Diversity in Massively Multi-agent Systems: Concepts, Implementations, and Normal Accidents

Philip Feldman[1,3] and Antonio Bucchiarone[2](✉)

[1] University of Maryland, Baltimore County, MD, USA
[2] Fondazione Bruno Kessler, Trento, Italy
bucchiarone@fbk.eu
[3] ASRC Federal, Laurel, MD, USA

Abstract. Coordination for Transportation as a Service (TaaS) can be implemented on a spectrum, ranging from independent agents communicating exclusively through market exchanges to hybrid market/hierarchy approaches fixed hierarchical control systems. An overview of each approach is described and a detailed description of recent work in simulating a hybrid solution is presented. The use of diversity as a potential approach to reduce the impact of catastrophic Normal Accidents is discussed.

Keywords: Diversity · Multi-agent systems · Transportation as a Service · Market systems · Hierarchical control · Distributed control

1 Introduction

Through most of history, the allocation of transportation resources has not been an issue. The trouble arose once we started to ride horses, sail boats, ride trains and travel in cars. Transportation resources can be expensive. In 2016, the average US consumer spent $8,427 on vehicles [2], or approximately 20% of the median US household income of $43,290 for that year [3]. Clearly, using transportation services more efficiently can create enormous savings for the individual, while simultaneously reducing congestion and pollution in areas where these efficiencies are achieved.

Transportation as a Service (TaaS) is the application of information technology to the movement of people at the individual level. Scheduling and allocation that was previously only cost effective for transportation of users as groups can now be allocated down to the level of an internet-connected, GPS equipped bicycle or e-scooter. Fifteen years ago, the integration of internet-connected, GPS equipped trucks disrupted the trucking industry, allowing for the emergence of markets that allowed individual owner-operators to bid competitively across a number of freight exchanges [5].

But people are different from cargo. They have agency, and the cost of even relatively minor errors can be high. They also have requirements that cargo doesn't, like cognitive load and status. How can these transportation needs be efficiently met? In this paper, we introduce TaaS as a *massively multi-agent system* able to cover a diverse technological spectrum ranging from tightly structured hierarchies to open markets. We then describe in detail research into a middle ground consisting of loosely connected ensembles of hierarchies. Lastly, we discuss some of the implications that can arise from building densely connected, highly responsive transportation networks, particularly with respect to unanticipated, extreme conditions.

2 The TaaS Spectrum of Coordination

Optimizing transportation isn't just hard, it's *NP-Hard*. As seen with just a single traveling salesman, the number of paths scales geometrically with the number of towns to visit. Even such apparently simple transport problems such as determining the staging and stops for a set of elevators in a skyscraper remain unsolved, and have recently been analyzed using machine learning techniques [14]. With distributed systems supporting potentially billions of people utilizing millions of devices, a closed form solution is clearly impossible. Rather, we need to focus on attainable benchmarks to evaluate potential and actual systems.

Wellman, in his work on market-oriented programming [35] suggests the following criteria for evaluating distributed systems:

- What is the quality of the allocation of resources?
- How computationally intensive is the allocation process?
- How easy is it to design and specify a system?

Since these criteria were developed in 1994, large scale wireless networks have become a daily reality along with hacking and security breaches. As such, we suggest adding the following criteria:

- How much bandwidth is needed? Particularly in situations where communication can be unreliable, the speed and number of bytes needed to achieve a complete transaction needs to be considered.
- How resilient is the system to unforeseen conditions? Can the system adapt rapidly and effectively to conditions that significantly disrupt normal transportation patterns, such as evacuations, natural disasters and even wars?
- How secure is the system? Is the system vulnerable as a whole or parts? Can the system be hacked to the point that a vehicle becomes a danger to its passengers and others?

With these criteria in mind, we now look to the three main regions that define this spectrum - market-based, hybrid, and hierarchical.

2.1 Market Systems

The Oxford English Dictionary defines “stock exchange” as A market for the buying and selling of public securities; the place or building where this is done; an association of brokers and jobbers who transact business in a particular place or market [1].

Although we could find no online exchanges for TaaS for people, there are online transportation exchanges for cargo have been in existence for about 20 years, and three particular categories have emerged; clearing houses, auctions, and freight exchanges [25].

Clearing houses collect the loads posted by the shippers or capacity posted by carriers. Both parties search for their preferred choice and negotiate one-on-one for the price of delivery. Lyft, Uber and other transportation network companies (TNCs) tend to incorporate a private version of this model as private clearing houses for matching user requests and ride providers.

Auction houses engage both carriers and shippers to sell their capacity or delivery services at the best price. The auction system has been found to improve their occupancy rate of transport vehicles while shippers can obtain better rates under spot market circumstances.

Freight exchanges let shippers post their demands and carriers posts their capacity in an online marketplace where each of them will be allocated to their respective services required at a competitive price.

Market systems are generally regarded to be an extremely efficient way to allocate resources. Agent-based simulation [33] shows that transportation markets can achieve Pareto equilibrium, where no user can improve their position without making another agent’s position worse. Markets can be gamed however. For example, in eBay auctions users tend to pile on bids an the last few seconds of an auction in a process called sniping. This tends to force a lower price, benefiting the bidder.

The amount of computation scales geometrically with the number of users and providers. Multiple heuristics can be applied to the data to reduce the amount of computation. For example, riders and providers that are near enough to each other need be evaluated. That being said, in dense urban environments, that could still be a computationally intensive task, where “good enough” answers would have to be accepted.

Since private clearing houses for TaaS have been deployed, it is clear that such systems can be built and deployed at scale. Lyft and Uber both offer APIs for external developers to integrate other products into their respective corporate ecosystems. This is only front end interaction though. The full stack that performs global scale interactions clearly depends on hundreds of developers.

Bandwidth for market systems does not need to be large. At a minimum there needs to be a request by the consumer and a response by the provider. Additional, market-specific information such as time remaining on an auction adds very little data to a given payload. No real-time communication is *required* for the market, though meeting particular deadlines can be critical. Where bandwidth permits, companies often provide UIs where real-time information like vehicle location to the user.

Security in regulated exchanges such as the New York Stock Exchange seems surprisingly effective, even though glitches such as Flash Crashes occur [20]. One reason is that these systems are often on their own networks, and have circuit breakers that halt all trading. Also, each transaction is directly associated with a registered user who is financially liable for all transactions. This fiscal obligation has resulted in massive losses due to software glitches [13], so there is considerable motivation on the part of traders to self-police.

Markets can adapt quickly to changing conditions as long as the basic framework of the market remains intact. A power failure at a server farm could shut down a centralized exchange. Another issue is the fundamental nature of a market, where prices fluctuate, based on supply and demand. In the case of an evacuation order, a market-based TaaS could easily favor the rich as the price for transportation rises inversely with respect to supply. An example of this is surge pricing, which raises the cost of a ride at times of high demand. A market can't adjust organically to such issues, so specific policies have to be put in place. For example, Uber put a cap on fares during the evacuation of sections of Florida prior to the landfall of hurricane Irma [22].

Research Challenges. As stated in the previous section, a private, central exchange is vulnerable to a sufficiently widespread catastrophe. Distributed, public exchanges could address this weakness, but building distributed trustworthy systems is difficult. Yuan and Wang develop a blockchain-based mechanism for intelligent transportation services [37], but there are high computation costs associated with creating ledgers. Furthermore, blockchain requires all transacting computers to be connected. An orphan network, such as might occur during a catastrophe would have to halt transactions. Other, tangle-based systems could be more resilient, and support isolated networks [28]. This would be important research, because a system that could support isolated markets could support any technology that can interact with the exchange, for example, the use of horses could emerge in the case of chronic fuel shortages.

2.2 Hierarchical Control Systems

Hierarchical control predates digital embodiments, with examples as diverse as companies, armies, and governments. As such, it is an intuitive concept for control systems that was described in considerable detail by Roth, in 1962 [30]. In his article, Roth describes the major components, communication requirements, separation of responsibility, and human integration that are still the basis for today's systems.

The National Institute of Standards and Technology has implemented a framework for large scale real-time control, the *NIST Real-time Control (RCS) Reference Model Architecture* [29]. NIST formalized RCS as a standard reference architecture and implemented this framework across multiple domains, ranging from vehicle control to robot control to manufacturing.

RCS is based on the concept of hierarchical task decomposition. A complicated task, such as painting a car, can be broken down along levels of abstraction.

At the highest level is the overall command paint the next car red. This command is then broken down into commands that are issued to the subcomponents, such as the painting robot and the auto body transport elements. At the lowest levels of abstraction the servos that move the various actuators are controlled. At the lowest levels of the system, updates rates are thousands of times a second. At the highest levels updates need only occur every few minutes.

The logic to perform a task is contained in an RCS *Controller Module*. All controllers have the same structure:

1. A command buffer, which contains the command (e.g. MOVE_TO_START), and a serial number.
2. A response buffer which contains an echo of the command, a status (e.g. WORKING, DONE, ERROR) and a serial number.
3. A set of command and response buffers for any child controllers.
4. A window to world data, that contains environmental information. Sensors controlled modules that could be useful to other modules are published here.
5. A preprocess that reads in any parent commands, child responses, and environmental data.
6. A decision process, that takes the command and the current state of the controller and decides what task is active.
7. A collection of finite state machines that perform the amount of the task within the update rate.
8. A postprocess where responses to the parent, commands to the children, and any useful sensor data is published.

Modules are connected in a strict hierarchy - no module may have two parents. Because each command has a serial number that is echoed back in the response, all direct interaction between modules is deterministic, and can tolerate poor communication - a command can't be sent until the response echoes back the serial number. RCS can also be used in simulations. The controllers interact with a physics based environment that provides enough information for the sensors and actuators to behave within reasonable parameters. Because controllers contain all process knowledge related to a task at their level of abstraction, communication between controllers is typically minimal. This in turn affords easy modification and adjustment of the hierarchy, so as a task or technology changes, only small parts of the running system need to be modified.

Hierarchies do not allocate resources well. The top-down nature of the control stands in opposition to the bottom-up self organization of market systems. What this means is that the allocation scheme has to be encoded in the structure of the particular control hierarchy. When this is done, and for those explicit instances, allocation can be extremely rapid and efficient. The moment the problem envelope exceeds the ability of the hierarchy to accommodate it, the control hierarchy can no longer adapt.

Control hierarchies can be designed to be optimally efficient, since the entire structure is known. Further, each component is trusted, additional work to determine trustworthiness (e.g. Blockchain calculations) is not needed. Short of a monolithic system, a control hierarchy should be able to embody the lowest computational intensity for well-defined tasks.

NIST RCS in particular is designed specifically to reduce cognitive load. Controllers handle a single task, using the same *preprocess/decision process/state table/post process* pattern. Developers quickly learn this methodology and can easily contribute to developers working on different controllers. Debugging tools that monitor the commands and responses between controllers provide high level views of the functioning of the overall system, while drilldown into the common state table operations within each controller can also be visualized using tools that understand the RCS implementation.

A properly designed hierarchy has very low bandwidth requirements due to the compartmentalization of the tasks within controllers. In places like high-speed servo control, where multiple child controllers may need to react to rapid commands by a parent controller, the system can be designed such that all hardware shares a high-speed communication channel.

Because they are designed to deal with a particular environment, a factory floor, a submarine, an autonomous vehicle, hierarchical control systems do not have an inherent capacity to adapt to a different control environment. If new hardware replaces old hardware on the factory floor, the control system must be adjusted too. A good hierarchy makes this easy to do, with minimum impact on the rest of the running system, but that is different from expecting the hierarchy to adapt to the new hardware.

Due to the explicit design of the system and the reuse of common components such as controllers, it is possible to design and build an extremely secure control hierarchy. That being said, if the top level controller is hacked, the rest of the system will blindly follow. As a rule, the risks of broken security lessen as the control system moves away from the hierarchical side of the spectrum.

Research Challenges. A great deal of research has been performed on adaptive hierarchical control systems [19,32]. Hierarchical systems have inherent disadvantages in that they need complete information across sub-systems to coordinate control down to the individual actuators. As such, a designer of a large scale, low response time system (such as a nuclear reactor) has to be aware of all possible interactions within the system, since an actuator far down a one branch of software involved with emergency response may vent high-pressure radioactive steam into a section of plumbing normally involved with the steam used for powering turbines [27]. The issue here is one of the combinatorial explosion of possibilities that can occur in monolithic systems. Working though all potential combinations is possible on small systems, but rapidly becomes uncomputable as the hierarchy grows.

If, on the other hand, small, testable hierarchies can be linked so that rapid response and control happens within the hierarchy, but looser interactions can exist *between hierarchies* then more resilient systems can be designed. These sets of smaller hierarchies can operate in clusters or ensembles could have the ability to operate using local information and respond in more adaptive, flexible ways to a scoped set of problems. This approach is discussed in the next section, where we use urban mobility to explore how adaptive ensembles of hierarchies can blend market flexibility and hierarchical control.

3 Ensembles of Hierarchies

Modern cities are complex socio-technical entities that exist to provide services effectively to their residents and visitors. Networks for water, electricity, communications, and finance permeate the urban environment. Further, people need to travel quickly and conveniently between locations at different scales, ranging from a trip of a few blocks to a journey across town or further. Each trip has its set of requirements. Time may be of the essence. Cost may be paramount and the convenience of door-to-door travel may be important. In each case, the transportation infrastructure should seamlessly provide the best option. A modern city needs to flexibly integrate transportation options including buses, trains, taxis, bicycles and cars. The combinatorial complexity of all these possibilities negates the option of a single, monolithic control system. How would a grouping, or ensemble of hierarchies perform in this situation?

In this section, we consider a simplified urban mobility system (UMS), that comprises several means of transportation that are collectively managed. We focus on the aspect of adaptivity in situations where computational agents and affected human (e.g passengers, drivers) collectively reach adaptation decisions. In the following we describe the scenario and demonstrate the challenges it poses to collectively adapting socio-technical systems like UMS.

Our UMS consists of the following means of transportation:

- *Regular bus* service, a network of fixed bus routes with fixed timetable;
- *Flexible Bus (FB)*, a service that collects trip requests from customers and organizes on-demand routes that efficiently serve the requests;
- *Car Pool*, a service to share car journeys so that more than one person travels in a car;
- *Taxi*, a conventional taxi service;

Each means of transportation has a complex internal substructure. For example the FB service allows third party minibus owners to register their availability for serving trips, and for customers to register trip requests (e.g., location, time). The service dynamically creates routes on the basis of time and location of the trips requested and the availability of vehicles. Each FB route is an unit comprised of the vehicle (or FB driver) that is supposed to serve the route, and passengers traveling within similar time and location spans. A FB route is supervised by the FB company that provides all necessary infrastructure. It is easy to see that a FB route is a good example of collaborative behavior: passengers sacrifice part of their flexibility in order to travel cheaper, compared to a taxi, and quicker compared to conventional buses.

As shown in Fig. 1, our simulation of this system connects transportation with a set of agents that can interact in different ways. Agents can be part of several possible ensembles (i.e., from E1 to E9) according to their needs. Figure 1 shows the topology of the UMS example. It includes a hierarchical pattern but also direct relations, as in the Ensemble E9 composed by the FlexiBus Company (FBC) and the Car Pool Company (CPC).

To illustrate this, let us consider a messy but not unlikely scenario: A passenger is late for her FB, so the bus waits until she arrives. A current passenger, fed up by the extra waiting, leaves the bus to walk the remaining distance. To make up time the driver speeds and is involved in an accident, blocking traffic and requiring the FB company to rout around the congestion. This is not only a bad day for the passengers, it is an extremely complicated and expensive problem for the FB company. Its options include, among others:

1. refund passengers who cannot reach their destinations in time
2. reroute the running buses to prioritize the most affected customers
3. reroute the running buses so that the largest number of passengers reach their destinations on time
4. reassign passengers to other routes
5. reassign (groups of) passengers to other means of transportation

In the following subsection we analyze the challenges posed by this UMS scenario.

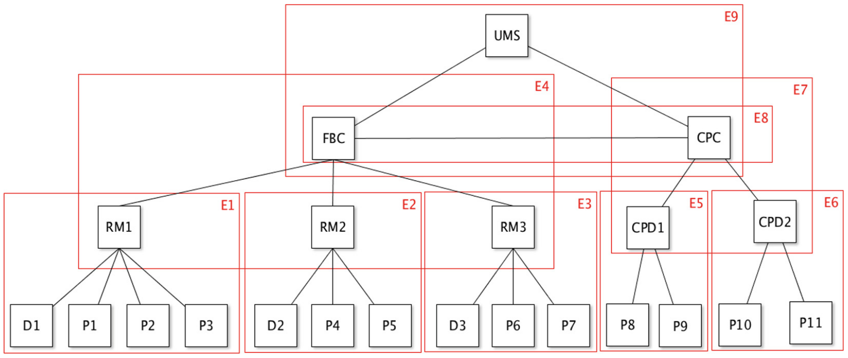


Fig. 1. Types of agents and ensembles in the UMS.

3.1 Research Challenges

The principal agents in the UMS scenario (i.e., passengers, FB drivers, FB company, etc.) are generally autonomous and act independently. This makes the system highly dynamic and distributed. The surrounding environment of an agent changes frequently and unpredictably (e.g. as other agents change their minds) and therefore the system requires constant monitoring and adaptation.

Existing approaches [34,36,39], normally deal with multi-agent adaptive systems through isolated adaptation: each agent adapts itself independently from each other. However, in our scenario the problem is complicated by collective behavior. Even though agents are generally autonomous, they dynamically form collaborative groups, called *ensembles*, to gain benefits that otherwise would not be possible. The example of such an ensemble is a FB route (E1 in Fig. 1) which coordinates the adaptation behavior of multiple agents (FB driver, passengers, and FB company) and in return gives them certain benefits (e.g., cheap and fast transportation). Membership of an ensemble may temporarily reduce the flexibility of its agents. Within this context, isolated agent self-adaptation is not effective. We can easily imagine what happens if a passenger books a trip with a FB and then silently changes their mind and decide not to travel. It is likely to cause unnecessary delay for the route (e.g. the bus will have a redundant stop) and raise the cost of the trip for the remaining passengers, including potential charges for the canceling passenger.

Even more serious consequences arise if a bus gets damaged: isolated adaptation by the bus driver could totally break the passengers' travel plans. Adaptation has to take into account not only customers trip requests but also customers constraints and preferences. For example, a particular passenger may want to avoid traveling through unsafe areas in the city, but a possible re-planned route may pass through such area.

The term *ensemble* has recently been introduced in the literature to denote very large-scale systems of systems that may present substantial socio-technical embedding [17,38]. They typify systems with complex design, engineering and management, whose level of complexity comes specifically from bringing together and combining in the same operating environment many heterogeneous and autonomous components, systems and users, with their specific concerns. To be robust against the high degree of unpredictability and dynamism of their operating environments, and to sustain the continuous variations induced by their socio-technical nature, ensembles need to self-adapt.

In adaptive systems with collective behavior, new approaches for adaptation are therefore needed that allow (i) multiple agents to collectively adapt with (ii) negotiations to decide which collective changes are best.

Collective adaptation also raises a second important challenge: *which parts of the system should be engaged in an adaptation*? This is not at all trivial, since solutions for the same problem may be generated at different levels. For instance, a passenger's delay may be resolved in the scope of a FB route, by re-planning the route, or in the wider scope of the FB company, with the engagement of other routes, or even in the scope of the whole UMS, with the engagement of other means of transportation such as a car pool. The challenge here is to understand these levels, formalize them and create a mechanism that decides the right scope for an adaptation for a given problem.

Within our scenario, we can identify several levels of abstraction that operate at different scales in time and space. An FB route combines passengers with a driver, a Flexibus company combines FB routes, and an UMS combines a

Flexibus company and other means of transportation. The higher the level of abstraction, the wider the scope of adaptation.

The *continuous and distributed adaptation* is a key feature of Collective Adaptive Systems (CAS), when it comes to operating in constantly changing environment. Concepts that are close to those introduced above, and that characterize CAS, have been studied in various domains such as, *Swarm Intelligence*, where actors are essentially homogeneous and are able to adapt their behavior considering only local knowledge [11,21], or *Autonomic computing*, where the actor types are typically limited and the adaptation is guided by predefined policies with the objective to optimize the system rather than evolve it [4], or *Service-based systems* where services are designed independently by different service providers and are composed to achieve a predefined goal (i.e., user tasks [6] or business goals [23]), or *Multi-agent based systems* where activities of different actors are regulated by certain collectively defined rules (i.e., norms) [12]. Most of the results obtained in these domains are tailored to solve a specific problem using a specific language or model and *lack of generality.*

At the same time, these studies tackle only some of the challenges for individual agents, while leaving decision-making for group, collective, and larger scales relatively unexamined.

For these reasons, we should move from *individual based applications* to *collective systems* with techniques that support adaptation of collectives. This will be achieved by defining *new software engineering methods* (i.e., models, theories and tools) that are highly flexible and can be specialized to fulfill different tasks in different ways. At the same time, they will introduce features for the collaboration and coordination among agents, as this is an essential prerequisite for building *collective adaptive systems (CAS).* The collective nature of software systems, with the important aspect of the *diversity* that different agents bring in, makes the theme issue completely new and different with respect to previous issues in the field of engineering complex and adaptive systems. Models for CAS need to be *adaptable by design*; this means that each agent in the system must be able to adapt its behavior taking into account the current context/situation. The model should be flexible and extensible, fusing a priori and learned knowledge. The local knowledge of an agent should be extendable during its lifetime, based on collaboration with other agents and the current context of the adaptation. Finally, the model should consider the *heterogeneity* and *diversity* of the agents, incorporating the specific roles that they play in the collective.

In the next sections we introduce our approach that addresses the challenges above in order to facilitate collective adaptation.

3.2 General Framework

Our approach addresses the challenge of collective adaptation by proposing a new notion of ensembles that enables systems with collective adaptability to be built as emergent aggregations of autonomous and self-adaptive agents. Key properties of our approach include (i) the emphasis on collaboration towards fulfillment of individual, diverse goals, and (ii) the heterogeneous nature of an ensemble with respect to roles, behaviors and goals of its participants. These properties distinguish our approach from other types of ensemble models, like for instance swarms, where all elements of a community have a uniform behavior and global shared goal [11,21], and multi-agent systems and agent-based organizations [16], where there may be several distinct roles and behaviors, but the differentiation is still limited and often pre-designed.

We define an ensemble in terms of a set of *roles* that can be taken by participating agents. A role can be seen as a component (or a type) that can be instantiated by agents of different types (e.g. a user can either play the role of a carpool passenger or a driver).

Each agent role (as depicted in Fig. 2, left-side) is modeled by a core process (i.e., *Agent Behavior*) and a *Scope* artifact used to understand when and how a role must be involved in a collective adaptation problem resolution. Each *Scope* is formed by an *Handler (H)* capable to catch an *Issue* and trigger the appropriate *Solver*.

Solvers model the ability of an agent to handle one or more issues. Each solver relates to the particular issue that it can handle. Moreover, each handler refers to a finite scope in the process of an agent, and it can be of two different types: (i) *external* handlers are used to catch issues coming from other agents in the system (both in the same or in a different ensembles); (ii) *internal* handlers are devoted to monitor internal property and catch the issues arising when this properties are violated.

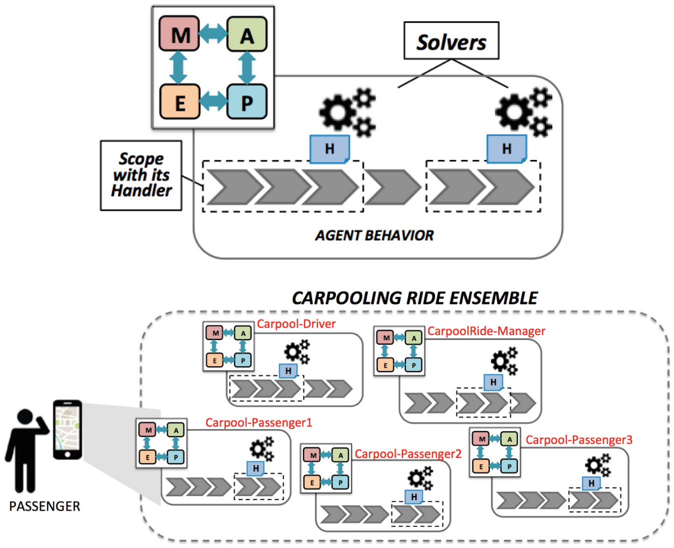


Fig. 2. Agent and ensemble models.

During the normal execution of the system, interactions between agents and ensembles are formed. Ensembles can be created spontaneously and change over time: different agents may join or leave an existing ensemble dynamically and autonomously. Their termination is also spontaneous: participants have reached their goals, or the ensemble itself has ceased to provide benefit.

For instance, in the carpooling scenario (see Fig. 2, right-side), users subscribe to a carpool ride by exploiting functionalities of the carpool manager, which has previously set up the ride and assigned a driver to it. In this way, the ensemble made by the carpool manager, the driver and the passengers is constructed[1].

During execution, the ensemble can evolve. New passengers can subscribe to the ride, while others can leave. However, to deal with unpredictable changes, local adaptation is not enough, since the scope of these changes goes beyond the single agent. Typical changes occurring in dynamic environments are characterized by the fact of affecting different agents, who can also belong to different ensembles:

- the agent directly related to the change (e.g., a ride interrupted directly affects the driver);
- the agents belonging to the same ensemble (e.g., both the passengers on board and the ones waiting at the pickup points);
- the agents involved as a consequence of the adaptation executed to solve the problem (e.g., the Carpool company provides a new plan for the waiting passengers).

This demonstrates the need for collective adaptation approaches able to deal with dynamic changes, and whose scope can be, in the worst case, the entire system. Thus, such an approach must provide one or more decision management strategies, to allow different agents to communicate and cooperate in a collective manner.

The collective adaptation process is handled in a decentralized manner by the agents involved, directly or indirectly, in an adaptation issue. Each agent implements a Monitor - Analyze - Plan - Execute (MAPE) loop [18] (as depicted in Fig. 3) that allows for the dynamic interaction with the other agents. We use a color code to distinguish the four phases of the MAPE loop. In the following, we highlight the most interesting states of the SM. In the *Monitoring phase*, each agent monitors the environment through active handlers. Issues can come either from the agent itself (*Issue Triggered*) or from a different agent, asking support for solving an issue (*Issue Received*).

The sequence begins with the *Analyze phase*, where the issue solver is called (*Local Solver Called*). In the *Planning phase*, if the solver has found a solution (*Solution Found*), the *Collective Planning phase* begins. All the agents involved in the issue resolution process will collectively collaborate to solve it.

In this example, a solution provided by the solver foresees the involvement of other agents, which are first found (*Targets Found*), and then triggered (*Issues Targeted*) to be involved in the resolution process. Once the current agent

[1] In this paper, we focus only on the collective adaptation aspect for agents. Their normal execution can be handled using the technique presented in [8], which is compatible with the approach we are proposing.

receives feedback from the triggered agents (*Solution Received*), it selects the best solution (*Solution Chosen*) (e.g., by applying approaches like [7,31]).

At this point, we should distinguish two cases. If the issue was triggered internally (root node edge), the agent asks the involved targets to commit their local best solution (*Ask Partners To Commit*), it waits for their commit to be done (*All Partners Commit Done*), and eventually it commits its local solution (*Commit Local Solution*). Otherwise, if the issue was coming from outside (not root node edge), the agent reports the feedback to the issues sender (*Solution Forwarded*), and it waits for a future commit (*Commit Requested*).

The agent can receive a positive or a negative reply for its proposed solution. In both cases, it executes a solution commit (*Commit Local Solution*), which will be empty in the negative case.

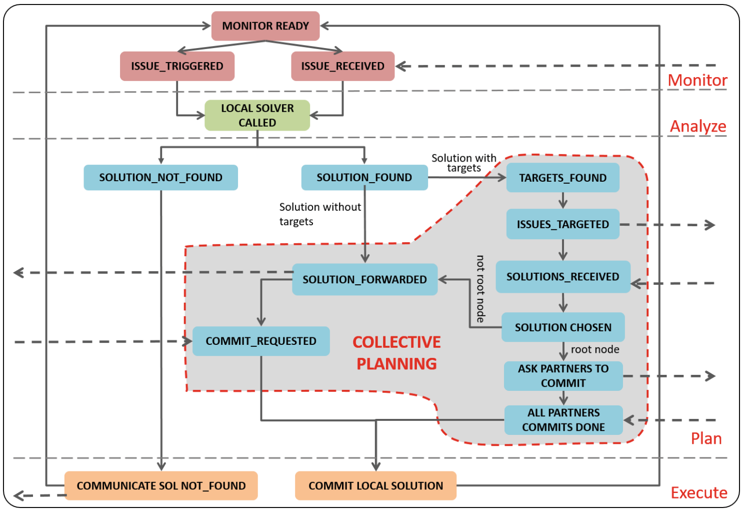


Fig. 3. MAPE state machine.

3.3 Hierarchical Adaptation

As we have seen so far, an agent instance resembles an ensemble instance in that both of them are essentially sets of role instances. However, an agent instance includes role instances belonging to different ensembles but played by a single agent. This architecture allows us to model more complex agents that can run multiple tasks simultaneously (e.g, a person can easily take/plan many activities at a time: travelling, visiting a cinema, organizing a meeting with colleagues etc.). However, a much more interesting application for agents with multiple roles is establishing links between different ensembles. If we assume that, similarly to ensemble instances, roles instances within an agent can also communicate with each other, this can be used to organize coordination of multiple ensembles.

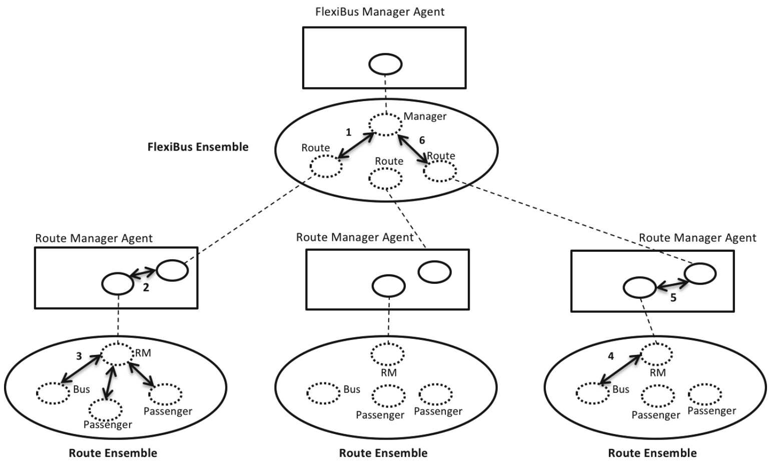


Fig. 4. Ensembles hierarchy.

To demonstrate this, let us consider the example in Fig. 4. Here we see two different ensembles: one (FlexiBus Ensemble) is devoted to managing the whole FlexiBus system and includes the roles of a manager and a route.

Multiple routes can exist at a time. Moreover, there is Route Ensemble devoted to managing a particular FlexiBus route. Multiple ensemble instances of this type can exist. If we try to place in this picture an agent (e.g., a piece of software) that manages a route, it is clear that its role will be twofold: on the one hand it is a subordinate (with role route) in a higher level ensemble that manages the whole FlexiBus system, and at the same type it is a leader in the lower level ensemble that manages participants of a single route.

As such, it plays two roles in these two ensemble simultaneously. By letting role instances within an agent talk to each other (e.g., using principles similar to issue communication), we can efficiently establish control links between ensembles. In the figure, it is exemplified with bold bidirectional arrows. For example, we can imagine that, for some intrinsic reasons, the FlexiBus manager requests a certain route to change its itinerary (e.g., in order to accommodate another passenger).

The issue communication is sent to the corresponding role instance (arrow 1). Trying to resolve this issue, the route role instance rethrows the issue to the RM role instance within the same agent (arrow 2). Consequently, the RM role instance triggers negotiation within the route ensemble to understand the possible solutions. Finally, throw the same links the resolution options are returned back to the FlexiBus manager, who makes the decision. We can see that everything works similarly to the collective adaptation within an ensemble, but now the changes happen on the inter-ensemble scale.

It is worth to remark that inter-ensemble communication can be used not only for propagating decisions to the lower level of abstraction, but also to scale up an issue in case it cannot be resolve on the lower level. For example, if the RM instance cannot resolve bus delay alone, it may scale the problem up to the level of the FlexiBus system to possibly find a solution that engages other routes (arrows 4, 5, 6). By joining all the communications into a single picture, we can derive even a more complex scenario, where the bus delay is resolved on the level of FlexiBus system (arrows 4, 5, 6) by reassigning some passengers to another route (arrows 1, 2, 3).

Even though our example shows how our architecture can be used to build arbitrarily large hierarchical systems with flexible collective adaptation, its use is not limited to hierarchies and can be exploited to design any topology based on peer-to-peer links between elements.

3.4 Collective Adaptation Engine

We have released a standalone Java implementation of the Collective Adaptation Engine (CAE), approach described in the previous sections. It has been first released as a standalone component[2], and used in the *DeMOCAS* framework [10][3].

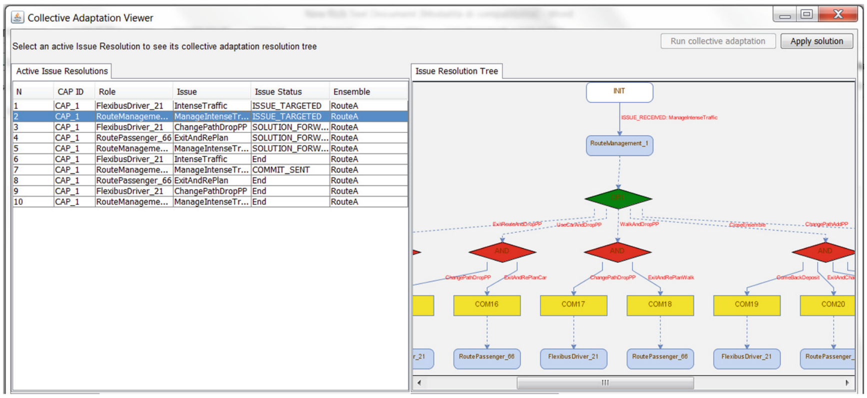
DeMOCAS is a framework for the modeling and execution of Collective Adaptive Systems (CAS). It includes mechanisms for services specialization and adaptation using the concept of Domain Objects [9]. This allows the system to model customizable and adaptable services. DeMOCAS is build around three main aspects:

– dynamic settings: each CAS is a collection of autonomous agents entering and exiting the system dynamically;
– collaborative nature of systems: agents can collaborate in groups (i.e., ensembles) for their mutual benefit;
– collective adaptation: multiple agents must adapt their behavior in concert to respond to critical run-time impediments.

In this framework, collective adaptation is used to handle unpredictable changes, which usually affect different running agents. Collective adaptation makes the system robust and resilient in the face of situations that could cause more rigid approaches to fail. The collective adaptation is performed by exploiting the handler and solver constructs, and by associating a MAPE (Monitor, Analyze, Plan, Execute) loop to each agent, as described in previous sections.

In Fig. 5 we show the *Collective Adaptation Viewer* of DeMOCAS. This screen capture shows a report issue resolution result for an *Intense Traffic Issue* triggered by a Flexibus Driver in mid-route. On the left side, all the agents involved in the issue resolution process are listed. The issue resolution tree of the Route Manager agents that owns the solver for the triggered issue, is shown on the right side.

[2] For the interested reader, the prototype is available in its entirety on a GitHub repository https://github.com/das-fbk/CollectiveAdaptationEngine.
[3] https://github.com/das-fbk/DeMOCAS.



Select an active Issue Resolution to see its collective adaptation resolution tree

N	CAP ID	Role	Issue	Issue Status	Ensemble
1	CAP_1	FlexibusDriver_21	IntenseTraffic	ISSUE_TARGETED	RouteA
2	CAP_1	RouteManageme...	ManageIntenseTr...	ISSUE_TARGETED	RouteA
3	CAP_1	FlexibusDriver_21	ChangePathDropPP	SOLUTION_FORW...	RouteA
4	CAP_1	RoutePassenger_66	ExitAndRePlan	SOLUTION_FORW...	RouteA
5	CAP_1	RouteManageme...	ManageIntenseTr...	SOLUTION_FORW...	RouteA
6	CAP_1	FlexibusDriver_21	IntenseTraffic	End	RouteA
7	CAP_1	RouteManageme...	ManageIntenseTr...	COMMIT_SENT	RouteA
8	CAP_1	RoutePassenger_66	ExitAndRePlan	End	RouteA
9	CAP_1	FlexibusDriver_21	ChangePathDropPP	End	RouteA
10	CAP_1	RouteManageme...	ManageIntenseTr...	End	RouteA

Fig. 5. Collective adaptation viewer.

This type of approach, where responsive hierarchies are loosely coupled into adaptive systems provides a war of addressing the types of system errors that can emerge when large numbers of components are tightly connected in geometrically complex networks. We discuss this in more detail in the following section.

4 Resilience Engineering for Normal Accidents

Charles Perrow established the concept of Normal Accidents [27] as property of complex, high-risk systems. These are unpredictable, yet inevitable combinations of small failures that build upon each other within an unforgiving environment. Normal accidents include catastrophic failures such as reactor meltdowns, airplane crashes, and stock market collapses. Though each failure is unique, all these failures have common properties:

– The system's components are *stiffly* connected. A change in one rapidly impacts one or more other components;
– The system is *densely* connected, so that the actions of one part affects many others, regardless of the speed of action;
– The system's internals are difficult to observe, so that failure can appear without warning.

The systems that live on the TaaS spectrum are complex - they consist of large numbers vehicles integrated in a complicated physical, electronic, and software webs. They are high risk, both at the individual level, as the vehicles themselves are inherently dangerous, and in a broader context, where misallocation of transport during an emergency could result in large scale suffering or death.

To see how easy it would be to create a single TaaS network, consider the case where blockchain-mediated transactions have become the exclusive payment scheme for transportation. There are many potential benefits for such a distributed payment system, among them a greater level of purchasing anonymity in an environment where every transaction can be tracked. But this decision to use a decentralized system means that every transaction has to have visibility to the distributed blockchain ledgers [15]. In essence, every node in the financial system is now closely coupled. If an accident occurs that breaks the network, financial transactions become impossible.

For example, consider a near-future case of a TaaS using self-driving vehicles that depends on blockchain in an island network created by an earthquake that has cut communication lines to the outside world. If the system were like the New York Stock Exchange (NYSE)[4], the entire system would suspend trading until the blockchain ledger servers could be reached, preventing evacuations. Or consider another example, where thousands of identical self-driving cars are subtly hacked so that they perceive squirrels as children in the street [24]. Adhering to the social consensus on trolley problem issues [26], thousands of self-driving cars crash into trees.

These particular accidents probably will not happen, but we can be confident that if the systems we design have Perrow's properties, something like them will. So how do we design systems for problems that are unknown? Ideally, the answer would be to ensure that any deployed systems are not densely and tightly connected, and that the elements control behavior are visible to those with the appropriate credentials.

It is not always easy to meet these three constrains short of legislation. But a proxy for addressing the stiffness and tightness of the connections in a *single* system is to ensure that *multiple, distinct* TaaS systems are always present in the communities they serve. Although each system may be dense, stiff hierarchy, the connections between the systems should be few and slack. This enforces a level of resilience at a minor cost in efficiency. Every vehicle and user doesn't need to be a individual competing across multiple markets, but neither should there only be one rigid hierarchy. We believe that *distributed ensembles is an appropriate compromise between responsiveness and resiliency.*

Acknowledgments. We'd like to thank Aaron Dant of ASRC Federal for his contribution to the direction and development of the market section of this paper.

References

1. Stock Exchange. Oxford University Press, oed online edn. http://www.oed.com.proxy-bc.researchport.umd.edu/view/Entry/190617?rskey=9zzLVE&result=2
2. Consumer spending on vehicles averaged 8427 in 2016, September 2017. https://www.bls.gov/opub/ted/2017/consumer-spending-on-vehicles-averaged-8427-in-2016.htm
3. Table 1. Median usual weekly earnings of full-time wage and salary workers by sex, quarterly averages, seasonally adjusted, October 2018. https://www.bls.gov/news.release/wkyeng.t01.htm

[4] https://www.nyse.com/.

4. Abeywickrama, D.B., Bicocchi, N., Zambonelli, F.: SOTA: towards a general model for self-adaptive systems. In: Reddy, S., Drira, K. (eds.) WETICE, pp. 48–53. IEEE Computer Society (2012)
5. Andres Figliozzi, M., Mahmassani, H., Jaillet, P.: Framework for study of carrier strategies in auction-based transportation marketplace. Transp. Res. Rec. J. Transp. Res. Board **1854**, 162–170 (2003)
6. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: A conceptual framework for adaptation. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 240–254. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28872-2_17
7. Bucchiarone, A., Dulay, N., Lavygina, A., Marconi, A., Raik, H., Russo, A.: An approach for collective adaptation in socio-technical systems. In: IEEE SASO Workshops, pp. 43–48 (2015)
8. Bucchiarone, A., Mezzina, C.A., Pistore, M., Raik, H., Valetto, G.: Collective adaptation in process-based systems. In: SASO 2014, pp. 151–156 (2014)
9. Bucchiarone, A., De Sanctis, M., Marconi, A., Pistore, M., Traverso, P.: Design for adaptation of distributed service-based systems. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 383–393. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_27
10. Bucchiarone, A., De Sanctis, M., Marconi, A., Martinelli, A.: DeMOCAS: domain objects for service-based collective adaptive systems. In: Drira, K., et al. (eds.) ICSOC 2016. LNCS, vol. 10380, pp. 174–178. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68136-8_19
11. C. Pinciroli et al.: ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In: IROS, pp. 5027–5034 (2011)
12. Cabri, G., Puviani, M., Zambonelli, F.: Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In: 2011 International Conference on Collaboration Technologies and Systems, CTS 2011, Philadelphia, Pennsylvania, USA, 23–27 May 2011, pp. 508–515 (2011)
13. Clearfield, C., Tilcsik, A.: Meltdown: Why Our Systems Fail and What We Can Do About It. Atlantic Books, Penguin Canada, 20 March 2018. https://books.google.it/books/about/Meltdown.html?id=46krDwAAQBAJ&redir_esc=y
14. Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 1017–1023 (1996)
15. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. Commun. ACM **61**(7), 95–102 (2018)
16. Far, B.H., Wanyama, T., Soueina, S.O.: A negotiation model for large scale multi-agent systems. In: Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006: Heuristic Systems Engineering, Waikoloa, Hawaii, USA, 16–18 September 2006, pp. 589–594 (2006)
17. Hölzl, M., Rauschmayer, A., Wirsing, M.: Engineering of software-intensive systems: state of the art and research challenges. In: Wirsing, M., Banâtre, J.-P., Hölzl, M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms. LNCS, vol. 5380, pp. 1–44. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89437-7_1
18. IBM: An architectural blueprint for autonomic computing. Technical report, IBM (2006)
19. Jones, A.T., McLean, C.R.: A proposed hierarchical control model for automated manufacturing systems. J. Manuf. Syst. **5**(1), 15–25 (1986)

20. Kirilenko, A., Kyle, A.S., Samadi, M., Tuzun, T.: The flash crash: high-frequency trading in an electronic market. J. Finan. **72**(3), 967–998 (2017)
21. Levi, P., Kernbach, S.: Symbiotic-Robot Organisms: Reliability, Adaptability, Evolution, vol. 7. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11692-6
22. Lima, D.: Uber caps prices ahead of Hurricane Irma's arrival, September 2017. https://www.bizjournals.com/southflorida/news/2017/09/07/ride-hailing-service-caps-prices-ahead-of.html
23. Marconi, A., Pistore, M., Traverso, P.: Automated composition of web services: the ASTRO approach. IEEE Data Eng. Bull. **31**(3), 23–26 (2008)
24. Mihajlović, M., Popoviċ N.: Fooling a neural network with common adversarial noise. In: 2018 19th IEEE Mediterranean Electrotechnical Conference (MELECON), pp. 293–296, May 2018. https://doi.org/10.1109/MELCON.2018.8379110
25. Nandiraju, S., Regan, A.: Freight transportation electronic marketplaces: a survey of the industry and exploration of important research issues (2008)
26. Noothigattu, R., et al.: A voting-based system for ethical decision making. CoRR abs/1709.06692 (2017). http://arxiv.org/abs/1709.06692
27. Perrow, C.: Normal Accidents: Living with High Risk Technologies-Updated Edition. Princeton University Press, Princeton (2011)
28. Popov, S.: The tangle, p. 131 (2016)
29. Quintero, R., Barbera, T.: A real-time control system methodology for developing intelligent control systems. Technical report (1992)
30. Roth, J.: The application of the hierarchy system to on-line process control. J. Br. Inst. Radio Eng. **24**(2), 117–125 (1962)
31. Saaty, T.L.: What is the analytic hierarchy process? In: Mitra, G., Greenberg, H.J., Lootsma, F.A., Rijkaert, M.J., Zimmermann, H.J. (eds.) Mathematical Models for Decision Support. NATO ASI Series, vol. 48, pp. 109–121. Springer, Heidelberg (1988). https://doi.org/10.1007/978-3-642-83555-1_5
32. Singh, M.G., Drew, S.A., Coales, J.F.: Comparisons of practical hierarchical control methods for interconnected dynamical systems. Automatica **11**(4), 331–350 (1975)
33. Tesfatsion, L.: Agent-based computational economics: modeling economies as complex adaptive systems. Inf. Sci. **149**(4), 262–268 (2003)
34. Vromant, P., Weyns, D., Malek, S., Andersson, J.: On interacting control loops in self-adaptive systems. In: IEEE/ACM SEAMS 2011, pp. 202–207 (2011)
35. Wellman, M.P.: Market-oriented programming: some early lessons. In: Clearwater, S.H. (ed.) Market-Based Control: A Paradigm for Distributed Resource Allocation, pp. 74–95. World Scientific, Singapore (1996)
36. Weyns, D., Malek, S., Andersson, J.: FORMS: unifying reference model for formal specification of distributed self-adaptive systems. TAAS **7**(1), 8 (2012)
37. Yuan, Y., Wang, F.Y.: Towards blockchain-based intelligent transportation systems. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pp. 2663–2668. IEEE (2016)
38. Zambonelli, F., Bicocchi, N., Cabri, G., Leonardi, L., Puviani, M.: On self-adaptation, self-expression, and self-awareness in autonomic service component ensembles. In: SASOW, pp. 108–113 (2011)
39. Zhong, C., DeLoach, S.A.: Runtime models for automatic reorganization of multi-robot systems. In: IEEE/ACM SEAMS 2011, pp. 20–29 (2011)