Multi-Agent Heterogeneous Digital Twin Framework with Dynamic Responsibility Allocation for Complex Task Simulation

Adrian Simon Bauer
German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC)
Weßling, Germany
adrian.bauer@dlr.de

Anne Köpken
German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC)
Weßling, Germany
anne.koepken@dlr.de

Daniel Leidner
German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC)
Weßling, Germany
daniel.leidner@dlr.de

ABSTRACT

To become helpful assistants in our daily lives, robots must be able to understand the effects of their actions on their environment. A modern approach to this is the use of a physics simulation, where often very general simulation engines are utilized. As a result, specific modeling features, such as multi-contact simulation or fluid dynamics, may not be well represented. To improve the representativeness of simulations, we propose a framework for combining estimations of multiple heterogeneous simulations into a single one. The framework couples multiple simulations and reorganizes them based on semantically annotated action sequence information. While each object in the scene is always covered by a simulation, this simulation responsibility can be reassigned on-line. In this paper, we introduce the concept of the framework, describe the architecture, and demonstrate two example implementations. Eventually, we demonstrate how the framework can be used to simulate action executions on the humanoid robot Rollin' Justin with the goal to extract the semantic state and how this information is used to assess whether an action sequence is executed successful or not.

CCS CONCEPTS

• **Computing methodologies** → *Distributed simulation*; **Multi-scale systems**; Real-time simulation; *Agent / discrete models*; *Cognitive robotics*; **Spatial and physical reasoning**; Reasoning about belief and knowledge.

KEYWORDS

Robotics; Simulation; Knowledge Representation; Digital Twin; Environment Models

ACM Reference Format:
Adrian Simon Bauer, Anne Köpken, and Daniel Leidner. 2022. Multi-Agent Heterogeneous Digital Twin Framework with Dynamic Responsibility Allocation for Complex Task Simulation. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

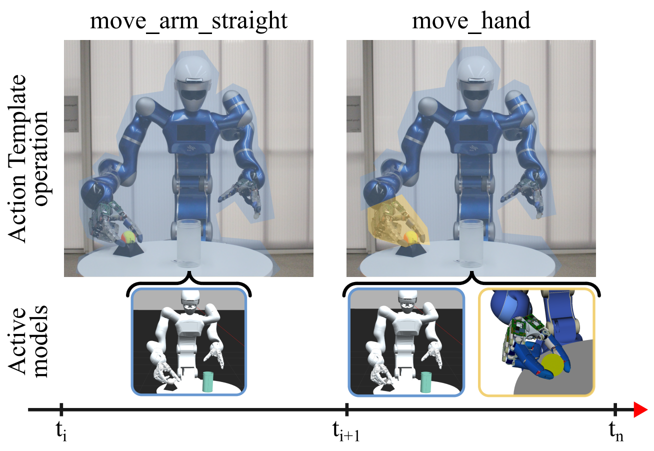


Figure 1: Example of a reconfiguration of a model ensemble due to a trigger. First all objects are simulated in the main physics simulation. Due to new action sequence information at t_{i+1} the in-hand localization takes over responsibility for the object next to the hand of the robot. Now both simulations share responsibility for the scene.

1 INTRODUCTION

It is understood that robots affect the environment they reside in. A robot that does not affect its surrounding cannot be of much practical use. Furthermore, the environment also affects the robot because its current state determines possible actions the robot can take. Thus, it is important that robots are able to determine the current state of their surrounding and understand how their actions change that state. Only this way they are able to create plans that let them achieve their goals.

While it is, on the one hand, important to understand how actions can change the state of the world in a *predictive* manner, it is, certainly not of lesser importance to have a deep understanding of the current world state in an *interpretative* manner. For planning it is important to know effects of actions to create a plan, but if the initial state was assumed wrongly, most plans will end up not being executable. Thus, our goal is to have a reliable model of the robot and its environment. Through the broad availability of simulation frameworks like gazebo [11], simulations are widely used to create such models, acting as digital twins of the robot [8].

However, simulations are usually limited in their representativeness since they are tuned for specific scenarios. Rigid-body simulations, for example, are not suited to simulate soft body contacts and often simulations have hard times dealing with multi-contact scenarios such as grasping. Therefore, previous work on inferring environmental states based on action execution data [1] had to confine to simple interactions due to the limitations of simulation.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Especially when grasps with multi-finger hands occur in simulations, researchers often have to take shortcuts by binding the grasped object rigidly to the robot manipulator in the simulation. Furthermore, simulations usually come with significant computational costs. Designers that represent the environment of a robot in a simulation are, thus, faced with a trade-off between high accuracy of their simulation and runtime.

A solution to the mentioned problems lies in using specialized simulations that are tuned to simulate the task at hand. The drawback of that solution is that specialized solutions usually do not integrate well into an existing digital twin of the system. Most simulations do not provide a simple coupling-mechanism with other simulations, thus, connecting specialized simulations with the digital twin is not straightforward. Therefore, we propose a framework for a Multi-Agent Heterogeneous Digital Twin simulation that combines multiple simulations of the robot and the environment into one digital twin. It allows to use an ensemble of specialized models for specific tasks and dynamically rearranges the mapping between objects and the models. Thus, it allows to transfer the responsibility for simulating parts of the robot or the environment based on external triggers. We'll refer to this ability as *dynamic responsibility allocation*.

The ensemble does not exclusively consist of simulations but may also involve simpler models like observers that only involve certain aspects of the environment. This draws inspiration from work showing that humans understand and predict their environment by creating models [10] and simulations [7]. The framework allows to connect different models together in order to model the agent and the environment on different levels of detail and with different foci.

The scenario shown in fig. 1 serves as a running example for our framework in this paper. It consists of the robot Rollin' Justin [3] being tasked to pick up a tennis ball. Standard physics simulations are not able to simulate the grasping of the ball with a four fingered hand out of the box, but tend to "explode" due to the high number of contacts. This can be circumvented but requires careful tuning of the simulation which comes at the cost of reduced accuracy of the simulation in other parts of the environment. In our case we have two models: one that is specialized at simulating grasping of objects and a general simulation. A trigger, that detects beginning grasps, triggers a rearrangement of the ensemble as the hand moves and the grasping of the ball is simulated in the specialized grasping model. The results of the specialized model are fed back into the general simulation and the task can be simulated accurately without impairing the accuracy of the general simulation.

The contributions of this paper are (i) a concept for a heterogeneous digital twin simulation, (ii) a prototypical implementation of the concept, and (iii) tests on the humanoid robot Rollin' Justin.

The rest of the paper is structured as follows: First, we introduce related research activities in section 2. Then, we describe the heterogeneous multi-agent digital twin framework in section 3. Next, we demonstrate how the framework can be used to interpret action effects based on action execution data from the robot in section 4, followed by a discussion in section 5. Finally, we conclude the paper in section 6.

2 RELATED WORK

This chapter links our work to related work in the field. We begin by describing related work that focuses on how simulations are used in decision making processes. Then we give a short overview of the commonly used simulations in robotics and finally present existing co-simulation approaches and multi-physics simulations.

2.1 The Role of Simulations and Models for Decision Making

While this work focuses on the application of simulations and models in robotics, it draws inspiration from psychological research on *mental models*. It is believed that humans create mental models as internal representations of their perceived reality and use them for decision making and to filter new information [10]. It has also been shown that these mental models are used in conjunction with mental simulations to interpret perceived scenes. In [7] participants were asked to perform tasks like predicting where the blocks of a tower would land if the tower tumbled, deciding whether moving dots behaved according to physical or social rules, or simply watching physical events. The authors of that study claim to have identified a brain region that is selectively engaged in these physic-related tasks, which they refer to as a "'physics engine' in the brain" [7].

Inspired by that, researchers investigated how to make use of simulations as internal models in robotics. One application they found, was to use simulations as a means to represent the belief of the robot about its environment which could be used to infer the results of actions in the environment when perception was not applicable [1]. Another approach used simulations to distill causal relations between actions observed from humans in a virtual reality environment [24].

Apart from using simulations to interpret effects and causal links between actions, they are also widely used to predict action outcomes. Any motion planner that checks for collisions using a model can be seen as a static "simulation" that evaluates whether a collision occurs based on a given configuration of the robot and the environment. An example for predicting high level action outcomes can be found in [12] where the authors use a simulation to test different parameterizations of an action regarding the desired effect. Another use case is described in [2] where the authors use multiple simulations to probabilistically predict the outcome of an action in conjunction with a learning-based algorithm.

2.2 Simulations in Robotics

Simulations are widely used in robotics not only because they can be used to generate mental models, but because they can be used to test algorithms before bringing them to hardware, because they offer cheap alternatives to real robots, and because they allow to generate huge amounts of training data fast and cheaply. There is a large amount of physics simulation engines and simulation environments specifically for robotics, for an overview see [4].

A drawback of the existing simulations and simulation frameworks is that they each only use a single simulation engine. A comparison of different simulation engines for robotics concluded with the insight that "each engine performs best on the type of system it was designed for" [6]. This reminds of the infamous "no free lunch theorem" for optimization [25] which states that no single optimization algorithm outperforms all other algorithms on an average over all types of optimization problems.

2.3 Co-Simulation Approaches

We are not the first ones to suggest to couple multiple simulations for increased representativeness of the coupled system. In the realm of factory and production planning, complex assembly lines or factories are simulated using an approach called *co-simulation* [5, 17, 22]. In co-simulations of production lines, all elements possess a simulation that is specialized for this specific element, e.g a simulation for a conveyor belt and simulations for multiple robots and multiple machining tools [5]. The simulations communicate via a unified interface known as *functional mock-up interface (FMI).*

The main drawback of this approach, for using it in robotics, is that there is a static mapping between elements in the world and simulators. We, however, want to be able to dynamically change the assignment of objects to models. This is particularly important when revisiting the example from fig. 1. An object that can be grasped should not always be simulated by a grasping simulation, but only when it is actively being grasped. Therefore dynamic responsibility allocation is one of the key concepts of the framework we present in the following.

2.4 Multi-Physics Simulation

Multi-physics simulations are simulations that combine multiple physics simulations to simulate setups with many objects.

A framework that implements the idea of combining multiple simulations into one big simulation is ARGoS [**?**]. ARGoS was designed to support and parallelize the simulation of large swarms of robots. Therefore, it consists of multiple physics simulations that each govern a part of the whole simulation space. Each simulation is responsible for simulating the objects residing in the space governed by it. ARGoS shares some ground principles with our framework but differs in two core design choices: First, ARGoS focuses on physics simulations while we propose a very general framework that supports any kind of model, not only physics based models. Second, the responsibility allocation in ARGoS follows a spatial subdivision of the simulated space. In contrast, our approach uses a more general trigger mechanism to decide which object is governed by which simulation.

Another multi-physics dynamics engine is Chrono [**?**]. The strength of Chrono lies in using parallelization to simulate large dynamic systems. The drawback for our scenario is that Chrono is a monolithic program and does not allow to combine multiple heterogeneous physics simulations. Thus it is impossible to reuse existing digital twin models and all implementation of models must happen in Chrono itself. Furthermore, similar to ARGoS, Chrono focuses on physics simulations while we aim to combine different types of models.

3 MULTI-AGENT HETEROGENEOUS DIGITAL TWIN SIMULATIONS

In this chapter we provide insights into the framework we developed for multi-agent heterogeneous digital twin simulations. First, we give a brief overview over the design choices behind the framework, then we present the modules in more detail. Whenever we speak of a set of models and the corrseponding Model Conductor interacting with each another, we refer to that as an *ensemble.* In contrast we will use the notion of *scene* to describe the state of the environment, e.g. in terms of objects and their positions.

3.1 Rationale Behind the Framework

Our framework is centered around the concept of *models.* Models in this context are representations of entities in the world, be it physical entities such as objects or virtual concepts. Our framework connects multiple models in order to have a more accurate and wholesome representation of the environment. From a cognitive science point of view, we are inspired by the insight that humans use models to understand and predict their surrounding (for an overview see [9]).

The framework consists of multiple components that are later described in detail. The central hub of communication and information sharing is the *Model Conductor* that coordinates and orchestrates all models in the scene. The number of models is not restricted but there must be one main model that times the updates of all other models. Furthermore, the framework allows the definition of *triggers* that are activated by the output of certain models and trigger a reconfiguration of the scene. Communication between the different modules is based on cyclic communication that is used to continuously publish data and acyclic communication implementing a simple request-response mechanism. This is in line with commonly known robotic middleware like ROS [19] or Links and Nodes [21].

In more concrete terms, we consider two types of models: simulations and observers. Simulations can be anything from a multi-body physics simulation to implementations of equations modeling the cooling of a cup of coffee. The core peculiarity of simulations is their ability to predict the next state given the history of old states and possibly any external input. In that sense, simulations are *predictive.* In contrast to that, observers are models that evaluate abstract predicates based on the current state. Thus, observers implement *interpretative* models.

It is important to note, that all simulations in the framework can dynamically allocate responsibility for certain attributes of other objects. Usually the main model is a multi-body physics simulation that is responsible for the positions of all objects in the world at the start. This represents the fact that all objects passively obey the laws of physics when they are not actuated. If now an object moves because it is actuated, it can allocate responsibility over its own position. Similarly an object can allocate responsibility for the position of another object, e.g. the model of the robot allocates responsibility of an object when it grasps that object.

The definition of the initial state of the scene is provided via a human-readable configuration file. It specifies the models in the scene, active triggers, and the initial allocation of objects to models.

3.2 Model Interfaces

In order to provide the functionalities presented above, models must offer a standardized interface to the Model Conductor. As every model implements a (partial) digital twin of an object or concept,

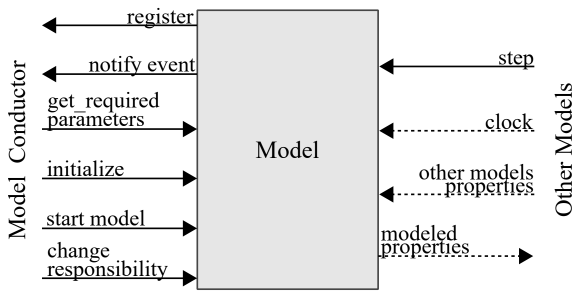


Figure 2: Interface of the model class. Arrows represent incoming and outgoing communication. Dashed arrows specify cyclic communication and solid arrows acyclic communication. Communication with the Model Conductor goes to the left, communication with other models to the right.

our interface is loosely inspired by the *Functional Mock-up Interface* for co-simulations[1]. The main interfaces of models are shown in fig. 2 and described in the following.

Registration at the Model Conductor is important to advertise the model and make it available also to other models. During the registration process the Model Conductor creates a unique id for each model that is later used to address the model.

Initialization of a model is performed in a multi-step approach. First, the Model Conductor queries the parameters required by the model, resolves them internally, and finally passes them in an initialization call to the model. Parameters can be any kind of information needed for a successful initialization of a model like object poses, object types, or ids of other models. Finally each model offers an interface method for starting the model.

During runtime each model can notify the Model Conductor of events that it observed. These events are used to activate triggers that re-allocate model responsibilities. A responsibility is defined as a parameter of an entity in the world, as for example the *position* of an *object*, the *temperature* of a *cup of coffee*, or the *charging state* of a *battery*. Handing over of responsibilities is managed by the Model Conductor and implemented via corresponding interfaces at the model level.

To actually run models, they must implement one of three possible interfaces. Either they are stepped via an acyclic service call, or they step internally based on an external clock signal, or they step without any external trigger. These three possibilities result in different behaviors. If the update of a model is triggered via an acyclic service, the main model blocks until the model finishes the update step. This ensures that the model has enough time to complete computation before the main model steps again. On the other hand, if the model triggers the update internally via a clock signal, the main model is not aware of the time required by the model for an update step. Thus, the main model can not wait for the model and they might run out of sync. Finally, if the model does not require any temporal information (e.g. because it does not model dynamics), it can run at its own pace.

[1]see https://github.com/modelica/fmi-standard/releases/tag/v2.0.2 (accessed 04.02.2022)

The difference between these types of updates results in a trade-off between runtime (blocking calls might slow down the whole framework substantially) and synchronization (internal updates via subscription can result in the model skipping update steps). In the end it is left to the designer of a model to decide how the step method of a model should be implemented.

Finally, the output of a model is made available to other models via cyclic communication. In the same way, any model can read properties of other models via incoming cyclic communication.

3.3 Triggers and Evaluators

Triggers and evaluators are a means to reconfigure a scene. Triggers react to certain events by triggering other events and evaluators are a shorthand form to specify preconditions for triggers and resolve variables.

3.3.1 *Triggers.*

Triggers either get activated by service calls or monitor certain topics and activate if the topic publishes certain values. Upon activation, a trigger may check for extra preconditions if specified. Only if the preconditions hold, the trigger activates an effect.

A possible effect of a trigger is to transfer responsibilities of objects between models. An example could be a trigger that activates when an objects slips out of the grasp of a robot. It connects to an observer that monitors the grasp status of the robot. While the object is grasped, its position is simulated by the robot model. As soon as the object is being released, the responsibility for simulating the position of the object is transferred from the robot model to the main physics simulation.

Triggers do not only transfer responsibilities but they are also used to trigger events in other modules. In the example above imagine that the object grasped by the robot was a glass. After the glass slipped out of the hand of the robot, it falls down and after some seconds gets in contact with the floor. Another trigger, that is activated whenever the glass makes contact with another object, becomes active. It then triggers the model of the glass to check whether the glass broke on impact or not.

A special case for service calls that activate triggers is action sequence information. In the *Action Template* framework of [14] for example, action execution data is annotated with semantic information about the action sequence. When executing an action, the robot does not only publish telemetry data, but also semantic annotations like an identifier for the operation it executes. Making use of that data, we can, for example, define a grasping model that gets activated when the robot executes a `move_hand` operation (see also fig. 1). We extensively make use of that feature in the evaluation in section 4.

As can be seen from the examples, triggers specify the evolution of a scene. With their preconditions and effects they encode the logic of the framework. Thus, care must be given when defining them.

3.3.2 *Evaluators.*

Sometimes it comes in handy to be able to reuse preconditions in our framework. For example, multiple triggers could depend on the precondition of a hand being `free`. Therefore, we introduce the concept of *evaluators* as shortcut. Instead of defining each time how exactly to check whether a hand is free, an

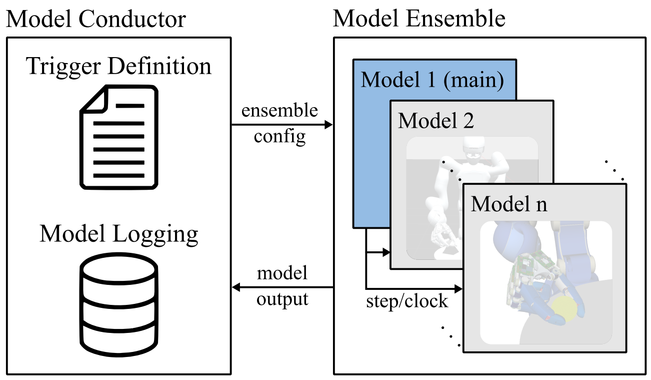


Figure 3: This figure shows an abstract overview of the modules of the simulation framework and the communication between them. The Model Conductor dynamically configures the model ensemble and logs its evolution.

evaluator is written once and is later reused in multiple preconditions.

Evaluators are not limited to only return boolean true or false values but can return arbitrary values. This is used to dynamically resolve arguments. Given a trigger for a `move_hand` operation that transfers the responsibility over the pose of the object that is being grasped to a simulation model. This trigger needs to evaluate which object is about to being grasped right now. Therefore it specifies an evaluator that returns the object closest to the fingers of the robot and assigns its value to a variable. This variable is then used to specify the effect of the trigger.

3.4 Model Conductor

The Model Conductor is key to our framework. It orchestrates the ensemble of models in the scene and forms the glue between them. To act as the central information hub, the Model Conductor loads a configuration from a human-readable .yaml file. This file specifies the scene by means of the models in the scene, the initial responsibilities, triggers, and evaluators. To the outside, the Model Conductor exposes a service to start simulating a scene. This service must be called with an initial geometric description of the scene in terms of the objects and their locations.

The input to articulated models of the scene can stem from either of two possible sources: Either the framework simulates the evolution of a scene based on input from a real robot, or it is based on prerecorded data. The origin of input data must also be specified in the call to the service starting the simulation. It is important to note, that each model in the scene must be able to step in real-time when the framework is connected to live data. If prerecorded data is used to control the models in the simulation, the playback of the data is organized in the main model and adapts speed automatically to all models that are stepped via a service call.

When the Model Conductor is called to simulate a scene, it first initializes all models. Then it sets up connections of the models to the specified data source and creates a process for the triggers. It registers all triggers to that process and creates a logging process to record the output of the models. Finally it starts the models. While the models are running, the trigger process constantly checks for signals that are connected to the existing triggers. When such a signal occurs, the corresponding triggers are activated. The simulation of the scene finishes when either the prerecorded data ends or a service for stopping the simulation is called.

Figure 3 shows an overview of the modules of the multi-agent digital twin framework and how they interact. The Model Conductor configures the model ensemble and logs its output. Configuration happens both initially and, in form of re-configuration, on-line. Online reconfiguration is provoked by triggers defined in the Model Conductor.

4 EVALUATION

Having introduced and described the heterogeneous simulation framework in the previous section, this section shows three exemplary use cases and their evaluation. Our focus in the evaluation lies on a *qualitative* evaluation. The goal of the developed approach is to enable a robot to correctly interpret or predict the (symbolic) effect of its actions. We do not focus on a *quantitative* evaluation of our simulations in terms of real-time factors because this metric depends strongly on the models that are used in the framework and not so much on the framework itself. Instead, the goal of the evaluation section is to demonstrate that our framework allows to simulate complex actions that state of the art methods fail to simulate. Thus, we evaluate our work based on the symbolic effects interpreted after simulating a scene using telemetry data and action sequence information. This is closely related to the work of [1] but with more complex actions than their framework, of using a single simulation, allowed for.

In order to give a complete picture of our approach, we briefly describe a scenario that shows the usefulness of our framework quantitatively. After that we focus on the qualitative evaluation by describing two scenarios and then report the findings for each of them in detail.

4.1 Quantitative Evaluation

As noted above, the quantitative evaluation of our approach is not at the center of our evaluation. It solely describes why it makes sense to subdivide complex simulations in multiple smaller modules as we propose in our approach.

The scenario we chose for this evaluation is the one of simulating the behavior of a fluid in a cup. The fluid is abstracted by multiple spheres in a cup that is placed in front of the robot on a table. The simulation is implemented in gazebo [11], using the ODE[2] engine for simulating physics. We decide to focus on gazebo as it is the most widely used simulation environment in robotics in terms of citations [4, Fig. 2].

We defined five setups and report the real-time factor achieved by gazebo for each of them. The first setup consists of only the robot, the table, and the cup without any particles ("bare simulation w/o particles"). This serves as a reference for the real-time factors as the real-time factor depends on many design decisions and we are solely interested in the difference between real-time factors. The second setup consists of setup 1 plus 50 particles in the cup ("robot with 50 particles"). This setup reflects the baseline for our approach

[2]https://www.ode.org/ (accessed 04.02.2022)

Table 1: Real-time factors for different simulation setups.

Setup	Real-Time Factor
bare simulation w/o particles	0.98
robot with 50 particles	0.55
50 particles w/o robot	0.97
100 particles w/o robot	0.75
100 particles w/o robot, 1.5* step size	1.09

of simulating the whole environment in a single simulation. The third setup consists of only the cup and 50 particles in the cup and the fourth setup consists of only the cup and 100 particles in it ("50 particles w/o robot" and "100 particles w/o robot"). These setups correspond to two possible models for our approach of different accuracy. Finally, the last setup is equal to the fourth setup but uses 1.5 times larger time steps in the simulation ("100 particles w/o robot, 1.5* step size"). The real-time factors for each of the setups are shown in table 1. The real-time factors for setup 3-5 are reported with a simulation of the robot and the environment running in parallel.

In general, the more particles are in a simulation, the slower it gets. This is because the simulation has to conduct more collision checks. The results show two main findings:

First of all, separating the simulation into multiple sub-simulation increases the real-time factor substantially. This is again because the number of collision checks is reduced. Especially the robot consists of a lot of non-convex elements which makes collision computation expensive. If the simulation is split up into a model for the particles in the cup and one for the rest of the environment, the real-time factor increases by 76%. Even if we double the number of particles in the "fluid simulation", the set of two simulations is still 36% faster than the single simulation.

Second, our framework allows to reduce the temporal resolution of a model without impairing the temporal resolution of other models. With this, as the last setup shows, we can run the additional simulation potentially at a lower pace without slowing down the main simulation by making use of parallelization.

4.2 Common Setup and Baseline Algorithms

For the evaluation we created an implementation of the concept described in section 3. The implementation realizes cyclic communication via topics and acyclic communication via services using the Links and Nodes [21] communication framework.

Both of the two exemplary scenarios share some common models. We use gazebo [11] as the main model and create a gazebo-plugin that implements the model of the humanoid robot Rollin' Justin [3]. Both scenarios also share a hand model for grasping of objects based on an in-hand localization [18] and a simple grasp monitor. The in-hand localization predicts the most likely position of the grasped object in the hand based on position and torque measurements of the fingers. Since the in-hand localization is a specialized model for prediction of grasp poses, it operates on the assumption that the object in focus is grasped. As we additionally want to verify this assumption, we add the grasp monitor that is tuned to observe whether anything is grasped but not how it is grasped.

We compare our results with state of the art methods for robot simulations. The first baseline approach is to only use a single physics simulation and simulate grasping together with the robot in that simulation. The second baseline is an extension to the first baseline by rigidly binding the grasped object to the manipulator of the robot upon a "bind" signal and releasing it upon a "release" signal. As a slight evaluation to the second baseline, the third baseline uses a gazebo-plugin that binds grasped objects to the manipulator without the need for semantic signals like "bind" and "release" based on simulated forces between the robot and the objects[3].

After simulating the actions, we extract the resulting semantic state using the framework from [1] as described in [2].

4.3 Interpreting Action Effects in an Uncertain Environment

The first scenario is based on data from [2]. In that work the authors recorded telemetry from a robot that grasped tennis balls from a stand on a table and dropped them over different containers. They learned success probabilities based on a Bayesian approach and used semantic similarity to previously executed actions to generate meaningful priors. The setup they used consists of Rollin' Justin, a table, the tennis ball on a stand, and one of four different containers as seen in fig. 4a. Relevant symbolic labels for this evaluation are `(in _container Ball)` specifying whether the ball ended up being in the container or not. Since the data set contains labels marking whether the task was executed correctly (i.e. whether the ball ended up in the container), we are able to evaluate the result of our framework against ground truth data and compare it with the baseline approach.

In the initial configuration of the ensemble, all objects are governed by the main model which is a gazebo simulation with the ODE engine. We additionally specified the following two triggers: The first trigger reacts to "move_hand" operations (c.f. [13]) in the action execution data and checks whether the manipulator being moved is free (via the grasp monitor) and whether an object is in graspable distance of that manipulator. If both preconditions evaluate true, responsibility of modeling the pose of the graspable object is allocated by the in-hand localization.

The first part of the evaluation is concerned with successful runs from the dataset, during which the robot was able to drop the tennis ball into the container. Therefore we randomly select five successful runs and replay them in our framework and with the baseline implementations. Four out of the five runs are successfully simulated by our framework. The first baseline approach of a single physics simulation does not even manage to grasp the tennis ball once. The spherical surface of the ball and the small surface of the fingers of the robot make the ball always roll out of the hand. Also the third baseline approach does not perform well. Similar to the first baseline, the ball starts moving upon the first contact with the robot finger such that the grasp never builds up enough internal force to activate the binding mechanism. Only the second baseline approach manages to reliably bind the ball successfully as it reacts to semantic triggers. When it comes to dropping the ball into the

[3]https://github.com/JenniferBuehler/gazebo-pkgs/tree/master/gazebo_grasp_plugin (accessed 04.02.2022)

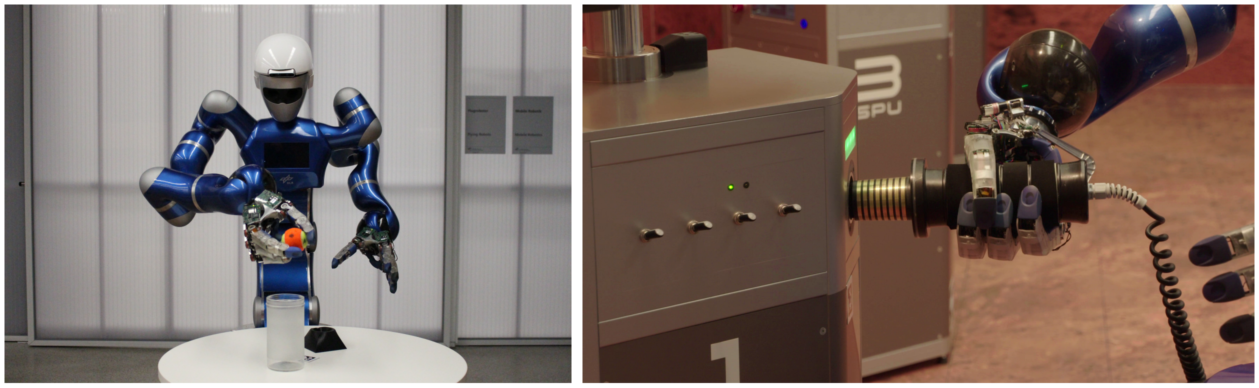

(a) (b)

Figure 4: Evaluation scenarios. (a) *Uncertain Environment:* Rollin' Justin in front of a table with a tennis ball in hand about to drop it into a cylindrical container. (b) *Insertion Action:* Rollin' Justin in front of the SPU, about to connect the DIP to the SPU.

containers, however, the second baseline only succeeds in 2 out of the five runs.

The second part of the evaluation is concerned with five unsuccessful examples, where the ball did not end up in the container. Reasons can be that either the robot dropped the ball early or the ball simply did not hit the container. Baseline one and baseline three again fail to grasp the ball from the stand. Our approach correctly predicts that the ball does not fall into the container all the times, the second baseline as well. The findings are summarized in table 2.

It should be noted that we chose not to report the number of times baseline 1 and baseline 3 ended up with the ball not being in the container when interpreting data of failed action executions in table 2. As they did not manage to simulate a grasp at all, it is understood that they never reported a ball in the container. However, this does not seem to imply any sort of correct result.

4.4 Simulating Inserting Actions

The second scenario is linked to a task from [20] on the SOLEX proving ground [15]. This scenario was set up to resemble a Martian solar farm. It was used in experiments where astronaut onboard the *International Space Station (ISS)* commanded Rollin' Justin to do maintenance tasks [20]. In the specific task, the robot is facing a *Smart Payload Unit (SPU)* and has to connect to it with its *Data Interface Probe (DIP)* as shown in fig. 4b. The crunch of this scenario is that it is hard to accurately model the snapping of the DIP in the tight hole of the SPU. Especially when the robot did not accurately localize the SPU, the chances are high, that the DIP in simulation does not perfectly fit into the SPU. On the real robot the problem of potentially bad localization is compensated to a certain degree through the use of a compliant controller.

Thus, we created a very simple additional model for the SPU that's only tasked with modeling whether the DIP is connected to the SPU. To do so it constantly compares the position of the DIP with respect to the SPU against the ideal position that represents a connection between DIP and SPU. When the DIP enters a cylindrical area defined around the ideal position, the SPU model allocates responsibility over the DIPs collisions to keep the physics simulation from becoming instable because of collisions between SPU and DIP. If the DIP enters another smaller cylinder around the ideal position, the SPU model also allocates responsibility over the position of the DIP, signaling that DIP and SPU are connected.

In this scenario we start with the DIP already grasped by the robot. We use data of five successful executions and replay them in our framework and with the baselines. With our framework it is possible to simulate successful connections three out of five times. When the robot does not successfully connect the DIP to the SPU it is so far away from the ideal connection point, that the SPU model does not bind the DIP. In the first baseline of using one physics simulation, the DIP falls out of the hands after the robot is moving a bit and can never successfully be connected to the SPU. The second and the third baseline successfully manage to keep the DIP grasped but when they get close to the hole in the SPU, the DIP collides with the SPU and the simulation becomes unstable.

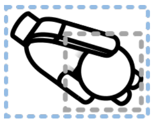
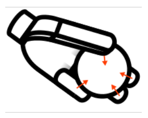
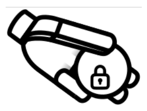
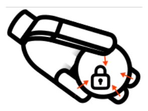
5 DISCUSSION

The evaluation shows that our approach is able to simulate complex tasks that can not be simulated with a single simulation alone. To achieve this result, we combine multiple models into a Multi-Agent Heterogeneous Digital Twin simulation. Due to the standardized interfaces between the models and the Model Conductor, we are able to extend the framework easily. In the following we'll discuss some properties of our framework.

5.1 Real-Time Factor of Simulations

Especially when using the framework as a digital twin for a running robot and its environment, the real-time factor of the simulation matters a lot. If the simulation is not able to run in real-time, the simulated state of the world deviates from the real world state. In our experiments the simulation ran with a real-time factor of ≈0.4 on a standard notebook. While this does not allow to run as a digital twin for a robot, it should be noted that we did not optimize our simulations for speed. Especially the in-hand localization used much of the resources on the system. In the end we believe that using multiple smaller simulations instead of one big simulation might

Table 2: Results from the first evaluation. The result of the simulations is compared to ground truth data about execution success. Numbers show correct interpretations out of all tests. Baseline 1 and 3 are not applicable for failed execution because they do not even manage to simulate a grasp of the tennis ball.

scenario	ground truth	 **our approach (model ensemble)**	 **baseline 1 (pure simulation)**	 **baseline 2 (rigid binding)**	 **baseline 3 (force binding)**
Uncertain Environment	execution successful	4/5	0/5	2/5	0/5
	execution failed	5/5	n/a	5/5	n/a
Insertion Action	execution successful	3/5	0/5	0/5	0/5

even be beneficial for the simulation speed. Instead of simulating the whole environment on a common level of detail, the modular approach lets users simulate different aspects of the surrounding at different levels of detail. In our example the models used very different timings. While the gazebo simulation ran with ≈ 400 iterations per second, the grasp monitor only ran at a rate of 15 iterations per second. Also running multiple models in parallel offers a simple approach to parallelized simulation.

5.2 Accuracy of the Framework

We evaluated our approach with the question in mind: "Is the framework able to interpret the correct symbolic state of the environment after a task execution?" Other metrics for simulation accuracy could be the distance between objects in simulations and ground truth positions. We did not focus more on accuracy since the overall accuracy of our ensemble of simulations is closely connected to the accuracy of the models themselves. It is up to the user of the system to create models that are as accurate as needed. However, we could show that very simple models, such as the model of the SPU, could greatly improve the outcome of our framework.

5.3 Scalability

The framework can be extended by adding models to it. In our setup in the evaluation we used at max 4 models: the main gazebo simulation, the in-hand localization, the grasp monitor, and the SPU model. Generally the framework is as fast as its slowest model. Therefore it is important to create light-weight models that are only activated selectively. In our example, the in-hand localization was not active all the time but only got activated when a grasp occurred. Following these two guidelines, light-weight models that are selectively active, we expect the framework to scale well to bigger problems.

6 CONCLUSION

In this work we presented a concept to create representative simulation of a robot and its environment by making use of an ensemble of models. We introduced the concept of a Multi-Agent Heterogeneous Digital Twin simulation and showed how it is used in a prototypical implementation to simulate tasks that are hard to simulate otherwise. The evaluation was based on the *interpretative* power of the framework, that is its ability to correctly interpret the course of action based on recorded data.

In the future we plan to work on two main topics related to this work, namely on improving the simulation framework itself and making use of the simulated outcomes. For the first topic, we plan to combine the robot model with a simulink controller to be able to consider forces in the simulation and make the framework also useful for *predictive* simulations. Furthermore, we aim to improve the accuracy of existing models and implement more complex models as, for example, fluid simulation to simulate and detect spilling when operating with fluids. We will also integrate more physics simulations with our framework such as MuJoCo [23], AMBF [16], or Isaac Sim[4]. Moreover, we are interested in learning triggers for reconfiguration of the ensemble. Last but not least, we plan to extend the framework to support parallel simulations in order to create distributions of results reflecting uncertainty in the initial state.

In the realm of exploiting the simulation results we are especially interested in using the outcomes to detect and explain erroneous task executions of the robot. By comparing the data from different simulation runs with the planned optimal task execution, we aim to detect discrepancies. We plan to use this approach to prevent errors by avoiding critical action parameterizations during planning (predictive) or to explain why an action failed in hindsight (interpretative).

ACKNOWLEDGMENTS

The authors would like to gratefully acknowledge the financial support and endorsement from the DLR Management Board Young Research Group Leader Program and the Executive Board Member for Space Research and Technology.

REFERENCES

[1] Adrian S. Bauer, Peter Schmaus, Alin Albu-Schäffer, and Daniel Leidner. 2018. Inferring Semantic State Transitions During Telerobotic Manipulation. In *Proc. 2018 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*. IEEE, Madrid, Spain, 5517–5524. https://doi.org/10.1109/IROS.2018.8594458

[2] Adrian Simon Bauer, Peter Schmaus, Freek Stulp, and Daniel Leidner. 2020. Probabilistic Effect Prediction through Semantic Augmentation and Physical Simulation. In *Proc. 2020 IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, Paris, France, 9278–9284. https://doi.org/10.1109/ICRA40945.2020.9197477

[4]https://developer.nvidia.com/isaac-sim (accessed 04.02.2022)

[3] C. Borst, T. Wimbock, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schaffer, and G. Hirzinger. 2009. Rollin' Justin - Mobile Platform with Variable Base. In *Proc. 2009 IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, Kobe, Japan, 1597–1598. https://doi.org/10.1109/ROBOT.2009.5152586

[4] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. 2021. A Review of Physics Simulators for Robotic Applications. *IEEE Access* 9 (2021), 51416–51431. https://doi.org/10.1109/ACCESS.2021.3068769

[5] Tim Delbrügger, Matthias Meißner, Andreas Wirtz, Dirk Biermann, Johanna Myrzik, Jürgen Rossmann, and Petra Wiederkehr. 2019. Multi-Level Simulation Concept for Multidisciplinary Analysis and Optimization of Production Systems. *Int J Adv Manuf Technol* 103, 9-12 (Aug. 2019), 3993–4012. https://doi.org/10.1007/s00170-019-03722-1

[6] Tom Erez, Yuval Tassa, and Emanuel Todorov. 2015. Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *Proc. 2015 IEEE Int. Conf. Robotics and Automation*. Seattle, USA, 4397–4404. https://doi.org/10.1109/ICRA.2015.7139807

[7] Jason Fischer, John G Mikhael, Joshua B Tenenbaum, and Nancy Kanwisher. 2016. Functional Neuroanatomy of Intuitive Physical Inference. *Proc. Natl. Acad. Sci.* 113, 34 (2016), E5072–E5081. https://doi.org/10.1073/pnas.1610344113

[8] Michael Grieves and John Vickers. 2017. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In *Transdisciplinary Perspectives on Complex Systems*. Springer, 85–113.

[9] P. N. Johnson-Laird. 2004. The History of Mental Models. In *Psychology of Reasoning*. Psychology Press.

[10] Natalie A. Jones, Helen Ross, Timothy Lynam, Pascal Perez, and Anne Leitch. 2011. Mental Models: An Interdisciplinary Synthesis of Theory and Methods. *Ecol. Soc.* 16, 1 (2011).

[11] Nathan Koenig and Andrew Howard. 2004. Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In *Proc. 2004 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*. Sendai, Japan, 2149–2154. https://doi.org/10.1109/IROS.2004.1389727

[12] Lars Kunze, Mihai Emanuel Dolha, Emitza Guzman, and Michael Beetz. 2011. Simulation-Based Temporal Projection of Everyday Robot Object Manipulation. In *Proc Int. Conf. Autonomous Agents and Multiagent Systems*, Vol. 1. Taipei, Taiwan, 107–114.

[13] Daniel Leidner, Christoph Borst, and Gerd Hirzinger. 2012. Things Are Made for What They Are: Solving Manipulation Tasks by Using Functional Object Classes. In *Proc. 2012 IEEE-RAS Int. Conf. Humanoid Robots*. 429–435. https://doi.org/10.1109/HUMANOIDS.2012.6651555

[14] Daniel Sebastian Leidner. 2019. *Cognitive Reasoning for Compliant Robot Manipulation* (first ed.). Springer Tracts in Advanced Robotics, Vol. 127. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-04858-7

[15] Neal Y. Lii, Daniel Leidner, André Schiele, Peter Birkenkampf, Ralph Bayer, Benedikt Pleintinger, Andreas Meissner, and Andreas Balzer. 2015. Simulating an Extraterrestrial Environment for Robotic Space Exploration: The METERON SUPVIS-JUSTIN Telerobotic Experiment and the SOLEX Proving Ground. In *13th Symp. Advanced Space Technologies in Robotics and Automation (ASTRA)*. Noordwijk, The Netherlands, 1–7.

[16] Adnan Munawar, Yan Wang, Radian Gondokaryono, and Gregory S. Fischer. 2019. A Real-Time Dynamic Simulator and an Associated Front-End Representation Format for Simulating Complex Robots and Environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1875–1882. https://doi.org/10.1109/IROS40897.2019.8968568

[17] Maurizio Palmieri, Cinzia Bernardeschi, and Paolo Masci. 2018. Co-Simulation of Semi-Autonomous Systems: The Line Follower Robot Case Study. In *Software Engineering and Formal Methods (Lecture Notes in Computer Science)*, Antonio Cerone and Marco Roveri (Eds.). Springer International Publishing, Cham, 423–437. https://doi.org/10.1007/978-3-319-74781-1_29

[18] Martin Pfanne and Maxime Chalon. 2017. EKF-Based in-Hand Object Localization from Joint Position and Torque Measurements. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2464–2470. https://doi.org/10.1109/IROS.2017.8206063

[19] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. ROS: An Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*, Vol. 3. Kobe, Japan, 5.

[20] Peter Schmaus, Daniel Leidner, Thomas Krüger, Andre Schiele, Benedikt Pleintinger, Ralph Bayer, and Neal Y. Lii. 2018. Preliminary Insights From the METERON SUPVIS Justin Space-Robotics Experiment. *IEEE Robot. Autom. Lett.* 3, 4 (Oct. 2018), 3836–3843. https://doi.org/10.1109/LRA.2018.2856906

[21] Florian Schmidt and Robert Burger. 2014. How We Deal with Software Complexity in Robotics:'Links and Nodes' and the 'Robotkernel'. In *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

[22] Sebastian Thiede, Malte Schönemann, Denis Kurle, and Christoph Herrmann. 2016. Multi-Level Simulation in Manufacturing Companies: The Water-Energy Nexus Case. *Journal of Cleaner Production* 139 (Dec. 2016), 1118–1127. https://doi.org/10.1016/j.jclepro.2016.08.144

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A Physics Engine for Model-Based Control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

[24] Constantin Uhde, Nicolas Berberich, Karinne Ramirez-Amaro, and Gordon Cheng. 2020. The Robot as Scientist: Using Mental Simulation to Test Causal Hypotheses Extracted from Human Activities in Virtual Reality. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 8081–8086. https://doi.org/10.1109/IROS45743.2020.9341505

[25] D.H. Wolpert and W.G. Macready. 1997. No Free Lunch Theorems for Optimization. *IEEE Trans. Evol. Comput.* 1, 1 (April 1997), 67–82. https://doi.org/10.1109/4235.585893