Empowering Self-Adaptive Systems with Global Forecasting Capabilities

Sylvain Lejamble[1,2][0000−0003−1043−4745], Ilham Alloui[2][0000−0002−3713−0592], Sébastien Monnet[2][0000−0002−6036−3060], and Flavien Vernier[2][0000−0001−7684−6502]

[1] R&D Department, SaGa Corp, Paris, France
[2] LISTIC, Université Savoie Mont Blanc, Annecy, France {sylvain.lejamble, ilham.alloui, sebastien.monnet, flavien.vernier}@univ-smb.fr

Abstract. Integrating intelligence into software that is not initially designed for it is a challenging task. This enhancement, however, can lead to significant improvements in both the software's functionality and the user experience. To address this, the Wise Object Framework (WOF) has been introduced. The framework efficiently generates agents to manage the input/output processes of actual software objects, doing so with minimal intrusion into the existing codebase. The main innovation of the WOF is that theses agents acts as components of third-party software. These agents are equipped with automatic actions and states logging, self-analysis, self-correction. In the philosophy of the WOF, agents have a selfish knowledge (they only know about themselves) that they acquire through the introspection process. Processing local knowledge involves local specialized models for each instance. It have many disadvantages such as coldstart, overfitting and huge memory consumption. In this paper we tackle this problem by creating a metalevel of knowledge. We also investigate how the system-level knowledge can enhance data processing algorithm and what kind of application it unlocks. We also show evidences and open discussions about the disadvantages of giving agents freedom to communicate such as synchronization, flows overloads, plan generation and verification.

Keywords: Self-adaptive systems · Agent systems · global forecasting · Wise Object · Event-driven architecture

1 Introduction

The increasing complexity of modern systems underscores the importance of system maintenance, a concept highlighted by Crow [9]. This leads to a higher need for trained professionals in maintenance and upgrades who are expensive but valuable experts. To optimally utilize their expertise, these professionals should focus on activities that yield greater business values. Self-adaptive systems, a branch of agent-based systems, aim to create intelligent systems that are capable of self-modification to accommodate new requirements or situations,

thereby lessening the dependency on input from developers or experts. In ever-changing environments, the ability of these systems to adapt quickly is crucial for maintaining their relevance and usability.

The Wise Object Framework (WOF) [5] provide the ability to add intelligence and adaptability to application that initially have not been designed for. WOF achieves this with minimal code intrusion, effectively transforming any software object into an agent with self-adaptation capabilities. In the present architecture of WOF, these agents accumulate knowledge about their operations and store this information within their internal knowledge base. Based on their knowledge, they are able to develop plans that can automatically adjust or correct themselves when required. Newly created agents initially face limitations in decision-making due to their insufficient knowledge base. Furthermore, the decision-making algorithms of these agents must not be overly complex to avoid the pitfalls of overfitting, which is a common issue in scenarios with limited data diversity.

A Centralized intelligence can resolve many of these issues by consolidating individual knowledge and processing it at the system level as needed. This paper outlines an architecture that allows agents to seek system-level decisions when faced with complex scenarios. It also examines the benefits and limitations of giving such communication protocol to agents.

2 Related work

2.1 Adaptive & Agent systems

An adaptive system can encompass a range of objectives. For instance, it may aim to comprehend and adapt to its environment, [7]. Alternatively, it may engage in transitioning between different models within a multi-model system to select the most suitable model for a given task, [16]. Furthermore, it have the capacity to effect changes in its components without necessitating protracted maintenance intervals to curtail human intervention, which could imply adjustments at the hardware, software, configuration settings, or even the overarching architecture level [19]. Lastly, in the event of low-grade performance, the system can engage in self-assessment and strategic reevaluation from a business perspective, as posited by [8].

ADELFE [6] stands out as the primary methodology for crafting Adaptive Multi-Agent Systems in dynamic or evolving environments. The paper demonstrates that introducing Adaptive Agents to a system can confer upon the entire system adaptability, owing to the attributes of agent autonomy, environment discovering, proactive behavior, and social interactions.

The intelligence of a self adaptive system can be led by one system agent or done in a collaboration perspective using multi-agent systems (each agent having its own expertise and only serving a small part of the system). A more general agent or a manager can then synchronize them and give them the rights they need to update the system. In [18], the authors classify the three categories of decision-making within agent-based systems ordering them from the least autonomous to

the most autonomous: The "command driven" where the decision is given from a master to a slave agent. The "consensus" where agent work as a team and share decision with equal decision rights. And the "local master": a scenario in which an agent independently makes decisions based on its individual expertise and may or may not issue directives to other agents. They also contend that agents can be intentionally designed to function at a singular level of autonomy when dealing with straightforward applications and given that the designer accurately anticipates the nature of the challenges the agent is likely to encounter. But when speaking about complex applications the optimal degree of autonomy can vary.

2.2 Local & Global forecasting

In multi-agent systems, the choice between global and local intelligence is crucial and comes with distinct advantages and disadvantages.

Local intelligence is the ability decentralizes decision-making to individual agents. Each agent operates based on its perception and understanding of the environment, which can lead to more adaptive and responsive behavior in specific, localized contexts. This approach enhances the system's robustness, as it reduces the impact of single points of failure and allows the system to continue functioning even if some agents fail or encounter issues. However, it is not feasible to simplify every situations to a model where a single agent operates with complete knowledge of the environment [11]. They often lack of a cohesive, system-wide strategy, which can lead to inconsistent or conflicting actions among agents. Coordinating and ensuring coherent behavior among agents becomes more challenging. There's also the potential for redundancy, as multiple agents might independently perform similar tasks or analyses, leading to inefficiencies.

Global intelligence, where decision-making and processing are centralized, offers several benefits. Firstly, it ensures consistency in actions and strategies across all agents, fostering uniformity in the system's behavior. This centralized approach also simplifies the management of the system, as changes and updates can be implemented uniformly. Moreover, Agents can consolidate their individual knowledge with other agents to learn more generic patterns than what local processing would lead. In [12], Prodromidis introduce two granularities of anomaly detection agents. The first is the "local fraud detection agent", which specializes in learning fraud detection patterns within a specific information system. The second is a meta-learning mechanism that leverages the collective knowledge of these local agents. In [13] the authors employ a similar architecture to identify anomalies in Industry 4.0. They demonstrate that agents are capable of undertaking sophisticated tasks, such as controlling machines, by acquiring knowledge from sensors and actively participating in communication with other agents. Global intelligence models have also advantages like reduced memory usage compared to fully decentralized smaller models [14]. Nonetheless, this system's knowledge level presents significant drawbacks. Centralized intelligence can lead to bottlenecks, where the system's performance hinges on a single point,

making it vulnerable to failures or attacks. It also poses scalability issues, as increasing the number of agents can overwhelm the central intelligence, leading to inefficiencies. One way to fix these problem is by creating communities of learners that are intermediate centralized points for agent to learn together. As described in [21], identifying similar interests and building communities can help agents to learn through collaborative learning [15]. To mitigate the single point of failure, various methods exist, including encrypted communication and processing [10], as well as data duplication to counteract server shutdowns, etc. However, each solution also introduces its own set of challenges.

2.3 Wise Object Framework

The WOF, as described in [5], is a software framework that enables object-based software systems to acquire intelligence with minimal intrusion into the business code, requiring only a simple decoration. During runtime, a dynamic proxy is instantiated for each decorated object. This proxy is designed to manage every inputs and outputs of the object. It is termed an agent or Wise Object (WO) due to the broad range of capabilities it possesses.

Controlling the object The concept of Wise Object is founded on the principle of introspection. Similar to humans, agents can engage in introspection to gain knowledge about the objects they represent, including fields, available methods, throwable exceptions, and more. This enable the sharing of information about the real objects for a future analysis. By controlling the object, the agent also controls the flows between objects themselves and between the real world and the object. Controlling flows have many benefits:

Catch any call within the application and store it Before any calls reach the real object, the agent is notified about incoming action. The agent automatically save the call in the object knowledge. Saving calls grants the possibility of using external log analysis tools. These tools can aid in comprehending the inner workings of the application, generating flowcharts, and enhancing the software's intelligence. Logs can also be sent to third party intelligent agents for complex analysis and expert insights.

Modify or block calls between objects if necessary The ability to manipulate interactions between object is essential in enabling the system to become self-adaptive. Once analytical agents have amassed a sufficient amount of knowledge, they can disseminate this knowledge to other agents and subsequently decide whether their involvement in the system's interactions is necessary. In cases where the interactions are deemed foreign to the system, agents possess the authority to obstruct them and transmit a report to the system's owner.

Modify or block calls between the application and the real world Finally, capturing and altering external calls to the application offers tangible benefits, including blocking effect on the real world. This approach can be used to prevent object from acting if an anomaly is detected in previous analysis.

Versatility of the WOF A key principle of the WOF is its versatility. Designed to integrate seamlessly with any application, the framework minimizes code intrusion. This is of the main motivation for the event-driven architecture of the framework. Additionally, the framework offers extendable classes, allowing for the incorporation of custom analysis and communication components.

Innovation of the WOF The WOF offers the ability to disconnect the real world from the business code. When the system is not actively serving the user (in the Awake state), it can transition into the Dream state. In this Dream state, objects are isolated from the real world, ensuring that the system's operations remain contained within itself. By preserving states, the system can revert to the exact state the user left it in, allowing for systematic exploration of the system's behavior. The concept of Dream is introduced around the application of IoT [4]. In such applications, real objects are managed by their corresponding avatars, which are software objects. The innovative aspect of the Dream state is its ability to disconnect the real object from the communication bus, allowing operations to be tested only on the avatar. This concept of an avatar can be extended to any software object by adding an extra layer of abstraction. The Dream state offers several advantages, including stress and unit testing, tracking flows, comprehending object interactions, employing simulated data on the live system for output analysis, and more.

Applications The relevance of the WOF have been showcased within the realm of IoT for creating an intelligent classroom with the capability to identify unusual behaviors. The outcomes presented in [4] are highly encouraging, underscoring the potential of the WOF as a foundation for crafting agent-based adaptive systems. The framework is also used in multi-camera environment. Some cameras are logging images to generate training data for an external agent responsible for classifying new pictures.

3 Contribution

3.1 Agent-to-System communication

The communication from agent to system uses the analysis workflow. The WOF provides a clear API to design analyzer that are triggered by pre-defined events. In figure 1 the system communication analyzer is called SystemBridgeAnalyzer. The SystemBridgeAnalyzer shares the HistoryKnowledge of the agent to the system. The HistoryKnowledge is the first output of analysis about logs (by setting the in order of arrival). It would be overflowing to share the object data to the system at each action of the agent. Therefore, we introduce TriggerKnowledge to act as the first part of the agent-to-system communication. The SystemBridgeAnalyzer is a listener of any types of TriggerKnowledge. The knowledge is created upon different configuration allowing the user to customize the communication

protocol. An example includes a CountTriggerAnalyzer, which generates a TriggerKnowledge after every n actions of the agent. Due to the modular nature of the WOF architecture, a variety of additional triggers can be implemented in just a matter of minutes.

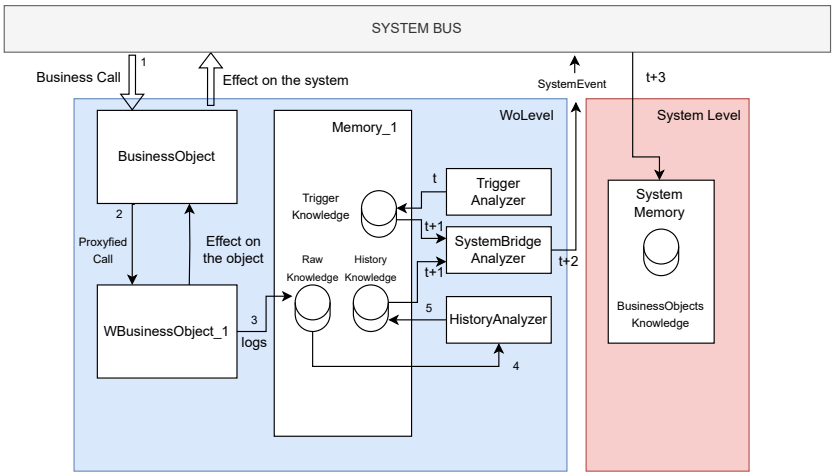

Fig. 1. Agent-to-System Communication

The primary challenge in this communication framework is determining how to compose knowledge, how the system automatically identifies compatible types of knowledge for consolidation and whether it should automatically trust data from agents. In the initial design of the agent-to-system communication, a straightforward approach is adopted: the system's memory records the Unique Identifier (UID) of each object along with its HistoryKnowledge. Knowledge of identical categories is aggregated into a singular data structure, simplifying the process of collective analysis. Composing knowledge, which involves merging different types of information to conceptualize non-existing entities like rooms in a sensor-filled environment, is crucial for understanding the overall functioning of the system, rather than just its individual components. Given the WOF's goal to achieve versatility, tackling the challenge of knowledge composition in an automated context proves to be quite challenging. In this paper, while we do not address this particular issue, it is important to note that its resolution would be highly valuable and merits further investigation in future research.

3.2 Agent-to-Agent communication

A key difficulty in consolidating data at the system level lies in achieving synchronization. Ideally, when processing data at a given moment, the system requires access to all agents' data in memory. However, due to the event system nature of

the WOF some agent will not be awaked by the trigger to share their information with the system. A practical solution to this issue is enabling direct communication between agents. Previously, agents lacked the capability to communicate directly. The communication protocol we introduce is designed for simplicity: when a TriggerKnowledge is created by an agent, that agent broadcasts a TriggerAnalyzerEvent to all other agents. The event is linked to the agent type, ensuring that only relevant agents process it. Once an agent receives an event, it's stored in its memory, activating the agent-to-system communication sequence. A crucial step is for agents to check if an identical Event has already been generated by them or others, preventing knowledge duplication which could distort analysis. Agents must confirm the uniqueness of a TriggerEvent by comparing it against existing ones before storing any new TriggerEvents. This measure is vital for preserving the accuracy and integrity of the analysis.

The communication between agents offers numerous advantages for various tasks, yet it also introduces several challenges, particularly in an event-driven architecture. In situations where certain agents are excessively solicited, broadcasting an event to request other agents to share their unchanged knowledge may be counterproductive. Such actions could potentially lead to an overload on the communication bus, making it less efficient compared to a system where communication is driven by specific agent actions. It is imperative for users to be mindful of this constraint and to develop a TriggerAnalyzer that is optimally aligned with the objectives of system-level analysis.

3.3 Communication consequences

In the first though, adding communication between agents might increase complexity as agents are constantly broadcasting. Access to system-level data enables the application of various machine learning (ML) algorithms, which are recognized for their significant business value. However, the complexity and resource demands of ML algorithms, in terms of both computational time and storage, are major concerns. In this section, we will explore the influence of communication on system performance, specifically focusing on aspects such as time and memory usage.

Experimental Settings In the following experiments, we will explore the selection of two distinct datasets that embody values characteristic of real-life applications. For each application, we have developed software designed to utilize this data, simulating a business application context. The chosen datasets significantly differ in the volume of data they contain as shown in Table 1. This variance is intentional, as it allows us to examine the effects of scaling either the number of agents or the amount of data generated by these agents.

Datasets We selected the *Wine* dataset [17] from the University of California, Irvine, (UCI) Machine Learning Repository, as introduced by [3], to serve as our baseline dataset. The dataset comprises 179 records of chemical metrics of

Table 1. Datasets description

Dataset	Number of records	Number of agents	Data dimension
$Wine-3$	179	3	14
$Ausgrid-20$	10000	20	4
$Ausgrid-100$	10000	100	4

wines originating from the same Italian region. The records are divided into three different cultivars. Records are sent to the application through a system bus and three cultivars objects are instanced as business application. Our study analyzes also a dataset from Ausgrid [1], featuring smart meter data from 300 clients over three years. We created two datasets: $Ausgrid-20$ with data from 20 clients and $Ausgrid-100$ from 100 clients, to investigate complexity increase with more agents.

Hardware settings Experiments are conducted on a 2017 MacBookPro 2,8 GHz Intel Core i7, 4 cores with 16 Go of 2133 Mhz LPDDR3 Ram.

Software settings The analytical component of the system is developed using Python 3.9, whereas the core of the Wise Object Framework (WOF) is currently maintained in Java 17. For the purpose of facilitating communication between the framework's core and the system, Fast API is employed, which is recognized for being among the most efficient frameworks available in Python [2].

Framework settings In each experiment we define a policy that is used by the WOF to instantiate analyzers at runtime. The policy got the basic analyzers needed to send logs to the system level such as HistoryAnalyzer, TriggerAnalyzer, AgentToAgentAnalyzer (needed for agent-to-agent communication) and SystemBridgeAnalyzer. For the self-corrective loop, we added a DreamGeneratorAnalyzer that ask the system to generate usable Dream data and a DreamPlanner to apply these plans. For each dataset, we employ various triggers designed to initiate an event after every 10% of the total data produced by the agents.

To test the complexity of the framework in real condition we tested 2 different system-level plan generation algorithms. Plans serve as data that enable agents to mimic their operational state during dreaming. We opted for replication plans over exploration plans to simplify data processing. The first algorithm, termed as *mean*, is a straightforward plan generator that produces new data by calculating the average of historical data. Additionally, we employed a more sophisticated algorithm renowned for creating synthetic tabular data, the Conditional Tabular Generative Adversarial Network (CTGAN), as outlined in [20], and referred to as *Gan*. In this example, the algorithm processes the entire dataset from the start of storage without employing time windows, meaning it considers all historical data for plan creation. For each dataset and both algorithms, we compare

metrics from individual-based approach, where data from each agent is used separately, and the collective approach, which accumulates data from all agents for a comprehensive analysis.

Time & space analysis stack To analyze differences between different configuration we chose to containerize the system part. This way no external parameter will have an impact on the recorded performances. We use Docker version 24.0.7 and cAdvisor/Prometheus/Graphana as the monitoring pipeline.

3.4 Results

Table 2. Time consumption

Dataset	**WOF consumption ms/call**	**Log Storing ms/call**	**Plan Generation ms/call**
$Wine - Local - mean$	0.18 (+/- 0.17)	2.61 (+/- 0.76)	0.26 (+/- 0.09)
$Wine - System - mean$	0.18 (+/- 0.17)	2.38 (+/- 0.75)	0.22 (+/- 0.07)
$Ausgrid - 20 - Local - mean$	0.20 (+/- 0.12)	2.30 (+/- 3.71)	0.21 (+/- 0.18)
$Ausgrid - 20 - System - mean$	0.20 (+/- 0.12)	5.42 (+/- 2.53)	0.189 (+/- 0.06)
$Ausgrid - 100 - Local - mean$	0.30 (+/- 1.6)	1.89 (+/-1.35)	0.21 (+/-0.13)
$Ausgrid - 100 - System - mean$	0.30 (+/- 1.6)	6.44 (+/- 3.98)	0.18 (+/- 0.06)
$Wine - Local - Gan$	0.18 (+/- 0.17)	2312.7 (+/- 927.9)	99.92 (+/- 53.78)
$Wine - System - Gan$	0.18 (+/- 0.17)	740.52 (+/- 1135.9)	68.62 (+/- 35.70)
$Ausgrid - 20 - Local - Gan$	0.20 (+/- 0.12)	2108.65 (+/- 2977.85)	26.63 (+/- 23.33)
$Ausgrid - 20 - System - Gan$	0.20 (+/- 0.12)	2579.68 (+/- 13057.48)	34.99 (+/- 36.52)
$Ausgrid - 100 - Local - Gan$	0.30 (+/- 1.6)	1736.81 (+/- 2480.07)	27.69 (+/- 23.69)
$Ausgrid - 100 - System - Gan$	0.30 (+/- 1.6)	171.55 (+/- 1818.20)	33.35 (+/- 62.20)

Table 3. Memory consumption

Dataset	**WOF** Metaspace	**System level** Analysis	**Local** Analysis
$Wine - mean$	21.3 MiB	54 MiB	53 MiB
$Wine - Gan$	21.3 MiB	275 MiB	192 MiB
$Ausgrid - 20 - mean$	22.0 MiB	94 MiB	99 MiB
$Ausgrid - 20 - Gan$	22.0 MiB	1.9 GiB	521 MiB
$Ausgrid - 100 - mean$	22.4 MiB	94 MiB	112 MiB
$Ausgrid - 100 - Gan$	22.4 MiB	1.9 GiB	521 MiB

This research presents the first assessment of the Wise Object Framework's (WOF) performance in different scenarios involving a large number of agents. Table 2 illustrates that the WOF processing time tends to rise with the increasing number of agents. The increase in standard deviation with $ausgrid-100$ suggests process queuing, aligning with the WOF's threading model. Increasing the thread number could lower this variability, potentially decreasing overall processing time but should increase memory consumption.

The memory footprint for analysis, as shown in Table 3, tends to increase with the complexity and additional dimensions of the data. The complexity of the data and the nature of the variables involved are key factors. This explains why the $Wine$ dataset requires more memory than the $AusGrid$ dataset.

System-level models undergo training once all agents have submitted their data, whereas agent-level models are trained with each request to store data. This varying training frequency contributes to the time disparity 2 between system storage/plan generation and individual agent processing. Variability in data sources and differences in analyzer configurations constitute the principal hurdles in comparing algorithms across varied datasets.

Running system-analyzers on a singular process means that plan generation at the system level significantly affects the performance of storage processes, as shown in the Gan experiment (Table 2). In the current philosophy of the WOF, objects in the IDLE state tend to request dream plans, which inadvertently leads to these idle objects consuming system's CPU and bandwidth by continuously receiving plan requests. A potential solution could involve isolate services into different processes, specifically separating the log storage (which requires speed and constant availability) from the optional dream plan generation. Alternatively, shifting the initiation of plan generation from the agent to the system itself could mitigate this issue, allowing for more effective resource management and ensuring the log storing process remains fully operational.

3.5 Limits

Although incorporating communication could enhance the system's efficiency, it also introduces several challenges, including potential server bottlenecks and concerns regarding data privacy. It is important to note that currently, communication between agents and the system within the Wise Object Framework (WOF) is not encrypted. While encryption is not a primary focus at this stage of research development, the implications of centralized communication, especially in terms of security, should not be overlooked. Agent synchronization is also a problem that may be solved by implementing the right notification protocol.

As we expected the system became a bottleneck when the WOF is tasked with sharing a lot of individual logs. It's crucial to highlight that our tests employed simulated time rather than real time, ensuring feasible testing conditions. In a production setting using the WOF model, it is anticipated that issues related to system overflow would be notably diminished, given the WOF's extended time range for operations.

The WOF times listed in table 2 reflect the performance of WOF wrappers and are based on a test client application that only sets fields. When using the WOF on real usecase, time spent by the client methods may vary impacting the WOF's processing time. Actual client application usage may result in different processing times for the WOF due to varying client method execution times.

4 Concluding remarks

This paper introduces a straightforward method for implementing a secondary (meta) knowledge level, which offers several benefits, including mitigating cold start problems and reducing overfitting. The experiment demonstrates that comparing data processing at the system and agent levels is challenging. The processing times listed in Table 2 and the memory usage in Table 3 are heavily influenced by the analyzers' configurations, such as training schedules and data posting practices. Nonetheless, there are key takeaways from this study in order to efficiently add system level analysis to MAS. Users should recognize the challenges posed by this communication framework including possible system overflow, issues with agent synchronization, and the risk of duplicating knowledge. Addressing these issues effectively requires a clear separation of concerns of the system's components.

It is worth noting that application complexity encompasses not only runtime duration and memory usage but also the time required for a new developer to comprehend the entire application. Incorporating this type of communication offers numerous benefits, yet significantly amplifies the complexity of interactions among objects.

This paper tests a global storage and plan generation layer, which raises philosophical questions concerning the WOF model's perspective on dreaming. It debates whether objects should request dream data upon entering the IDLE state and if a higher intelligence level should dictate which agents dream and the dream's content, aiming to minimize unnecessary communications and knowledge generation.

References

1. Solar home electricity data - Ausgrid, https://www.ausgrid.com.au:443/Industry/Our-Research/Data-to-share/Solar-home-electricity-data
2. Web Framework Benchmarks, https://www.techempower.com/benchmarks/#section=test&runid=7464e520-0dc2-473d-bd34-dbdfd7e85911&hw=ph&test=query&l=zijzen-7
3. Aeberhard, S., Coomans, D., De Vel, O.: Comparative analysis of statistical pattern recognition methods in high dimensional settings. Pattern Recognition **27**(8), 1065–1077 (Aug 1994)
4. Alloui, I., Benoit, E., Perrin, S., Vernier, F.: WIoT: Interconnection between Wise Objects and IoT. In: International Conference on Software Technologies. pp. 349–371. Springer (2018), journal Abbreviation: 13th International Conference on Software Technologies

5. Alloui, I., Vernier, F.: A Wise Object Framework for Distributed Intelligent Adaptive Systems. 12th International Conference on Software Technologies (2017)
6. Bernon, C., Gleizes, M.P., Peyruqueou, S., Picard, G.: ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) Engineering Societies in the Agents World III. pp. 156–169. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2003)
7. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Muller, H., Pezze, M., Shaw, M.: Engineering Self-Adaptive Systems through Feedback Loops. Software engineering for self-adaptive systems (2009)
8. Cheng, S.W., Garlan, D., Schmerl, B.: Evaluating the Effectiveness of the Rainbow Self-Adaptive System. ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (2009)
9. Crow, L.H.: Evaluating the reliability of repairable systems. Annual proceedings on reliability and maintainability (1990)
10. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178. ACM, Bethesda MD USA (May 2009)
11. Jennings, N.R., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems **1**(1), 7–38 (1998)
12. L. Prodromidis, A., Stolfo, S.: Agent-Based Distributed Learning Applied to Fraud Detection. Technical Report CUCS-014-99, Columbia Univ (1999)
13. Mateos García, N.: Multi-agent system for anomaly detection in Industry 4.0 using Machine Learning techniques. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal **8**(4), 33–40 (Sep 2019)
14. Montero-Manso, P., Hyndman, R.J.: Principles and Algorithms for Forecasting Groups of Time Series: Locality and Globality (Mar 2021)
15. Panait, L., Luke, S.: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems **11**(3), 387–434 (Nov 2005)
16. Ravindranathan, M., Leitch, R.: Heterogeneous intelligent control systems. IEE Proceedings-Control Theory and Applications (1998)
17. Stefan Aeberhard, M.F.: Wine (1992), https://archive.ics.uci.edu/dataset/109
18. Suzanne Barber, K., Goel, A., Martin, C.E.: Dynamic adaptive autonomy in multi-agent systems. Journal of Experimental & Theoretical Artificial Intelligence **12**(2), 129–147 (Apr 2000)
19. Weyns, D., Iftikhar, M.U., Malek, S., Andersson, J.: Claims and Supporting Evidence for Self-Adaptive Systems: A Literature Study. 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (2012)
20. Xu, L., Skoularidou, M., Cuesta-Infante, A., Veeramachaneni, K.: Modeling Tabular data using Conditional GAN. Advances in neural information processing systems **32** (Oct 2019)
21. Yang, F., Wang, M., Shen, R., Han, P.: Community-organizing agent: An artificial intelligent system for building learning communities among large numbers of learners. Computers & Education **49**(2), 131–147 (Sep 2007)