Collective Adaptation of Socio-Technical Systems

Antonio Bucchiarone*, Naranker Dulay†, Anna Lavygina†, Annapaola Marconi*, Heorhi Raik*, Alessandra Russo†
*Fondazione Bruno Kessler, Trento, Italy - {bucchiarone,marconi,raik}@fbk.eu
† Imperial College, London, UK - {n.dulay,a.lavygina,a.russo}@imperial.ac.uk

***Abstract*—Socio-technical systems are systems where autonomous humans and distributed computational entities collectively collaborate with each other and with the environment in order to satisfy their goals. To be resilient, socio-technical systems need to be able to adapt to the unexpected behaviours of humans as well as to exogenous changes in the environment. In this paper, we describe a novel framework for the development of social-technical systems where system adaptation is itself a collective process that is driven by awareness of the capabilities, goals, constraints and preferences of humans and entities, as well as knowledge of the environment. Our adaptation process is controlled by a multi-criteria decision making function that is combined with an analytic hierarchic process (AHP) to select best adaptation alternatives. The paper presents the formal model of our approach to collective adaptation and illustrates it for a smart mobility scenario supporting dynamically formed collectives of passengers, drivers, means of transportation, with service-based transportation providers.**

***Index Terms*—Socio-Technical Systems, Ensembles, Collective Adaptation, Multi-criteria Decision Making.**

I. Introduction

Socio-technical system has recently been introduced to denote systems that support humans as first-class entities and are capable of adapting to the subtle cause/effect loops that arise between computational behaviors and human behaviors [**?**],[**?**]. They are able to support entity self-adaptation, but also collective-adaptation where entities "opportunistically" join one or more *ensembles* that define how groups of entities should collectively adapt. This is essential in situations where the results of entity self-adaptions can adversely affect other entities. Ensembles provide a collective design-pattern that aims to ensure that collective adaptations lead to outcomes that are better than if each entity was left to self-adapt.

The work presented in this paper starts from the realisation that while collectiveness and adaptability are interesting when considered in isolation, they bring new challenges and better outcomes when considered together. Our concept of ensembles implies several key properties that include (i) the emphasis on supporting collaborations between human and computational entities, and (ii) the heterogenous nature of entities with respect to their capabilities, goals, preferences and behaviours. These properties distinguish our Socio-technical approach from other types of collective adaptive systems like swarms, where all elements of a community have a uniform behavior and a global shared goal [**?**], [**?**], and multi-agent systems [**?**], where there may be several distinct roles, but their behaviors are known and often predictable in advance.

In the paper we present a new approach to collective adaptation of socio-technical systems. An adaptation protocol is triggered within affected ensembles whenever an adaptation issue arises and results in entities, affected by the issue, to adapt with minimal impact on their own preferences. The ensembles issue adaptation protocol discovering which entities have the means the solve the issue and applies a distributed multi-criteria decision making function that is combined with an analytic hierarchic process (AHP) to select best adaptation alternatives, taking into account the individual preferences of the entities involved in the adaptation. The paper presents the formal model and algorithm used of our approach and illustrates it for a smart mobility scenario supporting dynamically formed ensembles of passengers, drivers, means of transportation, with service-based transportation planners and providers. This paper is organised as follows. Section II motivates the need for distributed and collective adaptation and discusses the challenges. Section III presents our solution which is based on the Ensemble concept. In Section IV we present the formal framework and the algorithm for collective adaptation that is applied and evaluated in Section V using a scenario in the urban mobility domain. We conclude the paper positioning it with respect to the related works in Section VI and discussing future work in Section VII.

II. Motivating Scenario and Research Challenges

Modern cities attempt to flexibly integrate transportation options for residents and visitors to use buses, trains, taxis, bicycles and cars. They play an important role in the economy of the city and the quality of life of its residents. In this paper 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 and are interested in situations where computational entities 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 UMSs.

A. Urban Mobility System

The 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 organises 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, and *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 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 essentially an ensemble composed of the vehicle (or FB driver) that is supposed to serve the route and passengers travelling within the same (or close) time and location span. 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 behaviour: passengers "sacrifice" part of their flexibility in order to travel cheaper, compared to a taxi, and quicker compared to conventional buses.

The following situations illustrate when a running FB route could trigger adaptation: one of its passenger is late for the bus; one of its passenger decides to no longer travel; the bus is damaged in an accident; the FB company decides to change the route in order to adapt to traffic conditions.

Even though, the cases above seem to be quite natural for any on-demand transportation service, tackling them is not always trivial. For example let us to consider the case when a bus is damaged or it is in a strong delay. In this case all passengers must be proposed alternatives and/or compensations. The solutions could be any or a combination of the following cases: 1) reassign passengers to other routes; or 2) reassign (groups of) passengers to other means of transportation (e.g., recombine the passengers into small groups and assign them to taxis or car pools).

B. Challenges

Even though entities 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: which coordinates the adaptation behavior of multiple entities (FB driver, passengers, and FB company) and in return gives them certain benefits (e.g., cheap and fast way of travelling).

Membership of an ensemble may temporarily reduce the flexibility of its entities. Within this context, isolated entity self-adaptation is not effective. We can easily imagine what happens if a passenger books a trip with a FB and then silently changes its mind and decides 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 probably extra charges for the cancelling 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 travelling through unsafe areas in the city, but a possible re-planned route may pass through such area.

In adaptive systems with collective behaviour new approaches for adaptation are therefore needed that allow (i) *multiple* entities 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 trivial at all, 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.

III. Overview of the Approach

A. Modeling Entities and the Ensemble LifeCycle

Our approach is based on the concepts of *entities* and *ensembles*. Entities (see Figure 1) are basic building blocks representing the different actors and components of the system (e.g. passengers, bus drivers, flexibus company). They have a repository of task models for the goals they may accomplish (e.g., meeting, trip, for the entity *person*). Task models may run in parallel creating sometimes also dependencies (e.g., the meeting task is related to the trip to reach the meeting point). Moreover, some goals are local (blue tasks inside *Person* entity), and others are achieved in collaboration with other entities (orange nodes) in the scope of the *ensemble* (e.g., joint trip using a FB Route) to which they belong.

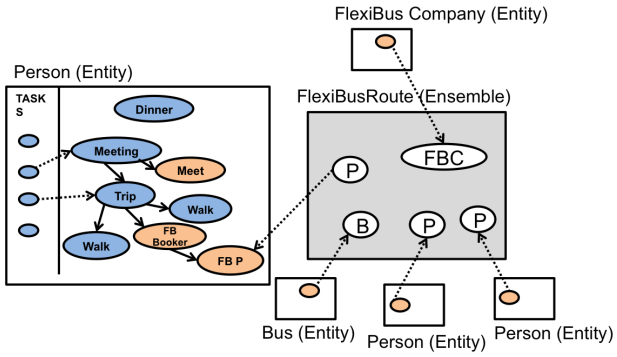

Fig. 1: Entity and Ensemble

In our view an ensemble defines a set of *roles* that can be played by participating entities, and an adaptation protocol that defines how conflicting goals are resolved in the scope of the ensemble.

The ensemble is created by some entity (the creator) whenever a need for collaboration emerges (e.g., FB route creation by the FB Company). It includes the specification of the different roles (i.e., passenger, FB driver and FB company) in terms of tasks that entities participating in it can execute. After creation, entities may dynamically join the ensemble taking any of the specified roles. Entities remain autonomous, that is, largely preserve their freedom of action, and continue operation within the ensemble, trying to achieve their goals. When an adaptation issue arises, it can be done autonomously

or can involve multiple entities that must adapt all together and transactionally. To avoid this last point some kind of negotiation must take place to decide on the changes to be applied on each side. Finally, entities exit the ensemble instance when their goals are achieved or when the participation in the ensemble is no longer beneficial.

B. Collective Adaptation

Although entities are autonomous and can self-adapt at any time, as participants in an ensemble they are able to adapt more efficiently or effectively by leveraging the adaptation capabilities of the other entities.

We extend the definition of an ensemble role presented in [**?**] introducing two concepts: *issue* and *solver*. An issue is used to define a critical situation that can happen to a role of an ensemble (i.e., flexibus delay for the role FB Company), while a solver reflects the ability of a role to handle certain types of issue (i.e., find a new means of transportation for the role FB Company). To resolve an issue our approach uses a procedure that includes two activities: *issue resolution* performed internally at an ensemble role, and *issue communication* performed by an ensemble role when it asks to other roles to resolve an issue.

The issue resolution protocol consists of four steps: (1) find a solution internally at the originating role; (2) see if the solution requires participation of other roles; (3) if so, communicate issues to all relevant roles and collect solutions from them; (4) if the solution does not require external activities, apply it, while if it requires external activities, commit only the best solution derived from the different roles. Looking at an example in the UMS scenario (see Figure 4), when the FB company must solve an issue (e.g., AssignPassengers) using its *FindNewTransport* solver, an issue communication *com3* is created. It detects all solvers (*ChangeRoute* from two alternative FB Drivers and *FindRide* from a CP company) that can handle the issue and sends them it. As soon as the solvers find a solution and send it back, the issue sender will decide which one is better and will commit it.

The issue resolution happening later in the FB2 Driver is more complex. When it receives the issue (i.e., AssignPassengers) via its *ChangeRoute* solver, it does the following: (1) issue resolution *IR4* is instantiated; the solution is calculated internally within FB2 Driver role (e.g., the new route); (2) the solution is interpreted in terms of issues to be raised: two passengers (PA and PB) are asked to change their FlexiBus assignment while the other two (PC and PD) are asked to change their trip details in terms of pickup point and scheduled time; (3) for each of these issues, an issue communication is created (from com4 to com7); (4) as soon as solutions are received from issue targets, the best solution is determined by the FB Company; finally (5) if the FB Company commits the solution, then the FB Company runs its own solution internally and it also commits the solutions for all the other roles.

To summarize, while most of the proposed solutions for collective adaptation work under the assumption that all the knowledge used to adapt a system is fully specified at design time (i.e., a predefined set of issues) and is centrally controlled by a specific component (i.e., a set of predefined solvers), our approach addresses collective adaptation problems in a decentralized fashion, at run-time, with new solvers that can be introduced at any time. At the same time, in highly dynamic and distributed environment, our approach provides a way to dynamically understand which parts of the system should be selected to help solve an adaptation issue while guaranteeing the highest utility for the roles involved.

C. Multi-criteria Decision Making

As mentioned in Section II one of the challenges of collective adaptation is the decision of collective changes that are beneficial to the participants involved in an adaptation process. In our approach, given a set of adaptation solutions and the preferences of the participants involved, the problem of finding the "best" adaptation solution is treated as a multi-objective problem where each objective (criteria) is the utility value of a solution for a participant. Utility of a solution for a participant are based on its preferences. For example, during delay issue resolution, the utility of a new trip for a passenger (PA or PB) can be defined by criteria like total travel time, trip price and walking distance.

A multi-objective function problem is typically solved by using a composite function that is a weighted sum of all objectives. This approach would, however, suffer of two major drawbacks [**?**]: (i) the values of the composite function are often difficult to interpret for complex problems with many criteria; and (ii) solutions are very much dependent on the weight-vectors and in different circumstances different weight-vectors have to be used. In this paper we use a different approach for solving a multi-objective function problem: we consider all objectives separately and use an *Analytic Hierarchy Process (AHP)* (see Section IV-B) for ranking alternative solutions [**?**]. The use of AHP allows evaluations to be performed for both qualitative and quantitative criteria, based on either subjective user opinion or actual objective measurements, and taking the relative importance of the criteria into account.

IV. Formal Model

A. Formal Model

Our model of collective adaptation is built around the concept of *ensemble*: it is a collection of autonomous entities which collaborate to perform certain tasks.

We introduce the notion of *entity* as a representation of a computational or human actor that can play multiple roles in different ensembles (i.e., a person entity can be a passenger of a FB Route ensemble or a driver in a CP Ride ensemble).

Definition 1 (Entity): An entity is defined by a set of roles it can play $y = \langle R \rangle$;

A role that an entity can play in an ensemble is primarily determined by the ways it collaborates with other roles. Collaboration consists in managing issues and responding to issues raised by others. As such, a role includes a set of issue types it can produce, and a set of solvers it provides. *Issues* generally correspond to different critical situations that can

happen to a role of an ensemble. Each issue type includes a set of parameters describing it:

Definition 2 (Issue Type): An issue type is defined by a set of parameters $u = uP$.

For example in the case of a FB driver, it can trigger an issue type $busDelay = \{delayTime, delayReason\}$.

Solvers reflect the ability of a role to handle certain issues. A solver defines a set of issue types it can handle and a set of constraints on issue parameters restricting the solver's capabilities:

Definition 3 (Solver Type): A solver type is a set of tuple $s = \{\langle u_1, uC_1\rangle, \ldots, \langle u_n, uC_n\rangle\}$, where u_i is an issue type this solver is compatible with and uC_i is a set of solver constraints (restricting parameters of acceptable issue type).

In our scenario the *FB Company* is able to find alternative transportation means (i.e., another FB route or a CP ride) when for example a FlexiBus is in delay. For this it has a solver type $FindNewTransport = \{\langle busDelay, \{NumOfDelayedPassengers\}\rangle\}$ where $busDelay$ represents the issue type it is able to solve, while $NumOfDelayedPassengers$ is a parameter that expresses the number of delayed passengers needed to solve the issue.

Each role is also able to express a set of preferences that will be taken into account during any collective adaptations where it is involved.

Definition 4 (Role Type): A role type is a tuple $r = \langle U, S, rS, rP\rangle$, where:

- U is a set of issue types that can be produced by a role;
- S is a set of solvers that are offered by a role;
- rS is a set of parameters that define the current state of a role,
- rP is a list of preferences available for a role.

In the UMS scenario, we can distinguish different roles: *Passenger*, *FB Driver*, *FB Company*, *CP Company*, etc.. Moreover, as we mentioned in Section II, passenger role may report delay and trip cancellation, this means $U = \{Delay, Cancellation\}$. Additionally, in order to realize the behavior described in the scenario, it is important for the passenger to provide solvers that would allow her to handle changes in their FB trip. $S = \{ChangePickUp, ChangeTime\}$ is about asking the passenger to approve changes in the assigned pickup point and in the scheduled time of her flexibus trip. The state of a passenger role can be described by the following parameters: $rS = \{Departure\,Point, Arrival\,Point, Departure\,Time, Arrival\ Time, Current\ Position\}$, and the following preferences are available for passenger role $rP = \{travel\,time, travel\,cost, walking\,distance\}$.

To represent collaboration of multiple roles we introduce the notion of *ensemble*. It describes a certain type of collective behaviour that may take place in the domain of interest.

Definition 5 (Ensemble Type): An ensemble type e is a set of role types R.

In our scenario the FB route and the CP ride are two examples of ensemble where $FB\ Route = \{Passenger, FBDriver, FBCompany\}$ and $CP\ Ride = \{Passenger, CPDriver, CPCompany\}$.

For most introduced concepts (i.e., issues, solvers, roles, ensembles), an instance can be defined as an object that is bound to a particular type and that additionally retains its state.

An issue instance corresponds to a particular situation occurring in an ensemble (e.g., flexibus delay that happened to a particular flexibus at a particular moment in time with a certain number of passengers waiting or already onboard). The issue instance belongs to an issue type and its state is determined by values assigned to issue parameters:

Definition 6 (Issue Instance): An issue instance is a tuple $ui = \langle u, L_u\rangle$, where u is an issue type, and $L_u : u.uP \to V$ is an assignment function for issue parameters.

Each role, collaborating in an ensemble, can provide one or more solvers. When it is invoked to solve a specific issue it runs a dedicated solver instance defined as follows:

Definition 7 (Solver Instance): A solver instance is a tuple $s_i = \langle s, ui, L_s\rangle$, where:

- s is the solver type;
- ui is the issue instance that the solver instance will solve and $ui.u \in s.U$;
- $L_s : s.uC \to V$ is an assignment function for solver parameters and V are all possible values that solver parameters can be assigned with;

A *role instance* is bound to a certain role type within an ensemble and its state is determined by its data (i.e., parameters and preferences) and by any ongoing issue resolution activities.

Definition 8 (Role Instance): A role instance is a tuple $ri = \langle ri_{id}, r, L_{ri}, P, \Uparrow_r\rangle$, where:

- ri_{id} is the role instance identifier;
- r is a role type;
- $L_{ri} : rS \to V$ is a function that assigns values to the state parameters;
- P represents preferences of a role instance: $P = \langle riP, \mathbf{C}\rangle$, $riP \subseteq rP$ is a list of preferences of a role instance, **C** is a pairwise comparison matrix of the relative importance of pairs of preferences (see section IV-B for more details);
- $\Uparrow_r$ is a set of active issue resolutions;

Finally, an ensemble instance is bound to some ensemble type and consists of role instances:

Definition 9 (Ensemble Instance): An ensemble instance is a tuple $ei = \langle ei_{id}, e, RI\rangle$, where ei_{id} is the ensemble instance identifier, e is an ensemble type and, RI is a set of role instances.

In our framework we have two types of activities that a role instance can execute during a collective adaptation problem resolution: *issue communication* and *issue resolution*. Issue communication is used to send an issue instance to a *target role instance* (see definition below) that is supposed to resolve it. The issue instance may be sent to multiple partners at a time in attempt to find a better solution. Issue communication comprises a few steps: 1) the issue is sent to all target roles; 2) the replies are received from the partners able to resolve the

issue; 3) the preferable solution is chosen; 4) the preferable solution is committed. Formally, a target role instance and issue communication are defined as follows:

Definition 10 (Target Role Instance): A target role is a tuple $t = \langle ri_{id}, s_i, p \rangle$, where:

- ri_{id} is the target identifier (role instance id);
- s_i is the solver instance invoked to solve the issue $s_i.ui$;
- p is the solution proposed by the target. It is a process that the target role will execute if it will become part of the overall issue resolution.

Definition 11 (Issue Communication): An issue communication is a tuple $\uparrow_u = \langle ui, T \rangle$, where ui is an issue instance communicated, and T is a set of target roles;

While the issue communication is a way to propagate resolution activities between partners, *issue resolution* corresponds to the high-level model of internal elaboration being done by role instances. In particular, we assume that the issue instance may either arise internally (when the issue originally occurs in this role instance) or is received by one of the role instance solvers. As soon as the issue instance is raised, the role instance may either resolve it locally or propagate issues to the other roles as a part of the resolution procedure. The issue resolution is formally described as follows:

Definition 12 (Issue Resolution): An issue resolution is a tuple $\uparrow_r = \langle ri_{id}, ui, \Psi \rangle$, where:

- ri_{id} is the identifier of the role instance, from which the issue instance arrived (null if the issue aroused internally);
- ui is an issue to be resolved;
- Ψ is a set of alternative solutions, each is a tuple $\psi = \langle ui, \Uparrow_u^O, p_{ext}, p_{int} \rangle$ where ui is the issue to be resolved, $\Uparrow_u^O$ is a set of outgoing issue communications, p_{ext} is a process (solution) that is supposed to be sent to the role instance associated with ri_{id}, while p_{int} is the internal process (solution) for the specific issue instance arrived.

If the resolution is fully local, in each solution $\psi \in \Psi$ the set of $\Uparrow_u^O$ of the outgoing issue communications is empty. Otherwise, $\Uparrow_u^O$ correspond to the communication of all subissues that must be resolved in order to resolve the original issue.

Following the previous example, in Figure 4 we show that the issue resolution procedure within an ensemble can be represented as a tree, which we call *issue resolution tree*. Indeed, the resolution procedure always starts from creating an issue resolution. It may instantiate further issue communications to resolve subissues. In turn, each issue communication may target a few role instances, each of which consequuently initiates an issue resolution and so on. In the figure rectangle nodes with blue balloons correspond to issue resolutions (AND and OR diamonds are internals of issue resolution and are not separate nodes), yellow rectangle nodes correspond to issue communications and edges correspond to relations between them (children of an issue resolution are its issue communications and children of an issue communication are issue resolutions initiated by its targets).

Definition 13 (Issue Resolution Tree): An issue resolution tree is a tree $\mathcal{T} = \langle n_r, \mathcal{N}_r, \mathcal{N}_c, \mathcal{L} \rangle$ (where $\mathcal{N} = \mathcal{N}_r \cup \mathcal{N}_c$ are tree nodes made up by issue resolutions ($\mathcal{N}_r$) and issue communications ($\mathcal{N}_c$) such that $\mathcal{N}_r \cap \mathcal{N}_c = \emptyset$, $n_r \in \mathcal{N}$ is a tree root and $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$ are parent-child links between nodes) that has the following properties:

- $n_r \in \mathcal{N}_r$, i.e., root is issue resolution;
- $n \in \mathcal{N}_c \rightarrow \exists (n, l, n') \in \mathcal{L}$, i.e., leaves are always issue resolutions;
- $n \in \mathcal{N}_c \rightarrow \forall (n, l, n') \in \mathcal{L} : n' \in \mathcal{N}_r$ and $n \in \mathcal{N}_r \rightarrow \forall (n, l, n') \in \mathcal{L} : n' \in \mathcal{N}_c$, i.e., all children of an issue resolution are issue communications and all children of an issue communication are issue resolutions;

An issue resolution tree is a very intuitive abstraction for understanding and analyzing how our approach works.

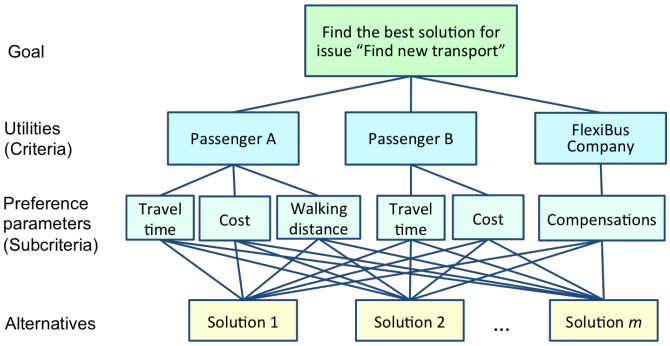

Fig. 2: AHP hierarchy

B. The Analytic Hierarchy Process

During the issue resolution process, the role instance that triggers the issue collects alternative solutions provided by the issue solvers (i.e., target role instances). Once alternative solutions are obtained, the role instance has to evaluate and rank them according to a number of criteria in order to find the best alternative, which will be used as an issue resolution.

We use the *Analytic Hierarchy Process* (AHP) in [**?**] for issue solutions ranking. AHP is a multi-criteria decision making approach that allows ranking and decisions to be made based on priorities using pairwise comparisons. The AHP works as follows. Given *n* evaluation criteria, and *m* alternative solutions that have to be ranked according to these criteria. First, weights of criteria are defined; higher weights correspond to more important criteria. Weights are calculated based on pairwise comparisons of the importance of criteria. Then, all alternatives are compared pairwise with respect to *each criterion separately*. Finally both weights of criteria and alternatives are synthesised to give final scores of alternatives that allows them to be ranked. The alternative with the highest score is used as the problem solution.

1) Criteria Evaluation: The first step in the AHP involves decomposition of the problem into a hierarchy of criteria and alternatives.

For the problem of finding the best issue solution we consider the utilities of the solution for each involved role instance as criteria for decision making. Each utility criterion can be broken down into a set of subcriteria that reflects preferences of the role instance - parameters that the role

Intensity of importance	Definition
1	Equal importance
3	Moderate importance
5	Essential or strong importance
7	Very strong importance
9	Extreme importance (the highest possible)
2, 4, 6, 8	Intermediate values
1.1, 1.2,1.3,...	Very close importance

TABLE I: Scale of relative importance of preferences

instance wants to maximize or minimize. Each role instance can define its own set of preferences as a subset of preferences available for its role. For example, for the "Find new transport" issue of the UMS scenario described above, the utility of a new trip for passenger A can be defined by such preferences (criteria) as total travel time, travel cost and walking distance, and utility of passenger B - by travel time and travel cost only.

Figure 2 shows the AHP hierarchy for issue "Find new transport", where the goal is to find the best journey alternative for passenger A and passenger B when their FlexiBus cannot proceed with the current trip. The utilities of passenger A, passenger B and the FlexiBus Company are taken into account as criteria when trying to find the best issue solution.

Once the hierarchy is built, the weights of the utilities of the participants and their preferences have to be defined.

Local weights of the role instance preferences represent the relative importance of the preferences for the role instance. To compute the local weights of the preferences, a matrix **C** of pairwise comparisons of preferences must be created. The matrix $C = (c_{jk})$ is of dimension $n \times n$, where *n* is the number of preferences and each element c_{jk} is the importance of the *j*th preference relative to the *k*th preference. The elements c_{jk} satisfy the constraint

$$c_{jk} \times c_{kj} = 1, \tag{1}$$

where $c_{jk} > 1$ indicates that the *j*th preference is more important than the *k*th preference. Consequently, in the case where the *j*th preference is less important than *k*th preference, we have $c_{jk} < 1$, and if the two preferences are indifferent we have $c_{jk} = 1$; which also implies that $c_{jj} = 1$. Saaty [**?**] suggests a numerical scale between 1 and 9 to express the importance of one decision criterion over another (see Table I) that we use to define the relative importance of the preferences.

Each role instance specifies its own list of preferences and the comparison matrix of preferences importance as specified in Definition 8.

Once the matrix **C** has been established, it can be used to derive the *local preference weight vector for the role instance* **w** using the equation

$$w_j = \frac{\sum_{l=1}^{n} \bar{c}_{jl}}{n} \tag{2}$$

where $\bar{c}_{jl} = c_{jl} / \sum_{k=1}^{n} c_{kl}$ is the normalized relative importance.

Similarly, if any role instances involved in issue resolution are more preferable than the others, different weights can be assigned to the utilities of the role instances. In this case, the role instance that triggers the issue (e.g. FlexiBus Company for issue "Find new transport") has to calculate these weights.

Once weights of the utilities and local preference weight vectors for all role instances are defined, the *global weights* of all preferences used in issue resolution can be calculated by multiplying their local weights by the weight of the corresponding utility:

$$\mathbf{g} = (w_1^0 \cdot w_1^1, w_1^0 \cdot w_2^1, ..., w_1^0 \cdot w_{m_1}^1, w_2^0 \cdot w_1^2, ...w_n^0 \cdot w_{m_n}^n) \tag{3}$$

where **g** is the global preference weight vector, $\mathbf{w}^0 = (w_i^0), i \in [1, n]$ is the weight vector of utilities, n is the number of role instances, $\mathbf{w}^i = (w_j^i)$ is the local preference weight vector of the *i*th role instance, m_i is the number of preferences of the *i*th role instance.

For example, let us assume that the preferences of passenger A include travel time, cost and walking distance. Passenger A defines cost as much more important than both travel time and walking distance, travel time as equally important to walking distance:

$$\mathbf{C} = \begin{pmatrix} 1 & 1/7 & 1 \\ 7 & 1 & 7 \\ 1 & 1/7 & 1 \end{pmatrix}$$

According to equation 2, the local preference weight vector of passenger A is $\mathbf{w}^1 = (0.11, 0.78, 0.11)$.

Assuming the local preference weight vector of passenger B is $\mathbf{w}^2 = (0.75, 0.25)$ (travel time is slightly more important than cost), for the FlexiBus Company is $\mathbf{w}^3 = (1)$, and the weight vector of utilities is $\mathbf{w}^0 = (0.43, 0.43, 0.14)$ (utilities of passengers are slightly more important than utility of the FlexiBus Company, but equally important to each other). Given this, the global preference weight vector $\mathbf{g} = (0.048, 0.333, 0.048, 0.321, 0.107, 0.143)$.

2) Solution Ranking: At this stage we calculate the scores of the alternative issue solutions with respect to each preference defined for the issue. To derive these scores we calculate a matrix of pairwise comparisons of alternatives $\mathbf{B}^j = (b_{ih}^j)$, where b_{ih}^j is the evaluation of the *i*th alternative compared to the *h*th alternative with respect to the *j*th preference.

Let x_i^j and x_h^j be the values of the *j*th preferences (e.g. travel time or cost) for issue solution alternatives *i* and *h* respectively.

If the *j*th preference must be maximized, then for all alternatives *i* and *h* with $x_i^j \geq x_h^j$, the element b_{ih}^j is computed by

$$b_{ih}^j = 8\frac{x_i^j - x_h^j}{x_{max}^j - x_{min}^j} + 1 \tag{4}$$

where x_{max}^j and x_{min}^j are the maximum and minimum values of the *j*th preference.

Similarly, if the *j*th preference has to be minimized, then for all alternatives *i* and *h* with $x_i^j \leq x_h^j$, the element b_{ih}^j is computed by

$$b_{ih}^j = 8\frac{x_h^j - x_i^j}{x_{max}^j - x_{min}^j} + 1 \tag{5}$$

Similar to the matrix **C**, the elements of $\mathbf{B}^j$ have to satisfy the constraint

$$b_{ih}^j \times b_{hi}^j = 1 \tag{6}$$

Having obtained $\mathbf{B}^j$, we can now calculate the score vectors $\mathbf{y}^j$ of trip alternatives with respect to each criteria $j \in [1, n]$. This calculation is done using Equation 2 but replacing the terms c_{jl} with b^j_{ih}.

The score vectors are then used to create the score matrix $\mathbf{Y} = [\mathbf{y}^1, \mathbf{y}^2, ..., \mathbf{y}^n]$.

Finally, the vector of global scores $\mathbf{v} = (v_i), i \in [1, n]$, can be calculated by

$$\mathbf{v} = \mathbf{Y} \cdot \mathbf{g} \tag{7}$$

The issue solution alternative with the highest global score is used as the overall solution of the issue resolution problem.

C. Issue Resolution Algorithm

To evaluate the feasibility of our approach, in Figure 3 we abstractly define an algorithm that covers the procedures for issue resolution and commitment (functions `resolve` and `commit` respectively). Function `resolve` is used recursively to trigger a distributed resolution procedure across multiple role instances within an ensemble. It takes as input a target role (Definition 10). The function is called locally by the role instance that originally detected a problem. Further recursive calls are propagated using a *Remote Procedure Call* (line 9). The function includes the following important steps:

Lines 2-3. The resolution issue is instantiated and its solutions generated. Function `build_solutions` is beyond the scope of this paper but may generally exploit various role-specific and domain-specific solvers;

Lines 4-9. For each solution returned, a set of subissues is identified (function `derive_coms` derives all subissues that must be resolved for a given solution in form of corresponding issue communications). For each issue communication, the set of potential solvers is identified across all reachable role instances (function `find_targets`). Finally, to understand how well the targets can handle subissues, the `resolve` function is called remotely on the targets (line 9);

Line 10. Once the solutions to subissues are obtained from the remote role instances, the Analytic Hierarchy Process (Section IV-B) is executed to identify the best solution ψ_{best};

Lines 11-17. If the current role instance is not the resolution tree root (i.e., $t_0.ri_{id}$ is not null), the issue resolution is stored locally (function `store`) and the calling role instance ($\psi_{best}.p_{ext}$) is returned. If the current role instance is the resolution tree root (line 15), `commit` is executed locally (target t_0). Function `commit` enacts a *distributed commit* of the best solution. It takes as input the target role corresponding to the issue to be resolved. It includes the following steps:

Line 18. Function `retrieve` is the opposite to function `store` on line 12;

Line 19. Function `extract_best_targets` extracts target roles corresponding to a given solution (best solution ψ_{best} in our case);

Lines 20-21. Commit is called for each of the target roles corresponding to the best solution. Commits are asynchronous, so as not to impede the solution execution on line 22;

Line 22. Role instance executes the internal process corresponding to the best solution.

```
1  function resolve(t_0)
2      ↑_r = <t_0.ri_id, t_0.ui, ∅>
3      ↑_r.Ψ := build_solutions(↑_r)
4      foreach ψ ∈↑_r.Ψ.solutions:
5          ψ.⇑^O_u := derive_coms(ψ)
6          foreach ↑_u∈ ψ.⇑^O_u
7              ↑_u.T := find_targets(↑_u)
8              foreach t ∈↑_u.T
9                  t.ψ = rpc(t.ri_id, resolve, t)
10     ψ_best := AHP(↑_r)
11     if t_0.ri_id != null
12         store(t_0, ψ_best)
13         return ψ_best.p_ext
14     else
15         commit(t_0)
16
17 function commit(t_0)
18     ψ_best := retrieve(t_0)
19     T_best := extract_best_targets(ψ_best)
20     foreach t ∈ T_best
21         commit(get_my_riid(), t)
22     execute(ψ_best)
```

Fig. 3: Issues Resolution Algorithm.

V. Evaluation

A. UMS Scenario Execution

In this section we show how our approach to collective adaptation works in the case of the UMS scenario. Figure 4 depicts the overall issue resolution tree. FB route and the CP ride are ensembles FB1 is an instance of FB Route and includes two passenger role instances PA and PB traveling from Trento to Verona airport and the FB1 Driver. FB2 and FB3 are two more FB Route ensemble instances with the same destination but different starting points and different routes. FB2 (with driver FB2 Driver) starts from Merano and arrives at Verona Airport collecting passengers PC and PD. FB3 (with driver FB3 driver) starts from Rovereto and arrives at Verona airport collecting passenger PE. Finally we have an instance of a car pool ride ensemble composed of a driver CP1 and passenger PG. The preferences of the various roles used in this example are depicted in Table II.

When a *FB1 Driver* experiences a delay, that compromises the trip requirements of its passengers (PA and PB), she instantiates two issues: $busDelay_1$ and $TrafficReport_1$. While in the first issue PA and PB are strongly related, the second issue does not involve any role instance and can be resolved internally to the UMS System (i.e., traffic info report). To resolve the $busDelay_1$ issue, the FB1 Driver will create an issue resolution structure like $IR1 = \langle \{\}, busDelay_1, \{com_1\}, init \rangle$.

At this point, the system has to perform the issue communication. For that purpose, all ensemble partners are examined for the solvers that can resolve this type of issue with this parameters. These solvers are checked not only for functional compatibility but also for parameter compatibility (e.g., the delay time constraint of the solver should be greater that the actual delay time for the issue). Let's assume that the relevant solver is found in the $FB\ Company$, and so the communication is specified like this: $com_1 = \langle busDelay_1, \{\langle FBC_1, FindNewTransport, null \rangle\} \rangle$.

$FB1\ Driver$ waits unitil it recieves the solution from its target. After the solution is received, $FB1\ Driver$

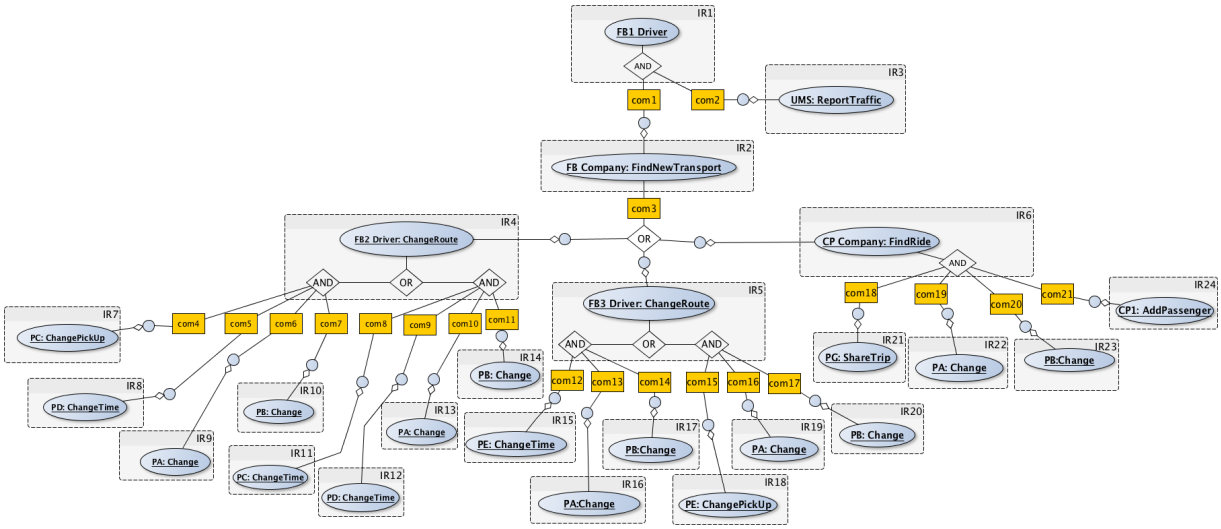

Fig. 4: Collective Adaptation Resolution in the UMS

makes decision and commits it (communicates the decision to the target selected). In the case of com_1 the resolution is more complex. The $FindNewTransport$ solver of the $FBCompany$ will build an internal solution from it internal adaptation procedure to manage the issue and an issue that it triggers. In this case the FBC_1 will trigger an issue $AccommodatePassengers_1$. The related Issue resolution $IR2$ instantiated looks like this: $IR2 = \langle\{FB\ Company\}, AccommodatePassengers, \{com_3\}, \emptyset\rangle$.

At this point, the system has to perform the issue communication $com3$ and its relevant solvers are found in $FB2\ Driver$, $FB3\ Driver$, and $CP\ Company$.

The $ChangeRoute$ solvers of the $FB2\ Driver$ and $FB3\ Driver$ and the $FindRide$ solver of the $CP\ Company$ will build their internal solution. Both $FB2\ Driver$ and $FB2Driver$ will produce two alternative solutions, respectively for $IR4$ and $IR5$ while $CPCompany$ only one for $IR6$. Each solution will trigger other issues that will be used to propose to each passenger involved in the collective adaptation, that they change the scheduled time (i.e., PD:ChangeTime), the assigned pickup point (i.e., PC:ChangePickUP) or the flexibus assigned (i.e., PB:Change).

When the issues in the bottom part of the tree are resolved, each corresponding solution will be sent to the upper layer (i.e., IR4) that will decide what its the best solution. This process will continue up to the root node that will decide and commit its overall solution. As we have seen in this example we can have multiple points in the tree where a decision should be taken among possible alternative solutions.

B. Decision Making

In this section we show how the Analytic Hierarchy Process is used in our algorithm for each specific issue resolution in the UMS scenario. The *FB2 Driver:ChangeRoute* solver generates two alternative solutions (new routes) in *IR4* . The first route (solution 1) requires Passenger *PC* to change the pickup location (IR7) and passenger *PD* to change time of the trip (IR8). The second route (solution 2) requires changing the time of the trips for both passengers *PC* and *PD* (IR 11 and IR12). In both cases *FBCompany* gives passengers *PC* and *PD* some compensation (e.g. a discount, free future trip).

Preferences of passengers *PA*, *PB*, *PC* and *PD* (see Table II for preferences) have to be taken into account when deciding which resolution out of these two is more preferable than the other. For *FB2 Driver* the utilities of its passengers are 5 times more important than the utilities of passengers *PA* and *PB* (see Table I), and the utility weight vector is $\mathbf{w}^0 = (0.083 \quad 0.083 \quad 0.417 \quad 0.417)$ based on Equation 2.

Scores of the solutions with respect to the preferences of the passengers were calculated based on Equation 5. Corresponding parameters of the solutions are given in Table III.

The final vector of global scores is calculated based on Equation 7. The global scores of solution 1 and solution 2 are 0.539 and 0.461, respectively. Therefore, solution 1 is more preferable, thus is returned as resolution of *IR4* issue by solver *FB2 Driver:ChangeRoute*.

FB3 Driver:ChangeRoute. This solver generates solution 3 (requires passenger *PE* to accept the trip time change (IR15)) and solution 4 (requires passenger *PE* to change the pickup point (IR18)). In both cases passenger *PE* is offered compensation.

This solver has to consider preferences of passengers *PA*, *PB* and *PE*. Similar to *FB2 Driver:ChangeRoute*, the utility of the passenger *PE* is more important than the utility of passengers *PA* and *PB*, and the utility weight vector is $\mathbf{w}^0 = (0.143 \quad 0.143 \quad 0.714)$.

The vector of global scores of solutions 3 and 4 is $\mathbf{g} = (0.679 \quad 0.321)$. *FB3 Driver:ChangeRoute* thus returns solution 3.

CP Company: Find Ride. Generates one solution (solution 5), thus no decision making is needed and solution 5 is used as a resolution of *IR6* issue.

Role Instance	Preferences P
PA	*riP*={time,cost,walking}, **C**=(1 0.143 1; 7 1 7; 1 0.143 1)
PB	*riP*={time,cost}, **C**=(1 3; 0.333 1)
PC	*riP*={time,cost,walking}, **C**=(1 7 3; 0.143 1 0.333; 0.333 3 1)
PD	*riP*={cost,walking}, **C**=(1 1; 1 1)
PE	*riP*={time,cost,walking}, **C**=(1 1 0.2; 1 1 0.333; 5 3 1)
FBCompany	*riP*={compensations}, **C**=(1)

TABLE II: Preferences of role instances

Solver	Solutions	Solution parameters
FB2Driver: ChangeRoute	Solution 1	*PA*: time=30min, cost=0€, walking=500m *PB*: time=30min, cost=0€ *PC*: time=30min, cost=-1€, walking=500m *PD*: cost=-1€, walking=400m
	Solution 2	*PA*: time=20min, cost=0€, walking=100m *PB*: time=20min, cost=0€ *PC*: time=40min, cost=-1€, walking=250m *PD*: cost=-1€, walking=400m
FB3Driver: ChangeRoute	Solution 3	*PA*: time=15min, cost=0€, walking=500m *PB*: time=15min, cost=0€ *PE*: time=15min, cost=-1€, walking=0m
	Solution 4	*PA*: time=20min, cost=0€, walking=100m *PB*: time=20min, cost=0€ *PE*: time=10min, cost=-1€, walking=500m
FBCompany: FindNew-Transport	Solution 1	*PA*: time=30min, cost=0€, walking=500m *PB*: time=30min, cost=0€ *FBCompany*: compensation=2€
	Solution 3	*PA*: time=15min, cost=0€, walking=500m *PB*: time=15min, cost=0€ *FBCompany*: compensation=1€
	Solution 5	*PA*: time=15min, cost=5€, walking=0 *PB*: time=15min, cost=5€ *FBCompany*: compensation=10€

TABLE III: Solution parameters for role instances preferences

FB Company: Change Route. Collects resolutions from solvers *FB2 Driver:ChangeRoute*, *FB2 Driver:ChangeRoute* and *CP Company: Find Ride*. It has to choose the best solution from solution 3,5 and 7 taking into account the utilities of Passengers *PA* and *PB* and the *FB Company*. It defines utilities of passengers as slightly more important (intensity of importance = 3) than the utility of the *FB Company*, thus $\mathbf{w}^0 = (0.429 \quad 0.429 \quad 0.143)$.

The final vector of global scores for solutions 1, 3 and 5 is $\mathbf{g} = (0.285 \quad 0.470 \quad 0.245)$. Therefore, *FB Company: Change Route* returns solution 3 as the most prefereable with Passengers *PA* and *PB* assigned to *FB3* and passenger *PE* asked to change the time of her trip.

C. Algorithm Complexity

In this section we evaluate the complexity of the solution search algorithm and outline general ways to optimize it. We use the following conventions: R – the total number of role instances; E – the average number of role instances in an ensemble; S, C, T – the average number of solutions per issue, subissues per solution and targets per subissue respectively; M the average cost of a peer-to-peer communication between role instances. The further discussion relies on the concept of issue resolution tree (Def. 13, Fig. 4) and the issue resolution algorithm (Section IV-C).

The first important observation that we make is that the issue resolution tree may grow infinitely. Even if R is finite, a single role instance may run infinite number of issue resolutions within the same tree. For instance, A sends an issue to B, and while resolving it B triggers subissue and sends it back to A, and so on, thus forming a cycle. A natural (and reasonable) way to prevent this behaviour is to restrict the number of issue resolutions per role instance per issue resolution tree to one or, alternatively, to introduce cycle detection (e.g., by tracing the ID of the root issue). This restrict the tree size and also makes it easier to guarantee consistent behaviour for a role instance (multiple cross-dependent issue resolutions executed by a single role instance for the same tree make it hard to ensure solution consistency).

To make an issue resolution tree simpler to analyze, we collapse together every issue resolution node with its issue communications. In the resulting tree, a node corresponds to a single role instance, and its children correspond to role instances immediately targeted by this node to resolve subissues (edges are communications between nodes). The number of nodes in such a tree is $O(R)$ (linear to the total number of role instances). Similarly, each role instance (tree node) has to be communicated by a parent role instance exactly once (except for the root), so the number of communications is also $O(R)$. To evaluate the overall solution search time, we have to evaluate the time complexity of a single inter-node communication, and of a single run of an issue resolution. While the first is initially defined as C, the second is obtained by analyzing function `resolve` in issue resolution algorithm.

At this point, it is important to notice that the average size of an ensemble (E) generally does not depend on the total number of role instances (R) (indeed, the number of passengers in the route does not depend on the population of the city and the number of means of transportation changes insignificantly with the size of the city). Since, the functions `build_solutions`, `derive_coms`, `AHP`, `find_targets` always operate on the scale of a single ensemble, thier worst case running complexity can be treated as constant on the scale of the whole system (even if these functions have exponentioal complexity $O(2^E)$ and for reasonable systems $E << R$, the overall complexity can be evaluated to some constant maximum X). Consequently, the number of calls to `build_solutions` is $O(1)$ (constant), to `derive_coms` is $O(S)$, to `find_targets` is $O(S*C)$, to `rpc` is $O(S*C*T)$ and to `AHP` is $O(1)$. Since the O-notation eliminates the constant factor, the overall complexity of `resolve` is $O(S*C*T)$ or $O(B)$, where $B = S*C*T$ equals the average number of children in our tree with collapsed nodes, and is known as a *tree branching factor*.

With total number of nodes $O(R)$ and the need to perform one issue resolution calculation and one communication per node, the overall calculations needed to run an issue resolution tree is $O(R*(B+C))$. However, in systems with virtually unlimited computing and networking capacities (such as an elastic cloud or a network of nodes posessing computing power, e.g., a network of smartphones), the real delay between a problem detection and the moment when a solution is found is much less. Indeed, the branches of the issue resolution tree may be explored simultaneously: if a node sends subissues to

several targets, the communication and the calculations on the target nodes go in parallel. And so, the overall delay dependes on the height of the tree $O(log_B R)$ rather than its size. The search delay is then $O(log_B R * (B + C))$, which proves the system to be highly scalable (degraded trees with $B \approx 1$ are very unlikely and can be eliminated from consideration).

We remark that since constant factor X may be big it may still make search delays too long (although the complexity class remains the same). One way to treat it is to restrict the tree height. In our preliminary experiments with the smart mobility domain suggest that it does not make sense to consider a resolution tree beyond level 5, since this involves too many role instances in resolving an issue and such a solution is unlikely to be acceptable.

VI. Related Work

Notions of collective adaptive systems have been presented in the literature in various different forms. In [**?**] Agents can participate to several coalitions at the same time and an adaptation mechanism is in charge of adjusting agents in a current coalition to minimise the agent penalty when it joins new coalitions. This approach is somewhat similar to our approach where role instances can be members of multiple ensemble instances. In our approach, any role instance can triggers adaptations and alternative collective adaptations are automatically identified to maximise the preferences of role instances involved in a current ensemble.

A coalition of coordinating agents can also be seen as a *choreography*. Several work have faced the problem of adaptation in service choreography [**?**], [**?**]. Choreography reconfigurations correspond to the addition or removal of some interactions or a simplification of the original choreography. Although these existing work provides a mechanism for deciding when the reconfiguration can take place, it does not suggest how to synthesize the new choreography in order to deal with the changes of the environment. Moreover, contrary to our approach the adaptation solution are defined at design time. Closely related to our approach is the notion of *ensemble*, as group of interacting agents, presented in [**?**]. In this work a formal language, called SCEL, supports abstractions for autonomic systems in terms of behaviors, knowledge, aggregation and policies. In SCEL it is possible to define an ensemble as a set of components, and the choice of which components to include at runtime is based on the satisfaction of certain predicates. [**?**] presents an ensemble based component model in which components can bind and communicate only via an ensemble. [**?**] gives instead a formal foundation for ensemble modelling, according to which an ensemble is defined in terms of roles and role connectors. A role can be seen as a meta-component (or a type) that can be instantiated by components of different types. Runtime behaviours of an ensemble is given by means of an automaton.

VII. Conclusion and Future Work

We have presented an approach for the collective adaptation of socio-technical systems. We have defined an algorithm to solve adaptation issues within an ensemble, which discovers which entities have the means to solve an issue and to apply adaptation with minimal impact on their own preferences. Finally we have evaluated the complexity and applicability of the algorithm using a scenario in the smart mobility domain. An important element of our near-term future work is the implementation of the UMS example as an ensemble system, which can be experimented with and evaluated. We will pursue that example towards a comprehensive case study and demonstrator in field conditions. To improve the approach further, the following aspects can be considered: (i)*Limiting response time*: this will allow to speed up the adaptation process by terminating branches of the resolution tree that require much time for evaluation, (ii)*Setting expiration time of the resolutions*: this will prevent the cases of the adaptation failure when the resolutions become unavailable if the decision maker takes too long to make a decision, and (iii) *Using the "common good"* concept as an additional criteria to compare solutions received from the solvers.

Acknowledgment

This work is partially funded by the 7th Framework EU-FET project 600792 ALLOW Ensembles.

References

[1] A. Bucchiarone, C. Mezzina, M. Pistore, H. Raik, and G. Valetto. Collective adaptation in process-based systems. In *SASO 2014*, pages 151–156, 2014.

[2] C. Pinciroli et al. Argos: A modular, multi-engine simulator for heterogeneous swarm robotics. In *IROS*, pages 5027–5034, 2011.

[3] R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *TAAS*, 9(2):7, 2014.

[4] Mila Dalla Preda et al. Developing correct, distributed, adaptive software. *Sci. Comput. Program.*, 97:41–46, 2015.

[5] T. Bures et al. DEECO: an ensemble-based component system. In *CBSE 2013*, pages 81–90, 2013.

[6] Leonardo A. F. Leite etl al. A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, 7(3):199–216, 2013.

[7] B. H. Far, T. Wanyama, and S. O. Soueina. A negotiation model for large scale multi-agent systems. In *IRI*, pages 589–594, 2006.

[8] R. Hennicker and A. Klarl. Foundations for ensemble modeling - the helena approach - handling massively distributed systems with elaborate ensemble architectures. In *Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi*, pages 359–381, 2014.

[9] J. Hillston, J. Pitt, M. Wirsing, and F. Zambonelli. Collective Adaptive Systems: Qualitative and Quantitative Modelling and Analysis (Dagstuhl Seminar 14512). *Dagstuhl Reports*, 4(12):68–113, 2015.

[10] William Ho and et al. Multi-criteria decision making approaches for supplier evaluation and selection: A literature review. *European Journal of Operational Research*, 202(1):16–24, 2010.

[11] M. Holzl, A. Rauschmayer, and M. Wirsing. Engineering of software-intensive systems. In *Software- Intensive Systems and New Computing Paradigms*, volume 5380 of *LNCS*, pages 1–44. Springer, 2008.

[12] P. Levi and S. Kernbach. *Symbiotic-Robot Organisms: Reliability, Adaptability, Evolution*, volume 7. Springer Verlag, 2010.

[13] Thomas L Saaty. *What is the analytic hierarchy process?* Springer, 1988.

[14] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[15] D. Ye, M. Zhang, and D. Sutanto. Self-adaptation-based dynamic coalition formation in a distributed agent network: A mechanism and a brief survey. *IEEE Trans. Parallel Distrib. Syst.*, 24(5):1042–1051, 2013.