DOI: 10.48011/asba.v2i1.1448

Towards Adaptive Discrete Event Control Based on PRD, PSS and Automatic Planner

Gabriel de Almeida Souza *
José Jean-Paul Zanlucchi de Souza Tavares **
José Reinaldo Silva ***

* *Faculdade de Engenharia Mecânica, Universidade Federal de Uberlândia, MG, (e-mail: gabrielsouzaworking@gmail.br).*
** *Faculdade de Engenharia Mecânica, Universidade Federal de Uberlândia, MG, (e-mail: jean.tavares@ufu.br)*
*** *Escola Politécnica, Universidade de São Paulo, SP, (e-mail: reinaldo@usp.br)*

Abstract:
Industry 4.0 technologies integrate devices and data, bring flexibility, efficiency and decision making, derived from decentralization. In a post pandemic society it is mandatory to reduce human presence in production and distribution of goods. This work implements some of Industry 4.0 characteristics by combining manufacturing elements such as Cyber Physical System (CPS) with passive entities that directly affect decisions in the same automatic planning domain. The proposal is illustrated by emulating a Block World problem, where it will be used a set of blocks with Radio Frequency Identification (RFID) each one containing its self goals, represented by predicates, an approach called PRD (Predicate inside RFID Database). A robot can identify objects by holding a RFID reader integrated with a Physical State Space (PSS). Since the robot controller has a local view of the process it is unable to compute the plan for the whole system by itself, so the planning process must be a Cloud service. Local planning must also be taken into consideration, solving any network issues. Thus, a generic solution has to be adapted to fit physical execution and domain constraints. Such solution detects changes in the physical environment and redo its plan, generating an adaptive discrete event controller.

Keywords: Adaptive Discrete Event Control, Automatic Planning, PRD, PSS, Cyber-Physical Systems.

1. INTRODUCTION

Cyber Physical Systems (CPS) are one of Industry 4.0 cornerstones, in the sense that information management, decision making and networking became deeply embedded into automated systems (Jazdi, 2014). A paradigm shift has been happening, since connectivity, bandwidth, computational power and accessibility increased significantly in recent years (Vuksanović et al., 2016). With greater processing power and information, more flexible systems are feasible, aiming at mass custom production (Rojko, 2017), supported by Internet of Things (IoT) and Cloud Computing, many industries and technical areas are changing towards smart, informed decision making. The recent improvements have to be further developed and aim at increased autonomy to avoid human contact in products, reducing health issues in the post pandemic society.

The fields of robotics and automated planning have to undergo adaptations to fit the post pandemic society, by exploiting the increased power in the CPS, specially robots. Simple state-machined solutions for robots are not enough, as flexibility and higher level of autonomy are now a mandatory requirement. Supported by computational power and relevant information, CPS must become more autonomous even under network malfunction.

Different decision making techniques, based on reinforcement learning, on classical automated planning, on data hungry methods, and others are currently being further developed and tested. Hofer (2017) uses Markov Decision process in the complex problems related to robot soccer, and Guérin et al. (2018) uses deep neural networks to process images and cluster objects for sorting. Both techniques require computational power that was not available until recently, and it could be helpful in decreasing direct human intervention in production and distribution of goods. In Michniewicz and Reinhart (2016) robot cells are virtualized and optimized regarding production flux and kinematics in an offline methodology. In Xue et al. (2009) kinematics and dynamics for grasping robots are used to improve grabbing by modelling relevant properties in the objects, offline. Tavares and Souza (2019) show PNRD/iPNRD (Petri Net Inside RFID Database and inverted Petri Net Inside RFID Database) integration solving Block World domain with three blocks in an adaptive discrete event control architecture, which is an online solution method and fully model based without training necessity, contrasting with most current architectures. PNRD/iPNRD data is held into each passive agent through a RFID tag and into each active agent inside

a RFID reader. When these components are combined, the iPNRD initial and objective states can be determined, as well as the physical positioning from a Physical State Space (PSS) represented by a Petri Space. A breadth-first search (Russell and Norvig, 2010) was implemented to find the iPNRD transitions required to reach the final state, generating an adaptive discrete event control. This approach does not deal with domains where the generation of the complete state space is prohibitive. In this case, it is necessary to use more powerful tools, such as automatic planners. Although PNRD/iPNRD approach is helpful in reducing human involvement, it does not scale well and is not very versatile, as the individual and global state-spaces are preconfigured.

The current approach is network independent and focused on discrete event dynamics, and does not deal with the optimization of the continuous part of movement. Instead, it offers an automated planning service both in Edge and Cloud computing. Nevertheless, an automated planning server requires high computational power to generate a plan and return execution steps; while a CPS controller is usually limited computationally.

IoT and Cloud Computing have made acting with continual online planning possible for CPS, as shown in da Silva Fonseca et al. (2016), where automated planning integrated with programmable logic controllers is discussed. Yet, an intermediary software for different functions is necessary, to bridge planning and action in an user comprehensive abstraction.

An open issue in automated planning field is that several works underestimate the importance and difficulty of deliberative acting (Nau et al., 2015) in real online applications, where acting on abstract plans and deriving concrete refined actions is computationally hard. The classical automated planning system is described in Figure 1. The system receives external information about the domain's current and objective states, meaning that the main planner must receive this data as input from a previous process. The planner feeds a command as a instruction string for the controller to actuate on the system. The controller gets feedback information from the system, as the plan outcome is disturbed by external events, so that the plan can be redone to deal with some perturbations. This conceptual architecture is adaptive in nature. This figure presents clearly that, currently, the main focus of automated planning research is in the planning itself (Ghallab et al., 2004), as perceiving and deliberating are not deeply explored in the planning community academia.

PDDL is a formal textual language used for describing domains (AIPS-98 et al., 1998), and it is used as input language to planners, as a portable system descriptor. It is the language used in this work, to send input into the planning service.

This paper shows how automatic planning can be modelled, adapted and offered as a service, using a comprehensive abstraction level to solve a problem that is close in complexity to real problems combined with RFID devices. The idea on identifying each object desired state is based on embedding predicates in RFID tags. Those predicates can be individually accessed and can be united into Grouped Individual State Predicates (GISP), which is used to create problem snapshots. This technique which

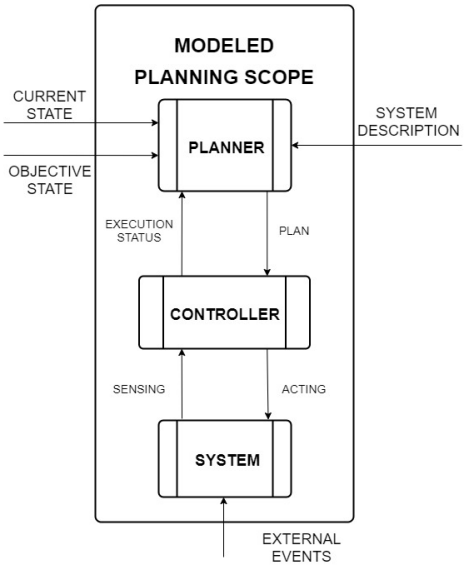

Figure 1. Classical Automatic Planner Model

integrates GISP and RFID readers and tags is called PRD or Predicate inside RFID Data structure. PDDL associated with PRD generates a powerful tool set, that can be used to solve bigger domains than previously explored by PNRD/iPNRD.

This work presents how GISP, implemented as PRD, integrated with automatic planning service can assist adaptive control on a PSS, through exception identification. Block World variants are explored for demonstration.

Next section presents the proposed infrastructure and control architecture. Section 3 shows how planning and acting can be integrated; followed by the PRD implementation with two examples, the first one for a single actor, and the second for two or more actors. Conclusions and further works are presented.

2. PRD PROPOSED ARCHITECTURE

The proposed architecture aims to improve the system flexibility, by being capable of receiving less constrained snapshots and importing planners to act on the real system.

2.1 PRD and PSS structures

As information, PRD and auxiliary data are represented using markings, with *:marking* notation. So a current state description can be inferred by searching for a *:current* and reading the information between brackets, usually written as predicates separated by commas. A series of consecutive objectives can be located using a *:objective-N* notation, where N references the $(n)^{th}$ objective. Auxiliary information can be addressed using the same system, as an example *:temperature* and *:priority* are markings that can be used to supplement or confront information regarding the objects past and future.

Figure 2 shows an example of PRD representation to represent current and objective states. In *:current*, three distinct predicates are present to describe the current situation, referencing other objects and entities in the domain, while in *:objective-0* two predicates are used to describe the desired state, with *status()* referencing a completion state.

The *:priority* marking can be used to indicate preference, if there is objective discordance between agents or constrained resources. It is the active agent job to process and infer information with the predicates.

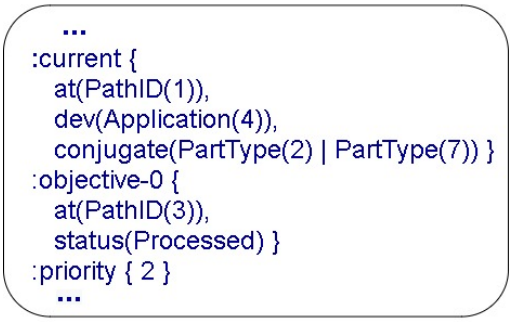

Figure 2. PRD data structure in passive agent

PRD is an approach used to store, read and write predicates into RFID tags, effectively giving decision power to passive agents. For instance, tagged *part P* in a production line, until it reaches a divergence, as shown in Figure 3, where active agent *robot R* needs to decide which path *part P* will take, *A* or *B*. There is no geometrical or external information to influence the decision, and any active agent should decide the same.

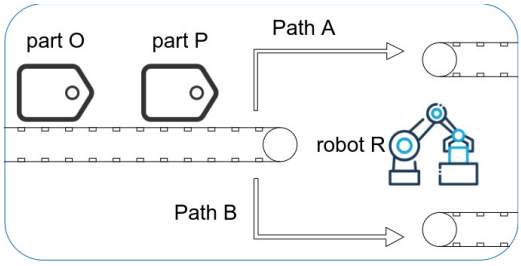

Figure 3. Production Line Divergence

This decision is possible if the passive agent itself provide information to the active agent, so it may act according to the passive agent information. Using PRD, *robot R* can get information on which path *part P* should follow, be it directly related to the path itself, or some destiny, like go to packaging or go to quality control. This information can be accessed in the *:objective-0* marking, as Figure 4 shows.

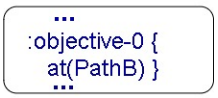

Figure 4. PRD *:objective-0* marking inside *part P*

In this case, *part P* explicitly informed *Path B* as objective. *robot R* will parse this marking and infer, using a database, that it has to act on a way that takes *part P* towards *Path B*. This method is more powerful than using other sources of information as Petri Nets, because the predicate representation aggregates flexibility as an uniform language between automatic planners and object data structure. As *R* moves any *part X*, it can update PRD data inside each *part X* tag.

Individual PRD data can be grouped and used to produce richer descriptions and more complex applications. To observe coupled effects on grouped data, imagine that *part O*, in Figure 3, has a predicate that specifies it must follow the same path as *part P* (Figure 5). This causes information in *P* and the final decision to also affect *O*, aggregating dependencies in causality, and complexity.

Figure 5. PRD *:objective-0* marking inside *part O*

2.2 Automated Planning Integration with PRD and PSS

In order to generate a feasible automated plan model, the system model must be a discrete domain type, and its environment must deal with disturbance.
PSS is a structure, normally represented by a digraph, used to represent discrete positions and adjacency relationships in the physical domain scope, useful to discretize continuous movement into simpler discrete transitions. PSS exploration and searching can be done using different techniques with different levels of complexity. Simple state machine exploration is the easiest to execute and requires minimal computation cost. Offline optimized methods may include conditionals, as discussed in Souza et al. (2019).
Online path planning is the most movement efficient approach regarding search in the PSS, but at higher costs than simpler methods. In practice, usually the PSS dimension is pre-established and the active agent has knowledge about its place and exploration limits (kinematic and dynamic constraints).
While the planner is responsible for decision making concerning model's internal choices, usually with an optimization criteria; the controller must have an interface for sensing and acting, which may include actuator driver interfaces. In order to structure the adaptive system, the PRD approach proposes that sensors are RFID tags in passive agents, and reader during reading activity; while actuators are active agents (as robot arm) coupled with RFID reader in writing activity.
The physical domain adds constraints and complexity that are not present in pure computational domains, that must be taken into consideration for the PRD architecture structure and models. Some physical concerns are: Snapshot configurations, as in identifying the passive agent current state, and inferring if its objective can be reached or not; movement dynamics and control, as in using the adequate level of abstraction; and partial observability, to understand that the view in the domain current state could be wrong due to unperceived disturbances. Even though the robotic arm runs a partially observed system, due to the fact that each block is read in different moments, exceptions may still be observed, when the system explicitly interacts with a non-conformant object or relationship. This PRD and PSS proposed architecture is presented in Figure 6.
The PRD and PSS implementation is divided into three modules, *RFID Sensing*, *Planning* and *Execution* (Figure 7). This process must initially identify all objects configuration to be fed into the planner, which outputs an action

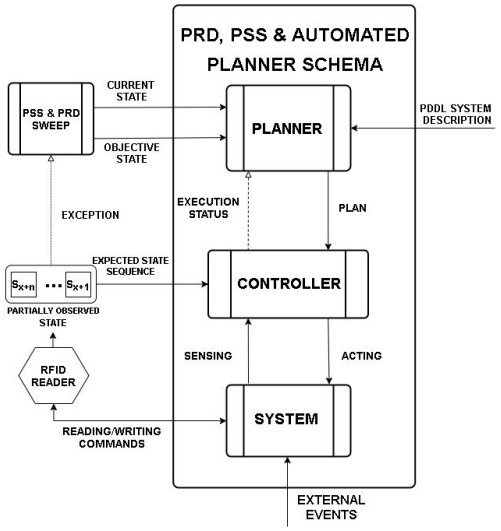

Figure 6. *PRD* and *PSS* integrated with Automatic Planner Conceptual Adaptive Discrete Event Control Schema

plan for the *Execution* module. These activities repeats itself until all object's objectives are successfully achieved, self-correcting if necessary.

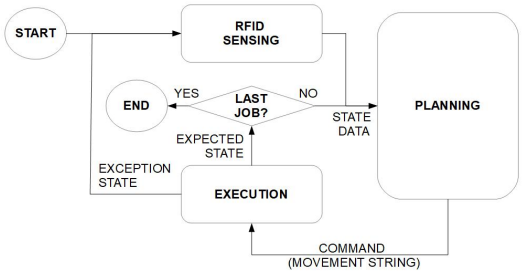

Figure 7. Adaptive Controller Integrated With Automatic Planner Flowchart

The *RFID Sensing* module observes the environment, by capturing the passive agents PRD data structure (Id, current and objective state) by moving the active agent integrated with RFID reader embedded into each discrete position following the PSS. This step is terminated only when the PSS is exhausted.
Next step is to infer about the grouped PRD data current state and objectives called *State Inference*. Current State inference can also be defined by information gathered from positioning in the PSS. This data can be confronted with *:current* tag predicates, to observe inconsistencies. Objective state is generated by grouping all individual *:objective-0* predicates. It must verify whether the objective state is valid.
Exception identification is a complex issue, due to many different situations, such as many configurations being impossible to physically realize, opposing objectives, missing referenced objects and other problems. General rules for quickly detecting exceptions can improve system reaction; however a general solution is still a challenge. If an exception arose the identification process must be restarted, else the observations are transferred as input (State Data) into the *Planning* module. Figure 8 details the *RFID Sensing* module.

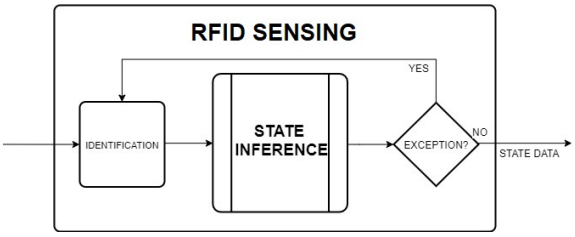

Figure 8. PRD's *RFID Sensing* Module

The *Planning* module is a two-level planner which is responsible for both the discrete state planning, and the physical movement planning. Figure 9 shows that both planners need a knowledge base, which is composed of a system description coupled with constraints and heuristics. It is better explained at section 3.
A planner requires two distinct files to work, one describing the domain and other the problem. A basic domain description is composed of existing classes declaration and existing actions as functions with preconditions and consequences. A basic problem description includes a repository of existing objects, current snapshot and objective conditions, gathered from PRD data capture.

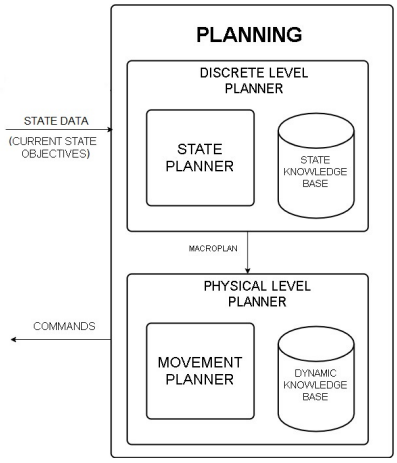

Figure 9. PRD's *Planning* Module

The *Execution* module is composed by the active agent controller (machine controller) and an interpreter knowledge base, which define the next action to be performed and inspected, similar to the *RFID Sensing* identification to check if the achieved state is equal to the desired one to update *PRD* data inside each tag (Figure 10). If a cycle has correctly ended a job conclusion query is performed to start the next execution or finish control as previously presented at Figure 7.

2.3 Planning as a Service

The planner can be either accessed as Edge or Cloud service, as long as the problem can be computed in effective time, and the network is in good conditions. Regarding

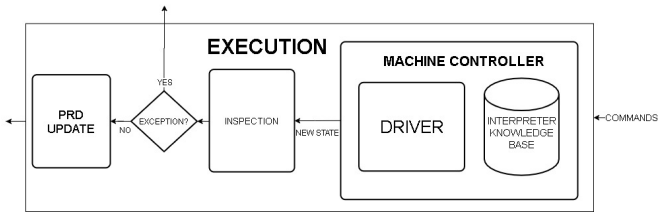

Figure 10. *Execution* Module

costs and speed, either can be more efficient, depending on the problem, network and target hardware. Ai et al. (2018) points out that Cloud computing has been consolidated as highly efficient and flexible, in a centralized architecture. But, due to the increasing presence of IoT applications, network time response, its constraints and derived problems cause the centralized paradigm to be inefficient. So Edge and Hybrid solutions should be investigated.
Empirically, it was decided that the best approach for this problem is to host the planner on the Cloud, and have a local machine in the Edge as a backup, in case of elevated latency or unavailability. This produces a redundant mechanism, to guarantee operation even under network failures. This topic needs to be explored in a future work.

3. INTEGRATING PLANNING AND ACTION

To integrate properly planning and action there is a need to generate a plan with sufficient level of refined abstraction that a machine may correctly interpret the steps (Ghallab et al., 2016). This identifies that the high abstraction steps do require more details, were those details can not always be completely embedded in its parent as constants, requiring methods for their definition, sometimes requiring sub planning.
As an example, imagine a general purpose robot. Upon receiving a high level task, this task has to be decomposed into refined steps, until they are simple enough for action. Figure 11 exemplifies the abstraction levels in deliberation, as a simple "bring object *o7* to place *room2*" command is too complex to be directly turned into action, until its abstraction level is sufficient for the robot to act.
To avoid working on multiple levels of abstraction, the

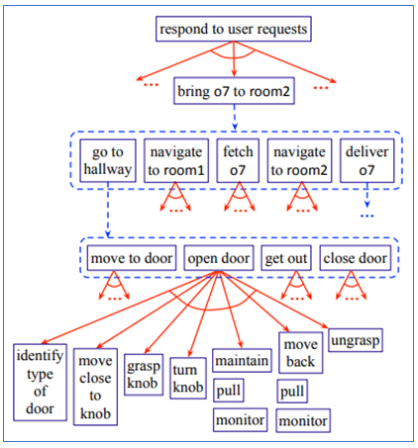

Figure 11. Multiple levels of abstraction in deliberative acting Ghallab et al. (2016)

proposed PRD approach is defined with two planning levels where the last one (movement planner) converts an action plan into control references for the robot working on the PSS.
PDDL is used in this work to feed the automated planner, on planning phase. It was chosen because it is an ever growing language widely used in the automatic planning community scope, as it was the official language of many occurrences of the IPC (International Planning Competition).
The chosen automated planner is Fast Downward, an open source full planner sustained by research groups (J and G, Seipp J and Roger G 2018). It supports many distinct input languages, search algorithms and heuristics, and outputs detailed information on parsing, searching and, of course, the generated plan.

4. PRD IMPLEMENTATION EXAMPLES

This section presents two Block World domain implementations using PRD approach, the first one with one robotic arm, and the second one with two.

4.1 Block World PDDL Domain Definition

Modelling Block World for automated planning can be done in several forms. This example follows Vaquero (2007) method. It is important to note that the final PDDL domain definition is highly coupled with architectural processes, PRD and the physical implementation, so predicate details may vary.
The model is composed of three classes *Position*, *Block* and the *RoboticArm*. *Position* and *Block* are passive entities, having no method, while the *RoboticArm* is an active agent and possesses methods for promoting movement changing. Two main versions of the PDDL domain are needed, one that supports only one actor, and another supporting multiple actors. The second domain aggregates complexity by including attributes to avoid collision between actors, making it less efficient in problems with only one actor.
There is also the physical implementation problem for multiple actors, were the architecture would either need some middleware to coordinate the actors and access the planning service or support decentralization. Those multiple actors issues must be discussed in a further work.
For both domains preconditions and conditional effects are declared. The domain specifications for one actor are defined in Figure 12. 10 (ten) predicates are declared, 4 (four) to create adjacency relations between *Position*, 2 (two) for *Arm/ Block* possession, 3 (three) as implicit constraints for moving and picking *Block* and 1 (one) to indicate if a *Block* is stacked atop another *Block*. The last described predicate is very useful, because it allows for relative objectives and facilitates the implementation of grouped individual objectives strategy. 6 (six) actions are declared, 4 (four) for moving and 2 (two) for picking and dropping blocks. All actions have unitary cost. The other moving actions were hidden because the only difference is in the adjacency precondition.
In the multi agent PDDL domain, the system has one additional predicate used to avoid agent collision and actions with more specifications.

```
; Block World One-Actor
(
  define (domain BlockWorld-Mecatronics)
  (:requirements :typing :negative-preconditions :disjunctive-preconditions
    :conditional-effects)
  (:types arm - agent
    block place - object)

  (:predicates
     (holdarm ?p - place ?a - arm)
     (holdblock ?x - (either arm place) ?b - block)
     (freetopick ?a - arm)
     (freetomove ?p - place)
     (freetomoveblock ?p - place)
     (adjacentup ?p1 ?p2 - place)
     (adjacentdown ?p1 ?p2 - place)
     (adjacentleft ?p1 ?p2 - place)
     (adjacentright ?p1 ?p2 - place)
     (stackedblock ?b1 ?b2 - block)
  )
  (:action pickblock
    :parameters (?a - arm ?b ?bd - block ?p1 ?p2 - place)
    :precondition (and (holdblock ?p1 ?b) (freetomove ?p1) (freetopick ?a)
      (holdarm ?p1 ?a) (adjacentdown ?p1 ?p2) (not (freetomoveblock ?p2)))
    :effect (and (not (freetopick ?a)) (not (holdblock ?p1 ?b)) (holdblock ?a ?b)
      (freetomove ?p2) (freetomoveblock ?p1)
      (when (holdblock ?p2 ?bd) (not (stackedblock ?b ?bd))))
  )
  (:action dropblock
    :parameters (?a - arm ?b ?bd - block ?p1 ?p2 - place)
    :precondition (and (holdblock ?a ?b) (not (freetopick ?a))
      (holdarm ?p1 ?a) (adjacentdown ?p1 ?p2) (not (freetomoveblock ?p2)))
    :effect (and (not (freetomove ?p2)) (freetopick ?a) (not (holdblock ?a ?b))
      (holdblock ?p1 ?b) (not (freetomoveblock ?p1))
      (when (holdblock ?p2 ?bd) (stackedblock ?b ?bd)))
  )
  (:action moveup
    :parameters (?a - arm ?p1 ?p2 - place ?b - block)
    :precondition (and (adjacentup ?p1 ?p2) (holdarm ?p1 ?a)
      (freetomove ?p2) (or (freetomoveblock ?p2) (freetopick ?a)))
    :effect (and (not (holdarm ?p1 ?a)) (holdarm ?p2 ?a))
  )
      ...
)
```

Figure 12. Domain specification with one Actor

```
; Block World One-Actor
(
  define (problem one-actor)
  (:domain BlockWorld-Mecatronics)
  (:objects
     blockA blockB blockC blockD - block
     place_0_0 place_0_1 place_0_2 place_0_3 place_0_4
     place_1_0 place_1_1 place_1_2 place_1_3 place_1_4
     place_2_0 place_2_1 place_2_2 place_2_3 place_2_4
     place_3_0 place_3_1 place_3_2 place_3_3 place_3_4 - place
     arm0 - arm
  )
  (:init
     ; adjacency
     (adjacentup place_0_0 place_0_1)
     (adjacentdown place_0_1 place_0_0)
      ...
     (adjacentleft place_3_4 place_2_4)
     ; blocks and arm
     (holdblock place_1_1 blockA)
     (holdblock place_2_1 blockB)
     (holdblock place_3_1 blockC)
     (holdblock place_3_2 blockD)
     (holdarm place_1_4 arm0)
     (freetopick arm0)
     ; unconstrained places
     (freetomove place_0_1)
     (freetomove place_0_2)
      ...
     (freetomoveblock place_0_1)
     (freetomoveblock place_0_2)
      ...
     ; stacks
     (stackedblock blockD blockC)
  )
  (:goal (and
     (holdblock place_2_3 blockA)
     (holdblock place_2_1 blockB)
     (stackedblock blockD blockB)
     (holdarm place_1_4 arm0))
  )
)
```

Figure 13. Problem specification for one robotic arm

4.2 Block World Problem Definition

The next step is to define the problem, it means, an object repository, the current snapshot and objectives into a single problem file. The repository indicates all objects that are involved. A snapshot is a set of objects and relationships defining a specific moment. The objectives are predicates that indicate desired attributes to be achieved for a set of objects.
A problem specification file for one robotic arm is presented in Figure 13. Basically it is divided into a header, where the problem receives a name and a father domain, the repository, current conditions and objectives. The problem is defined after obtaining information from the PRD in the distributed objects.

4.3 PRD example with one robotic arm

As previously informed, to locate all *Block* objects, the PSS is exhausted. Positions are mapped rightwards and upwards in a *Position(X,Y)* notation, starting at zero. In this example, *Block A, B, C, D* are found respectively in positions (1,1), (2,1), (3,1), (3,2) as presented in Figure 14. Inside each RFID tag, in the *:current* marking, predicates describing their current state can be found, to contrast with PSS information. In the *:objective-0* marking, a set of objectives can be collected, for each tag. In this example, each tag holds, at most, one objective predicate about itself, to be put in a certain position or relative to another block.
After locating and reading PRD data inside each tag, current predicates and objective predicates are united to generate the problem file. Information regarding adjacency and unconstrained places was hidden to reduce space. The corresponding domain file must also be sent to the planner. Figure 14 represents the current snapshot described inside the Problem file on a graphical representation. Bottom *Position* objects, in pink, have attributes that make them behave like a table (by setting false all movement predicates), so *Blocks* can be put atop and the *Robot* can not move into these positions. *Block A, B, C, D* are identified in blue and the *ARM* in green. All adjacency relationships are implied by *Position*.

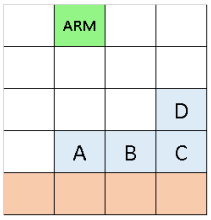

Figure 14. Initial Snapshot with one actor

The objective, described in natural language, is an *"AND"* composition of individual objectives, following the GISP paradigm: *Block D* atop *Block B*, *Block A* in (2,3), *Block B* in *(2,1)* and *ARM* at *(2,4)*. This objective description includes absolute positioning, as in *Block A* occupying *(2,3)*, relative positioning, as in *Block D* atop *Block B* and null objectives, as *Block C* did not state an objective. This means that many distinct configurations might successfully satisfy the objectives, as *Block C* can be put anywhere.

Afterwards, feeding in the Domain and Problem files, the Fast Downward parser will convert PDDL into structures suitable for searching. After generating 2.696 states, the optimal plan is composed of 16 steps suitable to robotic command, as shown in Figure 15. The optimal final configuration can be observed in Figure 16.

```
; Block World One-Actor
  movedown arm0 place_1_4 place_1_3 blocka
  movedown arm0 place_1_3 place_1_2 blocka
  moveright arm0 place_1_2 place_2_2 blocka
  moveright arm0 place_2_2 place_3_2 blocka
  pickblock arm0 blockd blocka place_3_2 place_3_1
  moveleft arm0 place_3_2 place_2_2 blocka
  dropblock arm0 blockd blockb place_2_2 place_2_1
  moveleft arm0 place_2_2 place_1_2 blocka
  movedown arm0 place_1_2 place_1_1 blocka
  pickblock arm0 blocka blocka place_1_1 place_1_0
  moveup arm0 place_1_1 place_1_2 blocka
  moveup arm0 place_1_2 place_1_3 blocka
  moveright arm0 place_1_3 place_2_3 blocka
  dropblock arm0 blocka blocka place_2_3 place_2_2
  moveleft arm0 place_2_3 place_1_3 blocka
  moveup arm0 place_1_3 place_1_4 blocka
Plan length: 16 step(s).
```

Figure 15. Generated Plan for One Actor

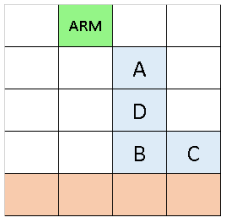

Figure 16. Generated Final Snapshot with one actor

Non-coherent problem specifications are still an open issue in identification, in the perception phase, prior to being fed to the planner, and are to be discussed in a further work. However the planner generally will accuse errors in the problem input file, if it can not be solved or a predicate regarding the same objects in *:init or :goal* is stated multiple times. This exception identification problem must be allocated inside the compiler that converts PRD data into problem files, or be a process afterwards.

4.4 PRD example with multiple actors

Figure 17 represents the initial snapshot, after collecting and inferring data from the PSS and PRD. The objective conditions are described, as an *AND* composition: *Block A* in (3,1), *Block B* in (1,1), *Block C* in (2,1), *Block D* in (1,2), *ARM0* in (3,4) and *ARM1* in (0,4). As all individual objectives are absolute in nature, the objective snapshot can be directly represented in Figure 17.

After generating 12.608.624 states, the optimal plan is composed of 36 steps.

4.5 Search Time Performance

The server is hosted in a computer with Intel$^{(R)}$ Core^{TM} i5-4210U CPU @ 1.70GHz x64, 4.0 GB DDR3 RAM using Linux with 4.15 generic kernel version. It was used Apache2 web server, programmed in PHP, to access through the Internet. A *C* language program was also built

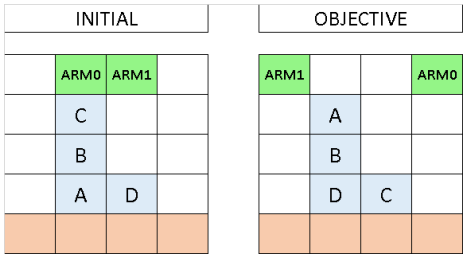

Figure 17. Objective Snapshot with two actors

to maintain a direct serial connection. The first scenario has planning time of 0.011 seconds while the second one was planned in 11.2 seconds.
Regarding the planning options, for both examples the A* search algorithm is used (Russell and Norvig, 2010) coupled with h_{max} heuristic (Keyder and Geffner, 2008). Another reason to avoid explicit axioms is that h_{max} does not keep its properties under axiom use (admissible, consistent and safe).
Those options were specified because the A* search assures optimal plan generation, and the h_{max} heuristic had the best performance for the example problems among all available options. Optimal plan generation is relevant because the bottleneck in time consumption for this application is expected to be the physical movement part.
Comparing the search results in the one actor example and multi actor example, it can be inferred that a small gain in complexity in the domain description coupled with the addition of an actor caused the problem to be significantly harder to solve optimally. The number of generated states and search time increased in magnitude by many orders, while the depth more than doubled. Time is to be taken as a relative comparative only, as both examples used the same computer.

5. CONCLUSION AND FURTHER WORKS

The proposed architecture integrates generic automatic planners into CPS using PRD and PSS. It is a bridge between automatic planning and adaptive discrete event control which can be applied in several other domains in an innovative way to reach multi agent goals and objectives, further increasing autonomy and reducing human presence. It differs from other architectures especially because of the PRD presence, which causes the objectives to already be present in the work space, not requiring an external interface to input the objectives. It also embeds decision power into passive entities. The generic automatic planning core in its planner block also gives it power to deal with many different scenarios and objectives, being more generic than architectures specialized solely for movement actions, the main limitation being the model itself.
The PRD is a flexible data structure for passive entity decision making. Its limitations are due to RFID memory size, because current tags have very constrained capacity, RFID reading issues, and the need of having a specialized data analyser in the RFID reader to correctly process each marking. It must be noted that information must be pre-recorded in the tags, so previous processes are necessary to insure sufficient data is recorded in each object.
The architecture of the controller described to be implemented in a physical version is composed by many pro-

cesses, each dealing with a specialized scope in solving the problem, where they can be optimized. This optimization, in turn, can be model costful or case specialized, so obtaining good metrics and insights for improving the system is challenging, being subject to further works.

An absolute generic marking processor is not possible, not because it would require general standardization, but because domains themselves vary greatly, so an abstraction under a certain context may be different from another. What is more feasible is to build processors for families of applications, where data bases and rules can be shared.

This approach deals with partial observation due to RFID reading characteristics. This causes uncertainty issues, where the domain could have been disturbed by external forces between actions. The system deals with partial observation by raising exceptions whenever something is in disagreement or unfeasible, returning to a previous point in control to correct. This could be improved by adding sensors and more powerful inference mechanisms to better track objects and detect exceptions earlier. Action behavior may also be specified to act as default, whenever an objective exception arises or the system is given instruction that can not be further processed by its data set.

The proposed architecture is not yet generic enough to be used in a diversity of domains, specifically those dealing with multiple decentralized actors or with action abstractions that can not be directly refined, requiring multiple levels of search. The planner was also built considering pure discrete dynamics, so including hybrid systems is an important generalisation step towards supporting domains with continuous dynamics that can not be modelled in discrete transitions. A supervision interface was not modeled, so outputting in real-time human readable data and allowing for human intervention are useful functions.

Vaquero (2010) points out about post-design analysis for AI applications and the refinement cycle, it means, a direction into improving domain definitions. Post-analysis can be useful in all described processes in this architecture.

This paper main contributions are to clearly show that automated planners can be applied in practice and in cases where regular solutions, usually state-machined ones, are not suitable.This can be observed by inferring that state-machine solutions are not capable of handling PRD under a least constrained practice.

A Cloud/Edge planning solution using PRD data must be implemented.

REFERENCES

Ai, Y., Peng, M., and Zhang, K. (2018). Edge computing technologies for internet of things: a primer. Digital Communications and Networks, 4(2), 77 – 86. doi:https://doi.org/10.1016/j.dcan.2017.07.001. URL http://www.sciencedirect.com/science/article/pii/S2352864817301335.

AIPS-98, Planning, Competition, and Committee (1998). PDDL: The Planning Domain Definition Language.

da Silva Fonseca, J.P., de Sousa, A.R., Ferreira, M.V.M., and de Souza Tavares, J.J.P.Z. (2016). Planpas: Plc and automated planning integration. International Journal of Computer Integrated Manufacturing, 29(11), 1200–1217. doi:10.1080/0951192X.2015.1067909.

Ghallab, M., Nau, D., and Traverso, P. (2016). Automated Planning and Acting. Cambridge University Press.

Ghallab, M., Nau, D., and Traverso, P. (2004). Automated Planning: Theory and Practice. Morgan Kauffman.

Guérin, J., Thiery, S., Nyiri, E., and Gibaru, O. (2018). Unsupervised robotic sorting: Towards autonomous decision making robots. International Journal of Artificial Intelligence and Applications.

Hofer, L. (2017). Decision-making algorithms for autonomous robots. Ph.D. thesis, Université de Bordeaux.

J, S. and G, R. (Seipp J and Roger G 2018). Fast downward stone soup 2018. Proc. Fourth International Planning Competition: International Conference on Automated Planning and Scheduling.

Jazdi, N. (2014). Cyber physical systems in the context of industry 4.0. 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, 1–4. doi:10.1109/AQTR.2014.6857843.

Keyder, E. and Geffner, H. (2008). Heuristics for planning with action costs revisited. ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece.

Michniewicz, J. and Reinhart, G. (2016). Cyber-physical-robotics – modelling of modular robot cells for automated planning and execution of assembly tasks. Mechatronics, 34, 170 – 180. System-Integrated Intelligence: New Challenges for Product and Production Engineering.

Nau, D.S., Ghallab, M., and Traverso, P. (2015). Blended planning and acting: Preliminary approach, research challenges. In B. Bonet and S. Koenig (eds.), INPROCEEDINGS of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, 4047–4051. AAAI Press.

Rojko, A. (2017). Industry 4.0 concept: Background and overview. International Journal of Interactive Mobile Technologies.

Russell, S. and Norvig, P. (2010). Artificial Intelligence: A Modern Approach. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Souza, G.A., Brito, H., and Tavares, J. (2019). Automatic column assignment for block world domain with *PNRD/iPNRD*. In 14th Brazilian Symposium on Intelligent Automation. SBA.

Tavares, J.J.P.Z.d.S. and Souza, G.d.A. (2019). Pnrd and ipnrd integration assisting adaptive control in block world domain. Inproceedings of the International Workshop on Petri Nets and Software Engineering 2019, 73–90.

Vaquero, T.S. (2007). Itsimple: Ambiente integrado de modelagem e análise de domínios de planejamento automático. master thesis, polytechnic school of the university of são paulo.

Vaquero, T.S. (2010). Post-Design Analysis for AI PLanning Applications. Ph.D. thesis, Polytechnic School of the University of São Paulo.

Vuksanović, D., Ugarak, J., and Korčok, D. (2016). Industry 4.0: the future concepts and new visions of factory of the future development. International Scientific Conference on ICT and E-Business related research.

Xue, Z., Kasper, A., Zöllner, J.M., and Dillmann, R. (2009). An automatic grasp planning system for service robots. International Conference on Advanced Robotics (ICAR).