Relational Activity Processes for Modeling Concurrent Cooperation

Marc Toussaint[1] Thibaut Munzer[2] Yoan Mollard[2] Li Yang Wu[1] Manuel Lopes[2]

***Abstract*— In multi-agent domains, human-robot collaboration domains, or single-robot manipulation with multiple end-effectors, the activities of the involved parties are naturally concurrent. Such domains are also naturally relational as they involve multiple objects, multiple agents, and models should generalize over objects and agents. We propose a novel formalization of relational concurrent activity processes that allows us to transfer methods from standard (relational) MDPs, such as Monte-Carlo planning and learning from demonstration, to concurrent cooperation domains. We formally compare the formulation to previous propositional models of concurrent decision making and demonstrate the planning and learning from demonstration methods on a real-world human-robot assembly task.**

I. Introduction

Concurrency is a very natural aspect of robotic domains. For instance, this work was primarily motivated by a human-robot collaborative domain where the robot has two manipulators to assist a human in assembling a piece of furniture. In this domain, the two robot hands (potentially also the two human hands) can be conceived as agents that can execute multiple actions in concurrency. Fluently assisting the human requires to plan ahead, ensuring that pieces are fetched in time, and actions are initiated and terminated at the right times. To give a second example: Any realistic robot system (say, implemented with ROS) will involve concurrently running activities. In the above example, one activity might execute a motor primitive controller for the left hand, another for the right hand, another activity might control the camera pose to focus on the human head, and also perceptual processes such as tracking the human eyes might be conceived as a concurrently running activity. While the coordination of such processes could be thought of as a software engineering issue, we think that reinforcement learning and probabilistic planning methods should be scaled to become applicable to such concurrent activity domains.

In this paper we propose a novel formalization of relational concurrent activity processes. While previous formulations such as coarticulation, concurrent action models (CAM) [13] and Concurrent MDPs (CoMDPs) [9] describe policies as choosing multi-actions, we describe a decision process in which different agents may initiates or terminates activities at different times, and which exploits a relational representation of the current activity state. We will review existing formalisms in detail in the following section.

This work was supported by the 3rdHand EU-Project FP7-ICT-2013-10610878.

[1]Machine Learning and Robotics Lab, University of Stuttgart, Germany. marc.toussaint@informatik.uni-stuttgart.de

[2]INRIA Bordeaux, France. firstname.lastname@inria.fr

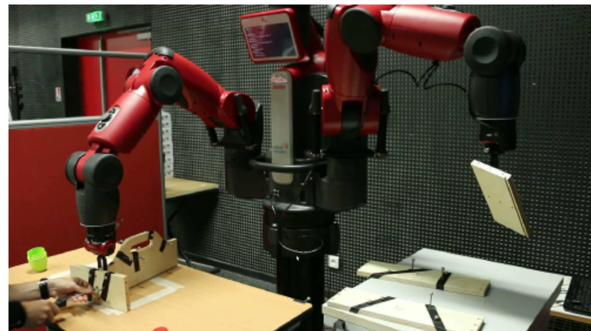

Fig. 1. A robot concurrently using its two arms to assist an operator. It can help human by feeding parts and by holding objects while the operator screws them together. See video at https://vimeo.com/139342248).

In a relational domain descriptions the state is described in terms of properties of, and relations between "objects". In this representation, models of the environment (the transition dynamics and reward function) as well as policies generalize over objects and naturally generalize to domains with different and varying number of objects. Therefore, statistical relational learning (regression & classification learning in relational domains [11]) as well as relational reinforcement learning (rRL) (including model-free rRL [4], model-based rRL [8], and inverse rRL [10]) showed great success in scaling to much larger domains than what would be possible with propositional representations.

In our case of concurrent cooperation, the "objects" we want to generalize over are not only real objects, but also the agents: The two hands of a robot both can potentially execute the same action if the preconditions are met. It is therefore particularly natural to express multi-agent cooperation domains (as well as policies and learned reward functions) in relational terms. Formally, objects as well as agents are equally represented as constants in the underlying first-order logic.

The main contributions of this work are, *first*, to propose a novel formalization of relational concurrent activity processes that is well-suited to model such concurrent cooperation. To our knowledge, this is the first formulation that considers initiation/termination decisions to describe the Markov process and exploit the relational state representation for this. Constraining the formalism to the propositional setting, we show that it is at least as general as the existing concurrent action model (CAM) [13]. *Second*, to transfer existing Monte-Carlo (MC) planning as well as learning from demonstration methods [10] to the concurrent relational domain using this formalism. The MC methods not only

allow us to estimate optimal decisions, they also provide a reward-weighted expectation over future decisions, for instance allowing the system to anticipate future activities of other agents. And *third*, to demonstrate the proposed methods on a real-world human-robot assembly task, where the system uses either the MC planner or a relational policy learned from demonstrations to initiate concurrent activities that assist the human.

In the following section we first briefly sketch the approach of the formalization and explain our distinct use of the words *action*, *activity*, and *decision* throughout the paper, to avoid confusion. Based on this we discuss related work. In Section III we present our model and discuss the relation to CAM and STRIPS. Due to this formalization, MC planning (Section IV) and learning from demonstration (Section V) can efficiently be transferred to the concurrent setting. In Section VI we report on the real-world human-robot collaboration experiments.

II. Overview & Related Work

A. Overview

In standard Markov decision processes (MDPs), the decision variables are also called actions, which have no duration (beyond one step) or concurrency. One of the standard generalizations to the concurrent setting, the concurrent action model (CAM) [13], introduces multi-actions (tuples M of concurrent actions). CAMs are a semi-MDP where the state variable is as before, the decision variable is a multi-action for each phase, phases last for multiple steps, and the termination time of a phase is stochastic.

In our formalism we instead speak of concurrent *activities*, and the set M of currently running activities is part of the relational state. A decision is the initiation or termination of an activity by only one or few agents. Every such decision corresponds to a step of the underlying semi-MDP on the relational state, but only certain decisions lead to an actual real time elapse. While in terms of the semi-MDP, all decisions are sequential, in terms of real time, the initiation and termination of activities may be synchronous or asynchronous and activities arbitrarily concurrent. Policies are single- or multi-agent initiation/termination rules instead of multi-action decision rules.

To avoid confusion, we therefore use the word *action* only when referring to the existing CAM model; in our framework we only speak of *activities* and initiation/termination *decisions*.

B. Related Work

We mentioned the CAM framework of Rohanimanesh et. al. [13] before and will discuss it in detail when explicitly relating it to the proposed formulation in Section III-C. Another core reference for concurrent action planning is [9], which discusses reductions of Concurrent MDPs (roughly, MDPs with multi-actions) and Concurrent Probabilistic Temporal Planning (CPTP) (planning with durative multi-actions) to planning in MDPs with extended state spaces. The relation or difference to CAM, especially between the *aligned* and *interleaved epoch* reductions of CPTPs and the *any*, *all*, and *continue schemes* of CAMs is not explicitly clarified, but they are very similar. Therefore our later comparison also holds w.r.t. CPTP.

Concerning planning algorithms, [14] proposed Temporal GraphPlan (TGP), [15] present a Generate, Test and Debug (GTD) algorithm, and [9] evaluated Real Time Dynamic Programming (RTDP) on the various MDP-reductions. [1] improve on this based on policy gradient methods. We are not aware of previous work using Monte Carlo (MC) methods, despite the recent success of MC methods in very large domains and games [3].

While concurrency has been extensively considered in a planning context, it has received less attention in a reinforcement learning context. Relational reinforcement learning [4], [8] has demonstrated the great benefit of exploiting relational representation in respective domains. But it has not yet been leveraged to learn in concurrent cooperative domains. Similarly, Munzer et. al. [10] presented efficient methods for inverse reinforcement learning in relational domains. The purpose of our work is to allow us to transfer these to the concurrent cooperation setting.

In summary, to our knowledge our formalism is the first to reduce concurrent cooperation planning to (semi-) MDPs by focusing on sequential initiations/terminations instead of multi-actions, leading to a considerably simpler formulation by avoiding the need to formalize mutexes and considering the activity as part of the relational state. Existing formulations have not exploited a relational representation to represent the activity state. Further, we are not aware that previous formulations have enabled to transfer MC planning and relational inverse RL to concurrent cooperation domains, or demonstrated their application in real-world robotic manipulation.

III. Relational Activity Processes (RAPs) for modeling concurrent cooperation

For brevity we first describe RAPs in the deterministic case. However, the reader familiar with non-deterministic decision rules [16] will anticipate how we represent stochasticity by probabilistic effects of all rules. The stochasticity of activity durations is less trivial and needs to be discussed in more detail later.

We consider a relational domain with a set of predicates $\mathcal{P}$. For a given set of constants $\mathcal{C}$ (referring to objects, agents, activities, and other constants) the state s is the conjunction of all true grounded literals.

A RAP is a semi-MDP on the state s, which we define in the following by first defining the set $\mathcal{D}(s)$ of possible decisions in each state, the transition function $(s, d) \mapsto s'$ (below, the transition probabilities $P(s'|d, s)$), then the duration model $(s, d, s') \mapsto \tau$ that defines the real-time duration of a Markov step and the reward function $(s, d, \tau, s') \mapsto r$, and finally optimality of a policy.

Decision Set: Given a set $\mathcal{A} \subseteq \mathcal{C}$ of activity constants, for each activity $a \in \mathcal{A}$ there exist one or multiple *initiation*

operators

$$o_i(a,\bar X):\ \texttt{pre}_i(a,\bar X) \to \texttt{go}(a,\bar X)=\tau_a, \texttt{post}_i(a,\bar X)$$

as well as *termination operators*

$$o_t(a,\bar X):\ \texttt{pre}_t(a,\bar X) \to \neg\texttt{go}(a,\bar X), \texttt{post}_t(a,\bar X)$$

where $\bar X$ is a tuple of logical variables[1], $\texttt{pre}_i(a,\bar X)$ a precondition (as in STRIPS), $\texttt{go} \in \mathcal{P}$ a special real-valued predicate detailed below, $\tau_{a,\bar X} \in \mathbb{R}$ an expected duration, and $\texttt{post}_i(a,\bar X)$ a list of effects (as in STRIPS). For instance

```
start(pickup X Y):
  { (go pickup X Y)! (hand X) (object Y)
            (inhandNil X) (busy X)! (busy Y)! }
  { (go pickup X Y)=2.1 (busy X) (busy Y) }
```

describes the option of agent X to initiate picking up Y. In general, initiation and termination operators may (with $\bar X$) refer to a single agent or multiple agents[2], and single or multiple objects or constants. In state s, the decision set $\mathcal{D}(s)$ includes all grounded initiation and termination operators s.t. $s \models \texttt{pre}(a,\bar X)$. In addition, the decision set includes a single special decision, the `wait` decision.

Transition model: Given state s, if the decision $d \in \mathcal{D}(s)$ is a feasible grounded initiation decision $o_i(a,\bar x)$, we define an intermediate state as $\hat s = \texttt{post}_{i,a}(\bar x) \circ s$, which applies the effects of the operator on s. We further assume a knowledge base KB that includes a set of first-order rules

$$r(\bar X):\ \texttt{pre}_r(\bar X) \to \texttt{post}_r(\bar X)\ .$$

The new state s' is given by the stable model under this KB (cf. answer set programming), that is, the result of forward chaining all rules on $\hat s$ until convergence. Analogously, if the decision is a feasible grounded termination decision $o_t(a,\bar x)$.

In the other case, when the decision is `wait`, the semantics is that all agents decide not to initiate or terminate anything further, and that real time progresses until the relational state changes and activities might terminate. We discuss below how this relates to the *any scheme* in CAM. We concretely define the state transition for a `wait` as the following procedure:

1) Find the go-predicate with the minimal time-to-go value τ_{min}.
2) Decrement all go-predicate-values by τ_{min}.
3) All zero-valued go-literals, $\texttt{go}(a,\bar x)$, are deleted from $\hat s$ and a corresponding the $\texttt{terminate}(a,\bar x)$ is added to $\hat s$.

This defines the intermediate relational state $\hat s$. Again, the new state s' is defined as the stable model under $\hat s \wedge \texttt{KB}$. The KB is assumed to include the rules that express the effects of termination.

[1] We generally use capital letters for variables, small letters for constants/substitutions of such variables, and bars for tuples.

[2] Some activities inherently involve multiple agent, such as one hand handing over an object to another hand, or robot holding a piece for the human to fixate.

Duration model: In the context of hierarchical RL and the standard CAM, steps of the sMDP correspond to the execution of an option, and the duration of the sMDP step is integer-valued, counting the steps of the underlying MDP. However, in general sMDPs the duration of one Markov step is real-valued, arbitrarily depending on (s,d,s'). In the concrete case of REPS we assume that initiation and termination decisions themselves have zero duration, while τ is equal to τ_{min} for the wait decision and therefore implicitly given by the go predicates in initiation operators.

Reward model: Rewards in RAPs are generally given as a relational mapping $(s,d,\tau,s') \mapsto r$. In our applications we represent this as a relational regression tree over (s,s') (e.g., to reward switching to a success state) and a separate (negative) reward function over τ, which are both added to define r.

Optimality: Unrolling a policy generates an episode $(s_0,d_0,\tau_0,r_0,s_1,..)$. We define the discounted return for an episode as

$$R = \sum_{i=0}^{\infty} \gamma^{\bar\tau_i} r_i\ , \quad \bar\tau_i = \sum_{j=1}^{i} \tau_j\ . \tag{1}$$

The objective is to find a policy that maximizes (the expectation over) this return.

A. Stochasticity

State transition stochasticity in RAPs is represented by probabilistic effects of initiation operators as well as KB clauses, exactly as done in NDRs [16]. We propose to generally define duration stochasticity via $P(\tau_{a,\bar x} \mid s,a,\bar x)$, that is, the probability over the time-to-go of an activity $(a,\bar x)$ depending on the current state. Let $M(s)=\{(a,\bar x) \mid s \models \texttt{go}(a,\bar x)\}$ be the set of current activities in state s (that is, the multi-action in the conventional formalisms). Then we define the effect of the `wait` decision by the stochastic procedure:

1) Sample a $\tau_{a,\bar x} \sim P(\tau_{a,\bar x} \mid s,a,\bar x)$ for each $(a,\bar x) \in M$
2) Select the minimal τ_{min} of these.
3) For all $\tau_{a,\bar x} = \tau_{\text{min}}$, delete the $\texttt{go}(a,\bar x)$ literal and add the $\texttt{terminate}(a,\bar x)$ literal to $\hat s$
4) For all $\tau_{a,\bar x} > \tau_{\text{min}}$, modify the value of the $\texttt{go}(a,\bar x)$ literal such that $\mathbb{E}\{\tau_{a,\bar x} \mid s,a,\bar x\}$ reduces by τ_{min}.

In practice, we use Gaussians with the mean defined by the $\texttt{go}(a,\bar x)$-value, which makes the last step simple to realize.

B. Generalization and Comparison to STRIPS

The above formulation differs from standard STRIPS (or its stochastic version, NDRs [16]) essentially in the `wait` operator and respective treatment of the go-predicate, as well as that the new state is the stable model under an additional knowledge base KB. We introduced the latter for representational convenience, allowing for a significantly more flexible declaration of environments. The `wait` operator, however, seems essential for the description of concurrent processes as it defines the relation between Markov steps and real time.

This seems out of the scope of what could be represented in plain STRIPS frameworks.

The special semantics of the initiation and termination operators, and that they necessarily need to set/delete a go-predicate, can be relaxed. An alternative formulation is the following: The rules in the KB are labeled as either "decision rules" or "auto rules"; decision rules are applied when the respective decision is made, auto rules are forward-chained as above to find the next stable model. The decision set is then the set of grounded decision rules s.t. s models $\texttt{pre}_r$ plus the wait decision. What we introduced as initiation and termination operators become special case decision rules. However, the explicit definition of initiation and termination operators clarifies the semantics in concurrent activity processes better.

C. Comparison to Concurrent Action Models

The above formulation is a reduction of concurrent action planning in relational domains to sequential decision making in a relational sMDP. A plan (or unrolling of a reactive policy) will give a sequence of decisions, each referring to a different agent (or set of agents), that can be interpreted by a robot either as own decision or as anticipation of the other agent's decision. This is in contrast to multi-action policies, where it remains somewhat unclear how a single agent should actually react (do his own, single-agent decision) given that the another agent is observed to initiate his own activity.

We want to compare RAP in more detail to the Concurrent Action Model (CAM) as presented in [12]. This comparison becomes most explicit by reducing a CAM model to a RAP model:

Proposition 1: Every CAM process can be represented as a RAP; an optimal of this RAP can be translated back to an optimal CAM policy.

We sketch a prove of this proposition by construction, making the reduction explicit. To this end we limit our RAP model to the case of a propositional state s. We first consider the **decision set**. In CAM, the decision space is the set of multi-actions $M \in \mathcal{M}(s) \subseteq \mathcal{A}^*$ in a subset $\mathcal{M}(s)$ of the power set of the action set $\mathcal{A}$. The subset $\mathcal{M}(s)$ depends on the state and is defined via mutex conditions, expressing that certain actions cannot be chosen concurrently (potentially depending on s). In RAP, let us define a decision episode as a sub-sequence of decisions $D = \langle d_1, .., d_m, \texttt{wait} \rangle$ where all decisions d_i are initiation or termination decisions, except for the last, which is a wait. We need to show that the preconditions of initiation operators can be chosen such that every feasible multi-action decision M can be reproduced by a decision episode D, and that feasible D exist that create an infeasible M. In other words, can the preconditions express constraints that are equivalent to the mutex conditions? If the preconditions are general propositional logic expressions this is clearly the case. But in the concrete case of mutex conditions that are literally mutexes of pairs or tuples of actions, this can very naturally be encoded as negative literals in initiation preconditions. Therefore we can construct an equivalence between $\mathcal{M}(s)$ in CAM and the set $\mathcal{D}(s)$ of feasible decision episodes in RAP. If the constructed initiation precondition allow for all possible permutations of initiation decisions, the $\mathcal{D}$ is larger than $\mathcal{M}$. This could be excluded by construction. However, even then a specific policy in RAP can always generate decision episodes equivalent to any multi-action decision.

In CAM, the **transition and duration model** is jointly and very generally given in terms of $P(\tau, s' \mid M, s)$ for $\tau \in \mathbb{N}$. We can reproduce this in RAP by creating rules in the KB that reproduce this transition whenever the last wait decision of a decision episode generates a terminate predicate; as the state $\bar{s}$ in RAP includes all information of (s, M) in CAM, any (probabilistic) mapping from (s, M) to s' can be realized by the KB. An interesting aspect of the CAM model are the three alternative termination schemes *any*, *all*, and *continue* (see [12] for details). The *any* scheme can be reproduced in RAP when the KB "deletes" all go literals (empties M) on a wait; the *continue* is reproduced simply by not deleting all go literals; and the *all* by introducing a blocked predicate that renders all initiations infeasible, becomes true after wait if there are still activities, and false if there are no left. Note that all three schemes are reproducible only by initiation decisions; allowing also for termination decisions generalizes these schemes.

In terms of the reward and notion optimality CAM and RAP do not differ. In conclusion, every CAM can be expressed as a RAP. Assuming that a RAP planner computes and optimal RAP-policy, it is straight-forward to construct a CAM policy that chooses the multi-action M to be the activity state $M(s)$ after an decision episode D of the RAP policy.

IV. Monte-Carlo Planning in RAPs

A standard approach to planning in single-agent non-concurrent relational domains is UCT (UCB1 applied to Monte-Carlo Tree Search) [7]. As our formulation sequentializes the decision process we can readily apply UCT or other MCTS variants also for planning in concurrent relational domains. For the purpose of the experiments in this paper we utilized the simplest option, namely plain MC estimates of the Q-function over the decision set $\mathcal{D}(s)$ in every step. We empirically found in our specific domains that plain MC behave more robust than other MCTS variants (UCT using MC backups and plain UCB1, UCT using Bellman backups and UCB1 [6]) in terms of not getting stuck in sub-optimal tree branches. We believe this to be a rather special effect of the domains we consider, where "success" is rare. Further, Plain MC has the advantage of allowing us to also compute an unbiased reward-weighted distribution over future states and decisions, for instance allowing one agent to anticipate the future decisions of another.

For generating finite rollouts we need to decide on a termination condition. In many domains, including our example domains, there is a natural termination condition reflecting success or failure. In addition, we always terminate a rollout

in a *dead-end* state which we define as a state without go-predicates (no current activities) and $\mathcal{D}(s) = \{\texttt{wait}\}$. If no natural termination conditions are given, one typically constrains rollouts to a maximal horizon H. If rewards are bounded (as they need to be for UCB1) and $\gamma < 1$, one can choose H to ensure a small upper bound on return that could be collected beyond H.

Finally, when comparing to CAM we mentioned that the outcome of decision episodes may be invariant under permutation of initiation or termination decisions. This seems to introduce large redundancy in the decision tree in comparison to a tree spanned by CAM-multi-actions. While in plain Monte-Carlo this redundancy has no effect, in general MCTS multiple nodes are created for the same state (the effect of a decision episode) which compromises the efficient collection of statistics for this node. This can be corrected for by modifying MCTS to become somewhat like graph search: after a `wait` decision we uniquely sort the decision episode and hash if the same has been sampled before.

V. Learning from Demonstration in RAPs

As for planning, the RAP formulation allows us to transfer existing learning methods to the relational cooperation scenario. We consider both, Direct Policy Learning and Inverse Reinforcement Learning (IRL).

A. Direct Policy Learning

For direct policy learning we consider a data set $D = \{(s_i, d_i)\}_{i=0}^N$ of state-decision pairs from expert demonstrations. Recall that in our case these are multi-agent concurrent activity demonstrations where d_i encodes which agent is (or agents are) initiating or terminating which activity. From this data we learn a policy $\pi : s \mapsto d$ that predicts expert decisions for a novel state.

We propose to use TBRIL [11] within the RAP framework to learn a relational cooperation policy. The TBRIL algorithm represents the policy as a relational regression tree and uses gradient tree boosting [2], [5] to train the model. The training objective is the likelihood under the probabilistic policy model

$$\pi(d|s) = \frac{e^{\beta\psi(s,d)}}{\sum_{d' \in \mathcal{D}(s)} e^{\beta\psi(s,d')}} ,$$

where $\psi(s, d)$ are the features implicitly defined as the leaves of the regression trees.

B. Inverse Reinforcement Learning

In inverse reinforcement learning (IRL) we are given expert demonstrations and assume a generative model that considers these demonstrations as optimal w.r.t. some unknown reward function. The goal is to uncover this reward functions; in our case a relational regression tree that represents such a reward function. Note that without further constraints this is an ill-posed problem as many solutions exist, for example, the reward function equal to zero for any inputs will always be solution of the problem.

We propose to use RCSI [10] within the RAP framework to uncover a relational reward function from expert demonstrations of cooperation. This algorithm decomposes the problem in two steps: First, find a Q-function $Q(s, d)$ such that expert decisions are maxima of the Q-function (i.e. a discriminative function describing the expert policy). This first step is equivalent to direct policy learning and we use the TBRIL algorithm as above. Second, compute a reward consistent with the Q-function by inverting the bellman equation. There is a one to one correspondence between the Q-function and the reward function.

Given the learned reward function we can use planning methods to compute optimal decisions in novel states, with the potential to generalize much better than direct policy learning.

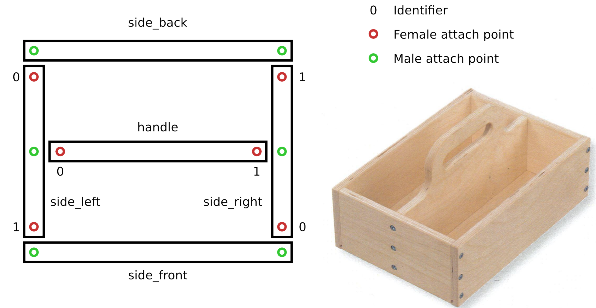

Fig. 2. Schematics representation and photo of the toolbox.

VI. Experiments

We will use two domains to showcase the capability of the proposed model.

A. Example domain: Concurrent assembly assistance robot

In this domain a robot has to assist a human in an assembly task. We have two agents, the two end-effectors of the robot, and a human, which we model using the `KB` as part of the environment. The aim is that the robot fluently assists the human in assembling a box composed of 5 parts.

In order to assemble the box the different pieces have to be 1) put in the human workspace 2) positioned and 3) attached. To put a piece in the human workspace the robot can initiate two activities, *pick(hand, piece)* and *give(hand, piece)*. A third activity, *wait_for_human*, waits for the human to position the next piece. The activity *hold(hand, piece, identifier)* will hold a piece at a specific point allowing the human to screw them together. Two additional activities, *go_home_left* and *go_home_right*, put the robot's arms back in homing position. All activities last one unit of time except for hold which lasts two. Each arm can only be involved in one activity at any time.

As illustrated in Fig. 2, the five pieces are *handle*, *side_right*, *side_left*, *side_front* and *side_back*. The first three have two female attach points, *side_right* and *side_left*. The last four have one male attach point. *side_front* and *side_back* have one male attach points. When talking about positioning or attaching two pieces (because we do not consider impossible builds), 3 arguments are needed to avoid ambiguity (and are used in this order by convention): the object with the

female attach point, the object with the male attach point and the identifier of the female attach point used.

The state of the domain is represented with the following predicates: *attached/3*, *positioned/3*, *in_human_ws/1*, *picked/1*, *at_home/1*, *busy/1*, *free/1*, *occupied_slot/2*, *human_can_do*.

The starting state is always the state where nothing is on the human workspace. This domain is challenging from the planning point of view because a high number (41) of decisions are necessary before reaching the goal state. For learning, on the other hand, the fact that the starting state is unique makes it easier.

B. Example domain: Concurrent blocksworld

This domain is an adaptation of the standard blocksworld domain where two activities can be realized at the same time. Five blocks can be put on top of each other or on a surface (called floor) by two robotic arms.

The domain is represented with the following predicates: *on/2*, *clear/1*, *busy/1* and *in_hand/2*. The activities are *pick(arm, block)* and *put(arm, block, block)*, both of them last one unit of time and both of them can be realized by either arm.

The goal of the task is to stack all blocks in one tower. We generate a random starting state by first sampling the number of initial towers and then uniformly select one state that respect the total number of blocks. In the start state no activities are active.

This domain presents different difficulties than the assistance robot one. The task is shorter in terms of number of decisions which simplifies the planning problem. On the other hand, there are activities with a negative impact, that put the agent further from the goal so random walk is not an effective strategy. Another challenge in this domain, more for the learning problem, is that there are many different paths to reach the goal depending on the start state. Thus, it is important to generalize well from the demonstrations as the policy will often reach states not observed in demonstrations.

C. Simulation Results

Before showcasing the RAP model in a real-world robot assistance domain we evaluate the transferred planning, direct policy learning and inverse RL methods in simulation.

a) MC Planning: For the assistance robot domain, results are presented in Fig. 3. We compare the performance of the planner, in terms of real task execution time and success rate, to the optimal policy and to a random policy. There are 50 trials for every parameter value and results are averaged. After 1000 decisions, if the goal state is not reached, we stop the trial and consider it to be a failure, the success rate is computed as the percentage of success. Two rewards function are used for planner. The first one only rewards the final state while the second one tries to guide the exploration by giving intermediate reward depending on the number of *attached/3* predicate. With few rollouts, the MC planner is noisier than the optimal policy, resulting in longer task execution times. With 200 rollouts and the guided

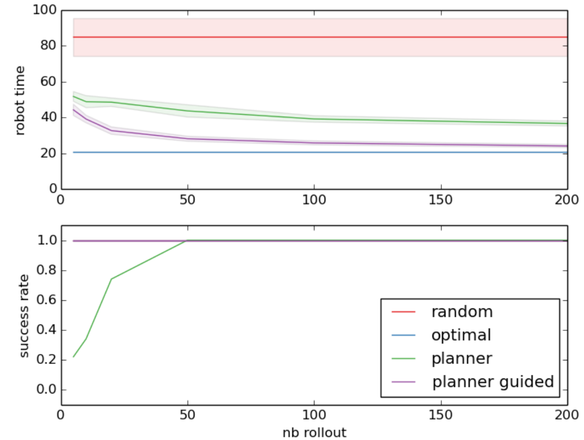

Fig. 3. Performance of the planner on the assistance robot domain. Shaded areas represent standard error.

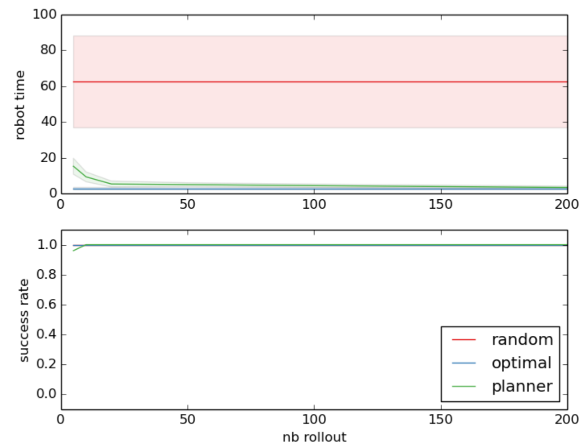

Fig. 4. Performance of the planner on the blocksworld domain. Shaded areas represent standard error.

reward, the planner finds a policy to reach the goal in 24 robot time units against 21 for the optimal policy.

As stated earlier, the planning problem for the blocksworld domain is easier. Fig. 4 shows that with 20 rollouts performance are almost optimal and at 200 rollouts performance are optimal.

TABLE I

PERFORMANCE OF DIRECT POLICY LEARNING FROM DEMONSTRATIONS ALGORITHMS ON THE ROBOT ASSISTANCE DOMAIN.

	robot time	success rate
optimal policy	21.0	1.0
random policy	84.96	1.0
policy learned (1 demos)	21.3	1.0
policy learned (2 demos)	22.25	0.96
reward learned (1 demos)	22.0	0.8
reward learned (2 demos)	21.6	1.0

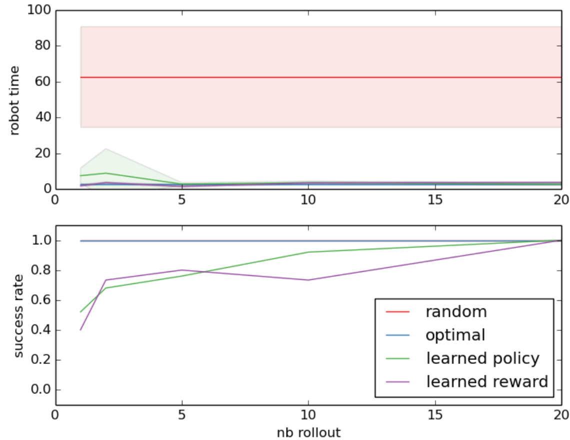

Fig. 5. Performance of the learning algorithms on the blocksworld domain. Shaded area represent standard error.

b) Direct Policy Learning: For the assistance robot domain we show in Table I the efficiency of policy learning from demonstration. With only few demonstrations the policy learned achieves near optimal behavior.

For the blocksworld results are presented in Fig. 5. This domain is more challenging in terms of learning from demonstration because of the number of possible starting states. However, the TBRIL learns the correct behavior with 20 demonstrations.

c) Inverse Reinforcement Learning: The results for IRL are presented in Table I and Fig. 5 and are really close to the ones of policy learning. This is to be expected as we use TBRIL for the first step of RCSI and use the full knowledge of the dynamic model of the world.

D. Robot application

We realized the robot assembly domain on a Baxter robot. To this end we additionally had to implement 1) a sensing module that allows us to detect the truth values of the different predicates and 2) an activity module with a routine for each activity.

Note that now, in real-world, the `wait` decision really waits until the first sensing or activity module reports a termination of an activity or change of state. This replaces the model assumption made in the definition of the RAP model, namely the effect of a `wait` decision in simulation.

For each predicate, a detector has been coded. For example:

- *positioned/3*: The 6D pose of the objects are retrieved with an Optitrack system and compared to the ground truth which is provided to the system beforehand.
- *attached/3*: This is true if it was true in the previous state or if objects are positioned and the human is operating a screwdriver (tracked with Optitrack) nearby for 7 seconds.
- *in_human_ws/1*: We check if the object is in a cube around the human.

a) MC Planning: The planning on the real robot is achieved with a simple loop that retrieves the state of the scene and the re-plans to find the better next decision using the planning algorithm. The previous plan is not reused, the planner starts from scratch each time.

The results are presented in the video supplement at `https://vimeo.com/139342248`. A plan is computed online making only two mistakes resulting in a complete assembly of the box. More precisely, twice the robot holds at a place not needed. We assume that allowing the planner more rollouts would solve this problem. We used the guided reward in this setup.

b) Direct Policy learning: In order to learn the policy, we first recorded some expert demonstrations. A specific command line interface has been developed that allows to command the robot to execute any activity. When a command is given, and before it is executed, the state of the scene and the decision are logged and use later to learn the policy. Once the demonstrations are recorded, the policy is learned. It is then used to control the robot with a simple loop similar to the planning case.

We have successfully learned and played a policy from two demonstrations on the Baxter robot. This learned policy is presented in the video `https://vimeo.com/139342248`.

VII. Conclusions

This work proposes a model of concurrent cooperation that allows for efficient transfer of existing planning and reinforcement learning methods to such domains. While in other formalisms policies map to multi-actions for all agents, RAP describes a process of sequential initiation, termination and wait decisions that each involve only one or few agents and exploits the underlying relational state representation. The knowledge base and generality of activity duration distributions offer great representational flexibility. We compared the generality of RAP to previous concurrent action models. Using RAP we transferred MC planning, direct policy learning and inverse reinforcement learning to relational concurrent cooperation domains, which previously has not been demonstrated. We also illustrated the approach on a real-world robot assistance scenario, where the robot concurrently uses both end-effectors to assist a human in an assembly task.

References

[1] D. Aberdeen and O. Buffet. Concurrent probabilistic temporal planning with policy-gradients. In *ICAPS*, pages 10–17, 2007.

[2] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1):285–297, 1998.

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, and others. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[4] S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001.

[5] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29(5):1189–1232, 2001.

[6] T. Keller and M. Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *ICAPS*, 2013.

[7] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[8] T. Lang, M. Toussaint, and K. Kersting. Exploration in relational domains for model-based reinforcement learning. *The Journal of Machine Learning Research*, 13(1):3725–3768, 2012.

[9] Mausam and D. S. Weld. Planning with durative actions in stochastic domains. *J. Artif. Intell. Res.(JAIR)*, 31:33–82, 2008.

[10] T. Munzer, B. Piot, M. Geist, O. Pietquin, and M. Lopes. Inverse reinforcement learning in relational domains. In *International Joint Conferences on Artificial Intelligence*, 2015.

[11] S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86(1):25–56, 2012.

[12] K. Rohanimanesh and S. Mahadevan. Learning to take concurrent actions. In *Advances in neural information processing systems*, pages 1619–1626, 2002.

[13] K. Rohanimanesh and S. Mahadevan. Coarticulation: An approach for generating concurrent plans in markov decision processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 720–727. ACM, 2005.

[14] D. E. Smith and D. S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI*, volume 99, pages 326–337, 1999.

[15] H. a. L. Younes and R. G. Simmons. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS*, volume 4, page 325, 2004.

[16] L. S. Zettlemoyer, H. Pasula, and L. P. Kaelbling. Learning planning rules in noisy stochastic worlds. In *AAAI*, pages 911–918, 2005.