electronics

MDPI

Article

Improving Fault Tolerance and Reliability of Heterogeneous Multi-Agent IoT Systems Using Intelligence Transfer

Vyas O'Neill * and Ben Soh

Department of Computer Science and Information Technology, La Trobe University, Melbourne 3083, Australia
* Correspondence: v.oneill@latrobe.edu.au

Abstract: Driven by the ever-growing diversity of software and hardware agents available on the market, Internet-of-Things (IoT) systems, functioning as heterogeneous multi-agent systems (MASs), are increasingly required to provide a level of reliability and fault tolerance. In this paper, we develop an approach to generalized quantifiable modeling of fault-tolerant and reliable MAS. We propose a novel software architectural model, the Intelligence Transfer Model (ITM), by which intelligence can be transferred between agents in a heterogeneous MAS. In the ITM, we propose a novel mechanism, the latent acceptable state, which enables it to achieve improved levels of fault tolerance and reliability in task-based redundancy systems, as used in the ITM, in comparison with existing agent-based redundancy approaches. We demonstrate these improvements through experimental testing of the ITM using an open-source candidate implementation of the model, developed in Python, and through an open-source simulator that tested the behavior of ITM-based MASs at scale. The results of these experiments demonstrated improvements in fault tolerance and reliability across all MAS configurations we tested. Fault tolerance was observed to improve by a factor of between 1.27 and 6.34 in comparison with the control group, depending on the ITM configuration tested. Similarly, reliability was observed to improve by a factor of between 1.00 and 4.73. Our proposed model has broad applicability to various IoT applications and generally in MASs that have fault tolerance or reliability requirements, such as in cloud computing and autonomous vehicles.

Keywords: internet of things; multi-agent systems; intelligent agents; adaptive systems; fault tolerant systems; system reliability

Citation: O'Neill, V.; Soh, B. Improving Fault Tolerance and Reliability of Heterogeneous Multi-Agent IoT Systems Using Intelligence Transfer. *Electronics* **2022**, *11*, 2724. https://doi.org/10.3390/electronics11172724

Academic Editors: Rashid Mehmood, Aakash Ahmad, Mahdi Fahmideh and Juan M. Corchado

Received: 17 July 2022
Accepted: 27 August 2022
Published: 30 August 2022

1. Introduction

The Internet of Things (IoT) is becoming increasingly ubiquitous across various disciplines, including such diverse applications as wearable devices, blockchain, smart cities, smart homes, autonomous vehicles, urban transportation, smart farming, and commercial and industrial uses (IIoT) [1–4]. IoT platforms often exhibit a high level of heterogeneity, both in software and hardware [5]. This heterogeneity makes writing software for IoT applications particularly challenging because of the wide diversity of platforms and operating conditions in which the software must operate [6]. As their range of applications grows, IoT systems are becoming increasingly subject to fault tolerance and reliability requirements as part of their software engineering process [7].

From a software engineering perspective, IoT networks may be modeled as heterogeneous multi-agent systems (MASs) [8]. Thus, an emerging challenge for the software architecture of intelligent agents in IoT applications is modeling, designing, and constructing reliable and fault-tolerant MASs [9]. This has broad implications for IoT's ability to gain mainstream acceptance in applications that have dependability requirements.

Existing approaches to fault tolerance and reliability in MASs generally use agent-based redundancy models which are built on the replication of agents in order to achieve the desired redundancy. However, complete replication of an agent may be unnecessarily costly in terms of resource usage, overhead, and survivability. In this paper, we identify three

intelligent activities: an agent's situational awareness, logical decision-making capacity, and agency, and propose a model which decouples them from individual agents, allowing the MAS to implement its redundancy at the finer, task-based level rather than at the agent level. We show both theoretically and in empirical testing that such a model outperforms the benchmark of agent-based redundancy.

We make the following contributions:

- A novel software architectural paradigm, the Intelligence Transfer Model (ITM) that achieves MAS goals through the decoupling and inter-agent transfer of intelligent activities;
- A novel mechanism in fault-tolerant systems design: the *latent acceptable state*. This mechanism forms the basis for improved fault tolerance and reliability outcomes for MASs implementing the ITM;
- A novel approach to quantifying the fault tolerance and reliability improvement of a MAS implementing the ITM;
- A graphical language for modeling systems implementing task-based redundancy as used in the ITM;
- An open-source candidate implementation of the ITM;
- An open-source simulator for comparing fault tolerance and reliability outcomes of ITM and non-ITM, benchmark MASs. This simulator was applied to the models proposed in this paper to empirically study the improvement in fault tolerance and reliability in ITM MASs.

The applications of these contributions have broad applicability across various areas of research in software engineering and architecture of IoT and IIoT systems, cloud computing, edge computing, autonomous vehicles, and generally in MASs which are expected to provide a dependable level of service.

In Section 2, we survey existing approaches to fault tolerance and reliability in MASs. Then, in Section 3 we present the proposed ITM. In Section 4, we discuss an open-source candidate implementation for the ITM which was developed during this research. Then, in Section 5, we propose a novel approach to quantifying the fault tolerance and reliability improvements of a MAS implementing the ITM, including a graphical language for modeling systems implementing task-based redundancy. In Section 6, we present our open-source simulator and the results of simulations carried out to experimentally validate the ITM, quantifying its improvements to fault tolerance and reliability. A table of abbreviations used in the paper is given in the Abbreviations part.

2. Existing Approaches to Fault Tolerance and Reliability in MASs

In this section, we survey existing approaches in the literature to fault tolerance and reliability in MASs. Table 1 gives a relative comparison of the different approaches.

A wide range of software architectural models have been proposed for IoT MAS systems, including domain-specific and domain-independent algorithms [8], microservices architecture [10], sensing-as-a-service [11], and edge and fog computing [12]. Several frameworks and standards for the general design and development of intelligent MASs exist, including the Foundation for Intelligent Physical Agents (FIPA) [13–18]. There is also research interest in software engineering design and communication of IoT architectures, including developing specific modeling languages for this purpose [19].

The trend in IoT MAS architecture mirrors the recent general trend in software architecture towards more granular, decentralized, microservices-based designs, rather than large, monolithic systems [10]. As the system architecture becomes more distributed and granular, the required level of agent inter-coordination increases [20]. This decentralization of IoT MAS, as well as the particular characteristics of IoT applications, have driven the demand for IoT software to be "smart," utilizing software engineering techniques developed for intelligent agents to better enable the IoT device to participate in its heterogeneous, decentralized environment [21]. Thus, the software engineering principles which guide the development of intelligent agents and MAS, in general, are well-suited to improving the outcomes of IoT agents, comprising the software running on an IoT device [22].

As the applications for IoT networks grow, increasing demands are being made on their fault tolerance and reliability [7,9]. A fault is defined as "a physical defect, imperfection, or flaw that occurs within some hardware or software component" [23]. Fault tolerance is defined as "the ability of a system to continue to perform tasks after the occurrence of faults" [23]. Reliability is "a function of time . . . the probability that the system operates correctly throughout a complete interval of time" [23].

Fault tolerance and reliability in MASs are open research problems [8,9,22,24]. There have been several different approaches, implementing various fault tolerance techniques to achieve greater agent reliability. Some have approached the problem from the human coder perspective, creating software design and coding tools to facilitate the better design of MAS agents' code [14–19]. Others have proposed improvements to the communications protocols which underpin the MAS, including approaches for establishing consensus in leaderless MASs [25,26]. Researchers have also been interested in developing methods for the detection of faults using MASs [27–31].

Another area of research is the analysis of the effect that cascading faults in one agent have on others in the MAS [32,33]. In [32], the authors use dynamic fault trees and a combinatorial model to analyze the fault-propagation behavior of different components making up an IoT MAS. In [33], the authors propose a cross-domain, agent-based model to analyze the cascading failure effects associated with malware propagation in networked IoT systems. In [34], the authors propose using a multi-agent genetic algorithm to select robust network paths to mitigate malicious attacks and cascade failures.

Some researchers have approached the problem from the perspective of self-healing systems [35–37]. Self-healing systems seek to provide maximum reliability by preserving availability and containing faults [35]. In [35], the authors propose a MAS architecture where individual agents self-detect and repair their faults while communicating with a planning agent which manages the migration of service delivery to other agents in the MAS.

Other researchers have applied general MAS reliability techniques to the unique environment of cloud microservices, where the stateless nature of the agents in the MAS enables specific methods of achieving reliability and fault tolerance outcomes [38,39]. In cloud applications, microservices are often run in dedicated, stateless containers which can be instantiated and destroyed on demand [40]. Such an environment blurs the distinction between agent-based and task-based redundancy because each agent exists to carry out a single task [40]. Achieving fault tolerance and reliability then becomes a scheduling challenge, ensuring that enough agents are available at a given time to provide the necessary level of service [38]. For example, in [38], the authors propose to achieve this through the use of a heuristic algorithm for scheduling microservice tasks within defined resource constraints.

Generally, these existing approaches to reliable, fault-tolerant MAS design build on research conducted on individual hardware and software platforms before heterogeneous MASs became widely adopted. Consequently, we find that these approaches are generally agent-focused, considering how faults can be prevented from becoming failures at the agent level, or how faults can be contained at the agent level without causing the MAS to fail. The three-universe model [23] is one such example. However, the advantage of a MAS is that it may act as a "whole" greater than the sum of its parts, thus, additional work must be completed to extend existing fault tolerance and reliability models to make them inherently applicable to MASs [41].

Existing fault prevention and recovery strategies are often domain-specific, using agent-based redundancy to replicate individual agents, thereby improving the level of fault tolerance and reliability in the MAS [42–46]. However, the domain-specific nature of the solutions limits their applicability to other problem domains.

Many agent-based redundancy strategies attempt to identify those agents which are critical to the outcome of some task (high criticality), prioritizing them for replication and thereby reducing the costs due to unnecessary replication [17,45,47]. Under this perspective,

however, individual agents are still considered the atomic unit making up the MAS and must be replicated as self-container units in order to achieve improvements in redundancy.

Table 1. Relative comparison table showing how existing studies have approached the problem of MAS fault tolerance and reliability.

Paper	Research Context	System Architecture and Modeling	Decentralization	Software Design and Coding Tools	Communication Protocols	Fault Detection and/or Propagation	Domain-Specific Algorithms	Agent Criticality and/or Replication
[4]	IoT/Smart Cities	X	X		X			
[8]	General MAS	X	X		X	X	X	
[9]	IoT					X		
[10]	IoT	X						
[11]	IoT	X			X			
[12]	Fog/Edge Computing	X	X					
[13]	Microgrid	X	X				X	
[14]	General MAS	X			X		X	
[15]	General MAS	X		X				
[16]	General MAS	X		X				
[17]	General MAS	X		X				X
[18]	General MAS	X		X				
[19]	IoT	X	X	X				
[20]	General MAS		X					
[21]	IoT		X		X			
[22]	Cyber-Physical Systems	X				X		
[24]	Smart Cities	X				X		
[25]	General MAS				X	X		
[26]	General MAS	X			X	X		
[27]	Linear Parameter-Varying Systems	X				X		
[28]	General MAS					X		
[29]	General MAS					X		
[30]	General MAS		X			X		
[31]	General MAS					X		
[32]	IoT	X				X		
[33]	IoT	X				X		
[34]	Networked Systems	X	X					
[35]	General MAS					X		X
[36]	Distribution Networks		X			X	X	
[37]	Distribution Networks		X			X	X	
[38]	Cloud	X					X	X
[39]	Cloud	X				X		X
[42]	Distribution Systems					X	X	
[43]	General MAS	X						X
[44]	Distribution Systems	X					X	X
[45]	General MAS							X
[46]	Microgrids	X				X	X	
[47]	General MAS	X						X
[48]	General MAS	X		X		X		
[49]	Datacenters					X		
[50]	Real-time and embedded systems	X		X		X		

3. Proposed Intelligence Transfer Model

In this section, we propose a novel software architectural model, the ITM, to improve upon agent-based redundancy models through dynamic task-based redundancy.

3.1. Decoupling an Agent's Intelligent Activities

Intelligence can be defined as "one's ability to learn from experience and to adapt to, shape, and select environments" [51]. In [8] an agent is defined as "an entity which is placed in an environment and senses different parameters that are used to make a decision based on the goal of the entity. The entity performs the necessary action on the environment based on this decision". In [52], an agent is defined as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators".

Applying these definitions to agents in a MAS, we can identify three core *intelligent activities* that characterize intelligent agents: the ability to experience their environments, a mechanism for decision-making, and some agency (a set of possible actions the agent may take) by which the agent can shape and influence its environment. Thus, if we consider an intelligent agent at some point in time, we can define it in terms of three scopes of information: the situational awareness comprising all the sensory input the agent has received from the environment; the logical image comprising the decision-making model of the agent; and agency, or the set of all possible actions the agent knows how to perform.

By taking the agent to be the unit of the MAS, as is generally the case in existing models, the three intelligent activities, manifesting as three structurally independent software concerns, are treated as a single component at the agent level. This limits the potential for redundancy to the level of the individual agents (agent-based redundancy), rather than the finer level of *task-based redundancy*, defined as the number of agents in a MAS that are able to carry out a specific task at a given time. The advantage of the task-based redundancy approach is that it may better leverage the heterogeneous nature of the agents, which may have overlapping capabilities at the task level.

The ITM aims to improve upon agent-based redundancy models by reframing the atomic unit of the MAS in terms of the intelligent activities carried out in an agent-independent manner. It proposes to decouple these concerns, treating each independently and distributing them over the MAS in a decentralized manner.

In this way, by abstracting situational awareness, decision-making capability, and agency, constraints arising from their co-location at individual agents are removed. Critically, additional acceptable states which would otherwise not exist can be created in the fault tolerance model of the MAS. Furthermore, by improving the level of task-based redundancy, the reliability function of the system can be likewise improved.

The ITM is designed to support agent heterogeneity while creating redundancy at the task level rather than at the agent level (task-based redundancy). Thus, the ITM does not assume the ability to simply replicate entire agents, making it particularly useful for heterogeneous MAS where full agent replication may not be possible for practical, intellectual property, ownership, strategic or security reasons, etc. Rather, the individual logical tasks are replicated between the agents according to the MAS Intelligence Transfer strategy.

3.2. Transfer of Intelligence

The aim of the ITM is to facilitate the *transfer of intelligence* between agents in heterogeneous MASs. This mechanism is then applied to the problem of fault tolerance and reliability at the most abstract, generalized level of the MAS, with the aim of quantifying and improving these factors independently of any specific problem domain.

We define the *transfer of intelligence* to mean the transfer between agents, in whole or in part, of the necessary logic to accomplish one or more of the intelligent activities required to achieve some *task*. A *task* refers to some outcome or accomplishment desired from the MAS. These definitions are broad by design to accommodate the wide range of potential MAS applications.

The decoupling and distribution of these three critical sets of information from the local agent level empower other agents in the MAS to use different algorithms to make local decisions in support of the global task, to collaborate, and cooperatively achieve goals in a decentralized manner while providing an improved level of redundancy for the intelligent activities which make up tasks at the MAS level. Such a MAS is more resilient, fault-tolerant, and responsive to local and global changes.

In the ITM, the three intelligent activities (situational awareness, decision-making logic, and agency) exist in a distributed manner across the whole MAS (Figure 1). Parts of the information sets enabling each activity are communicated between agents in the MAS, as necessary, with each agent holding a portion of each in its local context. Thus, the system, as a whole, exhibits the desired state independent of the local context of each agent.

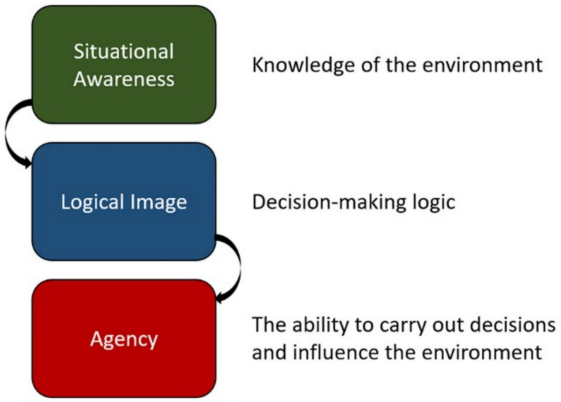

Figure 1. The three intelligent activities that make up the Intelligence Transfer Model (ITM).

3.3. Flow of Control in an Agent Implementing the ITM

Agents implementing the Intelligence Transfer Model behave according to the following process:

1. The process begins with a trigger. This may be a pre-programmed timer poll, an event-driven framework where messages sent through some communication protocols are tested against locally defined signatures in the agent, a response to a change in the environment, human input, or some other mechanism;
2. The local *situational awareness* is constructed according to data from on-board sensors, data received through communication with other agents in the MAS, and memory of past state information;
3. A decision is taken in the *logical image*, based on the agent's situational awareness;
4. The decision is executed as an action carried out locally or potentially by some other agent through a cooperative or delegative process;
5. Where logic required to perform an intelligent activity is missing, unavailable, or out of date, an agent may contact other agents in the MAS who, through Intelligence Transfer, can transfer the required logic to the requesting agent.

3.4. Software Architecture of an ITM-Compatible Agent

The general software architecture of an ITM-compatible agent is shown in Figure 2. Such an agent must provide the following functionality:

1. Handle the local hardware requirements of the agent (as applicable);
2. Process incoming ITM messages from the network;
3. Update the situational awareness from local and MAS sources;
4. Update the logical image and execute any decision-making logic;
5. Carry out the actions decided upon by the logical image;
6. Communicate any ITM messages generated during this process to the MAS.

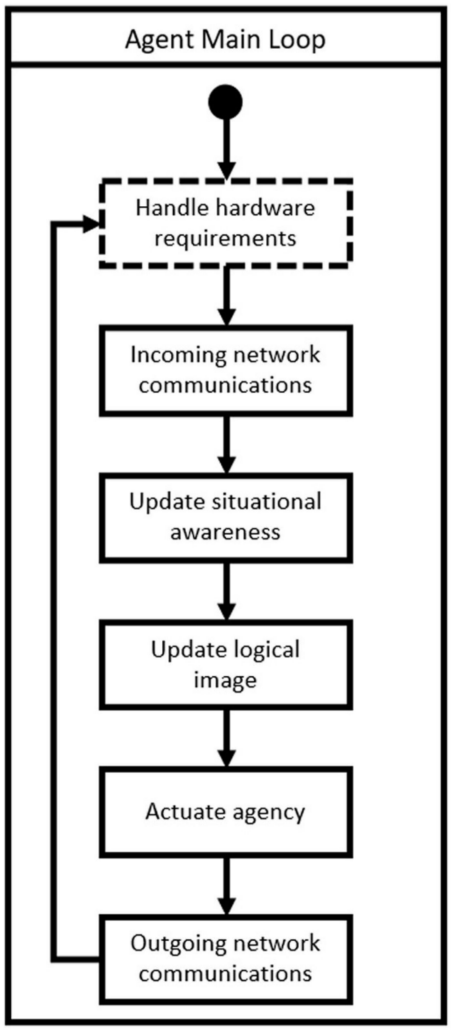

Figure 2. The general software architecture of an ITM-compatible agent.

This architecture is simplified; concrete implementations may achieve this with concurrency and other design patterns, depending upon the technologies used to implement the local agent. Additionally, the separation of concerns between situational awareness, logical image, and the agency is a software engineering paradigm designed to facilitate different Intelligence Transfer strategies based on whether the activity involved is related to enriching the agent's understanding of its environment, making decisions based on that understanding, or executing the selected course of action. The separation may not necessarily be at the functional level; fundamentally all three are implemented as executable code.

Section 4 details how this architecture was realized in a candidate implementation of the ITM.

3.5. Example of the Intelligence Transfer Process

Let us consider a simple example of an IIoT MAS implementing the ITM to illustrate the potential benefits that may be achieved. Suppose we have a MAS representing an economy where customer agents engage with manufacturing agents, each corresponding to a robot whose job it is to mix paint. The robot has four paint nozzles for each primary color (cyan, magenta, yellow and black) and a water hose which is used to clean the mixing drum at the conclusion of each batch.

When the robot was designed and first installed at the factory, the paint manufacturer expected all paints to be produced at the same concentration, thus all the possible actions in the scope of the robot's agency refer to different combinations of the primary colors.

Now, suppose a customer arrives with a special requirement for the paint to also be diluted. Though this requirement is outside the design parameters and known agency of the paint-mixing robot—it does not understand the meaning of "dilution"—another agent somewhere in the MAS understands that dilution simply requires the addition of a certain amount of water. This second agent may not even necessarily be a paint-mixing robot.

The second agent possesses the *knowledge* of how to mix the desired type of paint, while the first agent possesses the physical *capability* to mix diluted paint (using the water hose intended for cleaning the drum), but not the knowledge of how to use this capability to produce diluted paint.

Using the ITM, the second agent can transfer the knowledge of how to dilute paint to the first agent, which can then carry out the necessary task. This is an example of Intelligence Transfer, specifically, Transfer of Agency. The same process could also be applied to resolve missing capabilities in the situational awareness and logical image of the agent.

To represent tasks in the ITM, we use component diagrams based on the notation developed in [47]. In our notation, a complex task such as mixing paint may be comprised of unique code, as well as calls to several self-contained, reusable component tasks across situational awareness, logical image, and agency scopes.

There is functionally no distinction between tasks and component tasks, which may be further decomposed ad infinitum. Under the ITM, each task can be transferred between agents in the MAS using Intelligence Transfer. In our notation, a red rectangle indicates agency, a blue circle indicates the logical image, and a green cut-off rectangle indicates situational awareness.

Figure 3 illustrates the use of a task decomposition diagram for the *Mix Paint* task and its component task dependencies. Table 2 shows the meaning of the different symbols used in the diagram.

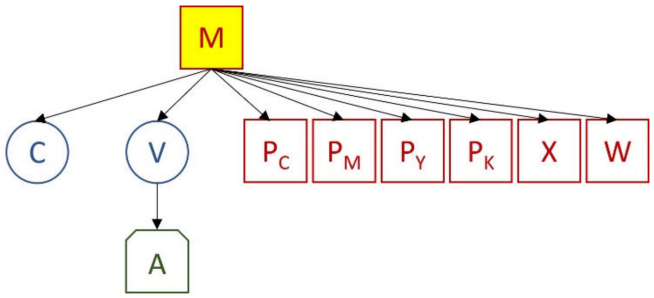

Figure 3. Task decomposition diagram for the paint-mixing robot agent showing the Mix Paint task (highlighted) and its component task dependencies.

This task decomposition diagram and its corresponding digital notation (we elected to use an adjacency matrix for this purpose) make it possible for agents to represent, modify, and transfer representations of the steps to accomplish a task independently of the code to actually carry them out, and agnostic of any distributed or co-operative task execution behavior in the MAS. Thus, the knowledge of how to execute a task is distributed across different agents in the MAS, facilitating improved task-based redundancy. It is this decoupling that forms the basis for the mechanism of Intelligence Transfer in the ITM.

Table 2. Component tasks for the paint-mixing robot example.

Symbol	Scope	Description
M	Agency	Mix paint
C	Logical Image	Calculate primary color breakdown
V	Logical Image	Verify paint can be mixed with available resources
A	Situational Awareness	Access paint-tank sensors
P_C	Agency	Add cyan paint
P_M	Agency	Add magenta paint
P_Y	Agency	Add yellow paint
P_K	Agency	Add black paint
X	Agency	Empty mixing bucket into customer's container. If no container is present, bucket contents are jettisoned
W	Agency	Add water to the mixing bucket
D	Agency	Dilute paint

When modeling tasks, we use *knowledge* to refer to the specific steps required to achieve the task and use the *capability* to refer to the availability of the component tasks which make up the task (i.e., the child nodes in the task decomposition diagram) at the local agent.

Decoupling the general *knowledge* of which steps are necessary to execute some tasks from the specific *capability* to actually execute them is particularly useful in MASs where the cost of spawning a new agent is low, but the cost of reconstructing information lost through non-redundant agent death is high. This cost differential typifies many software-based implementations of MASs, especially in AI, computer vision, IoT, and deep learning applications where the agents' behavior is evolved over a long time in response to stimuli.

Continuing the example, Figure 4 shows the task decomposition diagram of the *Dilute Paint* task. The difference between the two task decomposition diagrams is highlighted in yellow. The operations to be performed in the *Mix Paint* task are shown in pseudocode in Figure 5, and the operations of the *Dilute Paint* task are shown in Figure 6.

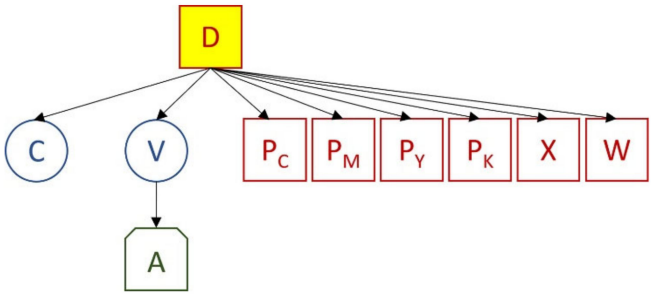

Figure 4. Task decomposition diagram for the paint-mixing robot agent showing the Dilute Paint task (highlighted) and its component task dependencies.

```
M: # Mix paint task
  C # Calculate color breakdown
  if V: # If paint is available
    PC # Add Cyan
    PM # Add Magenta
    PY # Add Yellow
    PK # Add Black
    X # Deliver the paint
    W # Wash the bucket
    X # Discard water
```

Figure 5. Pseudocode for the Mix Paint task (M).

```
D: # Dilute paint task
  C # Calculate color breakdown
  if V: # If paint is available
    P_C # Add Cyan
    P_M # Add Magenta
    P_Y # Add Yellow
    P_K # Add Black
    W # Dilute the paint with water
    X # Deliver the paint
    W # Wash the bucket
```

Figure 6. Pseudocode for the Dilute Paint task (D) with additional step highlighted.

Figures 5 and 6 show that, while the specific control flow of component tasks to be performed differs between M and D (highlighted in yellow in Figure 6), their task dependencies are the same. This is expressed graphically in Figures 3 and 4. Thus, an agent capable of executing M will be able to execute D, provided that the logic for D is acquired through Intelligence Transfer. In our example, the second agent may transfer the logic in D to the first agent, enabling it to provide the required diluted paint to the customer agent using the existing capabilities already available at the local agent.

The same Intelligence Transfer mechanism that was used to transfer agency-scope code between two agents in the MAS can likewise be applied to the situational awareness scope (where information about the environment is transferred) and logical image scope (where decision-making logic would be transferred). The distinction between the scopes reflects the different nature of the code being transferred from a cognitive intelligence perspective [51]. The purpose of this architectural distinction is to enable different acquisition/resolution strategies to be employed based on the cognitive purpose of the code being transferred, as we demonstrate in our candidate implementation discussed in Section 4.

4. Candidate Implementation of the ITM

A candidate implementation of the ITM was developed to demonstrate and evaluate its applicability to MAS design. This candidate implementation, named ItmPy, was developed in the Python 3 programming language and utilized an aspect-oriented software design (AOSD) to achieve the functionality of the ITM without introducing ITM-related code to the static flow of execution of the agents. This design was selected after experimentation with a purely object-oriented design because it provided significant benefits in terms of code complexity and abstraction of the code related to the Intelligence Transfer Model. Using ItmPy, we were able to successfully create a MAS capable of achieving tasks using the ITM which would not have otherwise been possible without the ITM.

4.1. Software Architecture of ItmPy

ItmPy is a software framework that provides the following support for MASs implementing the ITM:

1. An *ItmAgent* base class that provides a contract for the functionality required by the framework at the agent level;
2. An *ItmScope* base class in which the executable code for each of the situational awareness, logical image, and agency is located. The aspect weaver (implemented in the *__getattr__* method) matches pointcuts on missing attributes and methods in Scope instances;
3. An *ItmStrategy* base class that implements the Strategy design pattern to encapsulate the logic by which the ItmPy framework resolves references to missing attributes and methods;
4. An *ItmNetworkStack* class that handles network communication with other agents in the MAS, providing the practical implementation of the Intelligence Transfer process.

Figure 7 shows a simplified high-level software architecture of the ItmPy framework to illustrate the relationships between these classes. Figure 8 shows the aspect-oriented behavior of the ItmPy framework in response to the matching of a pointcut on a missing member of an *ItmScope*.

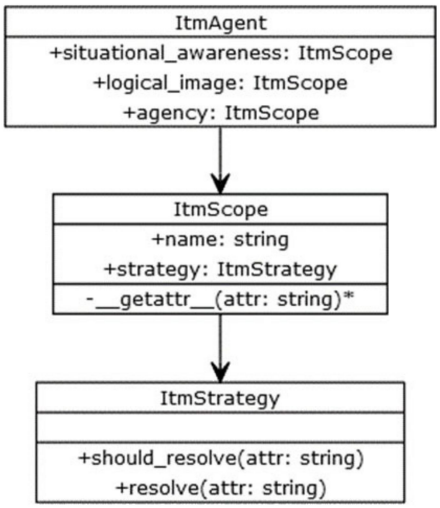

Figure 7. Simplified high-level architecture of ItmPy (an asterisk is used to indicate the method where the aspect weaving process takes place).

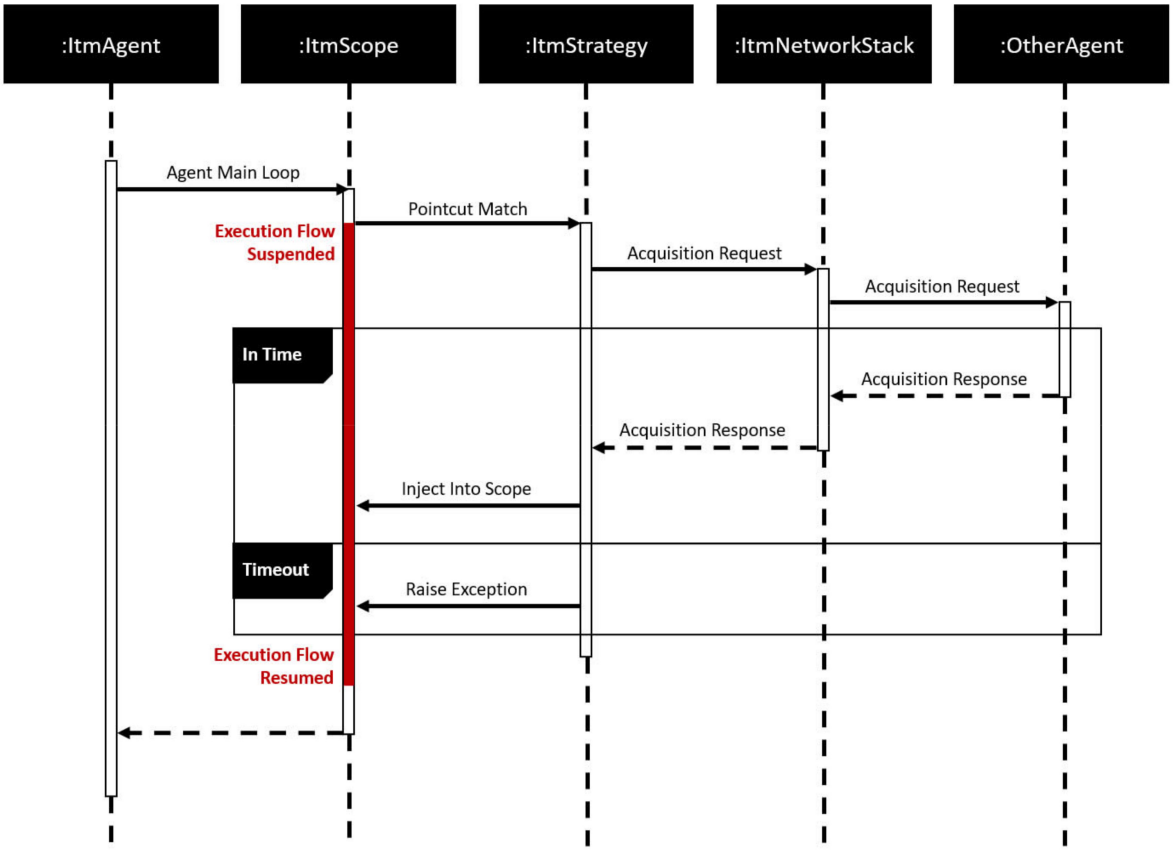

Figure 8. Aspect-weaving behavior of the ItmPy framework matching a pointcut on a missing member.

During the normal flow of execution of an *ItmAgent*, the agent's code accesses the domain-specific functionality in each of its *ItmScope*s to carry out its tasks. When an attempt is made to access or call a property or method in an *ItmScope* which does not

presently exist at the local agent (either by the agent's code or from another method in an *ItmScope*), a pointcut in the *ItmScope* class is matched and the aspect-oriented design of the ItmPy framework intervenes. The normal flow of execution is suspended, and the ItmPy framework uses *ItmScope*'s attached *ItmStrategy* to resolve the missing reference through inter-agent communication in the MAS.

The encapsulation of the resolution strategy in the *ItmStrategy* class enables different strategies to be switched at run-time, perhaps even themselves through inter-agent communication.

In our architecture, we utilized three scopes, one to represent each of the intelligent activities as per the ITM. Technologically, however, there is no limit as to the number of scopes an agent may have. This will enable the expansion and future development of the ITM and potentially alternative theories which may change the cognitive model of agent intelligence to organize the core intelligent activities differently while preserving the concept of Intelligence Transfer.

Similarly, by separating the three intelligence activities as per the ITM into separate scopes, we can use different resolution strategies for each intelligent activity, reflecting the difference in each scope's objectives which may be realized in terms of different methods of resolving missing information.

4.2. Inter-Agent Communications

Inter-agent communications in ItmPy are achieved through the use of a custom application-layer protocol, ITM Protocol (ITMP), implemented over User Datagram Protocol (UDP) and Internet Protocol (IP), as shown in Figure 9.

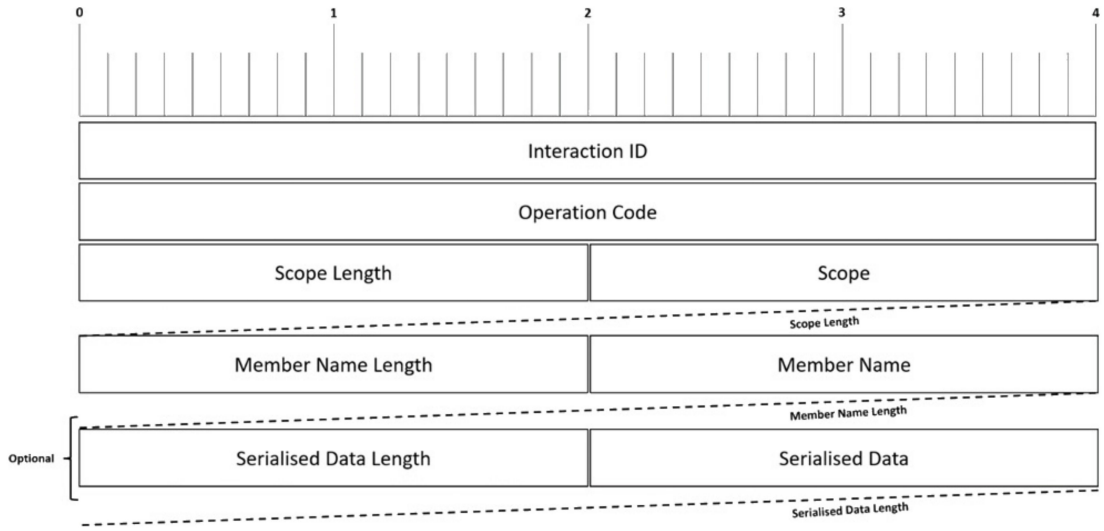

Figure 9. Bit-level packet diagram of the Intelligence Transfer Model Protocol (ITMP), as used in our candidate implementation.

In the ITM network stack, IP addresses were used to designate individual agents in the MAS, while the MAS itself was addressed using a classless inter-domain routing (CIDR) subnet. The subnet broadcast address was similarly used to address messages to all agents in the MAS. An arbitrary UDP port was selected for ITMP which was used by all agents in the MAS.

Intelligence Transfer functionality was implemented in the application layer in ITMP. In our implementation, we used the following fields (ITMP may be easily extended to accommodate additional information in alternative models):

1. Interaction ID—a 4-byte field that can be used to track requests and responses;
2. Operation code—a 4-byte field that represents the type of operation being conducted. In the candidate implementation, we implemented only member acquisition requests;

however, future research may expand this to include other operations to facilitate co-operative and delegative behavior;
3. Scope length—the length in bytes of the "scope" field including null-termination;
4. Scope—the name of the scope to which the knowledge being transferred belongs (situational awareness, logical image, agency);
5. Member name length—the length in bytes of the "member name" field, including null-termination;
6. Serialized data length—the length of the "serialized data" field;
7. Serialized data—the payload of the packet which contains the executable code or other information being transferred between agents.

In the design of ITMP, we provided only the minimal functionality required for the candidate implementation. We did not consider fragmentation, encryption, trust, integrity, quality of service, denial of service protections, and similar issues which were outside the scope of the research. However, these considerations could potentially be added in the future at the session and presentation layers.

4.3. Proof-of-Concept MAS

To test the candidate implementation, we developed a sample, proof-of-concept (POC) MAS using the ItmPy framework. The goal of the POC was to demonstrate a complete Intelligence Transfer process, as shown in Figure 8, from the normal execution flow of the agent, through to the matching of a pointcut on a missing member, resolution of the missing member, aspect-weaving [53–55] to inject the resolved member into the agent's codebase, and finally resumption of normal flow of execution.

A simple MAS of two agents was created to demonstrate the software paradigm of the ITM applied to the paint-mixing robots scenario described above. One agent represented the paint-mixing robot, and another represented a customer. Knowledge of the dilution methodology was implemented in the customer agent for simplicity but could have equally been implemented in a third agent as per the example. The method for diluting the paint relied on functionality in the situational awareness and logical images of a paint-mixing robot agent, functionality that was not available in the customer agent.

Thus, the MAS, if considered as a single entity, contained the knowledge of the paint available in the system (in the paint-mixing robot's situational awareness), how to determine the correct amounts of each primary color to mix (in the paint-mixing robot's logical image), and the knowledge of how to carry out a mixing action which included a dilution (in the customer's agency).

However, these were localized at different agents in the MAS in a manner that did not enable completion of the global task, namely, dilution of paint. Only by using Intelligence Transfer, was it possible for the paint-mixing robot to request the knowledge of how to dilute paint from the MAS, with the customer agent able to respond dynamically with the required knowledge. This was achieved through the aspect-weaving behavior of the ItmPy framework and the paint-mixing robot's Intelligence Transfer Strategy.

At the point where the paint-mixing robot made a call to *dilute_paint*, a method that did not exist in the agency scope of the local agent, the ItmPy framework suspended the normal flow of control and began executing the Intelligence Transfer functionality. ItmPy activated the paint-mixing robot's Intelligence Transfer Strategy which broadcast the request for the missing knowledge of paint dilution to the MAS as a whole.

The Intelligence Transfer process was observed using several tools which were developed as part of the research, including:

1. Trace output from the agents in the system;
2. A "MAS Network Observer" program, created to parse and display ITMP messages sniffed from the network through a raw socket in human-readable form;
3. A custom Wireshark dissector for ITMP, which enabled both live observation and dissection of captured packets in offline analysis.

The trace output from the paint-mixing robot agent is shown in Figure 10, with the intervention of ItmPy's aspect weaving functionality highlighted by a red line. Figure 11 shows the output from the MAS Network Observer of the ITMP network communication which took place as a result of this intervention.

```
magenta changed by 0.5.
Tanks now hold {'cyan': 0.5, 'magenta': 0.5, 'yellow': 1.0, 'black': 1.0}
yellow changed by 0.5.
Tanks now hold {'cyan': 0.5, 'magenta': 0.5, 'yellow': 0.5, 'black': 1.0}
black changed by 0.5.
Tanks now hold {'cyan': 0.5, 'magenta': 0.5, 'yellow': 0.5, 'black': 0.5}
Bucket emptied
5L of water added to bucket
Bucket emptied
Attempted to access unknown attribute agency.dilute_paint
Resolved agency.dilute_paint
cyan changed by 0.5.
Tanks now hold {'cyan': 0.0, 'magenta': 0.5, 'yellow': 0.5, 'black': 0.5}
magenta changed by 0.5.
Tanks now hold {'cyan': 0.0, 'magenta': 0.0, 'yellow': 0.5, 'black': 0.5}
yellow changed by 0.5.
Tanks now hold {'cyan': 0.0, 'magenta': 0.0, 'yellow': 0.0, 'black': 0.5}
black changed by 0.5.
Tanks now hold {'cyan': 0.0, 'magenta': 0.0, 'yellow': 0.0, 'black': 0.0}
4L of water added to bucket
Bucket emptied
5L of water added to bucket
Bucket emptied
```

Figure 10. Trace output from the paint-mixing robot showing the intervention of the ItmPy aspect weaver (indicated by a red line).

```
Starting network server for agent MAS Observer 127.0.0.99:33108..
MAS Observer network server up.
MAS Observer is starting the agent main loop...
127.0.0.1 > 127.255.255.255: ACQUIRE agency.dilute_paint
127.0.0.2 > 127.0.0.1: RESPONSE [ACQUIRE agency.dilute_paint]
```

Figure 11. Output of the Multi-Agent System (MAS) Network Observer showing the Intelligence Transfer Model Protocol (ITMP) communication taking place from the advice executing on the pointcut indicated by the red line in Figure 10.

Having successfully resolved the call to *dilute_paint* through Intelligence Transfer, the ItmPy framework dynamically added the method to the agency scope of the agent before resuming the normal flow of control, shown in the trace output in Figure 10.

In this scenario, the MAS began execution in a failure state with respect to the dilution task. Over the course of the system's evolution, through the ITM's functionality, the MAS was able to transition from this unacceptable failure state to an acceptable state wherein it was able to execute the task.

The entire Intelligence Transfer process in the POC scenario was completely transparent to the paint-mixing robot agent by virtue of ItmPy's aspect-oriented design. No special code was required in the agents to deal with the Intelligence Transfer scenario. Rather, specific functionality telling ItmPy what strategy should be employed to resolve calls to missing methods was localized in the Intelligence Transfer Strategy class, enabling reuse across multiple agents. Similarly, the flow of control of the customer agent was not affected by the Intelligence Transfer process, with the functionality taking place in the ItmPy framework.

5. Modeling the Fault Tolerance and Reliability of ITM-Based MASs

In this section, we propose a novel way of modeling fault tolerance and reliability in the context of MASs and ITM. We begin by briefly reviewing existing approaches to fault tolerance at the individual agent level. Then, we develop a model to describe the static task-based redundancy of a MAS using a state machine. We then further develop the model to describe the dynamic task-based redundancy of a MAS, based on the novel proposition of a *latent acceptable state*. Next, we present a mathematical model for quantifying the improvement of fault tolerance using the ITM. Finally, we apply and extend this model to quantify the improvement in system reliability.

5.1. Fault Tolerance at the Individual Agent Level

Established methodologies exist for modeling fault tolerance in localized systems (some system which is not part of a MAS) that consider the fault status of the local system as a state machine [48,56,57]. The specific notation and emphasis vary between researchers but the fundamental concepts are the same. Under these methodologies, the local system may be in one of several different states, depending on the design of the local system, each of which we may classify as one of the following classes:

1. Fully operational;
2. Acceptable degraded operation;
3. Safe shutdown state;
4. Failure state.

Of these states, only the "failure" state is generally considered unacceptable. However, not all systems may necessarily have states in each of the categories. Furthermore, the environmental state of the system may be such that a safe shutdown is not possible. For example, an Unmanned Aerial Vehicle (UAV) control system may not have a safe shutdown state while airborne, as shutting down the control system would result in a crash. Some methodologies also map states in which the system is in a *latent fault state*, referring to a state in which an uncovered fault is yet to manifest as a failure [23,49,50].

Such models have partial applicability when considering MASs, as a MAS can be considered a system of different components. We propose to extend this model and adapt it to model MASs in two stages. In the first stage, we consider a general method of modeling fault tolerance in MASs using a task-based redundancy model. Then, in the second stage, we introduce a new type of state, a *latent acceptable state*, to describe the novel fault tolerance contributions of the ITM.

5.2. Static Task-Based Redundancy Model

Our analysis will consider the MAS's level of fault tolerance from the perspective of each individual task. For example, suppose that the MAS is expected to be able to complete two tasks, T_1 and T_2. If at some time, no agents are able to execute T_1, then the MAS may be said to be in a failure state with respect to T_1. However, agents in the MAS may be able to execute T_2. Thus, the MAS shall be in an acceptable state with respect to T_2.

We shall begin by analyzing MASs without Intelligence Transfer. When analyzing the task-based redundancy of a non-ITM-based MAS with respect to some tasks, we propose the following model, which we shall refer to as a *Static Task-Based Redundancy Model* (Static TBRM). We create a state machine to describe the task-based redundancy of a MAS at some time. Each state shall be defined in terms of four quantities reflecting the possible configuration of agents in the MAS (as shown in Table 3 and Figure 12):

1. Number of agents with both knowledge and capability to carry out the task (denoted as A);
2. Number of agents with the capability but not the knowledge to carry out the task (denoted as C);
3. Number of agents with the knowledge but not the capability to carry out the task (denoted as K);

4. Number of agents with neither the knowledge nor the capability to carry out the task (denoted as X).

Table 3. Definitions of agent classes in the Static Task-Based Redundancy Model (Static TBRM).

Agent Class	Knowledge	Capability
A	Yes	Yes
C	No	Yes
K	Yes	No
X	No	No

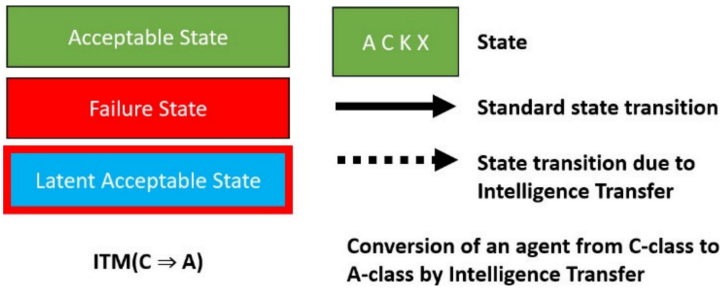

Figure 12. Legend for Static and Dynamic TBRM state diagrams.

Each state shall be defined to be either an *acceptable state*, where there exists at least one agent capable of executing the task, or a *failure state*, where there do not exist any agents capable of executing the task.

In our analysis, *birth* refers to an agent joining the MAS, *death* refers to an agent leaving the MAS, *recovery* refers to an agent, previously unable to execute the task, recovering this ability, and *failure* refers to an agent losing the ability to execute the task, perhaps due to a component failure on the agent's device or some change in its environment. Transitions between the states are defined by the following events.:

1. Birth/recovery of an A, C, K, or X class agent (denoted as B_A, B_C, B_K, and B_X, respectively);
2. Death/failure of an A, C, K, or X class agent (denoted as D_A, D_C, D_K, and D_X, respectively).

For simplicity of the examples, we shall treat birth and recovery as functionally equivalent, likewise, death and failure. We also exclude transitions arising from the birth/recovery of an A-class agent and the death/failure of an X-class agent and do not consider latent/uncovered fault states.

In this paper, we represent TBRMs graphically as state machine diagrams according to the legend given in Figure 12. Figure 13 shows a simplified Static TBRM for some tasks in a MAS consisting of three A-class and one K-class agent (only states and transitions which are significant to the analysis are shown). For the purposes of the example, we shall consider that in this particular MAS, agent recovery is not possible and thus any agents removed from the system are removed permanently. We also consider that the task being modeled depends on some learning outcome, which, after MAS genesis, can only be acquired from other agents already in the MAS. Thus, while new agents may join the MAS, they cannot join as either A-class or K-class agents. Similarly, if all agents are removed, this information cannot be recovered, and the MAS will be in an irrecoverable failure state.

From the diagram in Figure 13, we can see that as A-class agents are removed from the MAS (D_A transitions), the MAS's task-based redundancy (measured by the number of A-class agents in the MAS) is reduced. Similarly, the K-class agent has no impact on the redundancy model of the MAS. Once the MAS enters a failure state (state [0 0 1 0] in Figure 13), it cannot return to an acceptable state. This property is typical of all non-ITM-based MASs lacking a mechanism by which an agent may acquire the necessary knowledge to change from a C-class to an A-class agent.

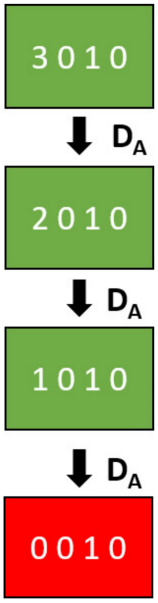

Figure 13. Static TBRM for the example task in a MAS not implementing the ITM.

5.3. Dynamic Task-Based Redundancy Model

The Static TBRM provides a foundation for defining and quantifying the level of fault tolerance in a MAS with respect to each task. Under the conditions we set for the example, agent birth of any class (except A, which is excluded) is inconsequential to the fault tolerance of the MAS. The lack of any mechanism to change an agent's class from C to A renders the TBRM static, functionally equivalent to existing fault tolerance models such as the model in [23].

However, by introducing the concept of Intelligence Transfer to the MAS, new states and transitions (see Figure 14) can be created by virtue of the capacity of agents to improve the set of tasks they can execute by changing from C-class to A-class agents, thereby improving the MAS's task-based redundancy.

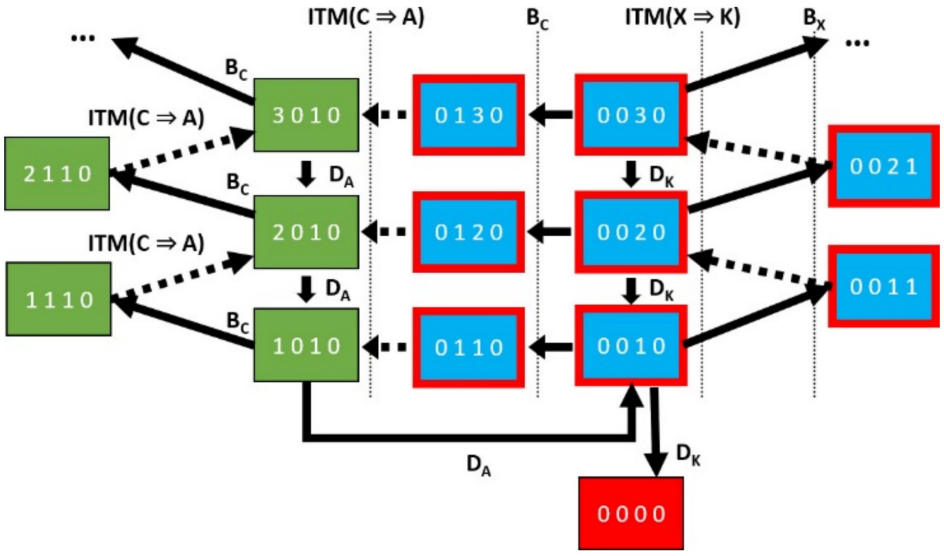

Figure 14. Simplified Dynamic TBRM for the example task in a MAS implementing the ITM.

We thus propose to extend the Static TBRM to a Dynamic TBRM to incorporate the capabilities of the ITM into the fault tolerance model. We term these TBRMs *dynamic* because the potential exists for the MAS to transition between failure and acceptable states, and to

improve its level of task-based redundancy through Intelligence Transfer irrespective of the capacity for agent recovery.

The key novel mechanism in a Dynamic TBRM is a new type of state which we shall call a *latent acceptable state*. In a latent acceptable state, the MAS is considered to be in a failure state with respect to some task (insofar as there are no A-class agents in the MAS, thus no agents exist that have both the knowledge and capability to carry out the task), but, using Intelligence Transfer, new state transitions could be defined to return the system from the latent acceptable state to an acceptable state.

The latent acceptable state is thus a distinct type of failure state because the MAS is incapable of executing the task, but, contrary to an unrecoverable failure state, the potential exists for the MAS to transition back to an acceptable state through the mechanism of Intelligence Transfer.

Specifically, this can be achieved if there are one or more K-class agents in the MAS with the knowledge but not the capability to carry out the task. Supposing that, at some future time, a C-class agent joins the MAS with the necessary capability but not the knowledge to carry out the task, the existing K-class agents in the MAS will be able to transfer the necessary knowledge to the new C-class agent through Intelligence Transfer, thereby changing the C-class agent to an A-class agent and returning the system to an acceptable state.

We shall refer to this process as Intelligence Buffering because the K-class agents in the MAS buffer the requisite knowledge of the intelligent activity until an agent with suitable capabilities joins the MAS. This is shown in Figure 14. Only when all A-class and K-class agents are removed from the system does it finally enter an unrecoverable failure state.

The implications of such a model for improving fault tolerance are wide-ranging because of the model's generality. At its core, a MAS based on the ITM utilizes every possible opportunity to improve its redundancy while it is operating in an acceptable state, and preserves its ability to recover to acceptable states even when it is operating in a failure state.

As the Dynamic TBRM in Figure 14 shows, there are essentially two redundancy processes taking place by virtue of the ability to engage in Intelligence Transfer between agents. When an ITM-based MAS is in an acceptable state, it takes advantage of the birth of new C-class agents to improve its level of task-based redundancy by converting them to A-class agents through Intelligence Transfer. In both the acceptable and failure states (only the latter is shown in Figure 14 for clarity), the ITM-based MAS takes advantage of the birth of new X-class agents to improve its ability to recover from a failure to an acceptable state by converting them into K-class agents through Intelligence Transfer.

The ITM-based MAS is thus found to be more resilient than a non-ITM-based MAS both because of its ability to improve its task-based redundancy and because of its ability to transition from a failure to an acceptable state through redundant latent acceptable states.

5.4. Quantitative Modeling of the Effects of Intelligence Transfer

In the preceding section, we modeled the fault tolerance of a MAS by considering its TBRM in terms of the discrete effects of adding and removing individual agents to the MAS. In this section, we propose a methodology for quantitatively modeling the effect that the Intelligence Transfer process has on the fault tolerance and reliability of a MAS. The goals of this model are to:

1. Provide a quantitative representation of the Dynamic TBRM of the MAS;
2. Provide a mathematical process by which the evolution of an ITM-based MAS can be studied;
3. Provide a metric by which different ITM-based MASs can be compared in terms of fault tolerance and reliability.

We shall define the following process by which the evolution of a general ITM-based MAS can be modeled from an arbitrary state to the maximally possible level of task-based redundancy:

1. Creation of an Existing Capacity Matrix (ECM);
2. Creation of a Dependency Matrix (DM);
3. Establishment of a baseline;
4. Creation of a Transfer Matrix (TM);
5. Creation of a Potential Capability Matrix (PCM).

In steps 1–3 of the analysis, we consider the capabilities of the agents in the MAS as they are initially. This gives us a baseline that we can use to determine what improvements to fault tolerance can be made through the process of Intelligence Transfer. In step 4, we identify which intelligent activities can be transferred between agents in the system to achieve the maximal level of task-based redundancy in the MAS as a whole. This culminates in step 5 which gives a representation of the maximal task-based redundancy of the MAS in the same dimensionality as the initial ECM, thus facilitating a comparison between the two and quantification of the improvement.

5.4.1. Creation of an Existing Capability Matrix (ECM)

The first step in modeling the current state of the MAS is to define the set of possible tasks and to create a mapping between the agents in the MAS and the tasks for which they have the knowledge to execute. We shall refer to this mapping as the ECM.

Let us consider an example MAS with nine possible tasks and four agents. We can thus define a matrix with nine columns and four rows to represent the mapping of agents to tasks. In this matrix, a cell with a value of "1" indicates that the agent represented by the cell's row number has knowledge of the steps required to execute the task represented by the cell's column number. A value of "0" in a cell indicates that the agent is not capable of executing the corresponding task.

For the following sections, we shall define an example ECM to represent a MAS with the mapping of agents to tasks given in Figure 15.

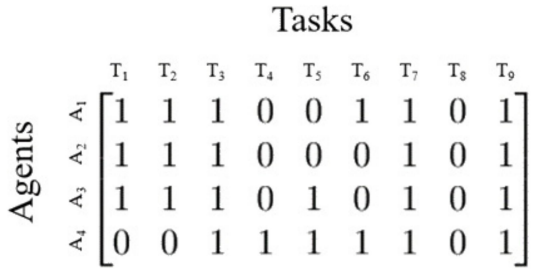

Figure 15. Existing Capability Matrix (ECM) for the example MAS.

5.4.2. Creation of a Dependency Matrix (DM)

Having created a representation of the mapping of agents to tasks in the ECM, we must now consider the task decomposition diagram representing the relationship between tasks and their dependencies as discussed earlier in Section 3. For this purpose, we shall define a new matrix, the DM, which shall be an $n \times n$ adjacency matrix representing the dependency relationships between the tasks in the MAS. Both dimensions of the DM are given by the number of vertices in the task decomposition diagram. The values of the cells are given by the edges.

Thus, the DM in our example will be a 9×9 matrix where a value of "1" in some cell $DM_{i,j}$ indicates that the task T_i depends on the task T_j. In our analysis, we shall assume that the DM is a transitive closure such that all transitive relations between dependent tasks have been resolved, i.e., dependencies of the dependencies of some task T have all been marked as dependencies of task T, etc. We shall also explicitly exclude circular dependencies because these would, in any case, mean it is not possible to actually execute the task due to infinite recursion in the executing agent.

We continue the example with the task decomposition diagram shown in Figure 16. Given this task decomposition diagram, the DM for the example is shown in Figure 17.

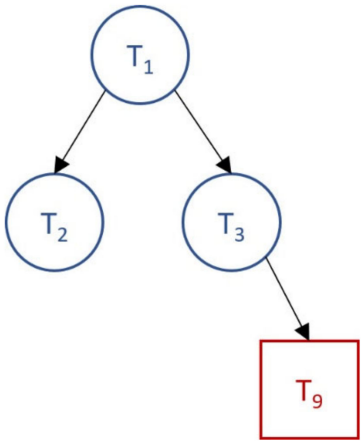

Figure 16. Task decomposition diagram for the example MAS. Tasks not shown are assumed to have no dependencies.

$$DM = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 17. Dependency Matrix (DM) for the example MAS.

5.4.3. Establishment of a Baseline Redundancy Matrix (B)

In the third step of the analysis, we create a representation of the baseline task-based redundancy of the MAS by considering which agents have both the knowledge and capability to execute a task (i.e., how many A-class agents exist for each task) using the following process.

In the ECM, we know whether the given agent possesses the knowledge of a task. To ascertain whether the agent possesses the practical capability to execute it, we must also consider whether the agent possesses knowledge of all the task's dependencies. This can be accomplished by comparing an agent's row vector in the ECM with the task's row vector in the DM.

By definition, leaf nodes have no dependencies; thus, there is no distinction between knowledge and capability—knowledge of a task implies the capability to execute it.

Having completed this process for each task in the MAS, we can create a Baseline Redundancy Matrix (B), defined as a row vector with one column for each task. The value of each cell is the number of A-class agents in the MAS having both the knowledge and capability to execute the corresponding task. We shall use matrix B, which represents the task-based redundancy in the MAS before Intelligence Transfer, as a control measurement with which we will be able to compare a similar measurement taken after Intelligence Transfer. Continuing the example, B would be as given in Figure 18.

$$\mathrm{B} = \begin{array}{c} \begin{matrix} \mathrm{T}_1 & \mathrm{T}_2 & \mathrm{T}_3 & \mathrm{T}_4 & \mathrm{T}_5 & \mathrm{T}_6 & \mathrm{T}_7 & \mathrm{T}_8 & \mathrm{T}_9 \end{matrix} \\ \begin{bmatrix} 3 & 3 & 4 & 1 & 2 & 2 & 4 & 0 & 4 \end{bmatrix} \end{array}$$

Figure 18. Baseline Redundancy Matrix (B) for the example.

5.4.4. Creation of a Transfer Matrix (TM)

Before we can begin to model the effect of Intelligence Transfer on the MAS represented with a given ECM and DM, there is one more consideration to address: the transferability of tasks between agents.

It is not necessarily the case that any task may be freely transferred between agents in the MAS, particularly when considering the heterogeneity that may be exhibited by the agents. The issue of transferability follows from the property of leaf nodes whereby knowledge implies capability. Leaf nodes represent the tasks of least abstraction, sitting at the boundary between the ITM and the local agent's (possibly physical) capabilities, thus it may not make sense to transfer these tasks between agents who do not have similar local configurations for the particular task.

For example, if we reconsider the paint-mixing robot example from earlier in the paper, it may not be possible to transfer task W between two agents in the system if the receiving agent does not have a water hose. Thus, it is necessary to define a Transfer Matrix (TM) which will hold the representation of which tasks may be transferred between agents and which tasks cannot because of some coupling to a local capability at the agent. In our model, we leave the decision over the specific nature of this coupling, and whether it is possible to transfer the task, at the agent level.

We shall define the TM as a matrix having the same dimensions as the ECM, where a "1" in a cell indicates that the corresponding agent can transfer knowledge of the corresponding task, whereas a "0" indicates that transfer is not possible. Let us continue our example with the TM given in Figure 19.

$$\begin{array}{c} \text{Tasks} \\ \begin{matrix} & \mathrm{T}_1 & \mathrm{T}_2 & \mathrm{T}_3 & \mathrm{T}_4 & \mathrm{T}_5 & \mathrm{T}_6 & \mathrm{T}_7 & \mathrm{T}_8 & \mathrm{T}_9 \end{matrix} \\ \text{Agents} \begin{matrix} \mathrm{A}_1 \\ \mathrm{A}_2 \\ \mathrm{A}_3 \\ \mathrm{A}_4 \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 19. Transfer Matrix (TM) for the example.

5.4.5. Creation of a Potential Capability Matrix (PCM)

Having defined a representation of the requisite information to model the Intelligence Transfer process, the final step in the analysis is to carry out the process on the representation of the current state of the MAS. The goal of this step is to reach a representation, compatible with the ECM, of the state of the MAS after all possible Intelligence Transfer activities have occurred. This will facilitate a comparison with the control (ECM) measurement to quantify the improvement in fault tolerance and reliability.

To achieve this goal, we shall define a new matrix, the Potential Capability Matrix (PCM), with the same dimensions as the ECM. The contextual meaning of the cells in the PCM is the same as for the ECM, differing only in that they represent the state of the MAS after the Intelligence Transfer process has run its course.

While acquiring the knowledge for additional tasks using Intelligence Transfer, the local agent must also ensure the dependencies of the desired tasks are satisfied locally. Thus, a recursive process is necessary: where the agent does not have one of the dependencies available it must also acquire that dependency. Fortunately, the DM gives us part of this

process by virtue of its definition as the transitive closure of the task dependencies. All that remains is, for each agent, to enumerate all the possible tasks in the system to see if it can satisfy the dependencies locally or through Intelligence Transfer (given by the TM).

Continuing the example, the PCM would be as given in Figure 20.

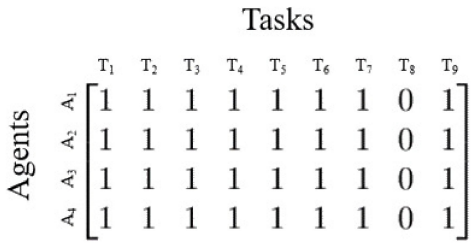

Figure 20. Potential Capability Matrix (PCM) for the example.

5.4.6. Quantifying the Improvement of Fault Tolerance through the ITM

The improvement in fault tolerance as a result of Intelligence Transfer can now be quantified by computing a Potential Redundancy Matrix (B*) for the PCM in the same manner that matrix B was calculated for the ECM. B* will contain a representation of the task-based redundancy in the MAS due to Intelligence Transfer. Subtracting matrix B from B* gives the improvement due to the ITM.

The matrix B* for the example would be as given in Figure 21. Subtracting B, we can find the improvement in fault tolerance for any desired task. For example, the task-based redundancy of T_1 has improved from 3 agents to 4 agents. We can thus create two TBRM diagrams: the first representing the state of the MAS before Intelligence Transfer and the second representing the MAS after Intelligence Transfer. Leaving aside, for the moment, the latent acceptable states which give the mechanism by which the improvement is achieved, we obtain the Static TBRMs as given in Figure 22 ("X" is used for do-not-care parameters, as we are comparing only the task-based redundancy given by the number of A-class agents).

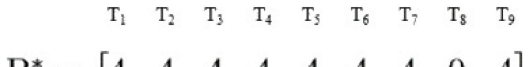

Figure 21. Potential Redundancy Matrix (B*) for the example.

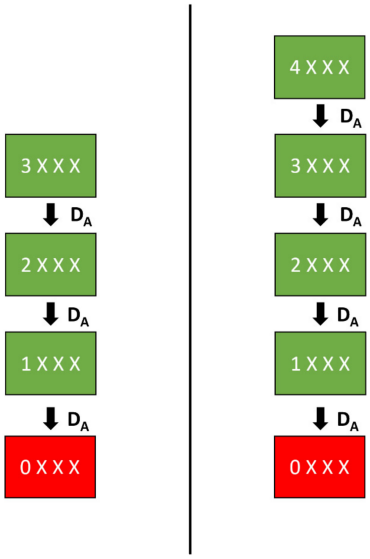

Figure 22. Static TBRMs for T_1 before (**left**) and after (**right**) Intelligence Transfer showing the creation of a new acceptable state in the MAS' fault tolerance model.

5.5. Modeling Reliability at the MAS Level

The previous section formalized the Intelligence Transfer process in a matrix-based mathematical model which can be used to quantify the level of fault tolerance improvement in MASs implementing the ITM. This model was used to show that through the mechanism of latent acceptable states, ITM-based MASs create new acceptable states in their fault tolerance model. In this section, we shall apply the same process (from ECM to PCM) in the context of reliability modeling.

5.5.1. Background to the Reliability Model

We define the reliability of some component, C, to be the probability that it will perform its intended function up to some time t. This can be defined mathematically as the function $R_C(t)$, where t represents some point in time up to which the component must operate as intended and τ represents the time at which failure occurs. The reliability function for a component with respect to time is given in (1).

$$R_C(t) = \Pr(\tau > t) \tag{1}$$

Generally, the probability that the component fails at a given time is represented by a probability density function, $f(x)$. Thus, the reliability function can be expressed as an integral of the failure probability density function as in (2).

$$R_C(t) = \int_t^\infty f(x)\, dx \tag{2}$$

We also can define a function $F(t)$ which represents the probability that a component fails before time t. The relationship between the two functions is given in (3).

$$R_C(t) = 1 - F_C(t) \tag{3}$$

When considering a system comprised of more than one component, it is possible to define a reliability function for the entire system, $R(t)$, and a corresponding failure function, $F(t)$.

Given a group of equations $F_1 \ldots F_n$ representing the failure functions of each of the components in the system, it is possible to calculate the reliability of the entire system, $R(t)$, by using a combinatorial model that reflects the series/parallel design of the components in the system [23]. For the purposes of our analysis, we shall consider an agent in the Intelligence Transfer Model as a component.

For a series design, the failure of any one component results in the failure of the system as a whole. Thus, we can calculate $R(t)$ to be the product of the reliability functions of the components [23]. For a parallel design, only the failure of all components results in the failure of the system as a whole. Thus, the failure function of the system, $F(t)$ is found to be the product of the failure functions of the components [23]. We can then find the reliability of the system as a whole using (3). Where a system comprises combinations of series and parallel subsystems, we simply use this process on each of the smallest subsystems to reduce them to a single function and then repeat the process at successively higher levels of the system.

5.5.2. Applying the Reliability Engineering Methodology to the ITM

We define *reliability* for a MAS as the probability that the MAS, as a whole, can successfully execute some task T during the period of time from the genesis of the system to some time t.

We assume agents to be independent, i.e., the failure of an agent is an independent event and each agent has an independent reliability function. Thus, from a reliability modeling perspective, for a given task T, the agents in the system are arranged in parallel.

This is shown in Figure 23, with $A_1 \ldots A_n$ representing the agents with the capability to execute the task.

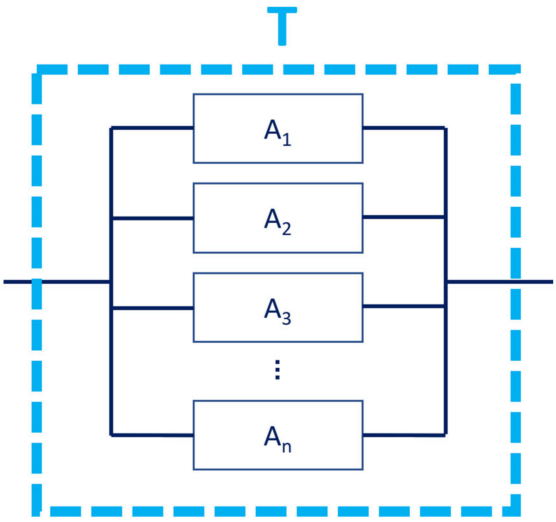

Figure 23. Reliability diagram for task T with respect to agents in the MAS.

We now continue the analysis of the example MAS in the previous section in terms of reliability. We shall begin by defining a Failure Matrix (FM), by replacing each element in each row of the ECM/PCM with the failure function of the agent represented by the row. Where an agent does not have the capability to carry out a task at all, that element shall be considered to always be non-functional and thus will be given the value "1" in the FM. The FM of the example MAS is given in Figure 24.

Tasks

$$\text{Agents} \begin{bmatrix} F_1(t) & F_1(t) & F_1(t) & 1 & 1 & F_1(t) & F_1(t) & 1 & F_1(t) \\ F_2(t) & F_2(t) & F_2(t) & 1 & 1 & 1 & F_2(t) & 1 & F_2(t) \\ F_3(t) & F_3(t) & F_3(t) & 1 & F_3(t) & 1 & F_3(t) & 1 & F_3(t) \\ 1 & 1 & F_4(t) & F_4(t) & F_4(t) & F_4(t) & F_4(t) & 1 & F_4(t) \end{bmatrix}$$

Figure 24. Failure Matrix (FM) of the example MAS.

Next, we can find the failure functions of the MAS for each task T_n by taking the product of all the elements in each column vector. This gives us a one-dimensional matrix (which we shall call the System Failure Matrix or SFM) where the nth element represents the failure function of the MAS, for task T_n. The SFM for the example MAS is shown in Figure 25.

Tasks

$$\text{SFM} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 \\ F_1F_2F_3 & F_1F_2F_3 & F_1F_2F_3F_4 & F_4 & F_3F_4 & F_1F_4 & F_1F_2F_3F_4 & 1 & F_1F_2F_3F_4 \end{bmatrix}$$

Figure 25. System Failure Matrix (SFM) of the example MAS.

Finally, using (3), we can convert the SFM to a System Reliability Matrix (SRM), shown in Figure 26. The resultant System Reliability Functions for each of the elements in the SRM are given in Figure 27.

Tasks

$$\text{SRM} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 \\ R_{T1}(t) & R_{T2}(t) & R_{T3}(t) & R_{T4}(t) & R_{T5}(t) & R_{T6}(t) & R_{T7}(t) & R_{T8}(t) & R_{T9}(t) \end{bmatrix}$$

Figure 26. System Reliability Matrix (SRM) of the example MAS.

$$
\begin{aligned}
R_{T1}(t) &= 1 - F_1F_2F_3 \\
R_{T2}(t) &= 1 - F_1F_2F_3 \\
R_{T3}(t) &= 1 - F_1F_2F_3F_4 \\
R_{T4}(t) &= 1 - F_4 \\
R_{T5}(t) &= 1 - F_3F_4 \\
R_{T6}(t) &= 1 - F_1F_4 \\
R_{T7}(t) &= 1 - F_1F_2F_3F_4 \\
R_{T8}(t) &= 1 - 1 = 0 \\
R_{T9}(t) &= 1 - F_1F_2F_3F_4
\end{aligned}
$$

Figure 27. System Reliability Functions for each element in the SRM of the example MAS.

Thus, given the failure functions for the agents in a MAS, we can quantitatively compute the reliability of the MAS with respect to some task T_n. We may also, given applicability in the MAS's problem domain, define a Composite System Reliability Function to compute the overall reliability of the MAS.

Supposing that the MAS is required to be able to complete all possible tasks in order for it to be in an acceptable state, the Composite System Reliability Function would be as in (4). Other configurations of MASs may use a different combination of series and parallel arrangements of tasks to arrive at a different Composite System Reliability Function as per the applicable problem domain of the specific MAS. As this will necessarily be application-dependent, for our analysis we consider reliability only at the task level.

$$R(t) = \prod_{i=1}^{n} R_{T_i} \tag{4}$$

5.5.3. Quantifying the Improvement of Reliability through Intelligence Transfer

Having established a reliability function, $R_T(t)$, for each task in the MAS, we can now use this methodology to quantify the improvement in this reliability function due to the ITM. This is achieved by applying the methodology to both the ECM and PCM states of the MAS and comparing the resultant reliability functions for the task.

We have shown that a MAS implementing the ITM can create new acceptable states in the fault tolerance model through the conversion of C-class agents to A-class agents. This process creates additional task-based redundancy by increasing the number of A-class agents in the system. This increased redundancy manifests in the SRM through the addition of A-class agents in parallel, thus, the reliability function for the task improves (assuming the new agents do not suffer an out-of-box failure and that the Intelligence Transfer process is carried out to its full potential). In any case, the reliability shall be at least as good as in the ECM state.

We can thus define a metric, the Reliability Improvement Factor, $\lambda(t)$, to compare the reliability of the MAS in the pre- and post-Intelligence Transfer (the ECM and PCM) states. This is given in (5).

$$\lambda(t) = \frac{R(t)}{R_0(t)} \tag{5}$$

6. Experimental Validation of the ITM

Thus far we have proposed the ITM and considered how it could theoretically improve both the fault tolerance and reliability of MASs. In this section, we present the results of three empirical studies which used the matrix-based model presented in Section 5 to practically simulate the effect that implementing the ITM had on the reliability and fault tolerance of samples of randomly generated MASs.

6.1. Summary of Experimental Method

We began by testing in idealized conditions, progressively modifying the criteria for the initial conditions of the MASs to more closely resemble how a MAS would be expected to behave in a real-world, practical implementation.

In the first scenario, we tested the outcomes for ITM-based MASs when the initial conditions were selected for an acceptable initial state. This enabled us to study the improvements, through Intelligence Transfer, to fault tolerance and reliability, based only on improved task-based redundancy.

In the second scenario, the acceptability condition was removed, and MASs were permitted to begin in an initial failure state. This enabled us to observe the effect of latent acceptable states through which MASs could move from a failure state to an acceptable state.

In the third scenario, we also enabled the MASs to begin in an initial failure state but constrained their TMs to prevent the transfer of leaf nodes. This was performed to more closely resemble the expected real-world conditions, where leaf nodes act as the interface between the ITM and the local agent.

6.1.1. Benchmark/Control Group

In each of the empirical studies (Scenarios 1, 2, and 3), the simulated MASs were divided into two groups: an experimental group, implementing the ITM according to the scenario configuration, and a control group, representing the benchmark against which the ITM MASs were measured.

The different MAS configurations used may be classified according to whether they implement task-based, or agent-based redundancy, as in Figure 28. The architecture of their software implementation is shown in Figure 29.

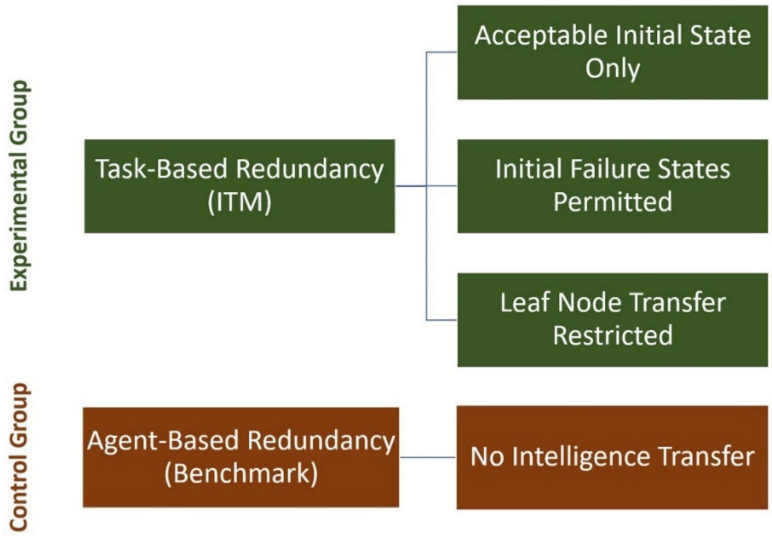

Figure 28. Selection of the experimental and benchmark/control MAS configurations classified as either task-based or agent-based redundancy.

The experimental (ITM) group used the dynamic task-based redundancy approach proposed in this paper. As agent-based redundancy models are widely used in the literature (see Section 2), we selected and implemented an agent-based redundancy MAS configuration as the benchmark against which the ITM configurations were tested.

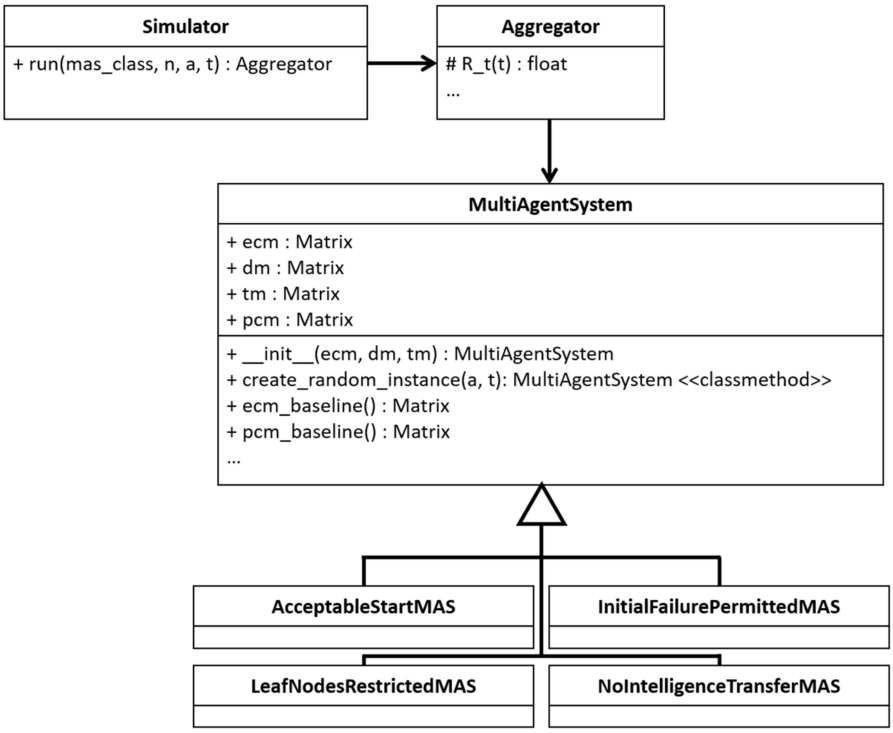

Figure 29. High-level system architecture of the Large-Scale Simulation Environment (LSSE).

In each individual test, a control MAS and an experimental MAS were spawned with the same agent-based redundancy configuration. Then, the experimental (ITM) MAS was permitted to engage in Intelligence Transfer, as per its configuration (acceptable initial state only, initial failure states permitted, or leaf node transfer restricted). Any improvements to the reliability and fault tolerance of the ITM MAS were recorded and compared with the measurement for the benchmark/control group.

6.1.2. The Simulation Environment

To practically test the outcomes of ITM-based MASs required a testing environment capable of generating large numbers of different MASs on-the-fly, running them through their system evolution, and measuring the observed fault tolerance and reliability improvements.

To this end, we developed an open-source tool, the Large-Scale Simulation Environment (LSSE), written in Python 3 using a multi-threaded, object-oriented design. The key components of LSSE are the *Simulator*, *Aggregator,* and *MultiAgentSystem* classes. The high-level system architecture of LSSE is shown in Figure 29.

The *Simulator* class is responsible for the multi-threaded generation of candidate MASs, inter-thread communication, synchronization, and progress monitoring. It provides an API for the generation of MASs which facilitates queueing and automatic execution of experiments without the need for manual intervention for experiments that run for extended periods of time.

The *Aggregator* class is responsible for measuring the MASs over the course of their evolution and condensing the results of the experiments into aggregate metrics. It also generates graphs for displaying reliability modeling. The *Aggregator* class contains the method which implements the reliability function for agents in the MAS.

The *MultiAgentSystem* class represents the MASs, maintaining their state over the course of their evolution. It is intended to be inherited to provide the specific behavior of each type of MAS in the experiment. For the experimental group, we used three types of MAS which overrode the *MultiAgentSystem* class:

1. MASs constrained to spawn in an acceptable state;
2. MASs which were permitted to spawn in a failure state;

3. MASs that were permitted to spawn in a failure state but were not permitted to transfer leaf node tasks.

In our experiments, we used an exponential function to model the increasing likelihood of "wear-out" failure of agents over the life of the system. The failure functions of each agent in our experiments were given by (6) and thus the reliability function was given as (7), as shown in Figure 30. The failure functions were selected because they reach approximately 99% after 5 s.

$$F(t) = 1 - e^{-t}, t \geq 0 \tag{6}$$

$$R(t) = e^{-t}, t \geq 0 \tag{7}$$

Figure 30. Graph of the reliability function, $R(t)$, of agents in the LSSE as a function of time (in seconds, shown in green).

6.2. Scenario 1: Acceptable Initial State Only

The first set of experiments was designed to evolve a sample of MASs, which began their evolution in an acceptable state, according to the ITM. Their resultant optimal states were compared with the states of the MASs at their genesis to quantify the improvement in fault tolerance and reliability.

A MAS was deemed to be in an acceptable state when, for any task that an agent was capable of performing, it was also capable of performing its dependencies. MASs where an agent possessed the knowledge of how to execute a task but not knowledge of how to execute its dependencies were excluded from the experiment (see Scenario 2).

The experiment consisted of the following steps, repeated $n = 1000$ times:

1. Random selection of acceptable MAS configuration;
2. Measurement of the baseline redundancy;
3. Computation of the baseline reliability function for each task;
4. Simulated evolution of the MAS with Intelligence Transfer;
5. Measurement of the resulting redundancy state of the MAS;
6. Computation of the resulting reliability function for each task.

In this experiment, the Transfer Matrix (TM) was set to be equivalent to the ECM, i.e., all tasks in the MAS could be transferred between agents.

The pseudo-random MAS generator produced a random distribution of task execution capabilities among agents in the system (the ECM), and a random interdependency between tasks (the DM). Thus, many of the configurations produced in this manner represented MASs that did not meet the criteria for the experiment. A filtering process was used to discard MASs with cyclical dependencies. These were identified by examining the transitive closure of the DM and rejecting those MASs with diagonal values set in their DMs, indicative of cyclical dependencies. MASs whose ECM was filled with zeros were also rejected.

To more closely resemble the expected behavior of real-world MASs, further treatment was made—where an agent was marked as capable of performing a task in the ECM, the transitive closure of that task's dependencies was also added to the agent's capabilities in the ECM. This was completed because the ECM represents the baseline state of the MAS without Intelligence Transfer. Agents in non-ITM-based MASs in an acceptable state would not be expected to have partial ability to execute a task. Finally, the ECM and transitive closure of the DM of each MAS were checked and duplicates of these were discarded.

LSSE was used to conduct a series of experiments with different numbers of agents and tasks. The evolution period in each case was 5 s, commensurate with the reliability function (7). The number of agents and the number of tasks varied separately within the limits of the computing hardware and time available.

To evaluate the improvement in fault tolerance, the mean TBRM Improvement Factor was recorded for each experiment, defined as the ratio between the number of acceptable states in the PCM and the number of acceptable states in the ECM. Table 4 contains the different configurations of MAS that were tested and their results.

Table 4. Mean Improvement in Acceptable TBRM States per MAS (Scenario 1).

	Tasks	**3**	**4**	**5**	**6**
Agents	5	1.63	1.46	1.32	1.29
	10	1.58	1.43	1.33	1.28
	15	1.57	1.43	1.34	1.27
	20	1.56	1.44	1.33	1.27

To evaluate the improvement in reliability, the Task-Based Reliability Functions for each task were individually recorded, as in Figure 27. These were then averaged to give a Mean Task-Based Reliability Function (MTBRF) which was used to compare reliability improvements between the control (ECM) and evolved (PCM) configurations.

Because Scenario 1 precluded the possibility of tasks transitioning from failure to acceptable states, the effect of Intelligence Transfer was realized in terms of improved task-based redundancy. Thus, we expected the reliability functions to begin at the same value. Then, due to the increased number of redundant agents per task, we expected the PCM MTBRF to be greater over the course of the MASs' evolution than the ECM MTBRF. This was observed to be the case in all MAS configurations tested, with an example result shown in Figure 31. The quantitative improvement is given in Table 5, which shows the range of the Reliability Improvement Factor, defined as the ratio between the values of the PCM MTBRF and the ECM MTBRF in each experiment.

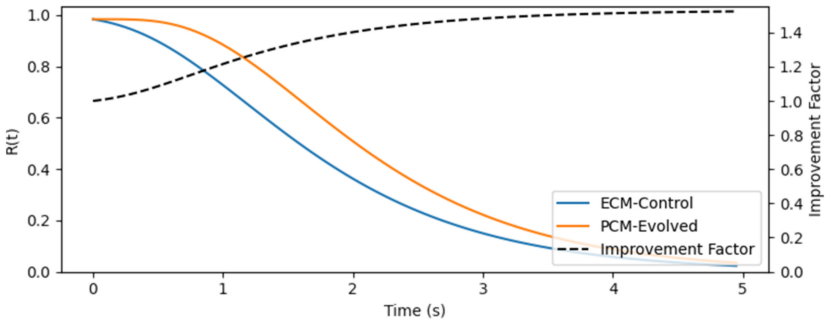

Figure 31. LSSE output for the Mean Task-Based Redundancy Function (MTBRF) and Reliability Improvement Factor of an experiment with 1000 MASs, with acceptable initial states, with 3 tasks and 5 agents in each MAS.

Table 5. Range of Reliability Improvement Factor (Scenario 1).

	Tasks	3	4	5	6
Agents	5	1.00–1.52	1.00–1.40	1.00–1.30	1.00–1.26
	10	1.00–1.51	1.00–1.38	1.00–1.31	1.00–1.26
	15	1.00–1.50	1.00–1.38	1.00–1.31	1.00–1.25
	20	1.00–1.49	1.00–1.39	1.00–1.30	1.00–1.25

In all configurations of MASs tested in Scenario 1, we found an improvement in both the fault tolerance (as measured by the ratio of the number of acceptable states in the TBRM of the ITM MASs in comparison with the benchmark/control group, given in Table 4) and in reliability (as measured by the Reliability Improvement Factor, as defined in (5), given in Table 5).

6.3. Scenario 2: Initial Failure States Permitted

The second set of experiments was designed to test the improvement in fault tolerance and reliability when the MASs were permitted to evolve from initial failure states (at the task level) to acceptable states through Intelligence Transfer.

The experimental methodology was as for Scenario 1, without the constraint which set the transitive closure of dependencies in ECM. This enabled the MASs to spawn having knowledge of how to execute a task but not its dependencies.

A new metric was introduced in this scenario which measured the number of tasks that transitioned from a failure to an acceptable state through the mechanism of Intelligence Transfer. The other metrics used in the previous scenario were also recorded.

Table 6 shows the observed TBRM Improvement Factor for Scenario 2. The results show a greater improvement in this metric across all experiments than the results obtained in Scenario 1, which we attribute to two factors:

1. New acceptable states created through improvement in the redundancy of tasks already in an acceptable state, as in Scenario 1.
2. New acceptable states created due to tasks transitioning from a failure to an acceptable state, then achieving their optimum redundancy as in the first case. Where a task is in a failure state its corresponding reliability function is zero, thus reducing the mean value of the reliability function. Transitioning the task to an acceptable state increases the overall value of the MAS's reliability.

Table 6. Mean Improvement in Acceptable TBRM States per MAS (Scenario 2).

	Tasks	3	4	5	6
Agents	5	3.74	4.50	5.07	6.34
	10	3.46	4.14	5.15	6.01
	15	3.27	4.01	4.75	5.66
	20	3.26	3.82	4.63	5.51

Table 7 shows the improvement in reliability in Scenario 2 and Figure 32 shows the MTBRF and Reliability Improvement Factor for one of the experiments conducted in this scenario.

Table 7. Range of Reliability Improvement Factor (Scenario 2).

	Tasks	3	4	5	6
Agents	5	1.17–2.70	1.32–3.23	1.50–3.70	1.72–4.42
	10	1.07–2.81	1.18–3.37	1.30–4.03	1.48–4.73
	15	1.03–2.81	1.09–3.39	1.20–4.01	1.33–4.68
	20	1.02–2.82	1.05–3.31	1.14–3.95	1.27–4.65

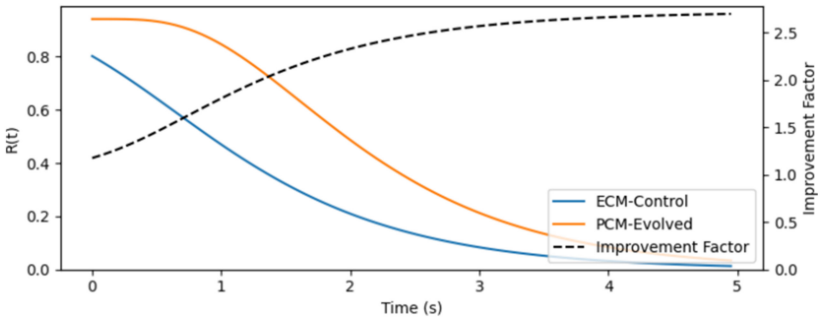

Figure 32. LSSE output for the MTBRF and Reliability Improvement Factor of an experiment with 1000 MASs, with initial failure states permitted, with 3 tasks and 5 agents in each MAS.

In contrast with the values in Table 5, Table 7 shows minimum values of the Reliability Improvement Factors greater than 1. We propose that the difference in the corresponding minimum values between Tables 5 and 7 may be used to quantify the improvement in reliability due to the transition of tasks from a failure to an acceptable state. The increase in reliability in comparison with Scenario 1 can also be seen by comparing Figure 32 to Figure 31.

6.4. Scenario 3: Leaf Node Transfer Restricted

The third set of experiments was designed to test the improvement in fault tolerance and reliability under the same conditions as in Scenario 2, but with the addition of a leaf-node transfer constraint. Whereas in the previous scenarios the TM permitted the transfer of all possible tasks using Intelligence Transfer, this would not be expected to accurately model real-world applications of the ITM, where leaf nodes in the task decomposition diagram act as the interface between the ITM and the local agent and are thus likely coupled at least partially to the configuration of the local agent.

To better model the expected non-transferability of leaf nodes, we introduced a constraint on the TM in Scenario 3 so that tasks without dependencies were marked as non-transferrable. The experimental methodology in Scenario 3 was as for Scenario 2, using the same metrics. Due to the introduction of a limit on the ability of the MASs in Scenario 3 to create new acceptable states through Intelligence Transfer, we expected to observe a lower level of improvement in the fault tolerance and reliability metrics.

Table 8 shows the TBRM Improvement Factor for Scenario 3. As expected, the results indicate a more modest improvement than in the ideal case in Scenario 2, but still better than the result obtained in Scenario 1 without the mechanism for transitioning from failure to acceptable states. In a real-world scenario, the transferability of leaf nodes would be determined based on their coupling to the local agent. Thus, we would expect a real-world result to fall somewhere between the results achieved for configurations of the applicable MASs tested using the methodology in Scenarios 1 and 2.

Table 8. Mean Improvement in Acceptable TBRM States per MAS (Scenario 3).

	Tasks	**3**	**4**	**5**	**6**
Agents	**5**	1.50	1.83	2.11	2.43
	10	1.51	1.86	2.12	2.45
	15	1.51	1.78	2.08	2.39
	20	1.49	1.73	2.07	2.40

Table 9 shows the Reliability Improvement Factor for Scenario 3. As in Scenario 2, the minimum values of the Reliability Improvement Factor were observed to be greater than 1 which we attribute to the improvement due to tasks transitioning from a failure to an acceptable state. The value of the minimum was observed to decrease as the MASs were configured with more agents.

Table 9. Range of Reliability Improvement Factor (Scenario 3).

	Tasks	3	4	5	6
Agents	5	1.15–1.38	1.29–1.61	1.46–1.85	1.64–2.07
	10	1.08–1.41	1.19–1.69	1.30–1.90	1.45–2.17
	15	1.04–1.42	1.09–1.65	1.21–1.90	1.32–2.16
	20	1.01–1.41	1.06–1.61	1.13–1.91	1.26–2.19

Because the ECMs were randomly generated, as more agents were added for a given number of tasks in the MAS, the likelihood of there being at least one agent with knowledge of each task increased, thus the Reliability Improvement Factor curve was expected to more closely follow the result from Scenario 1. This was observed to be the case and can be seen by comparing the results in Table 9 with those in Table 5.

In Scenario 1, the value of the Reliability Improvement Factor was observed to decrease as more tasks were added for a given number of agents. In contrast, in Scenario 3, the Reliability Improvement Factor was observed to increase, as was observed in Scenario 2. This can be seen in Table 9 and Figures 33 and 34. We propose that this is due to the increased number of acceptable states in the TBRM and the associated increase in task-based redundancy due to the ITM's mechanism for transitioning tasks from failure to acceptable states.

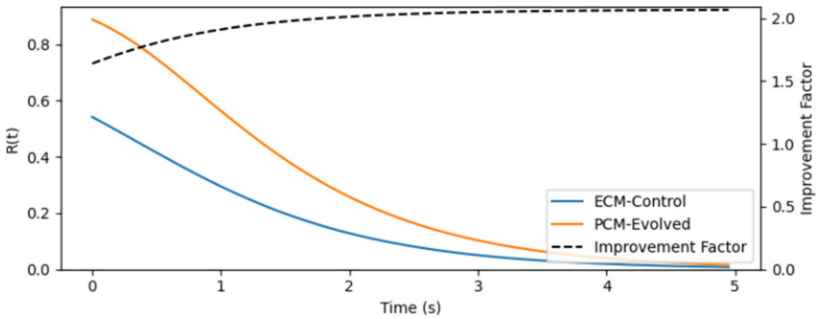

Figure 33. LSSE output for the MTBRF and Reliability Improvement Factor of an experiment with 1000 MASs, with leaf node transfer restricted, with 6 tasks and 5 agents in each MAS.

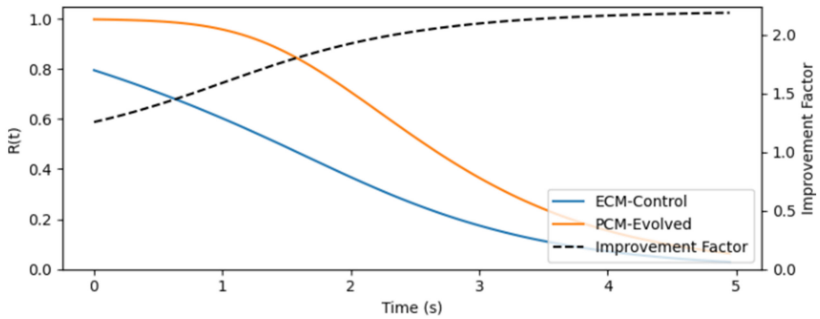

Figure 34. LSSE output for the MTBRF and Reliability Improvement Factor of an experiment with 1000 MASs, with leaf node transfer restricted, with 6 tasks and 20 agents in each MAS.

6.5. Discussion of Experimental Findings

In all the MAS configurations in the three experimental scenarios we tested (acceptable initial states, initial failure states permitted, and leaf node transfer restricted), we found an improvement in both fault tolerance, as measured by the TBRM Improvement Factor, and reliability, as measured by the Reliability Improvement Factor.

6.5.1. Comparison of Summary Results with the Benchmark/Control Group

Figures 35 and 36, and Tables 10–12, synthesize the data in Tables 4–9 in order to compare the ITM's task-based redundancy performance with the benchmark (agent-based redundancy). For comparison purposes, the values given are normalized with respect to the level of redundancy in the control group, thus, the control is always 1.00.

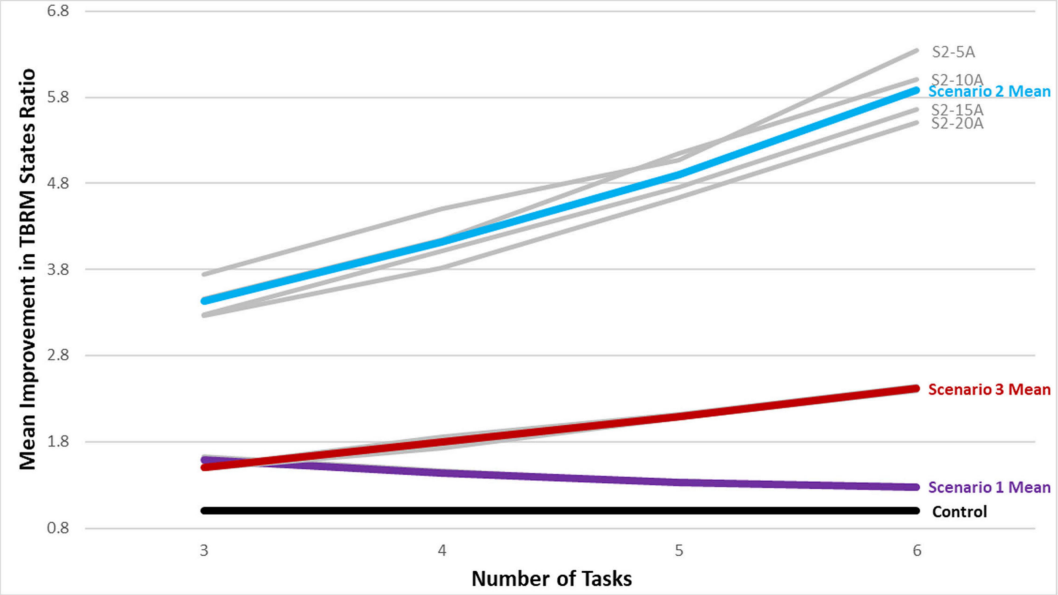

Figure 35. Mean results from each scenario in comparison with the control group with respect to the number of tasks in each MAS. Experiments with different agent configurations are shown in grey and labeled according to the scenario number and number of agents in each MAS. Individual experiments in Scenarios 1 and 3 are plotted but not labeled due to their close proximity to the respective scenario means.

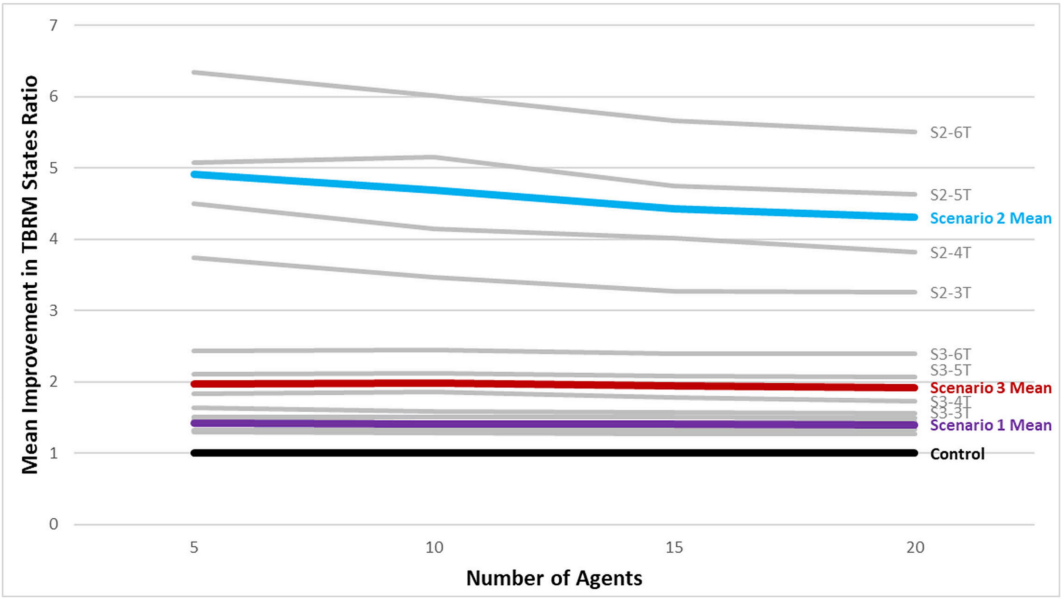

Figure 36. Mean results from each scenario in comparison with the control group with respect to the number of agents in each MAS. Experiments with different task configurations are shown in grey and labeled according to the scenario number and number of tasks in each MAS. Individual experiments in Scenario 1 are plotted but not labeled due to their close proximity to the scenario mean.

Table 10. Summary of improvement ratios for fault tolerance and reliability metrics used in the three studies, as compared with the benchmark/control group.

Scenario	Improvement in TBRM Ratio per MAS		Reliability Improvement Factor	
	Minimum	Maximum	Minimum	Maximum
Scenario 1	1.27	1.63	1.00	1.52
Scenario 2	3.26	6.34	1.02	4.73
Scenario 3	1.49	2.45	1.01	2.19

Table 11. Mean results from each scenario in comparison with the control group with respect to the number of tasks in each MAS.

Number of Tasks	Mean Improvement in TBRM Ratio per MAS			
	Scenario 1	Scenario 2	Scenario 3	Control
3	1.59	3.43	1.50	1.00
4	1.44	4.12	1.80	1.00
5	1.33	4.90	2.10	1.00
6	1.28	5.88	2.42	1.00

Table 12. Mean results from each scenario in comparison with the control group with respect to the number of agents in each MAS.

Number of Agents	Mean Improvement in TBRM Ratio per MAS			
	Scenario 1	Scenario 2	Scenario 3	Control
5	1.43	4.91	1.97	1.00
10	1.41	4.69	1.99	1.00
15	1.40	4.42	1.94	1.00
20	1.40	4.31	1.92	1.00

Table 10 shows the minimum and maximum improvement observed in each scenario. The data show that in all configurations tested, the ITM groups outperformed the control groups in each experiment. The magnitude of the improvement in fault tolerance as measured by the improvement in TBRM states per MAS ranged from a factor of 1.27 to 1.63 for Scenario 1; 3.26 to 6.34 for Scenario 2; and 1.49 to 2.45 for Scenario 3. The magnitude of the improvement in reliability, as measured by the Reliability Improvement Factor, ranged from 1.00 to 1.52 for Scenario 1; 1.02 to 4.73 for Scenario 2; and 1.01 to 2.19 for Scenario 3.

Figure 35 and Table 11 shows the mean results from each of the task-based redundancy (the ITM group) experiments in comparison with agent-based redundancy (the benchmark/control group), with respect to the number of tasks in the MAS. Figure 36 and Table 12 show the same comparison with respect to the number of agents in the MAS.

As the number of tasks was increased, the trend in Scenarios 2 and 3, where the use of latent acceptable states was allowed, diverged positively from the control (as shown in Figure 35).

6.5.2. Specific Scenarios

Scenario 1 precluded the possibility of transitioning tasks from failure to acceptable states, limiting the effect of the ITM to improvements in the redundancy of those tasks, which were already in an acceptable state at system genesis. Despite this limitation, ITM-based MASs were found to be equal to or better in fault tolerance and reliability in comparison with the control.

Scenario 2 tested the ideal conditions for ITM-based MASs, in which all tasks, including leaf nodes, could transition from failure to acceptable states. Under these ideal conditions, a greater improvement was observed in fault tolerance and reliability than in Scenario 1, as shown in Table 10.

The magnitude of the improvements in fault tolerance and reliability were observed to increase as more tasks were added to the system and to decrease as more agents were added. We propose that in the first case this is due to the mechanism of Intelligence Transfer transitioning tasks from a failure to an acceptable state, and in the second case, due to the increasing probability of at least one agent being randomly generated with knowledge of a given task as the total number of agents was increased.

Scenario 3 tested an approximation of the real-world conditions under which the ITM would be expected to operate, where some (or all) of the leaf nodes would be non-transferable. As expected, the results for this scenario showed an improvement in fault tolerance and reliability which fell between the results for Scenarios 1 and 2, as shown in Table 10.

7. Conclusions

In this paper, we proposed the Intelligence Transfer Model (ITM), a novel, generalized approach to modeling, quantifying, implementing, and testing fault tolerance and reliability in multi-agent systems (MASs) with particular applicability to the Internet of Things (IoT) MAS applications.

A candidate implementation was developed using an aspect-oriented design by which we were able to successfully demonstrate the completion of a task using Intelligence Transfer that would not otherwise have been possible to complete.

To model MASs implementing the ITM, we proposed an analytical methodology that quantifies the improvement to fault tolerance and reliability and implemented it in a simulator. Three scenarios were tested in the simulator: (1) an adverse case in which the mechanism of the ITM was restricted to improving redundancy in tasks that were already in an acceptable state; (2) an ideal case in which the mechanism of the ITM was allowed to operate to its fullest possible extent; and (3) a case which approximated the real-world conditions in which we would expect the ITM to operate. Fault tolerance was observed to improve by a factor of between 1.27 and 6.34 in comparison with the control group, depending on the scenario configuration. Similarly, reliability was observed to improve by a factor of between 1.00 and 4.73.

The novel software engineering model proposed in this paper has broad applicability across MAS applications that demand fault tolerance and reliability standards, such as in IoT and Industrial IoT (IIoT) software, cloud computing, edge computing, and autonomous vehicles.

8. Future Work

There are several additional areas of inquiry in which the research we have presented may be extended. These include:

1. Extension of the candidate implementation to support additional operations;
2. Experimentation with more complex resolution strategies;
3. Testing of larger and more complicated MAS configurations in ItmPy;
4. Improvement of ITMP to include support for fragmentation, encryption, trust, integrity, quality of service, denial of service protections, etc.;
5. At-scale testing of MASs with more tasks using LSSE with additional computational resources;
6. Implementation and testing of additional types of MAS besides the three configurations tested (acceptable initial states, initial failure states permitted, leaf node transfer restricted);
7. Experimentation with more complex reliability models such as Markov modeling;
8. Implementation of the ITM in a real-world case study of a MAS which would benefit from improved fault tolerance and reliability, such as in an IoT network, or cloud microservice architectures.

The ITM provides an abstract, generic framework for implementing the transfer of intelligence as a means of improving fault tolerance and reliability in MAS. To create

a testable version of the ITM, we developed a candidate implementation and model at increasing levels of concretization; however, future research may choose to extend any point in the hierarchy of concretization into a new implementation of the model.

Author Contributions: V.O. and B.S. contributed equally to the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: V.O. was supported in this research by an Australian Government Research Training Program Scholarship.

Data Availability Statement: Source code for the ItmPy framework and LSSE software developed in this research is publicly available at [58] and [59], respectively.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

AOSD	Aspect-Oriented System Design
B	Baseline Redundancy Matrix
B*	Potential Redundancy Matrix
CIDR	Classless Inter-Domain Routing
DM	Dependency Matrix
ECM	Existing Capability Matrix
FIPA	Foundation for Intelligent Physical Agents
FM	Failure Matrix
IoT	Internet of Things
IIoT	Industrial Internet of Things
IP	Internet Protocol
ITM	Intelligence Transfer Model
ITMP	Intelligence Transfer Model Protocol
LSSE	Large-Scale Simulation Environment
MAS	Multi-Agent System
MTBRF	Mean Task-Based Reliability Function
PCM	Potential Capability Matrix
POC	Proof-of-Concept
SFM	System Failure Matrix
SRM	System Reliability Matrix
TBRM	Task-Based Redundancy Model
TM	Transfer Matrix
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol

References

1. Lao, L.; Li, Z.; Hou, S.; Xiao, B.; Guo, S.; Yang, Y. A Survey of IoT Applications in Blockchain Systems: Architecture, Consensus and Traffic Modelling. *ACM Comput. Surv.* **2021**, *53*, 18:1–18:32. [CrossRef]
2. Dian, F.J.; Vahidnia, R.; Rahmati, A. Wearables and the Internet of Things (IoT), Applications, Opportunities, and Challenges: A Survey. *IEEE Access* **2020**, *8*, 69200–69211. [CrossRef]
3. Khanna, A.; Kaur, S. Internet of Things (IoT), Applications and Challenges: A Comprehensive Review. *Wirel. Pers. Commun.* **2020**, *114*, 1687–1762. [CrossRef]
4. Syed, A.S.; Sierra-Sosa, D.; Kumar, A.; Elmaghraby, A. IoT in Smart Cities: A Survey of Technologies, Practices and Challenges. *Smart Cities* **2021**, *4*, 429–475. [CrossRef]
5. Kollolu, R. A Review on Wide Variety and Heterogeneity of IoT Platforms. *Int. J. Anal. Exp. Modal Anal.* **2020**, *12*, 3753–3760. [CrossRef]
6. Erazo-Garzon, L.; Cedillo, P.; Rossi, G.; Moyano, J. A Domain-Specific Language for Modeling IoT System Architectures That Supports Monitoring. *IEEE Access* **2022**, *10*, 61639–61665. [CrossRef]
7. Bagchi, S.; Abdelzaher, T.; Govindan, R.; Shenoy, P.; Atrey, A.; Ghosh, P.; Xu, R. New Frontiers in IoT: Networking, Systems, Reliability and Security Challenges. *IEEE Internet Things J.* **2020**, *7*, 11330–11346. [CrossRef]
8. Dorri, A.; Kanhere, S.S.; Jurdak, R. Multi-Agent Systems: A Survey. *IEEE Access* **2018**, *6*, 28573–28593. [CrossRef]

9. Moore, S.J.; Nugent, C.D.; Zhang, S.; Cleland, I. IoT Reliability: A Review Leading to Five Key Research Directions. *CCF Trans. Pervasive Comput. Interact.* **2020**, *2*, 147–163. [CrossRef]
10. Razzaq, A. A Systemic Review on Software Architectures of IoT Systems and Future Direction to the Adoption of Microservices Architecture. *SN Comput. Sci.* **2020**, *1*, 350. [CrossRef]
11. Margi, C.B.; Alves, R.C.A.; Sepulveda, J. Sensing as a Service: Secure Wireless Sensor Network Infrastructure Sharing for the Internet of Things. *Open J. Internet Things* **2017**, *3*, 91–102.
12. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J. All One Needs to Know About Fog Computing and Related Edge Computing Paradigms: A Complete Survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
13. Khan, M.; Wang, J. The Research on Multi-Agent System for Microgrid Control and Optimization. *Renew. Sustain. Energy Rev.* **2017**, *80*, 1399–1411. [CrossRef]
14. Foundation for Intelligent Physical Agents (FIPA). Available online: http://www.fipa.org/ (accessed on 8 February 2022).
15. Java Agent Development Framework. Available online: https://jade.tilab.com/ (accessed on 8 February 2022).
16. Avancini, H.; Amandi, A. A Java Framework for Multi-Agent Systems. *SADIO Electron. J. Inform. Oper. Res.* **2000**, *3*, 1–12.
17. Perles, A.; Crasnier, F.; George, J.-P. AMAK—A Framework for Developing Robust and Open Adaptive Multi-Agent Systems. In Proceedings of the 16th International Conference on Practical Applications of Agents and Multi-Agent Systems, Toledo, Spain, 20 June 2018. [CrossRef]
18. Bellifemine, F.; Poggi, A.; Rimassa, G. Developing Multi-Agent Systems with a FIPA-Compliant Agent Framework. *Softw.–Pract. Exp.* **2001**, *31*, 103–128. [CrossRef]
19. Garcia, C.G.; Zhao, L.; Garcia-Diaz, V. A User-Oriented Language for Specifying Interconnections Between Heterogeneous Objects in the Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 3806–3819. [CrossRef]
20. Wang, Y.; Garcia, E.; Casbeer, D.; Zhang, F. Preface. In *Cooperative Control of Multi-Agent Systems*; Wang, Y., Garcia, E., Casbeer, D., Zhang, F., Eds.; John Wiley and Sons: Hoboken, NJ, USA, 2017; ISBN 978-111-926-612-9.
21. Rivera, D.; Cruz-Piris, L.; Lopez-Civera, G.; de la Hoz, E.; Marsa-Maestre, I. Applying an Unified Access Control for IoT-based Intelligent Agent Systems. In Proceedings of the IEEE 8th International Conference on Service-Oriented Computing and Applications, Rome, Italy, 19–21 October 2015. [CrossRef]
22. Calvaresi, D.; Marinoni, M.; Sturm, A.; Schumacher, M.; Buttazzo, G. The Challenge of Real-Time Multi-Agent Systems for Enabling IoT and CPS. In Proceedings of the International Conference on Web Intelligence, Leipzig, Germany, 23 August 2017. [CrossRef]
23. Johnson, B. *Design and Analysis of Fault-Tolerant Digital Systems*; Addison-Wesley: Reading, MA, USA, 1989; ISBN 978-020-107-570-0.
24. Rajawat, A.S.; Bedi, P.; Goyal, S.B.; Shaw, R.N.; Ghosh, A. Reliability Analysis in Cyber-Physical System Using Deep Learning for Smart Cities Industrial IoT Network Node. In *AI and IoT for Smart City Applications*; Piuri, V., Shaw, R.N., Ghosh, A., Islam, R., Eds.; Springer: Singapore, 2022; ISBN 978-981-167-497-6.
25. Chen, S.; Ho, D.; Li, L.; Liu, M. Fault-Tolerance Consensus of Multi-Agent System with Distributed Adaptive Protocol. *IEEE Trans. Cybern.* **2015**, *45*, 2142–2155. [CrossRef]
26. Li, L.; Ho, D.; Xu, S. A Distributed Event-Triggered Scheme for Discrete-Time Multi-Agent Consensus with Communication Delays. *IET Control. Theory Appl.* **2014**, *8*, 830–837. [CrossRef]
27. Chadli, M.; Davoodi, M.; Meskin, N. Distributed State Estimation, Fault Detection and Isolation Filter Design for Heterogeneous Multi-Agent Linear Parameter-Varying Systems. *IET Control. Theory Appl.* **2017**, *11*, 254–262. [CrossRef]
28. Davoodi, M.; Khorasani, K.; Talebi, H.; Momeni, H. Distributed Fault Detection and Isolation Filter Design for a Network of Heterogeneous Multi-Agent Systems. *IEEE Trans. Control. Syst. Technol.* **2014**, *22*, 1061–1069. [CrossRef]
29. Menon, P.; Edwards. C. Robust Fault Estimation Using Relative Information in Linear Multi-Agent Networks. *IEEE Trans. Autom. Control.* **2014**, *59*, 477–482. [CrossRef]
30. Ilic, N.; Stankovic, M.; Stankovic, S. Consensus Based Overlapping Decentralized Observer for Fault Detection and Isolation. In Proceedings of the 15th IEEE Mediterranean Electrotechnical Conference, Valletta, Malta, 26–28 April 2010. [CrossRef]
31. Hajshirmohamadi, S.; Sheikholeslam, F.; Davoodi, M.; Meskin, N. Event-triggered Simultaneous Fault Detection and Tracking Control for Multi-Agent Systems. *Int. J. Control* **2019**, *92*, 1928–1944. [CrossRef]
32. Zhao, G.; Xing, L. Reliability Analysis of IoT Systems with Competitions from Cascading Probabilistic Function Dependence. *Reliab. Eng. Syst. Saf.* **2020**, *198*, 106812. [CrossRef]
33. Xu, B.; Lu, M.; Zhang, H.; Pan, C. A Novel Multi-Agent Model for Robustness with Component Failure and Malware Propagation in Wireless Sensor Networks. *Sensors* **2021**, *21*, 4873. [CrossRef]
34. Shuai, W.; Liu, J. Designing Comprehensively Robust Networks Against Intentional Attack and Cascading Failures. *Inf. Sci.* **2019**, *478*, 125–140. [CrossRef]
35. Rajput, P.; Sikka, G. Multi-Agent Architecture for Fault Recovery in Self-Healing Systems. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 2849–2866. [CrossRef]
36. Guan, L.; Chen, H.; Lin, L. A Multi-Agent-Based Self-Healing Framework Considering Fault Tolerance and Automatic Restoration for Distribution Networks. *IEEE Access* **2021**, *9*, 21522–21531. [CrossRef]
37. Bagherzadeh, L.; Shayeghi, H.; Pirouzi, S.; Shafie-khah, M.; Catalao, J. Coordinated Flexible Energy and Self-Healing Management According to the Multi-Agent System-Based Restoration Scheme in Active Distribution Network. *IET Renew. Power Gener.* **2021**, *15*, 1765–1777. [CrossRef]

38. Liu, Z.; Yu, H.; Fan, G.; Chen, L. Reliability Modelling and Optimization for Microservice-Based Cloud Application Using Multi-Agent System. *IET Commun.* **2022**, *16*, 1182–1199. [CrossRef]
39. Colman-Meixner, C.; Develder, C.; Tornatore, M.; Mukherjee, B. A Survey on Resiliency Techniques in Cloud Computing Infrastructure and Applications. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 2244–2281. [CrossRef]
40. Mohamed, M.; Engel, R.; Warke, A.; Berman, S.; Ludwig, H. Extensible Persistence as a Service for Containers. *Future Gener. Comput. Syst.* **2019**, *97*, 10–20. [CrossRef]
41. Olaru, A.; Florea, A. Emergence in Cognitive Multi-Agent Systems. In Proceedings of the 17th International Conference on Control Systems and Computer Science, Bucharest, Romania, 26–29 May 2009.
42. Zidan, A.; Khairalla, M.; Abdrabou, A.M.; Khalifa, T.; Shaban, K.; Abdrabou, A.; El Shatshat, R.; Gaouda, A. Fault Detection, Isolation and Service Restoration in Distribution Systems: State-of-the-Art and Future Trends. *IEEE Trans. Smart Grid* **2017**, *8*, 2170–2185. [CrossRef]
43. Marin, O.; Sens, J.-P.; Guessoum, Z. Towards Adaptive Fault Tolerance for Distributed Multi-Agent Systems. In Proceedings of the European Seminar on Advances in Distributed Systems, Bertinoro, Italy, 14–18 May 2001.
44. Zidan, A.; El-Saadany, E. A Co-operative Multiagent Framework for Self-Healing Mechanisms in Distribution Systems. *IEEE Trans. Smart Grid* **2012**, *3*, 1525–1539. [CrossRef]
45. Fedoruk, A.; Deters, R. Improving Fault-Tolerance by Replicating Agents. In Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 15 July 2002. [CrossRef]
46. Ananada, S.A.; Gu, J.-C.; Yang, M.-T.; Wang, J.-M.; Chen, J.-D.; Chang, Y.-R.; Lee, Y.-D.; Chan, C.-M.; Hsu, C.-H. Multi-Agent System Fault Protection with Topology Identification in Microgrids. *Energies* **2017**, *10*, 28. [CrossRef]
47. Almeida, A.; Aknine, S.; Briot, J.-P.; Malenfant, J. Predictive Fault Tolerance in Multi-Agent Systems: A Plan-Based Replication Approach. In Proceedings of the 6th International Join Conference on Autonomous Agents and Multi-Agent Systems, Honolulu, HI, USA, 14–18 May 2007. [CrossRef]
48. Chen, X.; Jiao, J. A Fault Propagation Modeling Method Based on Finite State Machine. In Proceedings of the Annual Reliability and Maintainability Symposium, Orlando, FL, USA, 23–26 January 2017. [CrossRef]
49. Gabel, M.; Schuster, A.; Bachrach, R.-G.; Bjorner, N. Latent Fault Detection in Large Scale Services. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks, Boston, MA, USA, 25–28 June 2012. [CrossRef]
50. Song, J.; Parmer, G. C'Mon: A Predictable Monitoring Infrastructure for System-Level Latent Fault Detection and Recovery. In Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium, Seattle, WA, USA, 13–16 April 2015. [CrossRef]
51. Sternberg, R.J. Intelligence. *Dialogues Clin. Neurosci.* **2012**, *14*, 19–27. [CrossRef]
52. Russel, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd ed.; Prentice-Hall: Hoboken, NJ, USA, 2020; ISBN 978-013-207-148-2.
53. Wand, M.; Kiczales, G.; Dutchyn, C. A Semantics for Advice and Dynamic Join Points in Aspect-Oriented Programming. *ACM Trans. Program. Lang. Syst.* **2004**, *26*, 890–910. [CrossRef]
54. Popovici, A.; Alonso, G.; Gross, T. Just-In-Time Aspects: Efficient Dynamic Weaving for Java. In Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, Boston, MA, USA, 17 March 2003. [CrossRef]
55. Cassar, I.; Francalanza, A.; Aceto, L.; Ingolfsdottir, A. eAOP: An Aspect Oriented Programming Framework for Erlang. In Proceedings of the 16th ACM SIGPLAN International Workshop on Erlang, Oxford, UK, 8 September 2017. [CrossRef]
56. Mishra, K.B. *Reliability Analysis and Prediction*; Elsevier Science: Amsterdam, The Netherlands, 1992; ISBN 978-044-489-606-3.
57. Dubrova, E. *Fault-Tolerant Design*; Springer: New York, NY, USA, 2013; ISBN 978-146-142-112-2.
58. ItmPy Source Code. Available online: https://github.com/vyas-oneill/itm_py (accessed on 16 July 2022).
59. LSSE Source Code. Available online: https://github.com/vyas-oneill/lsse (accessed on 16 July 2022).