Going Cognitive: A Demonstration of the Utility of Task-General Cognitive Architectures for Adaptive Robotic Task Performance

Tyler Frasca and Zhao Han and Jordan Allspaw and Holy Yanco and Matthias Scheutz

Abstract— **It has been claimed that a main advantage of cognitive architectures (compared to other types of specialized robotic architectures) is that they are task-general and can thus learn to perform any task as long as they have the right perceptual and action primitives. In this paper, we provide empirical evidence for this claim by directly comparing a high-performing custom robotic architecture developed for the standardized robotic "FetchIt!" challenge task to a hybrid cognitive robotic architecture that allows for online one-shot task learning and task modifications. The results show that there is no disadvantage of running the hybrid architecture (i.e., no significant difference in overall performance or computational overhead compared to the custom architecture) while adding the flexibility of online one-shot task instruction and modification not available in the custom architecture.**

I. INTRODUCTION

Over the years, cognitive architectures have been repeatedly touted as meaningful additions to robotic architectures as they could enable the latter to have better high-level control and reasoning as well as interaction capabilities with humans (e.g., [1]). While such "hybrid systems" have been shown to be able to perform tasks on robots, the tasks were often limited in complexity and usually lacked a direct comparison to the performance of a mere robotic architecture in the same task which would have allowed for a substantiation of the claim that there is utility of the added complexity of the cognitive architecture.

In this paper, we address this lack of a direct comparison, and thus the missing support for the above claim about the utility of cognitive architectures on robots, by performing the first through empirical comparative evaluation of a purpose-made robotic architecture for the standardized task designed for the "FetchIt! The Mobile Manipulation Challenge" (at ICRA 2019)[1] with a hybrid cognitive-robotic architecture. Critically, the mere robotic and the hybrid architecture used the very same perceptual and action primitives, but the hybrid architecture utilized explicit task script representations that were accessible via natural language and could be modified on the fly. We demonstrate through empirical evaluations comparing both architectures in the "FetchIt!" task on the same Fetch robot that there is no significant difference in computational effort or task performance (both in terms of time-to-task-completion and number of completed runs) between the two architectures, while the hybrid architecture allows for various extended features such performance monitoring and task modifications through as spoken natural language dialogues that would require extra significant development effort on the purpose-made architecture. The evaluations provide evidence for the utility of a task-general cognitive architecture for robotic task performance, specifically showing that (1) the entire task can be instructed in spoken natural language (NL) instead of having to be programmed, (2) the system can be queried in NL throughout the task performance about its expected performance, next steps, goals, etc., and (3) the task can be modified during execution through NL dialogues with the modifications taking effect immediately without the need to stop the operation. These added features to task performing robots enabled by the addition of a cognitive architecture without any downsides will significantly improve the utility of robots for many application domains, including versatile manufacturing as demonstrated by the "FetchIt!" task.

[1]`https://opensource.fetchrobotics.com/competition`

II. MOTIVATION

Cognitive architectures like ACT-R, Soar, and others were originally developed as models of human cognition, consisting of various modules, representations, and processes that together by way of their interactions are able to perform a large variety of cognitive tasks [1]. Different from robotic architectures, which traditionally were developed for specific classes of (robotic) tasks (navigation, manipulation, etc.), cognitive architectures have from the beginning aimed to be *task-general*: given a set of task-specific percepts and actions, and a task description with a task goal, a cognitive architecture should be configurable to perform the task and accomplish the goal. Ideally, it should be possible to learn from the very same instructions that humans would receive if they were to perform the task (e.g., [2]).

Recently, it has been demonstrated that cognitive architecture can learn a significant set of different tasks from instructions and are even able to utilize the commonalities among the tasks [3]. Clearly, this kind of capability would be of great utility for the robotics community, as robots could be instructed to perform a variety of tasks that share a set of perceptions and actions. However, cognitive architectures have not been designed with robots in mind and are thus typically not able to easily control robots (e.g., for SOAR a special interface had to be developed and much of the perceptual system lives outside the architecture proper, similar adaptations had to be made for ACT-R, e.g., see [4]).

One way to overcome the various challenges of running cognitive architectures directly on robots – real-time processing of sensory data, temporally extended actions, parallel operating of different components, etc. – while preserving their advantages – task generality, instructibility, high-level reasoning, etc. – is to integrate them with robotic architectures that take care of the nexus between real-world sensors and actuators and the robotic architecture's high-level percepts and actions. Two main integration methods have been pursued over the year (e.g., see [4] for details). The first is to use the robotic architecture as an implementation environment for the cognitive architecture, transducing real-world sensory data into the particular high-level perceptual input form cognitive architectures require, and mapping high-level atemporal action outputs from the cognitive architecture onto time-sensitive real-time controllers or even complex action scripts that implement the intended action on the robotic platform. The second integration is to treat the cognitive architecture as an additional component in the robotic architecture that operates concurrently with all other components. Different from the first integration where the cognitive architecture is in control of the overall action execution, both robotic architectures are jointly in control (although the degree to which each has a say will depend on how they specifically interact).

III. The "FetchIt!" Mobile Manipulation Challenge

The goal of the "FetchIt!" competition tasks is for a Fetch robot [5] to assembly a kit of parts from different stations and transport it for inspection. The Fetch robot is a mobile robot with a chest-mounted seven degrees of freedom arm and a head-mounted RGBD camera. As shown in Fig. 1, to reach the goal, the robot needs to navigate in a narrow work cell by moving closely among different stations to pick mechanical parts with the large raw gear machined, place them into the correct compartments of one of the caddies on the top table, and transport the caddy to the table at the bottom left. Clockwise, the parts are small gears, large gears, gearbox tops, gearbox bottoms, and screws[2]. For the large gear, the robot needs to insert the raw version into the narrow lathe chuck of the SCHUNK machine located to the right to shape the gear thread; the robot can also push the door handle to close before insertion or pull to open before removing. After placing two screws, one small and machined large gear, one gearbox top and bottom into the caddy, the robot need to pick up the kit and transport it to the bottom-left table for inspection.

Three major challenges need to be addressed in order to reach the goal: navigation, perception, and manipulation. The dimension of the arena is 3×3 meters, which is relatively narrow for a Fetch with 0.6-meter-diameter round base: the arena can only accommodate 9 Fetch robots (3×3) with all the tables. In order to reach objects on the farther half of the table, the robot needs to stop in front of the table as close as possible without any collisions. This capability is important because Fetch's arm is short when not counting the long wrist link while grasping the caddy handle from a upright position. It is also vital for the large gear machining task for two reasons. First, the top door handle on the SCHUNK machine needs to be reachable during the whole open/close process. Second, the robot needs to see the whole lathe chuck side for detection and large gear insertion.

[2]The CAD models are available at `https://github.com/fetchrobotics/fetchit`

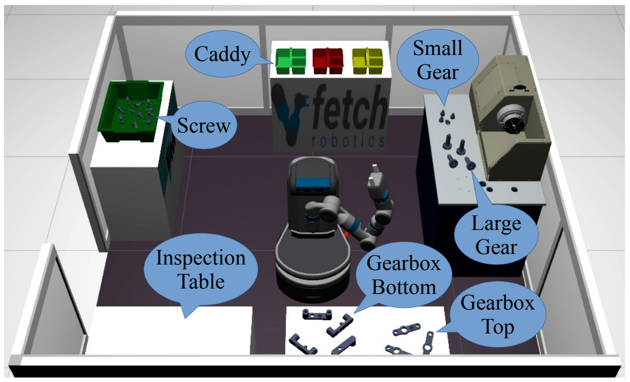

Fig. 1. The "FetchIt!" environment. The robot's goal is to place the irregular parts into the correct sections of the concave caddy on the top table, and transport the caddy to the bottom-left table for inspection.

IV. The UML-HRI Mobile Manipulation Framework and DIARC Architecture

Since we were interested in directly comparing a robotic architecture with a hybrid cognitive-robot architecture, we started with the high-performing robotic architecture specifically developed for the "FetchIt!" task (the architecture won second prize in the competition, see [6]). We removed its action sequencing component that implicitly encodes the task and replaced it with the complete DIARC architecture [7] which is then connected appropriately to sensory and action processing components to allow it to perform the task learning, task performance prediction, and task modification. In the following, we first describe the base robotic architecture, followed by a short overview of the employed DIARC architecture and how it was integrated with the base architecture.

A. The UML-HRI Mobile Manipulation Framework

The architecture we developed for the 2019 "FetchIt!" competition was not based on a particular robotic or cognitive architectural paradigm, but rather consisted of a set of existing and custom-built components utilizing the ROS middleware. One component (to be replaced in the hybrid architecture) is in charge of implementing the task flow and coordinating the activities among the other components (see Fig. 2) such as the detector components ("caddy", "dropoff", "schunkdoor", "chuck"), the manipulation components ("caddy", schunkdoor", "widget"), as well as the navigation components for moving within the arena.

We used the widely available ROS navigation stack [8], originally designed for indoor office environment. Using this technology, the robot is reluctant to move close to a table less than 20 cm due to a rough base model, which is also confirmed by [9]; it will get stuck and keep rotating for localization. In their system [9], Ciocarlie *el al.* allow for manual control by users to move the robot closer through a GUI. Since the "FetchIt!" competition requires the robot to

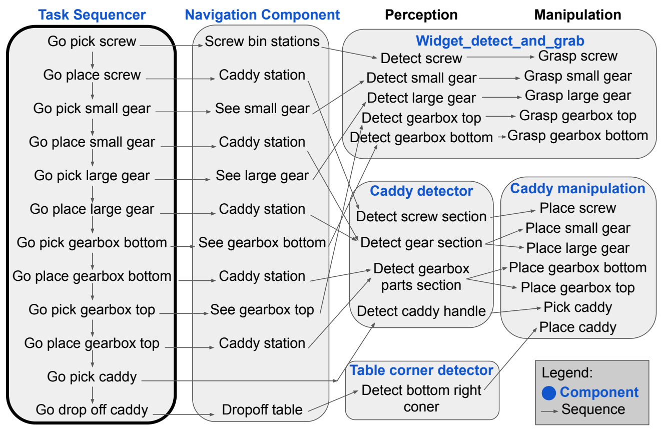

Fig. 2. Diagram for the task sequencer in the custom architecture and its relation to other perceptual, manipulation, and navigation components; the components are C++ libraries. In the hybrid approach, the task sequencer component is replaced with DIARC, and all other components are converted to ROS services for the DIARC integration.

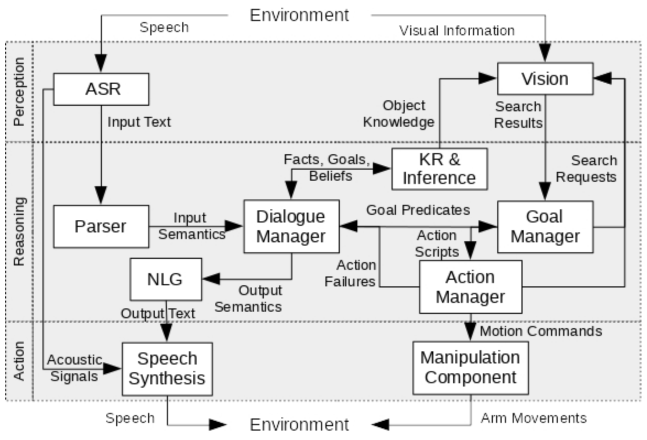

Fig. 3. A diagram of a particular instantiation of the DIARC architecture, showing the natural language processing components (ASR, Parser, Dialogue Manager, NLG, Speech Synthesizer) as well as the goal management components (Goal Manager, Action Manager with Action Interpreter).

be fully autonomous, we solved the issue by implementing autonomous manual base movement, which moves the robot towards the table after reaching an approximate goal and moves away after finishing the manipulation task.

For perception, the task-based objects are complex and irregular, unlike blocks or cylinder objects commonly seen in the robotics field. The caddy is a concave object and has a handle and three compartments. It is practically symmetric but one side has a divider that can be easily omitted due to occlusion, which is the case when the handle is horizontal and the divider is behind the handle.

Because we have access to the CAD file of the caddy, we tried to cluster the caddies and match the point cloud to the CAD model, but none of them (correspondence grouping [10], [11], hypothesis verification [12], hypothesis rejection [13]) can detect the orientation of the caddy correctly and consistently. We did not attempt the convolutional neural network method [14] as data collection is difficult for the unknown competition arena and the learned model using data from our test arena may not transfer to the competition arena environment. Instead, we used a height heuristic to crop the point cloud of the caddy handles, clustered them using Euclidean cluster extraction, determine the y axis using the principle component analysis (PCA) technique, and determined the divider by checking points at both sides of the caddy.

For other objects such as large gears and gearbox top/bottom, we used a simple plane model segmentation to find the table in front and isolated the region directly above the table. Then we obtained each cluster above the table, examining the width, length and height of each and comparing with the desired object, discarding any that do not match. In addition objects that were too close together were discarded. Since all objects involved have a main axis, we used the same PCA technique to determine the orientation of the object. For most objects, we could line up directly above and come straight down, however for the large gear our grasp strategy required us to grab the gear end at a slight angle. If no objects were detected within the first frame, the robot would keep scanning until a valid object was found.

When choosing which object out of all the valid objects to actually grasp, the strategy varied per object. Most objects were on a table with one other object, each on opposite sides. In those cases the chosen object would be the one furthest from the other objects side, to reduce the chances of grabbing the wrong object. In the case of screws, they were isolated in a bin so the most desired object were the ones located nearest the center of the bin. Once an object was chosen, the robot would first plan the grasp, including the final position. If no solution was found it would move on to the next best object. If all objects were exhausted without a valid solution, it would go back to the detection phase and attempt to find the objects again.

For the actual grasp, the robot would check the position of the gripper after grasping against the expected object size. If there is a significant difference, such as the gripper fully closed, the robot would release the gripper, go back up, and repeat the detection phase. Most actions had simple retry techniques if a failure was detected. For manipulation, if planning failed, or the robot failed to execute a valid plan, it would retry up to 5 times. If still unsuccessful, the error propagated up and the previous step would be repeated. This also applies to motion planning and higher level planning.

For full details of the framework and implementation details, please refer to our work [6].

B. The DIARC Architecture

DIARC is a component-based distributed architecture scheme (see Fig. 3) which can be configured for applications with different components present (a comprehensive overview of the architecture is given in [7]). Most relevant for the current setting, are two subsystems of DIARC: (1) the natural language subsystem and (2) the goal management subsystem, which we will briefly describe next.

The natural language subsystem is used for interactions with the user about any aspect of the robot, including tasking the robot (i.e., giving it new tasks and goals), querying the robot (e.g., about its knowledge, performance state, etc.), and teaching the robot new knowledge (e.g., new skills, tasks, objects, etc.). Here we utilize DIARC's instruction-based one-shot learning capabilities [15] which allow us to instruct the "FetchIt!" task in spoken natural language. Once, instructed, the robot stores the new task knowledge and is able to execute it at any subsequent time. We also utilize DIARC's capability to revise any type of knowledge through natural language dialogues to demonstrate the utility of explicit task representations that can be adapted during task performance when needed. For the purpose of this paper, we added the relevant nouns and verbs for the task-based objects (e.g., caddy, screw, gearbox) and primitive actions (e.g., fetch, navigate, place) to the lexicon of the semantic parser. And we marked the perceptual and action primitives available in the custom-architecture with the same semantic labels as the natural language words in the dictionary to effect a mapping between words for object and their detectors, and words for actions and their primitive action implementations. That way instructions that involve any of these words will be semantically interpreted correctly and result in the appropriate processing in the custom architecture.

The second relevant subsystem is the goal management subsystem, which consists of the Goal Manager (which stores all active goals and subgoals of the robot), the Action Manager (which stores the agent's procedural knowledge) and the Action Interpreter (which takes an action script, i.e., a procedural knowledge representation, and executes it). The Goal Manager can receive goals from the natural language subsystem as well as from other components in the architecture, and upon receiving a goal, determines its validity, and searches for known action scripts that will accomplish the goal (otherwise, if no script is available, it will consult a task planner to find a solution). Action scripts are sequences of actions (which themselves could be scripts or primitive actions) together with their pre-, operating, and post-conditions or control instructions (such as conditionals and loops). The Action Interpreter is responsible for executing an action script, performing the operations prescribed in the script. When an action is a primitive, the Action Interpreter requests its execution from the component in the architecture implementing the primitive. In the case of the action primitives in the "FetchIt!" task, control is passed to the custom architecture for their execution. When pre- or operating conditions are not satisfied, action execution fails and the Action Interpreter stop execution, if requested notifying the natural language subsystem to inform the user of the failure.

V. Architecture Comparison and Evaluation

Our evaluation focuses on a direct comparison between the performance of a purpose-made robotic architecture in the "FetchIt! The Mobile Manipulation Challenge" with a hybrid cognitive-robotic architecture. While there are many aspects that can be evaluated, we focus on two central components: (1) any performance-based differences between the two architectures, e.g., in terms of time-to-task-completion or completion failures, and (2) any computational differences with respect to CPU load and utilization. The first is a check that adding a task-general cognitive architecture does not lead to a drop in performance compared to a high-performing architecture specifically designed for the task, while the second attempts to quantify the computational overhead and cost of adding a cognitive layer. We hypothesized that **H1** the hybrid architecture would not show any statistically significant performance differences from the specialized architecture, and that **H2** the computational overhead of the hybrid architecture was not statistically significant. For **H1**, we thus measured the overall duration of the task as well as the task success and error rates (e.g., whether the robot failed to complete the task altogether, but also when it repeated actions due to action failures). For **H2**, we collected information about the CPU load during task performance.[3].

All experiments were performed on a Fetch mobile robot which has a built in computer with a quad-core Intel Core i5-4590S CPU @ 3.00GHz and 16 GB RAM. No external computing was used except to ssh into the robot and run the commands. First the navigation and motion planning components were started. The robot was given its initial starting position and drove to a center starting location. Then a roslaunch file started the detection nodes, ROS services, and the logging nodes. Finally depending on the condition, a launch file was started launching the appropriate architecture. The run was considered started when the first command was given to assemble a caddy. Upon completion of a caddy, the robot would report its success and the run would end.

A. *Experiment 1: Procedure and Results*

We performed 15 runs of the "FetchIt!" task with each of the two architectures. Since the hybrid architecture does not contain the task sequencing component implemented for the custom architecture, a human instructor taught the task to the hybrid architecture in spoken natural language before the first run using the instruction-based one-shot task learning capabilities of DIARC [15] (the remaining 14 trials were run without the need re-teach the robot as the architecture stored the learned task). To start each trial run of the hybrid architecture, the instructor then issued the spoken natural language command "assemble a caddy", while task performance in the custom architecture commenced automatically when the architecture was started.

As hypothesized in **H1**, the task completion times are very similar with both architectures: 16.27 ± 1.17 minutes for the hybrid and 16.15 ± 1.53 minutes for the custom. A two sample t-test finds no significant difference ($t(28) = .24, p = .81$), hence we cannot reject the null hypothesis. In fact, we obtain a Bayes Factor of 2.84 for the null hypothesis showing that the null hypothesis is about three times as

[3]The script for data collection is available at `http://bit.ly/2wbtaWR`

likely under the data than the alternative hypothesis. Fig. 4 shows the distribution of the task completion times for both architecture. Both architectures successfully completed 15 runs, there was no difference in task success and failure rates (although there was one operator override when the robot got stuck in the hybrid case, where it was not clear whether the robot would have gotten unstuck by itself). We also investigated failures of individual actions (that ultimately succeeded due to the built-in recovery methods that were the same in both architectures) and did not find any significant differences there either.

Surprisingly, a two sample t-test determined that the computational load in terms of the average CPU usage percentage per core for the hybrid architecture ($54.77\% \pm 1.97\%$) was significantly higher throughout the performance compared to the custom architecture ($29.56\% \pm 2.19\%$), thus requiring us to reject **H2** ($t(28) = 33.19, p < .001$). Since this difference was counter to our expectations, we attempted to track down the culprit of the additional load and found it to be in the ASR component running the Sphinx speech recognizer. To confirm that this was indeed the cause for the added load, we performed an additional set of experiments without the ASR, but with instructions entered in natural language through a GUI.

B. Experiment 2: Procedure and Results

We ran an additional set of 15 trials in the modified hybrid configuration with a GUI-based text input component substituted for the Sphinx speech recognizer. The task was then instructed by typing the instructions into the GUI instead of saying them out loud. Every other aspect of the procedure remained the same as in Experiment 1.

As shown in Fig. 4, the result with the modified hybrid architecture showed again no significant difference in time-to-task-completion in the hybrid architecture without speech recognizer (15.90 ± 1.43 minutes) compared to the custom (16.15 ± 1.53 minutes) with a two-sample t-test $t(28) = -0.46, p = .65$ yielding a Bayes factor of 2.68 again in favor of the null hypothesis. Moreover, all 15 runs of the modified hybrid architecture were successfully completed without any operator override. A two-sample t-test now also reveals no significant difference in terms of CPU load between the hybrid without speech recognizer ($30.18\% \pm 0.94\%$) and the custom ($29.56\% \pm 2.19\%$) architectures ($t(28) = -1.01, p = .32$), with Bayes factor of 1.98 showing that the null hypothesis is twice a likely under the data than the alternative, thus lending support to **H2**.

VI. Discussion

The results from the experiments demonstrate that a task-general hybrid architecture where the task is instructed at the beginning of the run by a human instructor – either through spoken instructions or through text-based written instructions – can perform the task as well as a high-performing custom-built architecture. Moreover, the experiments demonstrate that at least in some cases there will not be any significant computational overhead associated with the hybrid task-general architecture.[4] Overall, our evaluation shows that there does not seem to be any disadvantage to using task-general architectures.

On the flip side, there are several advantages to task-general architectures that we will now briefly discuss (e.g., see also [3]), the most obvious being that it can perform a large set of possible tasks even with a given set of percepts and primitive actions (as with the above hybrid architecture using only the percepts and actions used in the "FetchIt!" task). In contrast, an architecture with fixed task representations that are implicitly encoded in the architecture needs to be manually adapted for every task modification. Moreover, the fact that the task is not hard-coded and can be instructed allows non-expert instructors to interact with task-general architectures. This significantly widens the usability of the architecture by including task domain experts that would otherwise not be able to encode their knowledge in the system.

The explicit task representations generated as part of the task learning dialogues are also available for introspection, thus allowing the robot to recount what it has learned and permitting the human to determine whether the robot has learned the task correctly. In case there was a misunderstanding and some of the task aspects were not properly acquired by the robot, the human instructor then has the opportunity to make changes to the task, again through natural language instructions. This interactive way of adjusting the robot's task representations until they match the human expectation allow for an efficient form of debugging that does not require any programming or program inspection, and thus can again be performed by non-experts.

Here we briefly demonstrate a simple modification instructed by the human instructor after listening the robot's current task modification (see `https://streamable.com/9fyi0` for a video of the demonstration):

```
H: Hello Andy.
R: Hello Tyler.
H: Describe how to assemble a caddy.
R: To assemble a caddy, I fetch the screw, I fetch
   the screw, I fetch the small gear, I fetch the
   large gear, I fetch the gearbox top, I fetch the
   gearbox bottom, I pick the caddy, I navigate to
   the dropoff, I place the caddy on table, and I
   reset arm.
H: When you assemble a caddy replace fetch a small
   gear with fetch a large gear.
R: Okay.
H: Describe how to assemble a caddy.
R: To assemble a caddy, I fetch the screw, I fetch
   the screw, I fetch the large gear, I fetch the
   large gear, I fetch the gearbox top, I fetch the
   gearbox bottom, I pick the caddy, I navigate to
   the dropoff, I place the caddy on table, and I
   reset arm.
H: Assemble a caddy.
R: Okay.
```

[4]We have not yet experimented with different configurations of the ASR or different ASRs that might reduce the computational load in the spoken instruction condition, but expect to be able to ultimately bring down the computation needed (e.g., by detecting silence and turning off recognition when not needed).

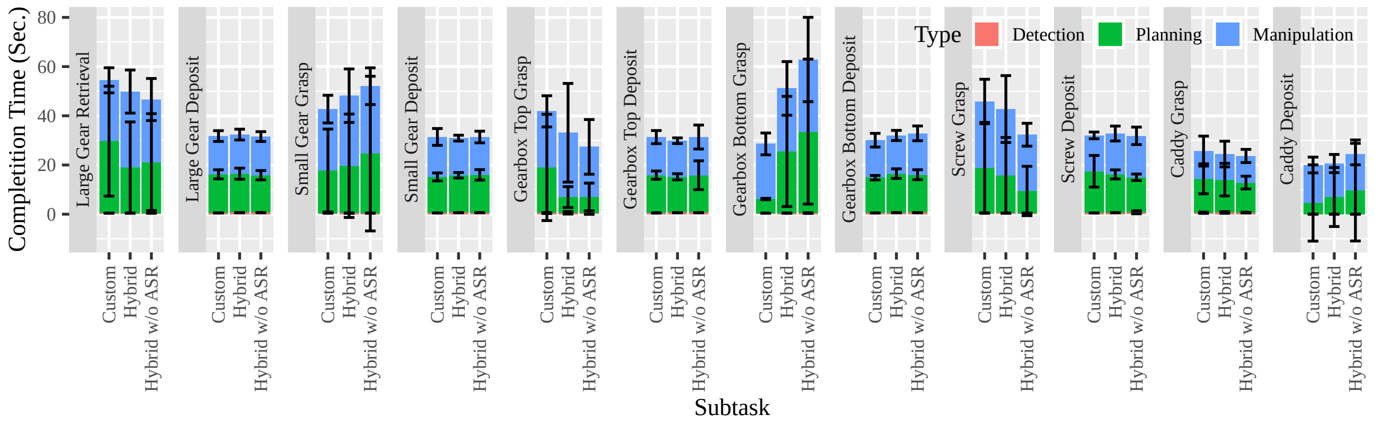

Fig. 4. The time-to-task-completion distribution for all subtasks in the custom and hybrid architecture with and without ASR.

It is also worth pointing out that task modifications are not restricted to dialogues prior to task execution, but can occur at any time during task execution. Hence, if circumstance for execution have changed, or if modifications to the task are required (e.g., because certain parts are missing for assembly or are no longer needed), the task can be modified online during task performance and the modification will take effect immediately. None of these capabilities are achieved with modifying the architecture in the custom-built case.

VII. Related Work

Only a few robotic researchers have been comparing and evaluating robotic systems through its functionality and overall architecture in the robotic domain. Some researchers attempted to introduce the architecture research work in software engineering to robotics by surveying to proposing practical strategy and practices.

Shakhimardanov and Prassler are among the few researchers to propose a formal comparative evaluation methodology for robotic architectures and demonstrated the method with an industrial case study [16]. Their approach comprises four stages that gradually shift from general to more specific aspects: non-functional quality attributes, attribute refinements (i.e., measurements), experimental scenarios, and scenario dissections. A material handling task with a robot arm on a conveyor belt is given as a case study for three different systems. There are performance and robustness quality attributes and experiments. The performance attribute is evaluated by cycle time (network time and execution time) and memory consumption. The robustness attribute is evaluated by hardware and software faults (disconnecting cables and software components).

In addition to manipulation tasks, researchers have evaluated architectures designed specifically for mobile robots, i.e., navigation. In 1993, Orebäck and Christensen, defined seven key requirements of a mobile robot architecture design and comparatively studied three designs [17]. The key requirements are *hardware abstraction* – high portability when hardware changes, *extendibility and scalability* – support for new software modules with efficient communication and well planned data flow, *run-time overhead* – memory, CPU, frequency and latency (deemed not important due to cheap modern hardware), *actuator control model* – behavior model, *general software characteristics* – simplicity both in implementation and interface, correctness, consistency, completeness, *tools and methods* – standardized tools, object-orientation, architecture visualization and modeling, communication and monitoring, and *documentation* – be rigorous and up-to-date. In the evaluation conclusion section, the authors also mentioned distributed processes to achieve scalability, extendibility and load balancing, and language binding. Sheikh *et al.* also compared different navigation control architectures with similar characteristics [18]. Matamoros *et al.* attempted to compare the complexity of component updating and response time between a centralized and a peer-to-peer mobile service robot architecture [19]. However, component updating is defined rather at a micro function level such as moving, splitting and merging in addition to the counterparts of a component.

Instead of identifying commonalities and differences among different architectures in a ad-hoc manner, Dittes and Goerick proposed to scientifically compare existing robotic system architectures by translating them into a high-level common formal language Systematica 2D (Sys2D) [20]. Sys2D is designed to have flexible expressions to translate different architectures and be formal to identify common patterns and architectural differences. In Sys2D, a robotic architecture is separated into layers of semantic function units or sub-architectures, which have dependencies modeled by interfaces and connections through input and output (I/O) ports. I/O ports, triggered by push or pull, have names, types and three requirement roles: required (i.e., Driving), optional (i.e., DrivingOptional) or modulatory (e.g., parameters). The authors evaluated the formal language by translating three robotic system architectures and found patterns and differences. Three common patterns are found: closed sensor-actuator loop for behavior generation (BG), scene decomposition to support BG, and a binding architecture to modulate the loop. For the differences, the authors found that architectures tend to be specific to differently shaped robots and applications, leading to various numbers of the loops and different models of behaviors generated. However, SYS2D focuses on the architecture concepts but omits robotic behavior modeling. It also does not center around tasks.

In addition to the methodology proposed by Shakhimardanov and Prassler and the modeling notation of SYS2D, other researchers has been investigating into architecture paradigms systematically. To manage complexities of a robot system, researchers have been in the unit of object, component, or service.

Ahmada and Babar performed a systematic mapping study to identify and classify software architectures for robotic systems [21]. After classification of the existing architectural solutions, 80% (35/44) of the researchers who modeled their architectures have used Unified Modeling Language (UML) diagrams as architectural notations. To model system structure and behavior, a number of notations are used: use case diagram for requirements modeling, component diagram for components interaction, sequence diagram for execution. According to Ahmada and Babar, four evaluation methods are identified: controlled experiment, simulation, framework evaluation, and real-world evaluation. In our work, we evaluate the two systems in real-world, a work cell.

Amorettia and Reggiani further reviewed three main different architecture paradigms with robotic applications and summarized a set of characteristics as evaluation criteria [22]. The architecture paradigms are Distributed Object Architecture (DOA) – an integration of object-oriented programming with distributed programming, Component Based Architecture (CBA) – compositions with contractual interfaces, and Service Oriented Architecture (SOA) – on-demand resource sharing. They are evaluated in terms of *specification and granularity* – module interfaces defining abstraction granularity, *coupling* – the dependency degree, *state* – memory storage, *workflow* – communication, *reusability* – required effort for adaptation to new environment, *extensibility* – quick adaptation, and *overhead* – e.g., communication or messaging.

VIII. CONCLUSION

In this paper, we have performed the first head-to-head evaluation between a high-performing custom-made task-specific architecture in a standardized robot competition and a task-general hybrid architecture. We showed in two experiments in the "FetchIt!" task on the Fetch robot that there was no performance difference between the two architectures with respect to overall time-to-task-completion and task success/failure rates. Moreover, we demonstrated that in some configuration (with text-based input vs. spoken natural language input) there was no difference in computational load either (the load differences in the spoken case being entirely due to the computational demands of the speech recognizer). The results demonstrate the utility of a task-general architecture where task can be instructed, immediately executed, debugged and modified at any time during task performance by non-expert users without the need for expert robot programmers to write code for every task modification. Thus, the current paper also lends evidence to the more general claim that cognitive architecture provided added value to robotic architecture and that such hybrid systems will be of great utility in a large range of future robot applications and domains.

REFERENCES

[1] P. Langley, J. E.Laird, and SethRogers, "Cognitive architectures: Research issues and challenges," *Cognitive Systems Research*, vol. 10, pp. 141–160, June 2009.

[2] M. Scheutz, E. Krause, and S. Sadeghi, "An embodied real-time model of language-guided incremental visual search," in *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

[3] J. R. Kirk and J. E. Laird, "Learning hierarchical symbolic representations to support interactive task learning and knowledge transfer," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 2019, pp. 6095–6102.

[4] M. Scheutz, J. Harris, and P. Schermerhorn, "Systematic integration of cognitive and robotic architectures," *Advances in Cognitive Systems*, pp. 277–296, 2013.

[5] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *IJCAI Workshop on Autonomous Mobile Service Robots*, 2016.

[6] Z. Han, J. Allspaw, G. LeMasurier, J. Parrillo, D. Giger, S. R. Ahmadzadeh, and H. Yanco, "Towards mobile multi-task manipulation in a confined and integrated environment with irregular objects," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[7] M. Scheutz, T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca, "An overview of the distributed integrated cognition affect and reflection diarc architecture," in *Cognitive Architectures*, ser. Intelligent Systems, Control and Automation: Science and Engineering, M. A. Ferreira, J. S. Sequeira, and R. Ventura, Eds. Springer, 2019, vol. 94.

[8] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 300–307.

[9] M. Ciocarlie, K. Hsiao, A. Leeper, and D. Gossow, "Mobile manipulation through an assistive home robot," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5313–5320.

[10] H. Chen and B. Bhanu, "3D free-form object recognition in range images using local surface patches," *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1252–1262, 2007.

[11] F. Tombari and L. Di Stefano, "Object recognition in 3D scenes with occlusions and clutter by hough voting," in *Fourth Pacific-Rim Symposium on Image and Video Technology*. IEEE, 2010, pp. 349–355.

[12] A. Aldoma, F. Tombari, L. Di Stefano, and M. Vincze, "A global hypotheses verification method for 3D object recognition," in *European Conference on Computer Vision (ECCV)*. Springer, 2012, pp. 511–524.

[13] A. G. Buch, D. Kraft, J.-K. Kamarainen, H. G. Petersen, and N. Krüger, "Pose estimation using local structure-specific shape and appearance context," in *IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2080–2087.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012, pp. 1097–1105.

[15] M. Scheutz, E. Krause, B. Oosterveld, T. Frasca, and R. Platt, "Spoken instruction-based one-shot object and action learning in a cognitive robotic architecture," in *Proceedings of the 16th International Conference on Autoomous Agents and Multiagent Systems*, 2017.

[16] A. Shakhimardanov and E. Prassler, "Comparative evaluation of robotic software integration systems: A case study," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2007, pp. 3031–3037.

[17] A. Orebäck and H. I. Christensen, "Evaluation of architectures for mobile robotics," *Autonomous Robots*, vol. 14, no. 1, pp. 33–49, 2003.

[18] U. A. Sheikh, M. Jamil, and Y. Ayaz, "A comparison of various robotic control architectures for autonomous navigation of mobile robots," in *2014 International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE)*. IEEE, 2014, pp. 239–243.

[19] J. M. Matamoros, J. Savage-Carmona, and J. L. Ortega-Arjona, “A comparison of two software architectures for general purpose mobile service robots,” in *2015 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2015, pp. 131–136.

[20] B. Dittes and C. Goerick, “Intelligent system architectures – comparison by translation,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 1015–1021.

[21] A. Ahmad and M. A. Babar, “Software architectures for robotic systems: A systematic mapping study,” *Journal of Systems and Software*, vol. 122, pp. 16–39, 2016.

[22] M. Amoretti and M. Reggiani, “Architectural paradigms for robotics applications,” *Advanced Engineering Informatics*, vol. 24, no. 1, pp. 4–13, 2010.