RobAuditor — A Methodology for Scalable and Context-Adaptive Task Execution Verification in Safety-Critical Robotic Processes

Franklin Kenghagho K.[1], Jean-Baptiste Weibel[2], Saifeddine Aloui[4], Miguel Prada[3], Michael Neumann[1],Clémence Dubois[4], Nirmal Raveendran[5], Mathieu Grossard[4], Anthony Remazeilles[3], Markus Vincze[2], and Michael Beetz[1]

***Abstract*— Robots autonomously exhibit more and more physical as well as mental capabilities in various sectors of our societies (e.g., healthcare labs, factories, households, shops). However, safety is essential in such human-centered environments and failing to address it only hinders the effective integration of these robots. In response to this observation, robot execution monitoring has been the problem of continuously checking the outcomes of executed actions and reacting to failures. Unfortunately, past works have mostly developed solutions for narrowed contexts, failing therefore to generalize across contexts given the high variability of contexts and the sensitivity of verification procedures to underlying contexts, where a robotic process context essentially comprises the process structure, the available computing resources and the participants to the process. This being said, this paper proposes RobAuditor, a methodology for scalable and context-adaptive task execution verification in safety-critical robotic processes. The methodology is demonstrated in the context of TraceBot, a project that tackles the robotization of sterility testing in medical laboratories.**

I. INTRODUCTION

Imagine human agents (top) and the robot (bottom) performing medical tasks, namely the sterility test of medical products as illustrated in Figure 2. Performing these tasks (e.g, handling and perceiving flexible tubes, tiny needles, transparent container or fluids) are not only computationally challenging (high uncertainty about motion and perception), increasing the chance of execution failure, but are also safety-critical with respect to human life and lost costs. This suggests and as required by the Quality Management System (QMS) standards (FDA QSR 820, ISO 13485), which are foundations of the Safety Management System (SMS) standards, that the performing agents should at least display three capabilities namely **(1)** process verification, **(2)** audit trail generation, **(3)** failure recovery and prevention (CAPA). While process verification ensures the system output's reliability, failure recovery and prevention reduces the damage costs, the generated audit trail systematically keeps track of everything that happened, why, how, when, which resources, for later and further inspections and appropriate reaction for instance. Unfortunately, referred to as execution monitoring in the literature [18, 19] and in this paper as functional

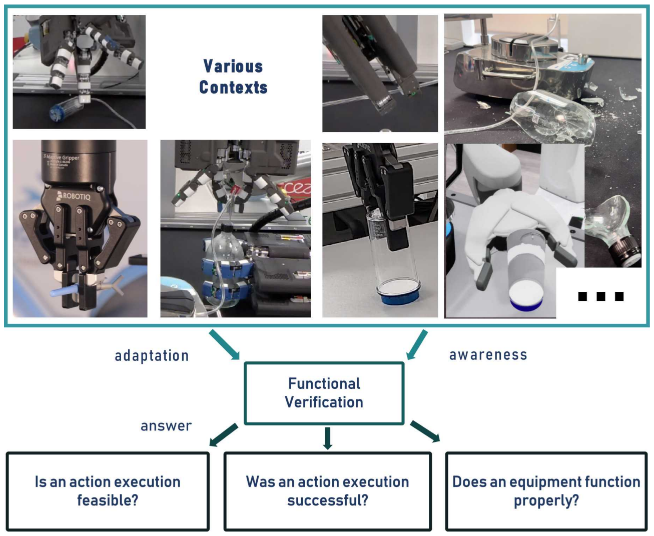

Various Contexts
adaptation
awareness
Functional Verification
answer
Is an action execution feasible?
Was an action execution successful?
Does an equipment function properly?

Fig. 1: Overview of functional verification problem. Even verifying the simple grasping action is not trivial as one might think. Context-awareness and -adaptation are required.

verification (see Figure 1), actual approaches, on the one hand, **(a)** do not model some of these capabilities. There are approaches performing verification without audit trail generation or without failure recovery and failure recovery without failure prevention. On the other hand, past works address them under strong assumptions and in an inflexible manner that make them difficult to cope with the complexity of the real world, such as suggested longtime ago by [8], namely: **(b)** The diversity of task execution structures. Most of these past works assume a static task execution structure though a task may be executed in infinitely many way. The robot can open the bottle before fetching the needle or the other way around. The robot can even proceed redundantly, picking the needle, placing it picking it, ... **(c)** The diversity of task execution contexts. the actual procedure to verify if a tiny needle or a flat sheet has been grasp would likely and drastically differ from the one to verify the grasp of a big bottle. And a particular verification procedure would require some resources such as sensors to be mounted on the robot. **(d)** Availability of computational resources. Verification can last longer than expected hindering the normal flow of the task execution. In this regard, the control program should be given the flexibility to start a verification when it desires,

[1] with Institute for Artificial Intelligence, University of Bremen, Germany fkenghag@uni-bremen.de; [2]Automation and Control Institute, TU Wien, 1040 Vienna, Austria; Tecnalia Research & Innovation, Spain[3]; Commissariat à l'énergie atomique (CEA), France[4]; Astech Projects Limited, United Kingdom[5];

even when the task execution is terminated. However, most recent actual works bake the execution monitor into the plan executive, rigidifying then the latter. In this paper, we introduce a flexible framework RobAuditor (Robot Auditor) that models the capabilities (1-3) while addressing the issues (a-b) in eight majors steps such as illustrated by Figure 2. This being said, the contributions of this paper are threefold namely:

- **Scalable and context-adaptive automated planning and execution of action verification tasks**: scalable to (task structure, context, computational resources), online/offline operational, plugin-like, and human- and machine-centered. We perform both feasibility and success verification where feasibility verification act as preventive measures.
- **Generation of rich human- and machine- understandable audit trail**: for understanding the course of the ongoing or executed process and eventually reacting appropriately (e.g., failure).
- **Proof of Concept of the proposed framework in the context of TraceBot**: a project for automation of medical lab with focus on sterility test

II. Related Work

Though task execution verification is not an old problem in industrial robotics, the literature is significantly sparse as far as autonomonous robotics in uncertain environments is concerned. Moreover, note that we are neither tackling the problem of (formal) verification nor validation of robot programs in this paper as controversial with respect to uncertainty in physical systems[14]. Now let focus on robot task execution monitoring.

Process Verification. As mentioned by [18], past works can be clustered into analytical, data-driven or knowledge-based method. Analytical verification methods rely on mathematical models of the system to produce expectations that in turn will be confronted to observations. Though accurate and straightforward, this approach becomes irrelevant when the system complexity grows and difficult to model mathematically. Data-driven approaches such as [10] also known as model-free operate statistically on observations (sensor data) to detect and classify failure. However, though they might rely on multimodal data, they do not only require huge amount of big data, but both data collection (what is a failure?) and training (high data entropy). become intractable as the semantic complexity of the task grow [1]. Moreover, existing data-driven approaches merely make use of the data online requiring them to be aligned with the control executive and changing the sensor setup on the robot appears to make them useless. Knowledge-based approaches on execution monitoring attempts to capture the high-level semantics of the task domain with symbolic formalism such as formal logic and then make use of low-level sensor data to ground primitive predicates. Though this approach is fundamentally flexible, existing works[6, 7] mostly focus on very simple task (e.g., block worlds) and overrely on situation calculus which is inefficient in realistic scenarios (complexity and coarse expressiveness)[13].

Failure Recovery & Prevention. Given that this step follows verification, only few works tackle it and preferably failure recovery. Another remarks is that the derivation of the recovery plan usually takes place with the same infrastructure used for verification. For instance, data-driven markov models for verification are usually expected to recover as they transit to a new state after failure[10]. This is also the case for knowledge-based approaches. Therefore the same limitations (a-d) apply. However, some works focus on providing natural explanation of failure to users for the sake of trustworthiness [16]. Failure prevention is revealed in the literature to be tackled as feasibility verification and is mostly only addressed by knowledge-based approaches which usually maintain symbolic preconditions of actions. As mentioned earlier on the situation calculus, these preconditions are not only coarse (e.g., grasp does only require object but forces) but become fuzzy and intractable when the task compleity grows [6, 7].

Audit Trail Generation. Though this topic is not new in computer systems, it is hard to find literature on audit trail generation by autonomous systems. Note that the audit trail should be understandable by humans and machines. Therefore, the mere use black-blox systems would be contraproductive. In this regard, work on failure explanation [16] can be viewed as preliminary steps to this goal.

III. RobAuditor: Workflow

Once again, the goal in this paper is to model the safety capabilities (1-3) in a flexible enough manner to address the challenges (b-d). We propose RobAuditor, whose worflow

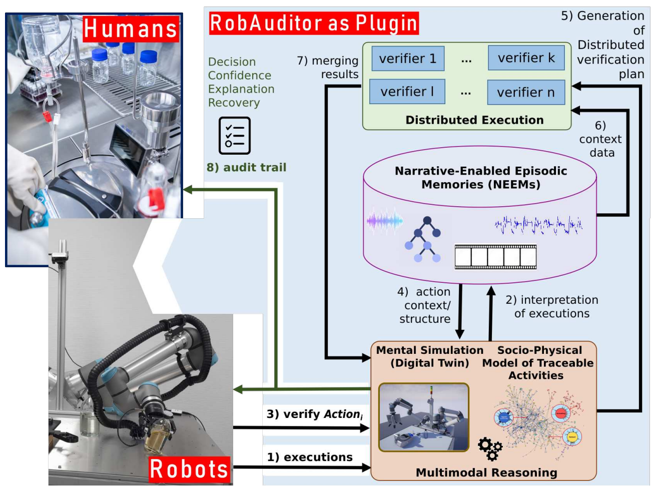


Fig. 2: Plug RobAuditor into robotic systems and easily adapt task execution verification to involved actions' structures, contexts and computational resources (1-8).

can be described in 8 steps such as illustrated by Figure 2. 0) For plugging RobAuditor into the target robot, a formal query-based language is provided for their interactions. (1) As the robot performs, it makes use of the interface to send execution traces to RobAuditor, (2) RobAuditor receives the traces, and augment or refine them through emulation of the actual real execution in a physico-realistic digital twin of the world (DT), which is grounded in a rich ontology of traceable activities (SOMTA). Then, the augmented execution traces are then interpreted into a meaninful story based on SOMTA and persistently stored as narrative-enabled episodic memories of the robot activities (NEEMs), consisting of robot experiences (e.g., sensor& motor data) grounded into activity narratives (i.e., action tree + world states), grounded themselves into SOMTA, where grounding enables understanding. (3) Any time (online/offline), the robot control program or human agent can issue a verification query (sucess/feasibility) for a given action using the interface, (4) RobAuditor's metareasoner will access the context (NEEMs+SOMTA+DT) of the task, (5) then generate a distributed verification pipeline, made up of reasoning units called verifiers, depending on the target task and the context based on an ensemble approach, (6) which will then be executed distributedly on various computers or same. (7) As the pipeline executes, reasoning units can make use of RobAuditor's interface to access the context (e.g., what is the diameter of the canister?). (8) Once, the pipeline has been executed, RobAuditor's metareasoner synthesizes the final verification result from all reasoning units with eventually a recovery plan in case of failure. This result is then cached into NEEMs. Note that each reasoning unit as well as the metareasoner returns for this verification a quadruplet (D_s, C_f, E_d, E_r), denoting respectively a boolean verification decision, a decision confidence, a decision explanation and a recovery plan eventually. Any time(online/offline), the robot or human can request an audit trail of the process through the interface, an RobAuditor will generate the audit trail from the context (NEEMs+ SOMTA + DT). Notice that we do not make assumptions on the task execution structure, context and computational resources, but we rather base on them. Moreover, we decouple RobAuditor from the robot control program and allow the latter to flexibly issue any query any time even offline. In the next sections, we discuss the knowledge representation, reasoning, learning and interface in RobAuditor. Finally, we present the application of RobAuditor in TraceBot.

IV. Knowledge Representation in RobAuditor

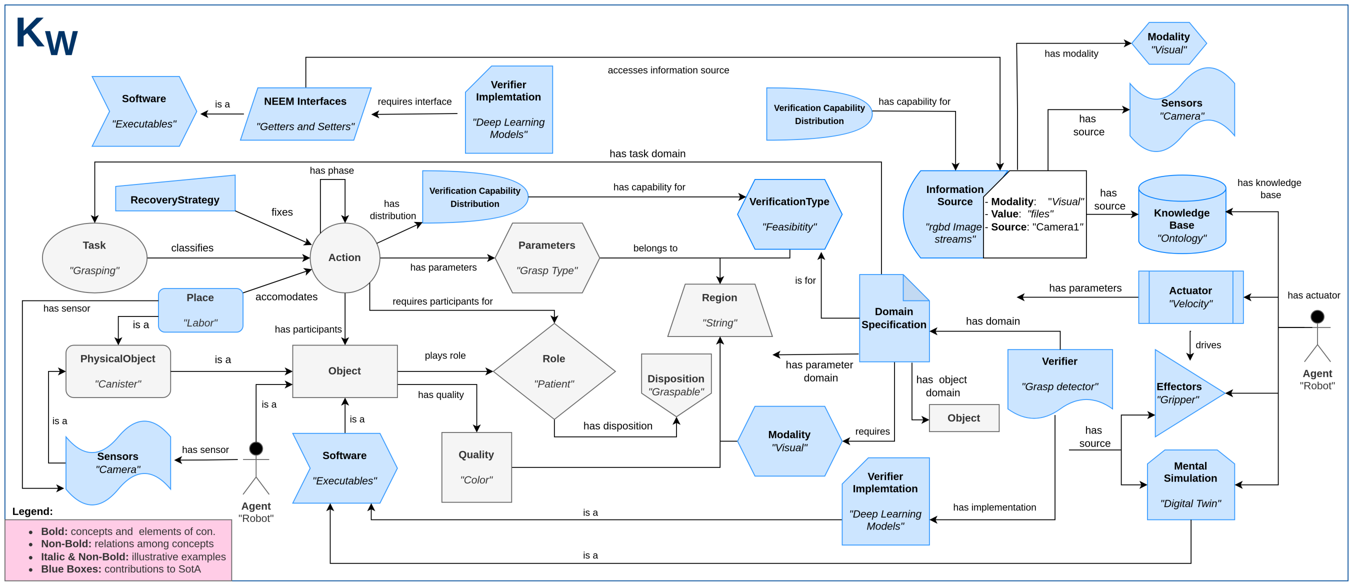


Fig. 3: SOcio-physical Models of Traceable Activities as grammar for generating and interpreting agents' activities (SOMTA). The ontology extends SOMA (gray items) with traceability specificities (blue items).

RobAuditor's knowledge base consists of ontologies (SOMTA), digital twins (DT) and episodic memories (NEEMs).

A. SOMTA: SOcio-physical Models of Traceable Activities

The SOcio-physical Model of Activities (SOMA) [5], implemented in OWL(Ontology Web Language) and SWRL(Semantic Web Rule Language) is an established ontological modeling approach for autonomous robotic agents performing everyday manipulation activities. It tries to catch the social (i.e., meaning of interactions) as well as the physical context (i.e., interactions) of activities. SOMTA extends SOMA with more traceability specificities for the sake of verification, recovery and audit trail generation.

1) Action Specification: In order to avoid imposing a structure on how actions are executed, the ontology only capture fundamental truths about them while avoiding assumptions such as inclusion relations among actions at the T-Box level but rather only at the A-Box level (i.e., NEEMs). Fundamentally, an action is performed to execute a task (e.g., grasping) with some particpants playing specific roles (e.g., agent) and given parameters (e.g., grasp type, velocity). Finally, probabilistic distribution of the relevancy of information sources related to verification of the task is specified (e.g., tactile is more relevant than acoustic data for checking grasp). such a distribution is also given for how the feasibility test of an action influence its success test. Notice that new actions can be added into the ontology over and over.

```
Grasping:              Transporting:     Inserting:
    roles:                 roles:            roles:
        - Patient              - Agent            ...
        - Agent                - Patient
        - Accomodation         - Accomodation
    parameters:                - Source
        - Grasp Type           - Destination
          ...              parameters:
    Verification:              - Transport Velocity
        - Tactile: 0.4           ...
        - Odometry: 0.38
          ...
        - Feasibility: 0.3
        - Success: 0.7
```

2) Information Source Specification: As the robot executes, there exists multiple sources of information with different modalities such as symbolic knowledge bases (e.g., ontologies), sensor data (e.g., visual, acoustic, odometric, tactile) and mental simulations (e.g., DT) which carry evidences of what is going on and can be used to perform verification of the target actions as shown by Figure 3. Given these information modalities, the goal is to specify information sources in the ontology in such a manner that the robot can automatically given the context of a task execution accesses information about the execution (e.g., What was the robot seeing when performing a certain action?). These sources can be tracked by checking if for an executed action, there is a participant which the source is attached to. Note that sources can flexibly be added over and over. Note also that sources can have different physical representation such as owl, excel files, ros topics, databases,... Getters/Setters can continually be pushed into RobAuditor's interface.

```
Tactile-Based: # Tactile-based info source
  topics:
    - source: Robot@Tracebot.RightRobotArm@UR10.
              RightRobotGripper@CEA
      value:
        thumb_finger: /lh/sr_tactile/touch/th
        first_finger: /lh/sr_tactile/touch/ff
        middle_finger: /lh/sr_tactile/touch/mf
        ring_finger: /lh/sr_tactile/touch/rf
        little_finger: /lh/sr_tactile/touch/lf
```

3) Verifier Specification — Reasoning Units: There is no single algorithm that can perform all the verification tasks. By transferring the key philosophy behind UIMA (Unstructured Information Management Architecture) [17], we call verifier an algorithm (e.g., data-driven, analytical or knowledge-based) which is expert at verifying in a specific domain of activity (also multi-domain). That is, for which actions (e.g., grasping), which participating entities (e.g., canister as patient) and information sources (e.g., odometry information) as shown by Schema IV-A.3. Beside its domain, a verifier's implementation must be also provided as a server (e.g., ros action servers, RPC servers, ...) taking as input the identifier of the target action and returning as explained earlier a quadruplet (D_s, C_f, E_d, E_r). Notice that verifiers can be added over and over, executed anywhere, anytime, are decoupled from the other modules (only action id as input).

```
verifier:
  name: KBGrabSuccessCheck
  description: "Verify success of grasping action"
  mode: Success
domains:
  - Grasping:
      participants:
        - Patient:
            - Canister
                - Material: [Glass, Steel, Ceramic]
            - Bottle
                - Shape: Cylindrical
                  ...
        - Actor:
            - Robot@Tracebot.RightRobotArm@UR10.
              RightRobotGripper@Robotiq]
      parameters: []
      modalities: [Odometry-Based, Visual-Based]
inputs:
   action_id: string
outputs:
   decision: bool
   confidence: float64
   decision_explanation: string
   recovery_explanation: string
```

4) Recovery Strategy Specification: Investigation of possible behaviors in case of failure revealed at least the following big categories of failure recovery strategies. (GOB) for Go back to failed action (e.g., reperceive after perception failed), which requires to just to a task again with just with eventually updated parameters and participants. (GBB) for Go back few steps before failed action, applies for instance when the failed action destroys the outputs of previous actions (e.g., reperceive before grasping after failed grasp moved object). (HIC) for Human intervention call (e.g., dead state), is needed when the robot is in a situation it cannot escape by itself. (INO) for Introduction of New Objects (e.g., replace broken bottle). Finally, (INA) for Introduction of New actions (e.g., fetch a new object, move to see). This investigations suggests that recovery strategies cannot be merely reduced to backtracking such as in behavior tree-based approaches or modifications of existing plan, but the synthesis of recovery plans as a generic solution. This being said, a recovery plan is a concatenation of instances of fundamental action model described earlier together with the strategy codes (see Figure 13) which are important information for the metareasoner on how to combine the suggestions from different verifiers, but also for the requesting robot or human agent for understanding the strategy behind the recovery plan. Note also that, a recovery plan can fail, then a recovery plan will be generated for the failure. To avoid infinite loop, a probabilistic sampling of plan is performed when multiple plans are derived. Dead plans such as HIC or INA for stopping the robot can be forced into the sampling pool with low sampling propabilities.

B. Physico-Realistic Digital Twin: Embodying SOMTA

The digital twin extends the symbolic concepts in SOMTA with physical appearances and dynamics so that fine-grained-reasoning about interactions can be achieved. For instance, from the SOMTA, it can might be expressed that the robot inserted the needle into the bottle cap. But when emulating the dynamics of both the robot and the objects, one realizes that the needle is actually broken due to force constraints at the cap surface. We make use of Unreal Engine 4 to model virtual worlds. Note that the DT is not compulsory in RobAuditor but rather an advantage. Moreover, scene models can be added over and over as you can see at Figure 11. Note that though Unreal Engine 4 is used for virtualization

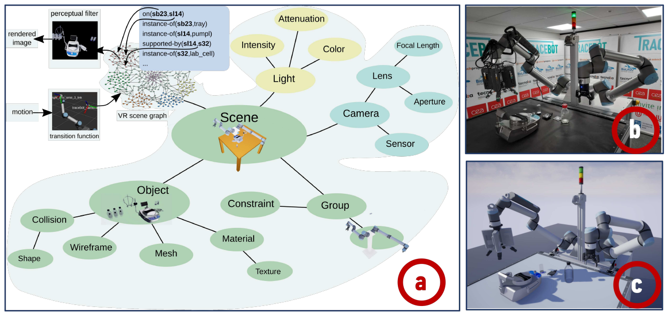
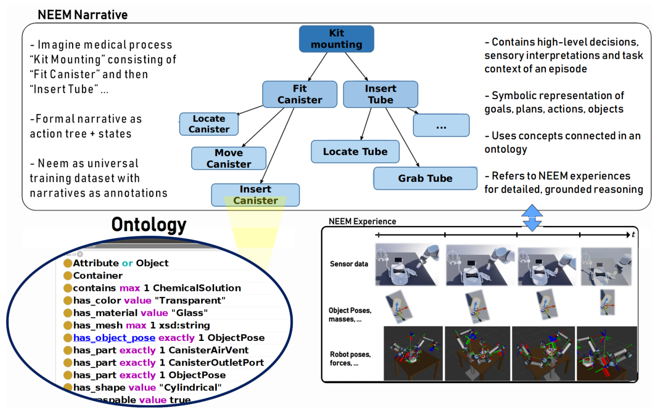


Fig. 4: Digital twin (c), real robot (b), grounding digital twin into SOMTA (a).

engine such as Mujoco or Unity can be used as soon as the DT interface (see Figure 6) are augmented.

C. *Narrative-Enabled Episodic Memories (NEEMs)*

NEEMs stands for Narrative-enabled Episodic Memories. They actually encode, as shown by Figure 5, chronological traces of activities at a symbolic (i.e., activity story) as well as at a sub-symbolic level (i.e., robot experience).


NEEM Narrative
- Imagine medical process "Kit Mounting" consisting of "Fit Canister" and then "Insert Tube"...
-Formal narrative as action tree + states
- Neem as universal training dataset with narratives as annotations
Kit mounting
Fit Canister
Insert Tube
Locate Canister
Move Canister
Insert Canister
Locate Tube
Grab Tube
...
- Contains high-level decisions, sensory interpretations and task context of an episode
- Symbolic representation of goals, plans, actions, objects
- Uses concepts connected in an ontology
- Refers to NEEM experiences for detailed, grounded reasoning
Ontology
Attribute or Object
Container
contains max 1 ChemicalSolution
has_color value "Transparent"
has_material value "Glass"
has_mesh max 1 xsd:string
has_object_pose exactly 1 ObjectPose
has_part exactly 1 CanisterAirVent
has_part exactly 1 CanisterOutletPort
has_part exactly 1 ObjectPose
has_shape value "Cylindrical"
graspable value true
NEEM Experience
Sensor data
Object Poses, masses, ...
Robot poses, forces, ...
Fig. 5: NEEMs as three-step constructions namely experiences grounded into narratives grounded in turn into ontologies.

Given this ability of NEEMs to systematically gather all the traces of an execution and glue them into a coherent story, one can answer powerful reasoning questions such as what is the robot visual experience after it inserted the canister into the tray. Such an image would be a strong evidence that insertion was (un)successful. Technically, NEEMs can be regarded as a sufficient statistics for the robot execution which can be used anytime as it was real robot execution. For this reason, they also constitute the foundation for the audit trail.

V. (META)REASONING IN ROBAUDITOR

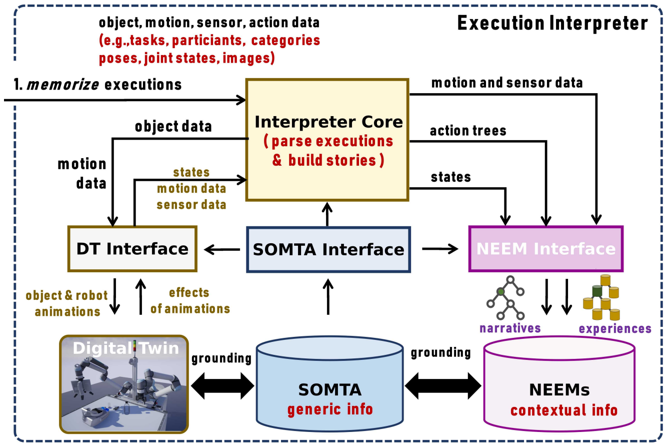


Fig. 6: (a) Augmenting robot execution traces through mental simulation, parsing the traces into networks of categories, and building stories as NEEMs out of the traces through SOMTA as activity grammar.

As far as reasoning is concerned, we extends KnowRob [4], a knowledge processing system designed for robots, with the following reasoning modules. In RobAuditor's reasoning, we distinguished between task execution interpretation and task execution verification.

A. *Task Execution Interpretation*

Task execution interpretation such as described earlier and by Figure 6 takes place either passively, actively or in dual mode. In passive mode, the robot makes use of the provided interface to send information about actions, states, sensor and motion data to RobAuditor. In active mode, RobAuditor only observes the robot execution's motion and sensor data, augment them through the DT and make use of SOMTA as activity grammar,based on contact states/events and the force-dynamics model of Talmy [5], to parse the resulting traces. Such a parser has been successfully proposed by [9]. Actually, RobAuditor only operates in passive and dual mode since the parser for pure active mode has not been integrated yet.

B. *Task Execution Verification*

We describe the planning of task execution verification pipelines as well as the execution of the elaborated plans. Furthermore, we elaborate on how formal recovery suggestions are synthesized from the verification results. Finally, we describe the audit trail generation which is the core output to the target robotic system or human agents.

1) Verification & Recovery.: (1) When a success or feasibility query is issued, (2) the verification executive first retrieves from NEEMs, as illustrated by Figure 7, the target's action tree/structure but (3,4) as well as the context namely the participants and the available information sources of

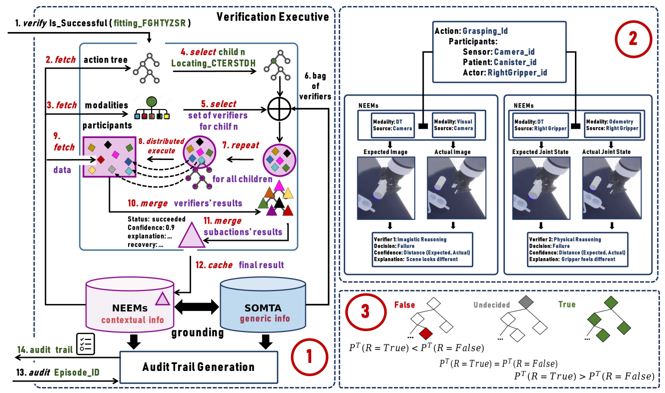
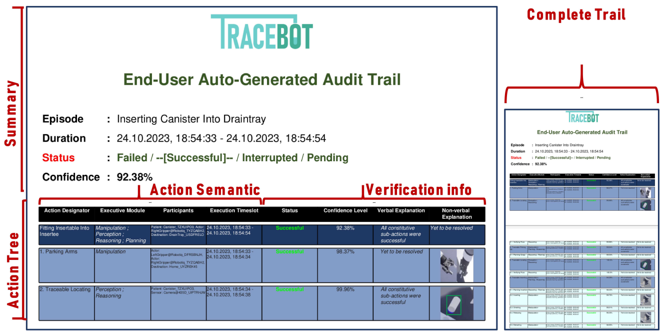

Verification Executive
1. verify Is_Successful (fitting_FGHTYZSR)
2. fetch action tree
4. select child n Locating_CTERSTDH
6. bag of verifiers
3. fetch modalities
participants
5. select set of verifiers for chilf n
9. fetch
data
8. distributed execute
7. repeat
for all children
10. merge verifiers' results
Status: succeeded
Confidence: 0.9
explanation: ...
recovery: ...
11. merge subactions' results
12. cache final result
NEEMs contextual info
SOMTA generic info
grounding
14. audit trail
13. audit Episode_ID
Audit Trail Generation
1
Action: Grasping_Id
Participants:
Sensor: Camera_id
Patient: Canister_id
Actor: RightGripper_id
2
NEEMs
Modality: DT Source: Camera
Modality: Visual Source: Camera
Expected Image
Actual Image
Verifier 1: Imagistic Reasoning
Decision: Failure
Confidence: Distance (Expected, Actual)
Explanation: Scene looks different
NEEMs
Modality: DT Source: Right Gripper
Modality: Odometry Source: Right Gripper
Expected Joint State
Actual Joint State
Verifier 2: Physical Reasoning
Decision: Failure
Confidence: Distance (Expected, Actual)
Explanation: Gripper feels different
3
False
Undecided
True
$P^T(R = True) < P^T(R = False)$
$P^T(R = True) = P^T(R = False)$
$P^T(R = True) > P^T(R = False)$

Fig. 7: (1) planning and execution of task execution verification, (2) distributed execution of verification plans, (3) merging of distributed executions).

information of each action node. (6) Then, given the domain specifications of available verifiers in SOMTA, (5) a set of domain-satisfied verifiers are loaded for each action node in the tree, (8) which are then executed distributedly where (9) verifiers can access (SOMTA + NEEMs + DT) during execution. After the execution, (10) each verifier return a quadruplet (D_s, C_f, E_d, E_r) (11) which is then merged by the metareasoner and (11) cached into NEEMs. Algorithms 1 and 2 give details information on how these steps operate. Notice that for the success verification, the feasibility verification is automatically performed. However, a feasibility verification would not trigger a success verification. Other algorithms are not provided however can be accessed directly from the code or intuitively understood from above algorithms. Notice that feasibility verification is an implementation of the prevention capability.

2) Audit Trail Generation: As you mentioned earlier, NEEMs contain so much information that it will not be trivial for a human or robot investigator to get at first glance an idea of what happens during the robot performance.



End-User Auto-Generated Audit Trail

Episode	:	Inserting Canister Into Draintray
Duration	:	24.10.2023, 18:54:33 - 24.10.2023, 18:54:54
Status	:	Failed / --[Successful]-- / Interrupted / Pending
Confidence	:	92.38%

Action Designator	Executive Module	Participants	Execution Timeslot	Status	Confidence Level	Verbal Explanation	Non-verbal Explanation
Fitting Insertable Into Insertee	*Manipulation ; Perception ; Reasoning ; Planning*	Patient: Canister_TZXUIPO3, Actor: RightGripper@Robotiq_TYZQABH2, Destination: DrainTray_USDFRELO	24.10.2023, 18:54:33 - 24.10.2023, 18:54:54	Successful	92.38%	*All constitutive sub-actions were successful*	*Yet to be resolved*
1. Parking Arms	*Manipulation*	Actor: LeftGripper@Robotiq_DFRSBNJH, Actor: RightGripper@Robotiq_TYZQABH2, Destination: Home_UYZREK45	24.10.2023, 18:54:33 - 24.10.2023, 18:54:34	Successful	98.37%	*Yet to be resolved*	
2. Traceable Locating	*Perception ; Reasoning*	Patient: Canister_TZXUIPO3, Sensor: Camera@435D_UIPTRHJW	24.10.2023, 18:54:34 - 24.10.2023, 18:54:38	Successful	99.96%	*All constitutive sub-actions were successful*	

Fig. 8: Overview of a pdf-rendered audit trail.

The idea of the audit trail is to summarize in a human- and machine-understandable manner what the robot has done, when, who participated, how it went and why it went so. This interface generates the audit trail from a given NEEM and saves it as pdf document (see Figure 8) to a specific given location, but also provides a machine-accessible json

Algorithm 1 *SuccessVerification*(A_{id}, I_c, K_w)

```
Require: K_w, knowledge base, i.e. ontologies and neems
         A_id, id of target action instance
         I_c, whether cached results should be ignored
Ensure: (D_s, C_f, E_d, E_r), respectively the decision, confidence, decision explanation
         and eventually recovery explanation
 1: $T_v = $"Success";  // set type of verification
 2: if $I_c = false$ then
 3:     if $IsAlreadyVerified(A_{id}, T_v, K_w)$ then
 4:         $(D_s, C_f, E_d, E_r) \leftarrow GetVerificationResults(A_{id}, T_v, K_w)$;  // cached results
            Exit(); //exit the program
 5:     end if
 6: end if
 7: $(Ds_0, Cf_0, Ed_0, Er_0) \leftarrow FeasibilityVerification(A_{id}, I_c, K_w)$; // feasibility test
 8: $R \leftarrow \{\}$; // init list of results
 9: $V \leftarrow \{\}$; // init list of verifiers
10: for $Verifier$ in $ListVerifiers(K_w)$ do
11:     if $Verifier.mode = $"Success" and $Verifier.domains \subseteq GetEpisode(A_{id}, K_w)$
        then
12:         $V \leftarrow V \cup \{Verifier\}$; // init list of verifiers
13:         $(Ds_1, Cf_1, Ed_1, Er_1) \leftarrow Run(Verifier, A_{id}, K_w)$; // call verifier
14:         $R \leftarrow R \cup \{(Ds_1, Cf_1, Ed_1, Er_1)\}$; // add result into list of individual results
15:     end if
16: end for
17: $(Ds_2, Cf_2, Ed_2, Er_2) \leftarrow MergeVerifierResults(R, A_{id}, K_w, V)$; // merging results
18: $R \leftarrow \{(Ds_2, Cf_2, Ed_2, Er_2)\}$; // init list of results again for direct subactions
19: for $A_s$ in $GetSubActions(A_{id}, K_w)$ do
20:     $(Ds_3, Cf_3, Ed_3, Er_3) \leftarrow SuccessVerification(A_s, I_c, K_w)$; // test of subactions
21:     $R \leftarrow R \cup \{(Ds_3, Cf_3, Ed_3, Er_3)\}$; // add result into list of individual results
22: end for
23: $(Ds_4, Cf_4, Ed_4, Er_4) \leftarrow MergeSubActionResults(R, A_{id}, K_w, T_v)$; //merging results
24: $R \leftarrow \{(Ds_4, Cf_4, Ed_4, Er_4), (Ds_0, Cf_0, Ed_0, Er_0)\}$; // feasibility and success
25: $(D_s, C_f, E_d, E_r) \leftarrow MergeFeasibilitySuccess(R, A_{id}, K_w, T_v)$; // merging results
26: $SetVerificationResults(A_{id}, T_v, K_w, (D_s, C_f, E_d, E_r))$; // caching results
```

Algorithm 2 *MergeSubActionResults*(R, A_{id}, K_w, T_v)

```
Require: $R = \{(Ds_0, Cf_0, Ed_0, Er_0), \ldots, (Ds_n, Cf_n, Ed_n, Er_n)\}$, list of individual results
         $K_w$, knowledge base, i.e. ontologies and neems
         $A_{id}$, id of target action instance
         $T_v$, type of verification
Ensure: $(D_s, C_f, E_d, E_r)$, respectively the decision, confidence, decision explanation
         and eventually recovery explanation
 1: $prob_true \leftarrow 1.0$;  // for joint true decision
 2: for $v \in R$ do
 3:     if $v[$"$D_s$"$] = true$ then
 4:         $prob_{true} = prob_{true} \times v[$"$C_f$"$]$;  // joint true decision
 5:     else
 6:         $prob_{true} = prob_{true} \times (1 - v[$"$C_f$"$])$;  // joint true decision
 7:     end if
 8: end for
 9: if $prob_{true} > 0.5$ then
10:     $D_s \leftarrow true$;  // final decision
11:     $C_f \leftarrow prob_{true}$;  // final confidence
12: else
13:     $D_s \leftarrow false$;  // final decision
14:     $C_f \leftarrow 1 - prob_{true}$;  // final confidence
15: end if
16: $E_d \leftarrow \bigoplus_{i,(D_s = R[i]["D_s"])} R[i]["E_d"]$;  //concatening for final decision explanation
17: $E_r \leftarrow \bigoplus_{i,(D_s = R[i]["D_s"])} R[i]["E_r"]$;  // concatening for final recovery explanation
```

version of the trail to the robot for eventual self-recovery.

VI. Learning verification expectations from NEEMs

Notice so far that the confidence from verification originates on the one hand from common sense at the ontology level and from verifiers. This can pose ethical issues as this reliability measure is only founded in the subjectivity of the ontology designer and the verifier designer. In order to address this problem, a probabilistic expectation of the success and feasibility of actions is estimated based on past observations. This expectation constitutes then a prior for future verification tasks. In this regard, the expressions below highlight the different distributions one could learn from such past data.

$P(S_t, S_{t-1}|A_t, A_{t-1})$, influence of neighbohring actions
$P(S_t|A_t, P_t)$, influence of participants
$P(S_{0:t}|A_{0:t}, P_{0:t})$, the whole episode

The first point highlights the distributions of success in a sequence of two actions. This is important to learn how temporally adjacent actions can influence each other with respect to success (e.g., manipulating fluids before grasping increases chances of failure because gripper wet and slippery). The second point indicates the distributions of success over actions conditioned on participants (e.g., higher chance of grasping canister than needle) and finally the third point showing how the general case of learning the success over an entire episode (e.g., canister insertion, tube insertion, bottle insertion, sterility testing, etc). In this project, we limited ourselves to the second case namely $P(S_t|A_t, P_t)$. Figure 9 illustrates the failure that can be encountered.

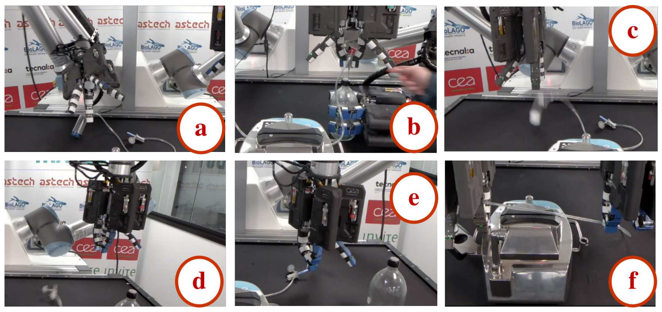


Fig. 9: Failure to (a) grasp canister, (b) insert needle, (c) verify grasping action, (d) stretch tube, (e) grasp needle cap, (f) insert tube.

This being done, it is known that with sufficiently large amount of data, one can learn any distribution over the data. However, the data collected in such tracebot processes and in the development phase are very sparse. For this reason, a parametric probabilistic model of success over actions conditioned on participants is carefully designed and the few model parameters are learned from the sparse data. In this specific case, the Poisson distribution is chosen to model the probability of occurrence of failure in actions given participants. This distribution is well known to model the probability of an event occurred in a fixed interval of time independently of other time intervals. Our time interval here can be an episode or multiple consecutive episodes. The probability function of a Poisson distribution is defined by:

$$P(x) = \frac{\lambda^x}{x!} e^{-\lambda}$$

In the probability function of a Poisson distribution, lambda is the mean and x is the variable. Using this, we compute the expectation of succeeding with a core tracebot actions (inserting, grasping, perceiving, simulating, verifying) on core tracebot objects (canister, needle, cap, tube, bottle) as shown by Figure 10.

This result aligns with observations as it shows on the

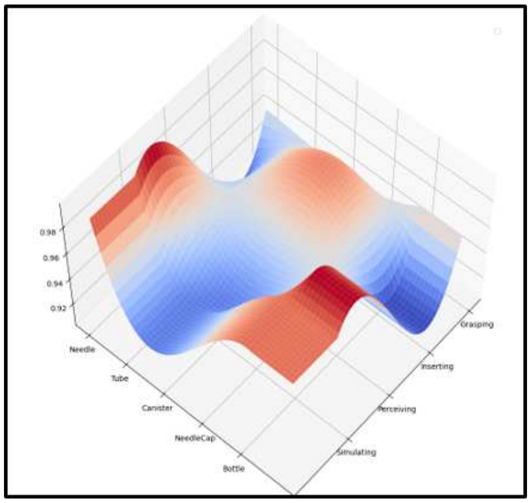


Fig. 10: Probability of succeeding actions in specific contexts (participants)

one hand that the simulation and perception of tube are hard whereas the grasping is stable. Moreover, it shows that the insertion of bottle is hard (due to lack of experiments and by default infinitely hard). But inserting the needle into the bottle is hard as well. Finally, grasping the needle cap and the needle is not trivial. On the other hand, it shows that the execution of all these actions is likely to succeed as the probability is above 0.90.

VII. Interface: Formal Query Language

We model the interface as formal and complete formal query language, in the sense that it can also to retrieve any fact from the knowledge base. Queries are very flexible as they can be used as three almost any informatio medium. We also extends KnowRob in terms of queries The language supports three type of sentences namely kb_call(P), kb_project(P) and kb_unproject(P) respectively to retrieve or construct facts P from, assert facts P in and retract facts P from knowledge base. P is a list of predicates in prefix form or triples in infix form from the ontology. To modify a fact, kb_unproject ∘ kb_project is applied. We distinguish:

A. Retrieval Queries.

A query for accessing the diameter of the canister will then be:

```
kb_call([subclass_of(Canister,A),
        has_description(A,exactly(has_size,1,B)),
        subclass_of(B,C),
        has_description(C,value(has_diameter,D))  ]).
```

The value of the canister's diameter will be stored in the variable D. The query states that A is a superclass of Canister and has exactly one size of type B. And C is a superclass of B and has a diameter of value D.

B. Memorization Queries

An example of query for saving information about executed action and even for recording NEEMs, which are foundations for audit trails, will then be:

```
kb_project([
newIri(Episode,soma:'Episode'),
newIri(Action,tracebot:'Grasping'),
newIri(Object,tracebot:'Canister'),
newIri(Agent,tracebot:'LeftUR10Arm'),
newIri(TimeInterval,dul:'TimeInterval'),
holds(Action,dul:'hasTimeInterval',TimeInterval),
holds(TimeInterval,soma:'hasIntervalBegin',
      StartTime),
isSettingFor(Episode,Action),
isPerformedBy(Action,Agent),
hasParameter(Action,Object)
newIri(Role,soma:'AgentRole'),
hasType(Role,soma:'AgentRole'),hasRole(Agent,Role)
]).
```

C. *Verification Queries*

These queries are highly encapsulated and basically consists of a predicate *Verify* taking as parameters the inputs and outputs of the verifiers presented in IV-A.3, but also the type of verification since it the verifier specification it is located in the domain specification.

```
kb_call([ ExecuteVerification(Tv,
          soma#Grasping_XZT49,D_s,C_f,E_d,E_r)]).
```

D. *Audit Queries*

Like the previous one, these queries are highly encapsulated for the same purpose.

```
kb_call([GenerateAudit(tracebot#Episode_ZUL13,A)]).
```

This means that as soon as an episode has been created in the episodic memory, an audit trail generation can be triggered.

VIII. On the High Flexibility of RobAuditor

We elaborate on the following core properties of RobAuditor:

Scalability. Again RobAuditor scales well with respect to computational resources (distributed computing and delayed verification), to action's structure and context: reasoning does not assume a structure, length, context of executed action but rather infers based on..

Offline/Online Operability. These modes are just consequences of anytime verification querying and the latter is importantly possible because NEEMs are rich enough to be viewed as real robot executions)

Plugin-like. RobAuditor acts a plugin to the target robot system. A formal query-based interface is provided to the robot to send execution traces to RobAuditor with levels of flexibility. These traces will then be mapped by RobAuditor through SOMTA and DT into NEEMs. SOMTA, NEEMs and DT are enough to operate.

Human- and machine-centered. RobAuditor has been described to interact with robots but also with human agents. The latter should only make use of RobAuditor's interface.

IX. Application: TraceBot

Use Case Description. The use case we selected, namely the sterility test of products through the membrane filtration, comes from the pharmaceutical domain, application field in which the traceability and the verification of the good execution of any process is of major importance [15].

System Description. Our complete implementation is

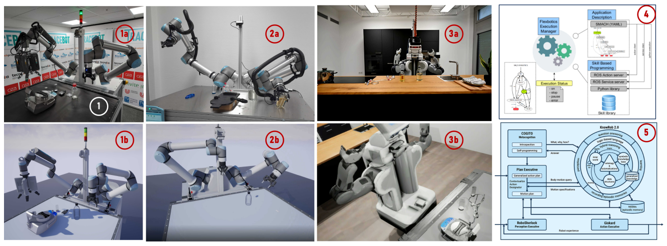
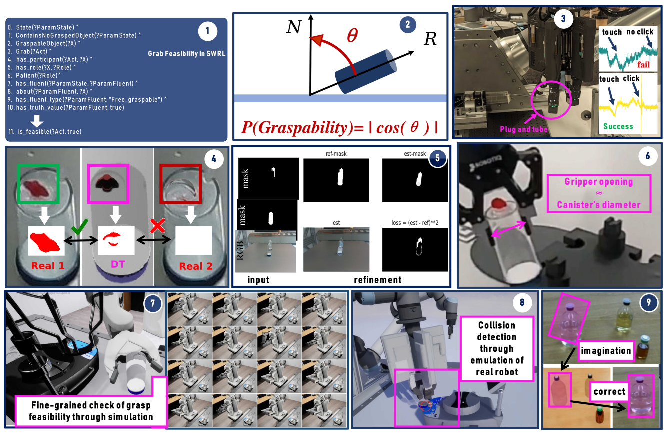


Fig. 11: Proof of concept through three real robot platforms (1a, 2b, 3c) and respective digital twins (1b, 2b, 3b) with two different control architectures (4,5).

available in these GitLab's repositories[1], [2] in ROS NOETIC. As illustrated by Figure 11, we provide a proof of concept for RobAuditor on three different robot platforms namely (2) dual UR10-arm + robotiq gripper, (1) dual UR10-arm + CEA gripper and (3) PR2. We also targeted two different control program architecture namely CRAM and a skill framework with SMACH[11].

Verifiers. We illustrate the logic behind some verification

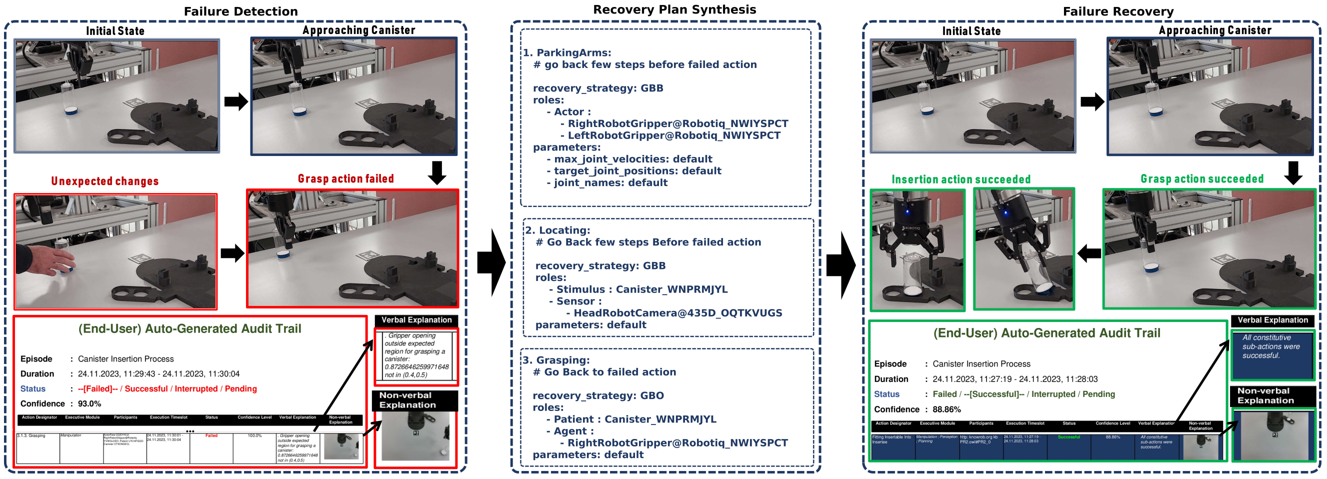


```
0.  State(?ParamState) ^
1.  ContainsNoGraspedObject(?ParamState) ^
2.  GraspableObject(?X) ^
3.  Grab(?Act) ^
4.  has_participant(?Act, ?X) ^
5.  has_role(?X, ?Role) ^
6.  Patient(?Role)^
7.  has_fluent(?ParamState, ?ParamFluent) ^
8.  about(?ParamFluent, ?X) ^
9.  has_fluent_type(?ParamFluent, "Free_graspable") ^
10. has_truth_value(?ParamFluent, true)
        ⇓
11. is_feasible(?Act, true)
```

Fig. 12: Demonstration of verifier variability in various contexts.

modalities. **tactile-based and acoustic-based verification** can verify a task execution based on contact events and force-dynamics (Figure 12.3). The **knowledge-based verification** makes use of SWRL rules to formulate coarse precondition and post-conditions of primitive actions(Figure 12.1) but also combined with vision (e.g., lying bottle cannot be grasped as in Figure 12.2). **Simulation-based verification** can easilly detect collision (Figure 12.8) or refine the pose of objects based on physics (Figure 12.9). **Visual-based verification** [3] employs inverse rendering to detect and eliminate deviations in the pose of transparent objects for which rgbd camera does not work (Figure 12.5). **Imagistic reasoning-based verification**[2] renders and expected state through the DT/simulation, then compares the rendering with real images (Figure 12.4) to detect potential failures or to estimate the state[12].

[1]See `https://tracebot.gitlab.io/tracebot_showcase`
[2]See `https://gitlab.com/tracebot`



(End-User) Auto-Generated Audit Trail

Episode	:	Canister Insertion Process
Duration	:	24.11.2023, 11:29:43 - 24.11.2023, 11:30:04
Status	:	--[Failed]-- / Successful / Interrupted / Pending
Confidence	:	93.0%

Action Designator	Executive Module	Participants	Execution Timeslot	Status	Confidence Level	Verbal Explanation	Non-verbal Explanation
...							
3.1.3. Grasping	Manipulation	ActorRole OGSYHVJI; RightRobotGripper@Robotiq YVWGLXED; Patient UYLNFGQO; Canister OTKDNGEQ;	24.11.2023, 11:30:01 - 24.11.2023, 11:30:04	Failed	100.0%	. Gripper opening outside expected region for grasping a canister: 0.8726646259971648 not in (0.4,0.5)	

Verbal Explanation

. Gripper opening outside expected region for grasping a canister: 0.8726646259971648 not in (0.4,0.5)

Non-verbal Explanation

```
1. ParkingArms:
   # go back few steps before failed action

   recovery_strategy: GBB
   roles:
      - Actor :
         - RightRobotGripper@Robotiq_NWIYSPCT
         - LeftRobotGripper@Robotiq_NWIYSPCT
   parameters:
      - max_joint_velocities: default
      - target_joint_positions: default
      - joint_names: default
```

```
2. Locating:
   # Go Back few steps Before failed action

   recovery_strategy: GBB
   roles:
      - Stimulus : Canister_WNPRMJYL
      - Sensor :
          - HeadRobotCamera@435D_OQTKVUGS
   parameters: default
```

```
3. Grasping:
   # Go Back to failed action

   recovery_strategy: GBO
   roles:
      - Patient : Canister_WNPRMJYL
      - Agent :
         - RightRobotGripper@Robotiq_NWIYSPCT
   parameters: default
```

(End-User) Auto-Generated Audit Trail

Episode	:	Canister Insertion Process
Duration	:	24.11.2023, 11:27:19 - 24.11.2023, 11:28:03
Status	:	Failed / --[Successful]-- / Interrupted / Pending
Confidence	:	88.86%

Action Designator	Executive Module	Participants	Execution Timeslot	Status	Confidence Level	Verbal Explanation	Non-verbal Explanation
Fitting Insertable Into Insertee	Manipulation ; Perception ; Planning	http: knowrob.org kb PR2.owl#PR2_0	24.11.2023, 11:27:19 - 24.11.2023, 11:28:03	Successful	88.86%	All constitutive sub-actions were successful.	

Verbal Explanation

All constitutive sub-actions were successful.

Non-verbal Explanation

Fig. 13: Demonstration of RobAuditor in the TraceBot sub-use case ”canister insertion into the drain tray”

Physical reasoning-based verification [12] relies on generative model of physics to anticipate the effects of causes and causes of effects such as the grasp parameters (Figure 12.7). **Odometry-based verification** leverages the robot joint states to verify (Figure 12.6). Multi-modal verifier can be developed such as (Figure 12.2). **Integrated Demonstration.** We demonstrate RobAuditor in one of the most completed sub-use case in the project, namely the insertion of the transparent canister into the drain train (tight hole). On the left of Figure 13, the grasp failed due to user unexpected move of the canister The use case is a bottleneck in the process . The failure is reported in the audit trail and a recovery plan is generated. On the right, the recovery plan is executed and the former plan is completed for insertion Note the incorporation of non verbal explanation in the audit trail such as explained in the NEEMs section IV-C (**See HD-Video attached to this paper for more details**).

X. CONCLUSION

In this paper, we exposed some standard requirements to meet when designing autonomous robots for safety-critical processes namely the process verification, audit trail generation and failure recovery and prevention. Then, we showed that actual works do not model some of these capabilities or do it but under strong assumptions and in an inflexible manner that cannot cope with real world processes namely the diversity of task execution structures, task execution contexts and the availability of computational resources. In response to that we design a flexible solution RobAuditor, whose proof of concept has been provided in the context of TraceBot, a project tackling the automation of sterility test in medical labs.prevention especially for addressing safety concerns hindering the successful integration of robots in our societies. As future work, we target the establishment of the framework as ready to use with more integrated tests. Another major point of concern is the regulation of verifier implementation (e.g., force them to observe some policies) since they can be implemented by any lambda.

ACKNOWLEDGMENT

This scientific work is partially funded by the projects DFG EASE CRC 1320 and EU TraceBot (grant agreement No 101017089).

REFERENCES

[1] Pieter Adriaans. “Learning as Data Compression”. In: 2007.
[2] Mania et al. “An Open and Flexible Robot Perception Framework for Mobile Manipulation Tasks”. In: 2024.
[3] Bauer et al. “VeREFINE: Integrating object pose verification with physics-guided iterative refinement”. In: (2020).
[4] Beetz et al. “Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In: ICRA 2018.
[5] Beßler et al. “Foundations of the Socio-Physical Model of Activities (SOMA) for Autonomous Robotic Agents1”. In: 2021.
[6] Bouguerra et al. “Active execution monitoring using planning and semantic knowledge”. In: 2007.
[7] Coruhlu et al. “Explainable Robotic Plan Execution Monitoring Under Partial Observability”. In: (2022).
[8] Doyle et al. “Generating Perception Requests and Expectations to Verify the Execution of Plans.” In: 1986.
[9] Haidu et al. “Automated acquisition of structured, semantic models of manipulation activities from human VR demonstration”. In: ICRA 2021.
[10] Hegemann et al. “Learning Symbolic Failure Detection for Grasping and Mobile Manipulation Tasks”. In: 2022.
[11] Herrero et al. “Skill based robot programming: Assembly, vision and Workspace Monitoring skill interaction”. In: (2017).
[12] Kenghagho et al. “NaivPhys4RP - Towards Human-like Robot Perception “Physical Reasoning based on Embodied Probabilistic Simulation””. In: 2022.

[13] Lin et al. “Chapter 16 Situation Calculus”. In: *Handbook of Knowledge Representation*. 2008.

[14] Luckcuck et al. “Formal Specification and Verification of Autonomous Robotic Systems”. In: (2018).

[15] Remazeilles et al. “Robotizing the Sterility Testing Process: Scientific Challenges for Bringing Agile Robots into the Laboratory”. In: 2023.

[16] Thielstrom et al. “Generating Explanations of Action Failures in a Cognitive Robotic Architecture”. In: 2020.

[17] Verspoor et al. “Unstructured Information Management Architecture (UIMA)”. In: 2013.

[18] “Execution monitoring in robotics: A survey”. In: (2005).

[19] Christian Fritz. “Execution Monitoring – A Survey”. 2005.