports are used to receive event from generators to eventually be considered in the environment.

- $Y : \{(p,v)|p \in [Out_0, Out_1], v \in V\}$ is the set of the two output ports Out_0 (resp. Out_1) used to send the initialization message (resp. (s,r,d)) to the agent model.
- $sigma \in \mathbb{R}^{+}_{0,\infty}$ is the variable introduced to manage the time advance function
- $state \in \{'IDLE','UPDATE\_ACTION','UPDATE\_EPISODE', 'UPDATE\_STATE\_ACTION\_MAP','UPDATE\_ALL'\}$ is the state space
- *goal_reward* is a variable used to specify if the search algorithm is oriented to the goal reward (see. Section 2.3.2.3)
- $state\_action\_map$ is the Markov states - actions dictionary.
- $init\_state$ is the initial state of the Environment.
- end_state is the end state of the Environment (if the goal reward option is enabled).
- s is the current state of the Environment.
- a is the action received from the Agent model.
- $inputs1N$ is the list of input messages detected in the input ports 1 to N (coming from generator models).
- $S : state \times sigma \times s$ is the set of sequential states
- $\delta_{int}(S \rightarrow S)$:

```
if done is True then
    $s \leftarrow init\_state$
    $init\_state \leftarrow end\_state$
    $done \leftarrow$ False
    state $\leftarrow$ ($'UPDATE\_EPISODE'$, 1.0, $s$)
else
    state $\leftarrow$ ($'UPDATE\_ACTION'$, $\infty$, $s$)
```

- $\delta_{ext}(Q \times X \rightarrow S)$:

```
    (p,v) $\leftarrow$ peek(X)
    if p is $Out\_0$ then
        $a \leftarrow v$
        state $\leftarrow$ ($'UPDATE\_ACTION'$, 0.00001, $s$)
    else
```

```
        $inputs1N \leftarrow v$
        if state is not $'UPDATE\_ACTION'$ then
            state $\leftarrow$ ($'UPDATE\_STATE\_ACTION\_MAP'$, $\infty$, $s$)
        else
            state $\leftarrow$ ($'UPDATE\_ALL'$, $\infty$, $s$)
    if all 1-N messages have been received
        if state is in $('UPDATE\_ALL','UPDATE\_STATE\_ACTION\_MAP')$ then
            $init\_state \leftarrow GetInitState(inputs1N)$
            $end\_state \leftarrow GetEndState(inputs1N)$
            $state\_action\_map \leftarrow GetStateActionMap(inputs1N)$
            state $\leftarrow$ (state, 0, $s$)
```

- $\lambda(S \to Y):$

```
If the state is in $('UPDATE\_ALL','UPDATE\_STATE\_ACTION\_MAP')$ then
    call $send(Out_1, state\_action\_map)$
else if state is in $('UPDATE_ALL','UPDATE_ACTION')$ then
    $s', r, d \leftarrow Step(s, a)$
    call $send(Out_0, (s', r, d))$
else if state is $'UPDATE_EPISODE')$ then

    call $send(Out_0, (s, 0.0, False))$
```

- $t_a(S \to \mathbb{R}^+_{0,\infty}) \leftarrow sigma$

The functions *GetInitState(inputs1N)*, *GetEndState(inputs1N)*, and *GetStateActionMap(inputs1N)* are used to define the initial state, the end state, and the state action map of the Environment model from the $inputs1N$ list of messages coming from generator models. The *GetStateActionMap* method calls the $GetReward(end\_state)$ method which returns the reward according to the chosen strategy (goal or penalty) that depends on the $goal\_reward$ Boolean variable.

```
function $GetReward(s)$
    if $goal\_reward$ is True Then
```

```
        return 0.0 // 1.0 is defined only for the end states
    else
        pass // define the reward algorithm
```

An example of the $GetStateActionMap(inputs1N)$ method can be presented as follows.

```
function $GetStateActionMap(inputs1N)$
$A \leftarrow dictionary$
    loop for $state$,$action$ in $state\_action\_map$
        if $state$ is $end\_state$ then
            $end \leftarrow end\_state$
        else
            $end \leftarrow None$
        $d \leftarrow$ {'end':$end$, 'action':$action$, 'reward':1.0 if $end$ is $end\_state$ else $getReward(end)$}
        if $end$ is $end\_state$ then
            $d[end\_state] \leftarrow True$
        if $end$ is $init\_state$ then
            $d[init\_state] \leftarrow True$
        if $state$ is not in $A$ then
            $A[state] \leftarrow [d]$
        else
            append $d$ to $A[state]$
```

The $Step(s,a)$ function is used to continue the learning process. Returns the set (s',r,d) depending on the current state s and the action a and can be implemented as follows.

```
function $Step(s,a)$
    loop for $t$ in $state\_action\_map[s]$
        If $a$ is $t['action']$ then
            $new\_state \leftarrow t['end']$
            $reward \leftarrow t['reward']$
            $done \leftarrow reward$ is 1.0
        break
    return $(new\_state, reward, done)$
```

Chapter 9

Efficient Frontier Semi Variance Implementation

Our ESV model has been coded in the Google Collaboration Python Environment [36]. The four-step methodology that has been followed in our ESV is introduced in the following:

1. **Choosing the sectors**: The four main indexes of the US market are chosen first. The chosen sectors are as follows;

   (a) Nasdaq Composite (noted IXIC)

   (b) Dow Jones Industrial Average (noted DJI)

   (c) S&P 500 (noted GSPC)

   (d) Russell 2000 (noted RUT)

2. **Data acquisition**: For each index, the historical prices of the four most critical stocks are extracted using the *DataReader* function of the *data* submodule of the *pandas_datareader* Python module. Stock prices are extracted from the Yahoo Finance site from September 1, 2012, and from September 1, 2014 to January 3 and September 1 of every year from 2015 up to 2022. There are five features in the stock data: open, high, low, close, volume, and adjusted_close. The current work is a univariate analysis and, hence, the variable adjusted_close is chosen as the only variable of interest.

3. **Derivation of the return and volatility**: The percentage changes in the adjusted_close values for successive days represent the daily *return* values. For computing the daily

returns, the *pct_change* function of Python is used. To calculate the portfolio variance and the standard deviation (volatility), the *np.dot* and *np.sqrt* Python functions are used, respectively. Assuming that there are 252 operational days in a calendar year, to calculate portfolio annual gains and losses for the stocks are found by multiplying the daily volatility by the *pct_change* function of Python is used.

4. **Construction of the Portfolio Optimization** (using PyPortfolioOpt library [126]):

   First, we pick the option of importing:

   - *EfficientFrontier* from PyPortfolioOpt library *pypfopt.efficient_frontier*,
   - *risk_models* from pypfopt,
   - *expected_returns, EfficientSemivariance* from pypfopt.

   Second, to calculate gains and losses, annual covariance and portfolio gains and losses, the functions *expected_returns.mean_historical_return* (μ) and *expected_returns.returns_from_prices* functions are used.

Then we calculate the best maximum Sharpe ratio based on the performance of William Sharpe on volatility [167]. Next, we compute the ESV using μ and *historical_returns*.

Finally, to build the best portfolio policy, we import *DiscreteAllocation* and *get_latest_prices* from *pypfopt.discrete_allocation* and then use the *get_latest_prices* function to get the latest prices, define the total value of the portfolio, and finally print *DiscreteAllocation* to obtain our optimized portfolio.

Chapter 10

LSTM Implementation

The LSTM model is introduced in the following two steps:

- **Data Description** The data used in this case study were collected from YFinance, which covers the index data from September 2012 to November 2022. Furthermore, LSTMs are sensitive to the scale of the input data, which is why the data was normalized to the range of 0-to-1. Data are divided into two parts: Training data sets with 85% observations are used to train our model, and the remaining 15% are used to test the accuracy of our model prediction.
- **Model Design** We create an LSTM model with two LSTM layers. The first layer has 1000 neurons and the second 500 neurons. The number of neurons was selected by a trail-and-error process. The output of the hidden state of each neuron in the first layer is used as input to our second LSTM layer. We have two dense layers, where the first layer contains 50 neurons, and the second dense layer, which also acts as the output layer, contains 1 neuron. The network is trained for 1 epoch, and a batch size of 1 is used. Adam optimizer and a mean squared error loss were used.