2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

Real-Time Software Module Design Framework for Building Self-Adaptive Robotic Systems

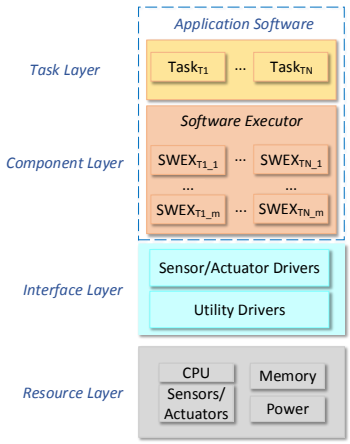
Yanzhe Cui, Joshua T. Lane, Richard M. Voyles

***Abstract*— We proposed ReFrESH in our previous publication. It is a self-adaptive infrastructure aimed at managing the performance of multi-robot systems through dynamically diagnosing and maintaining unexpected issues of modules. To integrate ReFrESH and robotic application-level software more conveniently, it is necessary to develop a module design framework to support implementation of self-adaptive real-time software. To this end, based on the port-based object abstraction and port-automation theory, we propose the *Extended Port-Based Object (E-PBO)*. E-PBO has two main advantages: (1) it builds the basis of a programming model to provide specific, yet flexible, guidelines to robotics application engineers for creating and integrating software modules; (2) it forms the basis of a self-adaption model to provide specific methods for evaluating the running task configuration and estimating the new but non-running task configuration (if required) without interfering with the running configuration. E-PBO has been incorporated into the Port-Based Object Real-Time Operating System (PBO/RT) and applied to a visual servoing robotic application, which is demonstrated here.**

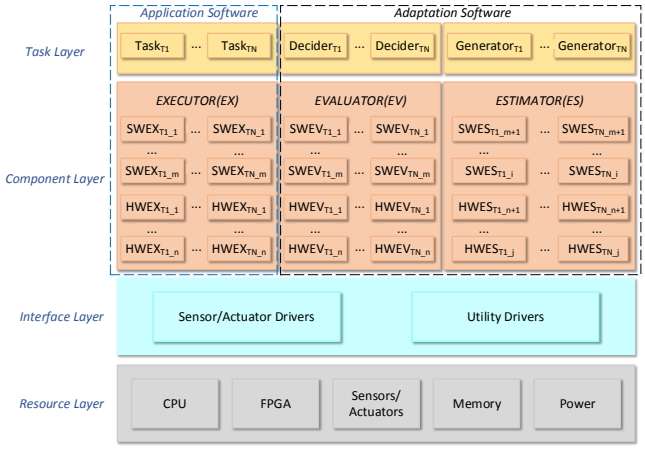
I. Introduction

Robotic systems are suited for the Component-Based Software Engineering (CBSE) design method due to certain domain characteristics, such as inherent complexity, flexibility requirements, execution in distributed environments, and the heterogeneity of hardware and operating systems [1]. Modularity, in low complexity systems, leads to greater robustness because one can rigorously validate the individual performance of modules prior to system assembly. However, as complexity of the composed system increases, such as robotic systems, modularity might actually decrease robustness because of the increased likelihood of adverse module interactions.

Our approach to the aforementioned robustness dilemma is the augmentation of conventional layered architectures (Fig. 1a) [2] [3] with new self-adaptation constructs at both the component and task layers that facilitate self-adaptation. These self-adaptation constructs form the basis of easy-to-design dynamically reconfigurable systems that are more agile and autonomic in the presence of uncertainty and performance degradation. We termed this infrastructure as **ReFrESH** (*Re*configuration *Fr*amework for distributed *E*mbedded systems for *S*oftware and *H*ardware), as shown in Fig. 1b. The details of ReFrESH were demonstrated in our previous publication to IROS2014 [4]. Although ReFrESH provides a new tool to monitor, diagnose and maintain performance from a module perspective, it lacked a standard module design framework that could guide robotic application engineers to conveniently integrate ReFrESH and functional modules. Therefore, in this paper we develop a real-time software module framework for assembling modules together to comprise a self-adaptive robotic system.



(a) Conventional layered architecture for building robotic systems.



(b) ReFrESH layered architecture for easy-to-design self-adaptive robotic systems.

Fig. 1: Comparison of conventional layered architecture and ReFrESH layered architecture.

The Port-Based Object (PBO) is a software module abstraction for designing dynamically reconfigurable real-time software [5]. Combining the PBO and Real-Time Operating

Yanzhe Cui is with the College of Engineering, Purdue University, West Lafayette, IN, 47907 USA cui56@purdue.edu
Joshua T. Lane are with the College of Technology, Purdue University, West Lafayette, IN, 47907 USA lane54@purdue.edu
Richard M. Voyles are with the College of Technology, Purdue University, West Lafayette, IN, 47907 USA rvoyles@purdue.edu

System (RTOS) mechanisms formed a software framework to design and implement real-time robotic systems [6]. However, when faults emerge in a system that cause immense performance degradation, the PBO abstraction alone does not provide any explicit interfaces to support detection of the faults or maintain the system dynamically. Therefore, to aid implementation of a self-adaptive robotic system, an Extended PBO (E-PBO) module design framework is proposed. This framework provides explicit interfaces or tools to evaluate the performance of the running modules themselves so as to monitor the system as a whole as well as estimate the performance of modules in a new non-running configuration without halting the running system. To sum up, the design targets of E-PBO are to:

1) extend the familiar layered execution model to include specific mechanisms for self-evaluation of the performance of execution units "in-vivo" at the component layer;
2) extend the familiar layered execution model to include specific mechanisms for estimation of the expected performance of execution units prior to execution "in-vitro" at the component layer;
3) extend the familiar layered execution model to include flexible mechanisms for deciding when performance has degraded and which hypothesized configurations are likely to exhibit improved performance at the task layer.

Section II demonstrates the E-PBO design from framework perspective. Section III introduces the implementation of E-PBO in PBO/RT operating system and provides an E-PBO template to guide robotic application designers in the use of E-PBO. Section IV demonstrates a case study on building a visual servoing task using E-PBOs with PBO/RT and ReFrESH. Finally, we summarize this paper in Section V.

II. Framework View of The Extended Port-Based Object

The Port-Based Object (PBO) is a module design abstraction which consists of an independent concurrent process whose functionality is defined by the methods of a standard object, as well as the ports: (1) using port-automation theory, one module's connection (communication) to other modules is restricted to its *variable input ports* and *variable output ports*; (2) the *configuration constants ports* are used to reconfigure generic components for use with specific hardware or applications; (3) the *resource ports* are for communication with sensors and actuators via peripheral drivers [5]. The Extended Port-Based Object (E-PBO) is modeled after the PBO in order to adapt it to the self-adaptive robotic system design.

A. Extended Port-Based Object

As shown in Fig. 2, to create the E-PBO the algebraic model of the port automation, which is similar to PBO is applied. Generally speaking, E-PBO is composed of two parts. One part is conventional **PBO Executor (EX)**, which defines the functionality and provides ports to communicate with ohter executing E-PBOs, reconfigure the module constants, or connect to sensors/actuators. The other part is the extended part, which consists of the **E-PBO Evaluator (EV)** and the **E-PBO Estimator (ES)**.

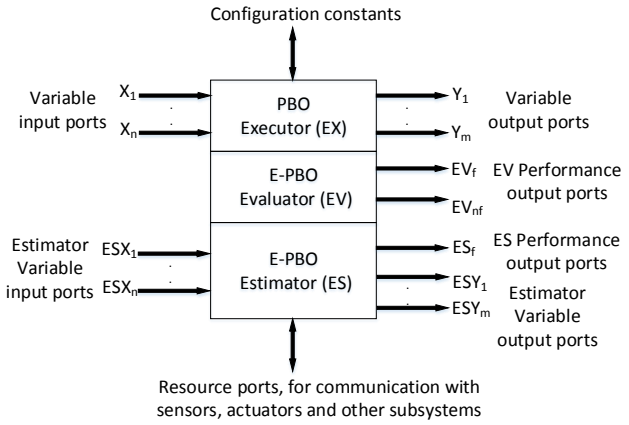


Fig. 2: Abstract view of an E-PBO. Subscript f denotes functional performance and nf denotes non-functional performance.

EV evaluates the performance of an E-PBO and it is triggered only if its corresponding E-PBO is executing. EV does not need not to take input(s) from other E-PBOs since it only evaluates performance based upon the variable output ports of its E-PBO. Thus, EV only includes *evaluator performance output ports*, which are needed to supply the functional performance (such as target detection accumulated error) and non-functional performance (required power usage versus system power capability). Furthermore, EV does not communicate with other E-PBOs, but instead connects to a system management unit, such as the *Decider* in ReFrESH [4]. The outputs of EV provide the evidence for self-adaptation to the system to determine the performance of a task configuration is satisfactory or not.

ES estimates the performance of an E-PBO and it can be used by both running and non-running E-PBOs. It is triggered only if the current running task configuration could not satisfy the performance requirement and a new task configuration is generated by a configuration generation unit, such as the *Generator* in ReFrESH [4]. ES includes the same functionality of its corresponding E-PBO as well as standalone sets of ports, *estimator variable input ports* and *estimator variable output ports*, which connect and communicate with other ES's, and *estimator performance output ports* that provide the functional performance values to a system management unit, such as the *Decider* in ReFrESH. The reason there is no non-functional performance output from the ES we already know the configuration candidates generated from the *Generator* satisfy the system non-functional requirement. After connecting all of the ES in a potential configuration, the management unit combines all the performance values from each ES to determine if the new task configuration is suitable or not. The ES of each E-

PBO provides the recommendation for self-adaptation so that it can determine which new configuration should be used in the presence of faults in place of the current configuration.

A sample library of E-PBOs for a robotic visual servoing application is shown in Fig. 3, which is a subset of the E-PBOs that were created in our laboratory at Purdue. During implementation, the information of an E-PBO, such as non-functional requirement (power, communication signal strength), functional requirement (target detection error), module name and port names, etc., are all specified in its own local structure.

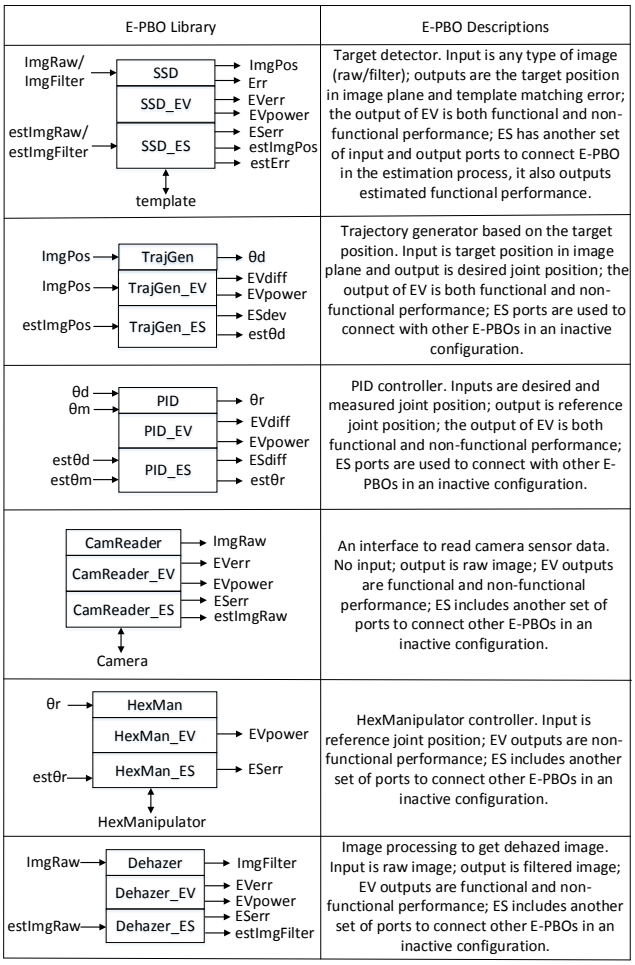


Fig. 3: Example library of E-PBOs for the robotic visual servoing application.

B. Configuration

In this paper, we define a task as consisting of a configuration and a finite state machine (FSM). Specifically, a configuration is a set of E-PBOs [1] interconnected together to provide the required system behavior and a FSM to control the transition states of these E-PBOs. Furthermore, based upon the port-automation theory, we define that a configuration is valid only if the data required by the input(s) of one E-PBO is produced by the output(s) of one and only one other E-PBO. In other words, if more than one E-PBO could supply the required data to the input(s) of an E-PBO, this configuration is invalid.

There are multiple approaches for interconnecting modules, such as those based on the data type of the port, based on the port name, or using shared memory. Since a port based connection approach is more flexible than shared memory when considering a distributed multi-robot application, and also because the port name matching method is more explicit and more convenient to implement, E-PBO uses port names to bind input and output ports. If two E-PBOs exist with matching input and output ports in a configuration, a communication link is created between them.

Fig. 5 illustrates a configuration that implements a visual servoing task where all of the modules used are from the sample library in Fig. 3. For this example, we assume that we are given only the exact E-PBOs required by the configuration and we need only connect them together. In regards to constructing a configuration automatically from a set of random E-PBOs, please check our previous publication [4]. Since the output of the CamReader module is matched with the input of the SSD module, there is a data flow link between them. The input port name of the TrajGen module is "ImgPos", so it will query all E-PBO outputs and then binds with the output of SSD whose port name is "ImgPos" as well.

III. Extended Port-Based Object Implementation

In this section, we show how we have implemented the E-PBO framework as a part of the Port-Based Object Real Time Operating Systems (PBO/RT). This allows robotic application designers to easily implement individual software modules using the E-PBO model.

A. PBO/RT

The PBO/RT API is modeled after the Chimera PBO interface for subsystems servers (SBS). In PBO/RT, processing data from input to output is the result of a specific event. The event is nominally a periodic, time-based trigger, but can also be an aperiodic trigger, such as an interrupt. Associated with each E-PBO is a set of methods that the RTOS calls to control real-time behavior. These methods include initialization, real-time re-initialization, starting, stopping, getting internal parameters, setting internal parameters, evaluating the module and estimating the module. PBO/RT is able to interact with each E-PBO through a local data structure for internal state, as shown in the last part of Section II-A.

Port automata can represent a broad class of arbitrary processes, so we will restrict our attention to tasks based upon them. The processes covered include all types of filters, real-time control algorithms, sampled data systems, and any function that computes outputs based on inputs and internal state at discrete intervals. Given the uniform structure and encapsulated nature of E-PBOs implemented in the PBO/RT operating system, it was only necessary to add functionality for downloading code dynamically, linking and

[1] All E-PBOs are currently implemented using C programming language.

locating that code dynamically, and providing hooks to the internal OS thread structures. These mechanisms guarantee software module self-adaptation. For details of software self-adaptation please refer to our previous publication [7].

B. Coding E-PBO

The E-PBO model is well structured so that the application designer can create a module simply without concern for any details of creating a real-time process. In this section, an E-PBO design template in PBO/RT is provided [2]. By using this template, all the user needs to do is filling in the user code (bold font part) to have the desired functionality.

The encapsulated information of the E-PBO is stored in the local structure called "module_localT". Part of this structure is required by PBO/RT to facilitate communication between nodes and connection to the management unit for evaluation and estimation, but the remainder of the structure is user definable. It can contain simple data types or specific structures, such as a definition for the functional performance output of an ES. The flexibility of the local structure allows a robotic application designer to include any data required to realize the desired functionality of the E-PBO.

The structure involved in creating an E-PBO provides a strict module design method for robotic application designers, guiding them exactly where to put what code. This decreases the amount of guesswork, reduces the amount of code they must write, and through PBO/RT, guarantees that synchronization and communication work from the beginning.

IV. Case Study

In this section, a visual servoing task is demonstrated to show (1) the construction of an application using the E-PBO module design framework; (2) the support of dynamic self-adaptation through E-PBO in ReFrESH; and (3) the use of E-PBO to easily modify the system configuration.

A. Hardware Platform

The robotic platform used in this case study is the *HexManipulator* that was built in our laboratory [8]. It is a form of Stewart-Gough platform [9] configured as defined by Uchiyama's HEXA-Parallel-Robot [10] [11]. The HexManipulator consists of six links where each link is a serial combination of a 1 DoF active rotary joint, a 2 DoF passive universal joint, and a 3 DoF passive spherical joint. All the links are connected to the base and traveling plate and are actuated by a total of six Futaba S3003 servo motors. An RGB camera which is capable of outputting 30 frames per second is fixed on the traveling plate as a vision sensor. The RecoNode [6] is selected to run the PBO/RT operating system. The RecoNode is a high performance *Reconfigurable Node*, whose multiprocessor architecture is based on the Xilinx Virtex-4 FPGA with low-power, hardcore PowerPC CPUs and is capable of up to 1600 MIPS which is more powerful than microcontrollers. The hardware platform setting for a visual servoing application is shown in Fig. 4.

[2]https://github.com/cuiyanzhe/ReFrESH/blob/master/epboTemplate.pdf

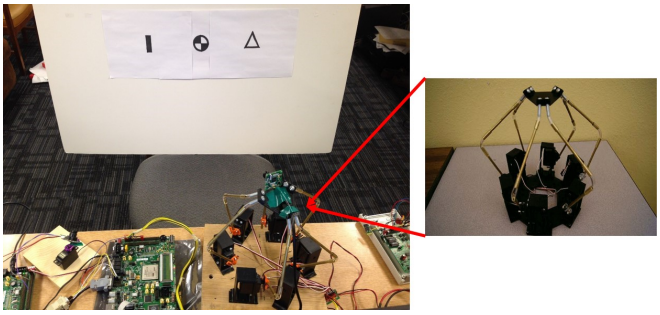


Fig. 4: The hardware platform setting for a visual servoing application.

B. Feasibility and Convenience of Using E-PBO to Build a Self-Adaptive Robotic System

1) Compose a System Using E-PBO: The visual servoing task in this case study aims to enable the HexManipulator platform to autonomously detect three targets cyclically and move from its current position towards a goal position with a specific angle to aim at the chosen target. This involves several functionality such as target detection (*SSD* E-PBO), trajectory generation (*trajGen* E-PBO), control of the physical HexManipulator platform (*HexMan* E-PBO), and runtime switching of the target template (using resource port "template" of SSD E-PBO).

Fig. 5 shows the initial system configuration that uses E-PBOs within ReFrESH. We use the same color code as Fig. 1b. ReFrESH contains the task finite state machine (FSM) and PBO EX in the conventional layered architecture. The E-PBO EV outputs provide evidence to the "Decider" (a part of the Management Unit) of ReFrESH to evaluate the task performance dynamically. The EV outputs consist of functional performance and non-functional performance, which are connected to *Func Performance Buffer* (dash line in Fig. 5) and *Non-Func Performance Buffer* (dash-dot line in Fig. 5) respectively to allow the "Decider" to easily monitor the system performance and distinguish the cause of performance degradation if a fault emerges in the system. For example, if a dying power supply causes the performance to degrade, the system configuration should be correct so the management unit only needs to generate task configuration candidates for module re-deployment (migrate some modules or sub-tasks to another robot which has more power). However, if the non-functional performance is acceptable, this means that the system configuration cannot meet the requirement of the current situation so the management unit should find a new system configuration.

2) Support of Self-Adaptation Using E-PBO in ReFrESH: After all E-PBOs in Fig. 5 have been connected, the system is ready to run. Simultaneously, the management unit of ReFrESH starts to monitor the task performance automatically by calculating both functional performance parameters and non-functional performance parameters. Since the functional performance of one module causes a cascading effect on the following modules, the product of the functional performance of each E-PBO reflects the whole system

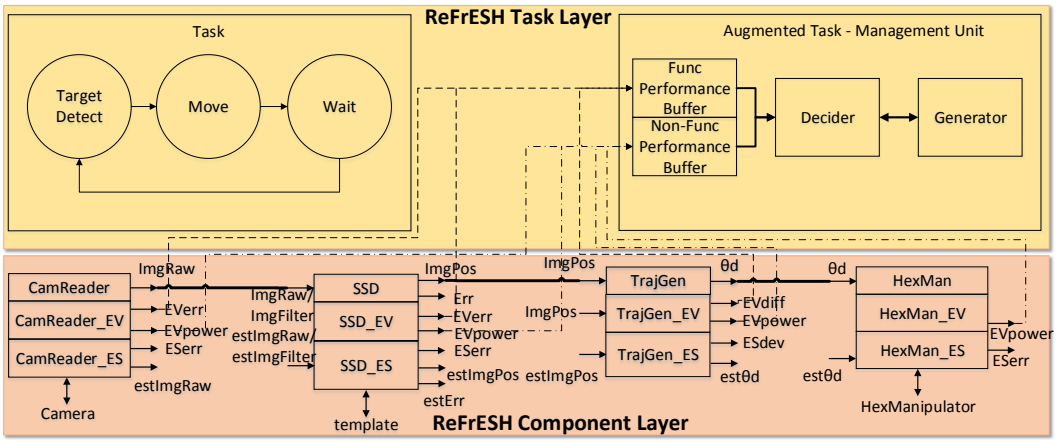


Fig. 5: The initial system built by E-PBOs.

configuration performance. Thus, Equation 1 is utilized in the "Decider" to evaluate the functional performance of the system configuration, where M denotes the number of E-PBOs in a system configuration and $P_{Func\_EPBO(i)}$ is the functional performance of the i_{th} E-PBO. Then the resulting P_F should be binarized to easily determine if the total system performance is acceptable by using Equation 2, where P_{F_Th} is the system performance threshold.

$$P_F = \prod_{i=1}^{M} P_{Func\_EPBO(i)} \tag{1}$$

$$P_F = P_F > P_{F\_Th} ? 1 : 0 \tag{2}$$

Unlike functional performance, the non-functional performance of one module affects the whole system directly. For example, a communication link failure of two E-PBOs that are located on two different robots causes the whole system to fail, not only a cascading affect on the modules. Equation 3 is utilized in the "Decider" to evaluate the non-functional performance of the system configuration, where M is the number of E-PBOs, j is the j_{th} non-functional characteristic (power, RSSI etc.), $P_{NF(j)\_EPBO(i)}$ denotes the j_{th} non-functional characteristic of the i_{th} E-PBO, $P_{NF(j)\_EPBO(i)\_Th}$ is the threshold setting of the required non-functional resource j of the i_{th} E-PBO. The result of Equation 3 is a binary value specifying if the j_{th} non-functional performance is acceptable or not.

$$P_{NF(j)} = \prod_{i=1}^{M} P_{NF(j)\_EPBO(i)} < P_{NF(j)\_EPBO(i)\_Th} ? 1 : 0 \tag{3}$$

Through Equation 4, the "Decider" can determine if the performance of the system configuration is acceptable or not. If $P = 0$, there are faults in the system configuration and the "Decider" can track back to the binary results of P_F and P_{NF} based on Equation 2 and Equation 3 to distinguish the functional or non-functional characteristic causing the performance degradation.

$$P = P_F \bigcap \prod_{j=1}^{N} P_{NF(j)} \tag{4}$$

For example, in this case study, to test self-adaptation support of E-PBOs within ReFrESH, we deliberately injected error into the system by adding noise into the *CamReader* module and we only consider power as the non-functional characteristic. As shown in Fig. 5, the "EVpower" output of each E-PBO satisfies the current situation, so $P_{NF} = 1$. However, due to the increased error of the SSD E-PBO, the output of "EVerr" causes $P_F = 0$. Therefore, based on Equation 4, the "Decider" detects a functional error in the current system configuration.

Once the error has been detected, the "Generator" in the "Management Unit" of the augmented task part should be called to generate the new feasible system configuration candidates. In this case study, one of the feasible candidates swaps in a *Dehazer* E-PBO. The "Decider" would call each E-PBO ES one time to connect the estimator variable inputs/outputs (red line in Fig. 6) and accumulate all the estimated performance values (red dash line in Fig. 6). The separation of PBO EX variable input/output ports and E-PBO ES variable input/output ports supports running the estimation process without interfering with the execution of the current system configuration. The "Decider" will calculate the candidate performance based on Equations 1 to 4 to estimate if the candidate can meet the task performance requirement. If the estimated performance of a candidate is better than the current running configuration, the ReFrESH will call self-adaptation approaches and run a new configuration; otherwise, the next candidate will be estimated.

After a new feasible system configuration was generated by ReFrESH, the "Management Unit" would provide the self-adaptation methods to reconfigure the system configura-

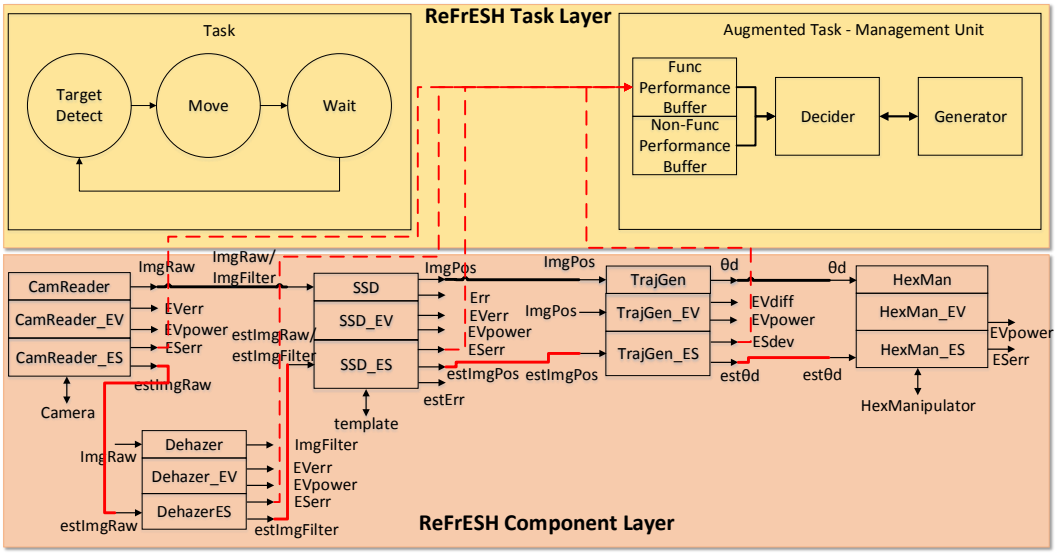


Fig. 6: The running task configuration and the estimation process for a new non-running task configuration composed by E-PBOs. We use red wires to distinguish current running configuration and non-running configuration that needs estimate.

tion [3]. It is then necessary to start the *Dehazer* E-PBO into the system between the "camReader" and "SSD" E-PBOs without modifying any other modules.

V. Conclusion

This paper demonstrates a module design framework, termed E-PBO. E-PBO defines both functional and non-functional characteristics of a module and, based upon port-automation theory, it creates a port binding mechanism. The Evaluator and Estimator parts of the E-PBO provide support for dynamic monitoring of the current running task configuration and dynamic estimation of the new non-running task configuration without interfering with the running configuration. Furthermore, by implementing it in PBO/RT operating system, E-PBO guarantees the real-time performance of robotic systems. Ultimately, E-PBO extends the conventional layered execution model to include self-evaluation mechanism for execution units at the component layer, to include system configuration estimation mechanism for execution units prior to execution at the component layer, and to include flexible mechanisms for deciding when performance has degraded and which hypothesized configurations are likely to exhibit improved performance at the task layer. The case study on building a visual servoing task shows the feasibility of the E-PBO to support build self-adaptive robotic systems in ReFrESH.

Acknowledgment

This work was supported in part by National Science Foundation Grants CNS-0923518, CNS-1450342 and IIS-1111568 with additional support from the NSF Center for Robots and Sensors for the Human Well-Being (RoSe-HUB).

[3]For details of the software and hardware reconfiguration method please check [7] [12].

References

[1] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, "Towards component-based robotics," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, Aug 2005, pp. 163–168.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[3] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand, "Genom3: Building middleware-independent robotic components," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, May 2010, pp. 4627–4632.

[4] Y. Cui, R. Voyles, J. Lane, and M. Mahoor, "Refresh: A self-adaptation framework to support fault tolerance in field mobile robots," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sept 2014, pp. 1576–1582.

[5] D. Stewart, R. Volpe, and P. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," *Software Engineering, IEEE Transactions on*, vol. 23, no. 12, pp. 759 –776, dec 1997.

[6] R. Voyles, S. Povilus, R. Mangharam, and K. Li, "Reconode: A reconfigurable node for heterogeneous multi-robot search and rescue," in *Safety Security and Rescue Robotics (SSRR), 2010 IEEE International Workshop on*, july 2010, pp. 1 –7.

[7] Y. Cui, R. Voyles, M. He, G. Jiang, and M. Mahoor, "A self-adaptation framework for resource constrained miniature search and rescue robots," in *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, Nov 2012, pp. 1–6.

[8] G. Jiang and R. Voyles, "Hexrotor uav platform enabling dextrous interaction with structures-flight test," in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, Oct 2013, pp. 1–6.

[9] B. Dasgupta and T. Mruthyunjaya, "The stewart platform manipulator: a review," *Mechanism and Machine Theory*, vol. 35, no. 1, pp. 15 – 40, 2000.

[10] F. Pierrot, M. Uchiyama, D. P., and F. A., "A new design of a 6-dof parallel robot," *Journal of Robotics and Mechatronics*, vol. 2, no. 4, pp. 308 –315, 1990.

[11] P. Last, C. Budde, and J. Hesselbach, "Self-calibration of the hexa-parallel-structure," in *Automation Science and Engineering, 2005. IEEE International Conference on*, Aug 2005, pp. 393–398.

[12] M. He, Y. Cui, M. Mahoor, and R. Voyles, "A heterogeneous modules interconnection architecture for fpga-based partial dynamic reconfiguration," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, July 2012, pp. 1–7.