Autonomous Task Planning for Heterogeneous Multi-Agent Systems*

Anatoli A. Tziola and Savvas G. Loizou

***Abstract*— This paper presents a solution to the automatic task planning problem for multi-agent systems. A formal framework is developed based on Nondeterministic Finite Automata with ϵ-transitions, where given the capabilities, constraints and failure modes of the agents involved, any initial state of the system and a task specification, an optimal solution is generated that satisfies the system constraints and the task specification. The resulting solution is guaranteed to be complete and optimal; moreover a heuristic solution that offers significant reduction of the computational requirements while relaxing the completeness and optimality requirements is proposed. The constructed system model is independent from the initial conditions and the task specifications, eliminating the need to repeat the costly pre-processing cycle, while allowing the incorporation of failure modes on-the-fly. A case study is provided to demonstrate the effectiveness and validity of the methodology.**

I. INTRODUCTION

Multi-agent systems operating autonomously in dynamical environments to perform complicated tasks have been one of the major areas of research interest during the last decade. High-level task planning using formal methods to define the system requirements is one of the promising approaches. Typical objectives arise from the multi-agent systems' behavior and requirements, such as sequential or reactive tasks, control, coordination and motion and task planning.

Several proposed methodologies address the high-level task planning problem using formal languages to express autonomous systems behavior. Some of the most common approaches include Linear Temporal Logic (LTL), sampling-based approaches and domain definition languages. Many of the existing works use LTL formulas to develop bottom-up approaches [1]–[4] where local LTL expressions are assigned to robots, or top-down approaches [5], [6], [7], where a global task is decomposed into independent subtasks that are treated separately by each agent.

In [1], a high-level plan is found by a discrete planner which seeks the set of system transitions that ensure the satisfaction of a logic formula. In [8], a formal method based on LTL has been developed to model specifications and implement centralized planning for multi-agent systems. Some related works suggest that hierarchical abstraction techniques for single-agent systems can be extended to multi-agent systems using parallel compositions [6], [7]. The fact that the transitions should be on the common events to allow parallel execution is very restrictive. In [3], temporal logic formulas are utilized to specify sub-formulas that could be executed in parallel by the agents, without a global task definition. In [9], the authors tackle the multi-robot task allocation problem under constraints defined by LTL formalism to concurrently plan tasks for robot agents. In [10], a formal method based on LTL has been deployed to model multi robot motion planning specifications. Common among the above works is the adaptation of formal verification techniques for motion planning and controller synthesis. The two main limitations of the above works are the exponential growth of the state space even for relatively simple problems and the extra computations required to express LTL formulas in Büchi automata. In [11], authors propose a formal synthesis of supervisory control software for multi-robot systems, while the scalability of the approach was improved in [12]. On the other hand, implementing Planning Domain Definition Language (PDDL) [13], [14] would be time consuming for real time applications.

In [15], a sampling-based approach is presented using directed trees to approximate the state space and transitions of synchronous product automata. The sampling process is guided by transitions that belong to the shortest path to the accepting states. However, these algorithms provide no solution quality guarantee. Our proposed approach seeks the optimal shortest path in a weighted graph and a case study focusing on the logistics flow aspects was chosen.

The main contribution of this work are:

- System modeling using nondeterministic finite automata with ϵ-transitions [16] expressing the system's behavior and combining the agents capabilities and constraints at the individual and group level, including failure modes.
- Determination of optimal task plans that satisfy any possible task specification from any initial condition, without the need to repeat the pre-processing cycle.
- Determination of reduced complexity sub-optimal solutions to the task planing problem.
- Incorporation of failure modes on-the-fly, after building the global system model (i.e. without the need to repeat the costly pre-processing step).

This work introduces a new approach the SuPErvisory Control Task plannER (SPECTER) for high-level task planning problems with respect to agent capabilities, constraints and failure modes. The problem formulation uses the environment model composed by individual agents' capabilities and constraints, considering the individual agent's failure modes to determine the optimal task plan. This work builds on top and expands the results of [17]. The problem is posed as a special case of module composition problem (MCP) [18] and then reduced to a combinatorial optimization problem

*This work was partially supported by European Union's H2020 research and innovation program under grant agreement 951813 (Better Factory).

The authors are with the Department of Mechanical Engineering and Materials Science and Engineering, Cyprus University of Technology, Limassol, CYPRUS, {anatoli.tziola, savvas.loizou}@cut.ac.cy.

of shortest dipath, which can be solved in polynomial time. The optimal module chain, combined with the Supervisory Control Theory (SCT) [19] can then be applied to synthesize control laws and communication strategies for efficiently accomplishing a global tasks [20]. Agents' navigation can be handled by a navigation methodology with performance guarantees that ensure compositionality, like the navigation transformation [21]. Agents' control and navigation are not under the scope of the current work. The active source code of the software developed is available at [22].

The rest of the paper is organized as follows: Section II presents the necessary preliminary notions, section III presents the problem formulation using the ϵ_0-NFA formalism while section IV exploits the ϵ_0-NFA formalism to reduce the problem to a Module Composition one. Section V presents the algorithms and the analysis of the methodology while section VI presents a case study to demonstrate the operation of SPECTER. Section VII concludes the paper.

II. PRELIMINARIES

A. Definitions

If A and B are sets, the cardinality of set A is denoted as $|A|$. The union and intersection of sets are denoted as $A \cup B$ and $A \cap B$ whereas set subtraction of set B from set A is denoted as $A \setminus B$. Operator $\wedge$ denotes conjunction.

We will use the definition of Deterministic Finite Automata (DFA) by [16] and represent them as $G \triangleq (X_G, E_G, f_G, \Gamma_G, x_{0,G}, X_{m,G})$.

Definition 1 (ϵ_0-NFA)**.** An ϵ_0-NFA is a Nondeterministic Finite Automaton with ϵ-transitions only from the initial state, defined as a six-tuple, ${}^{\epsilon_0}G = (X_G \cup \{x_0\}, E_G \cup \{\epsilon\}, {}^{\epsilon_0}f_G, \Gamma_G, x_0, X_{m,G})$, where ${}^{\epsilon_0}f_G : X_G \cup \{x_0\} \times E_G \cup \{\epsilon\} \to 2^{X_G}$ such that $\forall (x,e) \in X_G \times E_G : f_G(x,e) = {}^{\epsilon_0}f_G(x,e)$, where the variables with "G" subscript are as in the DFA Definition and $x_0 \notin X_G$ is the initial state.

Corollary 1. If ${}^{\epsilon_0}G$ is an ϵ_0-NFA then it can be converted to the DFA G by removing x_0 along with the associated ϵ transitions and assigning an $x_{0,G} \in X_G$ as an initial state.

Proof. Please see the Proof of the same in [23] □

Based on Corollary 1, we construct the following:

Definition 2. Let ${}^{\epsilon_0}G$ be an ϵ_0-NFA and $x_{0,G} \in X_G$ a state in X_G that we want to assign as an initial state. Then $\Delta({}^{\epsilon_0}G, x_{0,G}) \triangleq G$, where G is a DFA. Operator Δ performs the conversion described in Corollary 1.

We need to associate a cost function with the transitions:

Definition 3 (Transition Cost Function)**.** A transition cost function for an event set E_G, is defined as $g_G : E_G \to \mathbb{R}_{>0}$.

We define the inverse transition function as:

Definition 4 (Inverse Transition Function)**.** Let G be a DFA. Define $f_G^{-1} : X_G \times E_G \to X_G$ to be the inverse transition function, if such exists, such that $f_G(x_G, e) = y_G \wedge f_G^{-1}(y_G, e) = x_G$ for some $x_G : e \in \Gamma_G(x_G)$, $y_G : e \in \Gamma_G(f_G^{-1}(y_G, e))$.

For the automata operations that will be required in the sequel, we need to introduce the following concept:

Definition 5 (Compatible ϵ_0-NFAs)**.** Let ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$ be ϵ_0-NFAs. Let $E_C = E_G \cap E_B$. ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$ are compatible iff $\forall e \in E_C$ then[1], $f_G(x_G, e) = f_B(x_B, e)$ and $f_G^{-1}(y_G, e) = f_B^{-1}(y_B, e)$ for some $x_G : e \in \Gamma_G(x_G)$, $x_B : e \in \Gamma_B(x_B)$, $y_G : e \in \Gamma_G(f_G^{-1}(y_G, e))$ and $y_B : e \in \Gamma_B(f_B^{-1}(y_B, e))$. In such case it will also be true that $x_G = x_B$ and $y_G = y_B$. We denote such a compatibility relation as ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$.

Note that the above definition effectively forces the endpoints of common events to be common states.

B. Operations on compatible automata

Here we introduce a custom set of the basic operations on compatible ϵ_0-NFAs: union, subtraction and concatenation. The introduced operations differ from the ones found in the automata literature [16], [24], [25] and provide the necessary functionality for the subsequent developments.

Definition 6 (Union of Compatible Automata)**.** For the ϵ_0-NFAs ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$, assume ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$. Define the compatible automata union ${}^{\epsilon_0}\mathcal{P} \triangleq {}^{\epsilon_0}G \cup_{\asymp} {}^{\epsilon_0}B$ to be the six-tuple ${}^{\epsilon_0}\mathcal{P} = (X_\mathcal{P} \cup \{x_0\}, E_\mathcal{P} \cup \{\epsilon\}, {}^{\epsilon_0}f_\mathcal{P}, \Gamma_\mathcal{P}, x_0, X_{m,\mathcal{P}})$, where $X_\mathcal{P} = X_G \cup X_B$, $E_\mathcal{P} = E_G \cup E_B$, ${}^{\epsilon_0}f_\mathcal{P} : X_\mathcal{P} \cup \{x_0\} \times E_\mathcal{P} \cup \{\epsilon\} \to 2^{X_\mathcal{P}}$ such that $\forall (x,e) \in X_G \times E_G : {}^{\epsilon_0}f_\mathcal{P}(x,e) = f_G(x,e)$ and $\forall (x,e) \in X_B \times E_B : {}^{\epsilon_0}f_\mathcal{P}(x,e) = f_B(x,e)$, $\Gamma_\mathcal{P} : X_\mathcal{P} \to 2^{E_\mathcal{P}}$, $x_0 \notin X_\mathcal{P}$ and $X_{m,\mathcal{P}} = X_{m,G} \cup X_{m,B}$.

In contrast with the standard union operation on automata presented in the literature, the above operation produces the union instead of the cartesian product state space.

Definition 7 (Subtraction of Compatible Automata)**.** For the ϵ_0-NFAs ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$, assume ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$. Define the compatible automata subtraction ${}^{\epsilon_0}\Theta \triangleq {}^{\epsilon_0}G \setminus_{\asymp} {}^{\epsilon_0}B$ to be the six-tuple ${}^{\epsilon_0}\Theta = (X_\Theta \cup \{x_0\}, E_\Theta \cup \{\epsilon\}, {}^{\epsilon_0}f_\Theta, \Gamma_\Theta, x_0, X_{m,\Theta})$, where $X_\Theta = X_G$, $E_\Theta = E_G \setminus E_B$, ${}^{\epsilon_0}f_\Theta : X_\Theta \cup \{x_0\} \times E_\Theta \cup \{\epsilon\} \to 2^{X_\Theta}$ s.t. $\forall (x,e) \in X_\Theta \times E_\Theta : {}^{\epsilon_0}f_\Theta(x,e) = f_\Theta(x,e)$, $\Gamma_\Theta : X_\Theta \to 2^{E_\Theta}$, $x_0 \notin X_\Theta$, $X_{m,\Theta} = X_{m,G} \setminus X_{m,B}$.

Note that this is more in line with the set difference operator than with the intersection with the language complement that is used in regular languages.

Definition 8 (Concatenation of Compatible Automata)**.** For the ϵ_0-NFAs ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$, assume ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$. Define the compatible automata concatenation ${}^{\epsilon_0}\Phi \triangleq {}^{\epsilon_0}G \amalg_{\asymp} {}^{\epsilon_0}B$ to be the six-tuple ${}^{\epsilon_0}\Phi = (X_\Phi \cup \{x_0\}, E_\Phi \cup \{\epsilon\}, {}^{\epsilon_0}f_\Phi, \Gamma_\Phi, x_0, X_{m,\Phi})$, where $X_\Phi = \{uv | u \in X_G, v \in X_B\}$, $E_\Phi = E_G \cup E_B$, ${}^{\epsilon_0}f_\Phi : X_\Phi \cup \{x_0\} \times E_\Phi \cup \{\epsilon\} \to 2^{X_\Phi}$ s.t. $\forall (x,e) \in X_\Phi \times E_\Phi : {}^{\epsilon_0}f_\Phi(x,e) = f_\Phi(x,e)$, $\Gamma_\Phi : X_\Phi \to 2^{E_\Phi}$, $x_0 \notin X_\Phi$, $X_{m,\Phi} = \{uv | u \in X_{m,G} \wedge v \in X_{m,B}\}$.

[1]Note that due to Corollary 1 we have that ${}^{\epsilon_0}f_G(x_G, e) \equiv f_G(x_G, e)$ and ${}^{\epsilon_0}f_B(x_B, e) \equiv f_B(x_B, e)$.

Note that the operation in Definition 8 above, is similar to the well known parallel composition operation (see e.g. [16]) but differs in that the event sets undergo a disjoint union in our case to ensure that only one event can be activated at a time (no synchronization on common events).

The concept of compatible automata, gives rise to the following properties related to Definitions 6, 7 and 8:

Corollary 2. The automata resulting from the operations defined in Definitions 6, 7 and 8 are ϵ_0-NFAs.

Proof. Please see the Proof of the same in [23] □

We need to introduce the following operator that addresses individual states of concatenated states of Definition 8.

Definition 9 (Projection Operator)**.** Assume $x \in X_G$ is a state of G with $|x| = n$ and let x_i denote the i'th element of x. Define the projector b as an n-bit binary and b_i its i'th bit. The projection operator is defined as $proj(x, b) \triangleq \{x_i | b_i = 1\}$.

We will need some definitions from MCP literature as well as some new ones for our development. Using the module definition by [18], we state the module in an appropriate form for our development.

Definition 10 (Single Port Module)**.** Let G be a DFA. Consider the finite set of ports $P = P_{in} \cup P_{out}$, where P_{in} be a non-empty finite set of input ports and P_{out} be a non-empty finite set of output ports with $P \subseteq X_G$ and $P_{in} \cap P_{out} \neq \emptyset$. We define the single input/single output port (I/O) module T as a 3-tuple $\mathrm{T} \triangleq \{p, e, q\}$, where $p \in P_{in}$ is the input port, $e \in E_G : f_G(p, e) = q$ and $q \in P_{out}$ is the output port.

The cost associated with the module T is defined as: $c(\mathrm{T}) \triangleq g_G(e)$. We can also define the inverted module T^{-1} where its input becomes its output and vice versa such that Definition 4 holds.

III. PROBLEM FORMULATION

A. Agent Model

Let n be the total number of agents and ${}^{\epsilon_0}\mathcal{A}$ be the finite set of the agents ϵ_0-NFAs, where $n = |{}^{\epsilon_0}\mathcal{A}|$. We represent the i^{th} agent as ${}^{\epsilon_0}A_i$, $i \in \{1, \dots n\}$.

1) Individual Agent Capabilities: Let κ be the total number of individual capabilities of ${}^{\epsilon_0}A_i$ and ${}^{\epsilon_0}M_{\beta,i}$ the β'th individual capability of ${}^{\epsilon_0}A_i$, where $\beta \in \{1, \dots \kappa\}$.

2) Individual Agent Failure Mode: Consider a state $x' \in X_{M_{\beta,i}}$. We define a failure mode of ${}^{\epsilon_0}A_i$ as the inability to complete the transition from some $x \in X_{M_{\beta,i}}$ to x' with an occurrence of event $e \in E_{M_{\beta,i}}$. This describes a detected transition failure of ${}^{\epsilon_0}A_i$ which renders $f_{M_{\beta,i}}(x, e) = x'$ infeasible. This failure mode is modeled by the ϵ_0-NFA ${}^{\epsilon_0}F_i$ such that $X_{F_i} = \{x, x'\}$ and $E_{F_i} = \{e\}$.

We model the agent's capabilities as the subtraction of the agent failure from the union of individual agent capabilities.

3) Agent Capabilities: Considering κ compatible ϵ_0-NFAs such that ${}^{\epsilon_0}M_{\alpha,i} \asymp {}^{\epsilon_0}M_{\beta,i}$, $\alpha \neq \beta$ with $\alpha, \beta \in \{1, \dots \kappa\}$ and ${}^{\epsilon_0}F_i \asymp {}^{\epsilon_0}M_{\beta,i}$, the capabilities of ${}^{\epsilon_0}A_i$ are modeled by the ϵ_0-NFA ${}^{\epsilon_0}K_i$ as: ${}^{\epsilon_0}K_i \triangleq \left\{ \bigcup_{\beta=1}^{\kappa} \asymp {}^{\epsilon_0}M_{\beta,i} \right\} \backslash_{\asymp} {}^{\epsilon_0}F_i$.

4) Individual Agent Constraints: Let λ be the total number of individual constraints of ${}^{\epsilon_0}A_i$ and ${}^{\epsilon_0}N_{\xi,i}$ the ξ'th individual constraint of ${}^{\epsilon_0}A_i$, where $\xi \in \{1, \dots \lambda\}$.

We model the agent's constraints as the union of individual agent constraints.

5) Agent Constraints: Considering λ compatible ϵ_0-NFAs such that ${}^{\epsilon_0}N_{\xi,i} \asymp {}^{\epsilon_0}N_{\eta,i}$, $\xi \neq \eta$ with $\eta \in \{1, \dots \lambda\}$, the constraints of ${}^{\epsilon_0}A_i$ are modeled by the ϵ_0-NFA ${}^{\epsilon_0}D_i$ as: ${}^{\epsilon_0}D_i \triangleq \left\{ \bigcup_{\xi=1}^{\lambda} \asymp {}^{\epsilon_0}N_{\xi,i} \right\}$.

Considering ${}^{\epsilon_0}K_i \asymp {}^{\epsilon_0}D_i$, the agent i^{th} agent is modeled by the ϵ_0-NFA of ${}^{\epsilon_0}K_i$ after subtracting ${}^{\epsilon_0}D_i$. Thus, ${}^{\epsilon_0}A_i \triangleq {}^{\epsilon_0}K_i \backslash_{\asymp} {}^{\epsilon_0}D_i$.

B. Environment Model

We model the environmental capabilities as the concatenation of agents' capabilities to express the allowed environment state transitions.

1) Environmental Capabilities: For the ϵ_0-NFAs ${}^{\epsilon_0}K_i$ and ${}^{\epsilon_0}K_j$, $i \neq j$ with $i, j \in \{1, \dots n\}$, let ${}^{\epsilon_0}K_i \asymp {}^{\epsilon_0}K_j$. The environmental capabilities are modeled by the ϵ_0-NFA ${}^{\epsilon_0}\mathcal{K}$ as: ${}^{\epsilon_0}\mathcal{K} \triangleq \amalg_{\asymp i=1}^{n} {}^{\epsilon_0}K_i$.

We model the environmental constraints as the concatenation of agents' constraints to express the not-permitted environment state transitions.

2) Environmental Constraints: For the ϵ_0-NFAs ${}^{\epsilon_0}D_i$ and ${}^{\epsilon_0}D_j$, $i \neq j$ with $i, j \in \{1, \dots n\}$, let ${}^{\epsilon_0}D_i \asymp {}^{\epsilon_0}D_j$. The environmental constraints are modeled by the ϵ_0-NFA ${}^{\epsilon_0}\mathcal{D}$ as: ${}^{\epsilon_0}\mathcal{D} \triangleq \amalg_{\asymp i=1}^{n} {}^{\epsilon_0}D_i$.

Then, inter-agents capabilities and inter-agents constraints are modeled as follows:

3) Inter-Agents Capabilities: Let $\mathrm{co} \subseteq {}^{\epsilon_0}\mathcal{A}$ with $\|\mathrm{co}\| \leq n$ be a set of compatible agent ϵ_0-NFAs. Then, the ϵ_0-NFA ${}^{\epsilon_0}\mathcal{K}_A \subset \amalg_{\asymp i \in \mathrm{co}} {}^{\epsilon_0}A_i$ denotes the inter-agents capabilities[2] between the members of co.

4) Inter-Agents Constraints: Let $\mathrm{co} \subseteq {}^{\epsilon_0}\mathcal{A}$ with $\|\mathrm{co}\| \leq n$ be a set of compatible agent ϵ_0-NFAs. Then, the ϵ_0-NFA ${}^{\epsilon_0}\mathcal{D}_A \subset \amalg_{\asymp i \in \mathrm{co}} {}^{\epsilon_0}A_i$ denotes the inter-agents constraints between the members of co.

We model the global capabilities by adding the inter-agent capabilities to the environmental capabilities.

5) Global Capabilities: For the ϵ_0-NFAs ${}^{\epsilon_0}\mathcal{K}$ and ${}^{\epsilon_0}\mathcal{K}_A$, let ${}^{\epsilon_0}\mathcal{K} \asymp {}^{\epsilon_0}\mathcal{K}_A$. The global capabilities are modeled by the ϵ_0-NFA ${}^{\epsilon_0}\widetilde{\mathcal{K}}$ defined as: ${}^{\epsilon_0}\widetilde{\mathcal{K}} \triangleq {}^{\epsilon_0}\mathcal{K} \cup_{\asymp} {}^{\epsilon_0}\mathcal{K}_A$.

We model the global constraints by adding the inter-agent constraints to the environmental constraints.

[2]Inter-Agents Failure Modes can be defined so as to restrict Inter-Agents Capabilities

6) Global Constraints: For the ϵ_0-NFAs ${}^{\epsilon_0}\mathcal{D}$ and ${}^{\epsilon_0}\mathcal{D}_A$, let ${}^{\epsilon_0}\mathcal{D} \asymp {}^{\epsilon_0}\mathcal{D}_A$. The global constraints are modeled by the ϵ_0-NFA ${}^{\epsilon_0}\widetilde{\mathcal{D}}$ defined as: ${}^{\epsilon_0}\widetilde{\mathcal{D}} \triangleq {}^{\epsilon_0}\mathcal{D} \cup_{\asymp} {}^{\epsilon_0}\mathcal{D}_A$.

Considering ${}^{\epsilon_0}\widetilde{\mathcal{K}} \asymp {}^{\epsilon_0}\widetilde{\mathcal{D}}$, the environment model is constructed by the ϵ_0-NFA of ${}^{\epsilon_0}\widetilde{\mathcal{K}}$ after subtracting ${}^{\epsilon_0}\widetilde{\mathcal{D}}$. Thus, ${}^{\epsilon_0}\mathcal{S} \triangleq {}^{\epsilon_0}\widetilde{\mathcal{K}} \backslash_{\asymp} {}^{\epsilon_0}\widetilde{\mathcal{D}}$, where the cardinality of $X_{\mathcal{S}}$ is $\theta = \prod_{i=1}^{n} |X_{A_i}|$. The procedure of constructing agents' and environment models is illustrated in Fig. 1, as well.

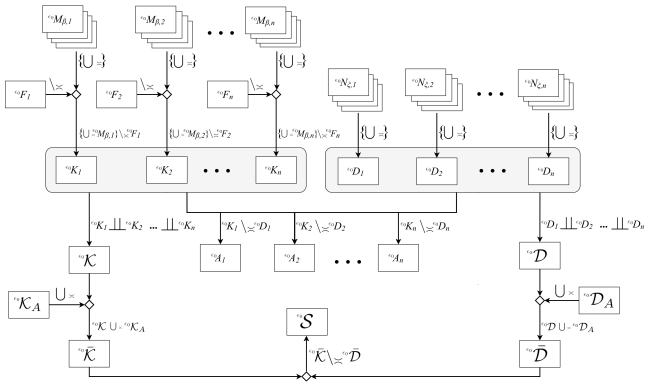

Fig. 1: Flow diagram of the operations to produce the models of agents ${}^{\epsilon_0}A_i$ and the environment ${}^{\epsilon_0}\mathcal{S}$.

C. Task Specification

Let $x \in X_{\mathcal{S}}$ and let b be an n-bit binary indicating the states of the agents that we are interested in specifying. The task specification is a projection $\gamma \triangleq proj(x, b)$ that indicates the desired state of specific agents in the system.

We can now proceed to formally state the following.

D. Problem statement

Given a set of individual and inter-agents capabilities and constraints including their failure modes, the initial state of the system and a task specification, determine a string that provides an optimal execution[3] (in terms of total transition cost) that brings the system from any initial state to any state satisfying the task specification.

IV. FORMULATION AS A MODULE COMPOSITION PROBLEM

Let us now consider the environment model ${}^{\epsilon_0}\mathcal{S}$ and assume that we would like to use $x_{0,\mathcal{S}} \in X_{\mathcal{S}}$ as the initial state of our system. Thus, $\mathcal{S} = \Delta({}^{\epsilon_0}\mathcal{S}, x_{0,\mathcal{S}})$. Let $P_{in} \in X_{\mathcal{S}}$ and $P_{out} \in X_{\mathcal{S}}$ be the non-empty finite set of input and output ports respectively, where $P = P_{in} \cup P_{out}$ and $P_{in} \cap P_{out} \neq \emptyset$. The module T_j is defined as: $\mathrm{T}_j \triangleq \{p_j, e_j, q_j\}$.

where $p_j \in P_{in}$, $e_j \in E_{\mathcal{S}}$ and $q_j \in P_{out}$.

Define the i^{th} task module as $\mathrm{T}_{0,i} = \{x_{0,\mathcal{S}}, e_{0,i}, x_{d,i}\}$ where $proj(x_{d,i}, b) = \gamma$, $x_{d,i} \in X_{\mathcal{S}}$ and $e_{0,i}$ a virtual transition from the initial to the final state. Observe that there are $|\gamma|!$ potential solutions.

[3]Automata execution (or run) as defined in e.g. [16]

To tackle the problem defined in the Problem Statement, we proceed to formulate our problem as a Module Composition Problem (MCP) [18]. Since we are using single-port modules, we will have a special case of the MCP that is solvable in polynomial time. We define $\mathcal{T}_i$ as the finite closed module chain containing $\mathrm{T}_{0,i}^{-1}$ describing the sequential environment states transitions during the execution defined as: $\mathcal{T}_i = \{\mathrm{T}_{0,i}^{-1}, \mathrm{T}_{1,i}, \ldots \mathrm{T}_{z,i}\}$, where $z_i = |\mathcal{T}_i|$. In the sequel we will drop the index i for notational brevity.

Let t_j denote the number of instances of T_j, $c_j = c(T_j)$ is the cost of implementing module T_j, then the discrete optimization problem can be posed as an integer programming problem (see [23] for details). Since in the current work we have implemented only single port modules, the above integer programming problem can be reduced to the shortest directed path problem [26], that can be solved utilizing the Dijkstra's shortest dipath algorithm [27]. The optimal solution composes the $\mathcal{T}$ that minimizes the cost of states transitions in order to fulfil the task specification γ.

The drawback of using singe port modules in the current work is that we are only limiting transitions to be performed by one agent at a time. Multi-port modules are currently being considered as a further research topic, to enable multiple agents to perform concurrent transitions and is beyond the scope of the current work. We have to also note here that the additional effort in casting the problem as a module composition one, enables us to seamlessly use the generated module chain as a model system for building supervisory controllers as in [17], and is currently under active research.

V. ANALYSIS

A. Task Planning Processes

In this section, we present the properties and process of SPECTER task planner to compute the optimal plan based on the agents' and environment models. The SPECTER's running operation phase includes the pre-processing phase and the problem solving phase.

The pre-processing phase includes the construction of the agents models, the environment model (see Fig.1) and the graph of ${}^{\epsilon_0}\mathcal{S}$ using adjacency matrix representation. The computational time to construct ${}^{\epsilon_0}A_i$ is $O(|X_{A_i}|^2)$ and $O(|X_{\mathcal{S}}|^3)$ for ${}^{\epsilon_0}\mathcal{S}$. It should be noted that ${}^{\epsilon_0}\mathcal{S}$ is only needed to be constructed once and can be used thereafter for all possible task specifications and initial states. ${}^{\epsilon_0}\mathcal{S}$ is converted to a weighted directed graph $\mathcal{H}_{\mathcal{S}} = (\mathcal{V}_{\mathcal{S}}, \mathcal{E}_{\mathcal{S}})$, where the set of nodes $\mathcal{V}_{\mathcal{S}}$ corresponds to the set of states $X_{\mathcal{S}}$, the set of edges $\mathcal{E}_{\mathcal{S}}$ is defined by $f_{\mathcal{S}}$ associated with its cost $g_{\mathcal{S}}$. The adjacency matrix representation of $\mathcal{H}_{\mathcal{S}}$ is a 2-dimensional array $X_{\mathcal{S}} \times X_{\mathcal{S}}$. Each element in the array stores the cost $g_{\mathcal{S}}$ related to the edge $f_{\mathcal{S}}(x \in X_{\mathcal{S}}, e \in E_{\mathcal{S}})$. The amount of space required to store the array is $O(|\mathcal{V}_{\mathcal{S}}|^2)$ in worst case. Computational efficiency can be succeeded if the pre-processing computations are made beforehand. The constructed environment model can then be used to solve all the possible task planning problems by arbitrarily choosing the initial and target states (projection) without the need to reconstruct the environment model.

The problem solving phase encapsulates the synthesis of the optimal task plan and the integration of individual agent failure mode. The Dijkstra's algorithm is implemented over the weighted graph $\mathcal{H}_{\mathcal{S}}$ to find the optimal task plan $\mathcal{T}$ as given in Algorithm 1. In the case where a fault is detected for the transition from state x_i to x_j of $\mathcal{S}$ (e.g. by some fault identification system) that affects agent ν, we can disable all the affected transitions by finding the set of states $X_i{}'$, $X_j{}'$, where $proj(x_i, b_\nu) \equiv proj(x_i{}' \in X_i{}', b_\nu)$ and $proj(x_j, b_\nu) \equiv proj(x_j{}' \in X_j{}', b_\nu)$. Then, we eliminate the transitions from all $x_i{}' \in X_i{}'$ to all $x_j{}' \in X_j{}'$. The above procedures incorporates on-the-fly a possible new failure mode into $\mathcal{S}$ without the need to repeat the costly pre-processing phase. The computational time required for this modification is $\theta' = \prod_{i=1,\ i\neq\nu}^{n-1} |X_{A_i}|$ in the worst case.

The task planning problem of minimizing the length of the plan is NP-hard [28]. Let $\mathcal{P}$ be the solution to this problem found after running Dijkstra's algorithm and let $\mathcal{P}_i$ denote it's i'th element, $i \in \{1, \dots |\mathcal{P}|\}$. In Algorithm 2, the optimal solution can then be found by running the algorithm for all states that satisfy the task specification, that is $\frac{\prod_{i=1}^{n} |X_{A_i}|}{|X_{A_\sigma}|}$ times in the worst case scenario, where σ denotes the agent index that was used for the task specification γ. The running time of Dijkstra's algorithm implementing the Complete Function of Algorithm 1 is $O((\mathcal{V}_{\mathcal{S}} + \mathcal{E}_{\mathcal{S}}) log \mathcal{V}_{\mathcal{S}})$.

Algorithm 1 Create module chain $\mathcal{T}$.

Require: ϵ_0-NFA ${}^{\epsilon_0}\mathcal{S}$, initial state $x_{0,\mathcal{S}}$, task specification γ, solution type ST ("Complete" or "Heuristic")
Ensure: Module chain $\mathcal{T}$
Initialize $\mathcal{H}$ to be a zero matrix, E to be an empty cell array and $\mathcal{T}$ to an empty set
$\mathcal{S} \longleftarrow \Delta({}^{\epsilon_0}\mathcal{S}, x_{0,\mathcal{S}})$
if $proj(x_{0,\mathcal{S}}, b) = \gamma \wedge x_{0,\mathcal{S}} \in X_{m,\mathcal{S}}$ **then**
  return $\mathcal{T}$
else
  for $i \in \{1, \dots |X_{\mathcal{S}}|\}$ **do**
    for $j \in \{1, \dots |X_{\mathcal{S}}|\}$ **do**
      if $\exists e \in E_{\mathcal{S}} : f_{\mathcal{S}}(x_i, e) = x_j$ **then**
        $\mathcal{H}[i][j] \longleftarrow g_{\mathcal{S}}(e)$
        $E\{i\}\{j\} \longleftarrow e$
      end if
    end for
  end for
  if ST = "Complete" **then**
    [$\mathcal{P}$, cost] $\longleftarrow$ COMPLETE($\mathcal{H}, \mathcal{S}, x_{0,\mathcal{S}}, \gamma$)
  else
    [$\mathcal{P}$, cost] $\longleftarrow$ HEURISTIC($\mathcal{H}, \mathcal{S}$, E, $x_{0,\mathcal{S}}, \gamma$)
  end if
  for $i \in \{1, \dots |\mathcal{P}| - 1\}$ **do**
    $\mathcal{T} \longleftarrow \mathcal{T} \cup \{\mathcal{P}_i,\ E\{\mathcal{P}_i\}\{\mathcal{P}_{i+1}\},\ \mathcal{P}_{i+1}\}$
  end for
  return $\mathcal{T}$, cost
end if

While this might be feasible for problems of relatively small size, a heuristic approach is proposed to reduce the computational time required to find a sub-optimal solution. The proposed heuristic, implemented in Algorithm 3, maintains the complexity to the one of Dijkstra's algorithm, while sacrificing optimality and completeness. The computational time of the proposed heuristic is $O(\mathcal{E}_{\mathcal{S}} log \mathcal{V}_{\mathcal{S}})$. We track the solution to the minimum element k where $proj(\mathcal{P}_k, b) = \gamma$

Algorithm 2 Complete Function.

function COMPLETE($\mathcal{H}, \mathcal{S}, x_{0,\mathcal{S}}, \gamma$)
  Initialize $cost_{min} \longleftarrow \infty$
  for $i \in \{1, \dots |X_{\mathcal{S}}|\}$ **do**
    for $j \in \{1, \dots |X_{\mathcal{S}}|\}$ **do**
      if $proj(x_j, b) = \gamma \wedge x_j \in X_{m,\mathcal{S}}$ **then**
        [$\mathcal{P}$, cost] $\longleftarrow$ Dijkstra($\mathcal{H}, x_{0,\mathcal{S}}, x_j$)
        if $\mathcal{P} = \emptyset$ **then**
          return Task infeasible.
        end if
        if cost $< cost_{min}$ **then**
          $cost_{min} \longleftarrow$ cost
          $\mathcal{P}_{optimal} \longleftarrow \mathcal{P}$
        end if
      end if
    end for
  end for
  return $\mathcal{P}_{optimal}$, $cost_{min}$
end function

and use the solution $\mathcal{P}^* = \{\mathcal{P}_1, \dots \mathcal{P}_k\}$. While there are some cases where the proposed heuristic recovers optimality, a classification of those cases and the conditions for feasibility of solutions is currently under consideration but is not part of the current work. The intuition here is to set the goal state x_d to be such that $proj(x_d, b) = \gamma$ and $proj(x_d, \bar{b}) = proj(x_{0,\mathcal{S}}, \bar{b})$ where $\bar{b}$ is the bitwise negation.

Algorithm 3 Heuristic Function.

function HEURISTIC($\mathcal{H}, \mathcal{S}$, E, $x_{0,\mathcal{S}}, \gamma$)
  Initialize x_d to an empty set
  Find $x_d \in X_{m,\mathcal{S}} : proj(x_d, b) = \gamma \wedge proj(x_d, \bar{b}) = proj(x_{0,\mathcal{S}}, \bar{b})$
  Initialize $c^* \longleftarrow 0$
  [$\mathcal{P}$, cost] $\longleftarrow$ Dijkstra($\mathcal{H}, x_{0,\mathcal{S}}, x_d$)
  if $\mathcal{P} = \emptyset$ **then**
    return No path to x_d.
  end if
  for $k \in \{2, \dots |\mathcal{P}|\}$ **do**
    $c^* \longleftarrow c^* + g_{\mathcal{S}}(E\{\mathcal{P}_{k-1}\}\{\mathcal{P}_k\})$
    if $proj(\mathcal{P}_k, b) = \gamma$ **then**
      $\mathcal{P}^* \longleftarrow \{\mathcal{P}_1, \dots \mathcal{P}_k\}$
      return $\mathcal{P}^*, c^*$
    end if
  end for
end function

B. Completeness and Optimality

We have the following results regarding the completeness of Algorithms 1 and 2:

Proposition 1. For a system constructed based on individual and inter-agent capabilities, constraints and failure modes of the multi-agent system as presented in Section III, the resulting environment model is complete, in the sense that it represents all and only those state transitions that are dictated by the automata capturing the individual and inter-agent capabilities, constraints and failure modes.

Proof. Please see the Proof of the same in [23] □

With the completeness property in place we can now state the following optimality and completeness result:

Proposition 2. For a system constructed based on individual and inter-agent capabilities, constraints and failure modes of the multi-agent system as presented in Section III, and assuming that only one agent is allowed to operate at any time instant, the solution of Algorithm 1 with the "Complete" solution type of Algorithm 2, produces the optimal sequence of actions to achieve task specification γ.

Proof. Please see the Proof of the same in [23] □

VI. CASE STUDY

The proposed methodology could be applied in various applications, such as manufacturing logistics. To demonstrate the applicability of the proposed methodology, several case studies were carried out and one of them is presented below. Additional case studies can be found in [23].

In this case study, we identified a pattern of two regular workflows concerning the manufacturing of semi-finished products 1 and 2 and final products 1. The construction of semi-finished products 1 require raw materials 1 and 2 whereas the semi-finished products 2 require raw material 3. The semi-finished products 1 are produced by the injection machine 1. The semi-finished products 2 are created by injection machine 2. Entities produced by injection machines are collected at stations J or C. The semi-finished products 1 are passed through the conveyor 2 to the packing machine. The final products 1 are created from semi-finished product 1, which are passed through the conveyor 2 to the packing machine. Packed entities are collected at station F and then are transported to Warehouse.

To model the two workflows requires to define the agents involved in the workflow processes. Considering that 9 agents are involved where ${}^{\epsilon_0}A_1, {}^{\epsilon_0}A_2, {}^{\epsilon_0}A_3$ model the raw materials 1, 2 and 3, ${}^{\epsilon_0}A_4, {}^{\epsilon_0}A_5$ model the semi-finished products 1 and 2, ${}^{\epsilon_0}A_6$ models the final product 1, ${}^{\epsilon_0}A_7$, ${}^{\epsilon_0}A_8$ model robots 1 and 2, and ${}^{\epsilon_0}A_9$ models the human-worker. The initial states of the agents are: Raw materials 1, 2 and 3 are stored in the Warehouse, each robot is at its docking station and the human is in Warehouse. The objective task is to produce the final product 1 and to store it in Warehouse in minimum time. Models of

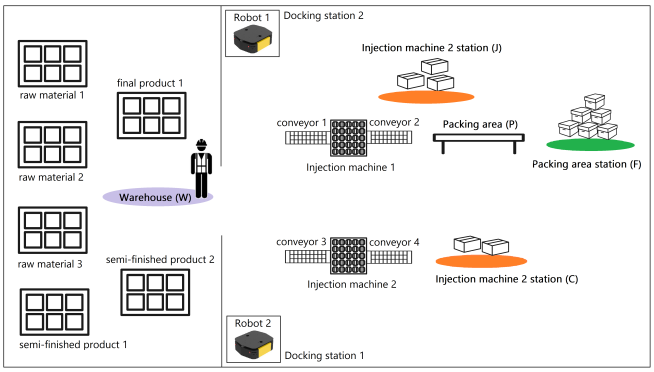

Fig. 2: Factory plant.

${}^{\epsilon_0}A_7$, ${}^{\epsilon_0}A_8$ and ${}^{\epsilon_0}A_9$ are constructed by combing the agents capabilities and constraints. Capabilities and constraints of ${}^{\epsilon_0}A_1, {}^{\epsilon_0}A_2, {}^{\epsilon_0}A_3, {}^{\epsilon_0}A_4, {}^{\epsilon_0}A_5, {}^{\epsilon_0}A_6$ are defined as inter-agent capabilities and inter-agent capabilities constraints. Thus, ${}^{\epsilon_0}\mathcal{S}$ is constructed, where $X_{\mathcal{S}} = \prod_{i=1}^{n} |X_{A_i}| = 1.875 \times 10^6$ states, with $|X_{A_1}| = |X_{A_3}| = |X_{A_4}| = |X_{A_5}| = |X_{A_6}| = |X_{A_7}| = |X_{A_8}| = 5$ states, $|X_{A_2}| = 6$ states, and $|X_{A_9}| = 4$ states. The task specification is defined as $proj(x_d, b) = JCW$, with $b = 000111000$ and $|\gamma| = 3$.

The solution $\mathcal{T}$ was computed with both Algorithms 2 and 3. Both solutions were identical. The computed $\mathcal{T}$ resulted to a composition of 24 open chain modules. In words, (T_1) raw materials 1, 2, 3 are stored in W, robots are at their docking stations, worker is at W, (T_2) robot 1 goes to W, (T_3) worker loads the raw material 3 to robot 1, (T_4) worker loads raw material 2 to robot 1, (T_5) worker loads raw material 1 to robot 1, (T_6) worker goes to injection 2, (T_7) robot 1 goes to injection 2 while carrying raw materials, (T_8) worker unloads raw material 3 from the robot 1 and loads the raw material 3 to the C, (T_9) robot 1 goes to injection 2 while carrying raw materials 1 and 2, (T_{10}) worker goes to injection 2, (T_{11}) C starts the production of semi-finished product 2, (T_{12}) worker unloads raw material 2 from robots 1 and loads the raw material 2 to J, (T_{13}) injection machine starts the preparation of semi-finished product 1, (T_{14}) worker unloads raw material 1 from robot 1 and loads the raw material 1 to J, (T_{15}) injection 1 starts the production of semi-finished product 1, (T_{16}) the final product 1 is produced at Fa, (T_{17}) semi-product 2 is produced by C, (T_{18}) worker goes to Fa, (T_{19}) robot 2 goes to Fa, (T_{20}) worker loads final product 1 on robot 2, (T_{21}) robot 2 goes to W, (T_{22}) semi-finished product 1 is produced by J, (T_{23}) worker goes to W, (T_{24}) worker unloads the final product 1 from robot 2 at W. Thus, the final product 1 is stored in Warehouse and the objective task is completed.

The pre-processing runtime to construct the agents and environment models is $3.081 \times 10^5 s$. The problem solving runtime to find $\mathcal{T}$ utilizing Algorithm 3 is $165.73s$ whereas with Algorithm 2 it required a total of $10,720s$ for all $15,000$ possible x_d's for which $proj(x_d, b) = JCW$.

VII. CONCLUSIONS

The methodology for a novel multi-agent supervisory control task planner is proposed. Given the capabilities, constraints and failure modes of the agents under the framework of NFAs with ϵ-transitions, SPECTER produces the optimal sequence of tasks for transporting the state of the environment from any initial to any given destination state. The developed algorithms can provide a complete solution with optimality guarantees whenever a solution exists. By relaxing the completeness property requirement, a significant reduction in the computational requirements is achieved, that provides suboptimal solutions through the use of efficient heuristics. The results of the case study demonstrate the applicability as well as the effectiveness and validity of the proposed methodology in successfully generating optimal executions for multi-agent systems with a predetermined set of individual capabilities and constraints as well as agent coupling capabilities and restrictions. Future work will focus

on combining SPECTER with supervisory control theory to enable reactive execution in dynamic environments.

REFERENCES

[1] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 3116–3121.

[2] ——, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] M. Guo and D. V. Dimarogonas, "Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 797–808, 2016.

[4] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[5] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "A formal approach to deployment of robotic teams in an urban-like environment," in *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 313–327.

[6] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2009.

[7] M. M. Quottrup, T. Bak, and R. Zamanabadi, "Multi-robot planning: A timed automata approach," in *IEEE Int. Conf. on Robotics and Automation*, vol. 5, 2004, pp. 4417–4422.

[8] J. Tůmová and D. V. Dimarogonas, "A receding horizon approach to multi-agent planning from local ltl specifications," in *2014 American Control Conference*, 2014, pp. 1775–1780.

[9] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The international journal of robotics research*, vol. 37, no. 7, pp. 818–838, 2018.

[10] M. Kloetzer and C. Mahulea, "Multi-robot path planning for syntactically co-safe ltl specifications," in *2016 13th International Workshop on Discrete Event Systems (WODES)*. IEEE, 2016, pp. 452–458.

[11] J. Goryca and R. C. Hill, "Formal synthesis of supervisory control software for multiple robot systems," in *2013 American Control Conference*. IEEE, 2013, pp. 125–131.

[12] R. C. Hill and S. Lafortune, "Scaling the formal synthesis of supervisory control software for multiple robot systems," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 3840–3847.

[13] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[14] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[15] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, pp. 812–836, 39(7), 2020.

[16] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[17] S. G. Loizou and K. J. Kyriakopoulos, "Automated planning of motion tasks for multi-robot systems," in *IEEE Int. Conf. on Decision and Control*, 2005, pp. 78–83.

[18] S. Tripakis, "Automated module composition," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2003, pp. 347–362.

[19] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM journal on control and optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[20] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM Journal on Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

[21] S. G. Loizou and E. D. Rimon, "Mobile robot navigation functions tuned by sensor readings in partially known environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3803–3810, 2022.

[22] "Business process optimization (bpo) module," https://github.com/rcdslabcut/mod.sw.bpo, accessed: March 2021.

[23] A. Tziola and S. Loizou, "Autonomous task planning for heterogeneous multi-agent systems," *Preprint submitted to arXiv.org*, September 2022.

[24] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Automata theory, languages, and computation*. Pearson, 2006.

[25] B. Khoussainov and A. Nerode, *Automata theory and its applications*. Springer Science & Business Media, 2012, vol. 21.

[26] S. Tripakis, "Automated module composition," *Lecture Notes in Computer Science*, vol. 2619, pp. 347–362, 2003.

[27] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[28] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, p. 263, 1998.