Comparative Evaluation of Robotic Software Integration Systems: A Case Study

Azamat Shakhimardanov
azamat.shakhimardanov@smail.inf.fh-brs.de

Erwin Prassler
erwin.prassler@fh-brs.de

Abstract—One might expect that after nearly 50 years of research in robot manipulation and mobile robotics the architectural design of robot systems has converged to a small set of best practice solutions, very much like in the area of operating systems, where the world is dominated by less than a handful systems. Quite the opposite is the case. It is only a small overstatement to say that almost every lab has brewed its own solution for robot control architecture, middleware and software integration concepts. One reason for this situation is the fact that the complexity and variety of systems and applications has grown considerably since then leading to a diversity of approaches. Another reason is clearly the lack of any sound methodology to measure and compare architectural designs and well defined standards.

This article tries to address the problem by proposing a comparison and evaluation methodology for robotics software systems and gives a view how it relates to standardization in robotics. The methodology facilitates an assessment of software systems through experimentation and takes into account the possible operational profile and quality attribute requirements of the robotic application.

The approach is validated through the evaluation of three software systems, i.e. GenoM, ORCA2 and GO. The results suggest possible applicability domains of each system and show the efficiency of the approach.

I. INTRODUCTION

One might expect that after nearly 50 years of research in robot manipulation and mobile robotics the architectural design of robot systems has converged to a small set of best practice solutions, very much like in the area of operating systems, where the world is dominated by less than a handful systems. Quite the opposite is the case. It is only a small overstatement to say that almost every lab has brewed its own solution for robot control architecture, middleware and software integration concepts. One reason for this situation is clearly the lack of any sound methodology to measure and compare architectural designs and well defined standards.

Whenever a new robotic system needs to be developed for whatever reason its designer is left with the questions “What architectural design fits best my needs and requirements?” Without a sound evaluation of the approaches which have been developed in the past it is tempting to develop yet another one. Very often this leads to a re-invention of the wheel and a significant waste of effort.

What is required to escape from this situation is establishing a culture and methodology of measuring and evaluating all aspects of architectural designs: communication and middleware aspects, software engineering aspects, planning, reasoning and control aspects, and last but certainly not least dependability aspects.

The following section briefly summarizes some earlier work in benchmarking and evaluation in robotics. It is then followed by analysis of software architecture evaluation methodologies established in software engineering. In the Section 3, we describe key points of software systems used in the experiments, with respective hardware and application setup. Section 4 summarizes our approach to evaluation of software systems in robotics. This is supported by the results in Section 5. Section 6 positions this evaluation methodology in possible standard defining activity and provides comparison with computer OS domain. Section 7 concludes the paper and gives an outlook to the future research.

II. STATE OF THE ART

A. Benchmarking and Evaluation in Robotics

Most of the research initiatives in robotics in the direction of benchmarking are related to the assessment of the algorithms applied, i.e. for vision, localization, planning, mapping etc. Some examples of such initiatives are presented in [1], where the authors attempt to establish a new taxonomy for benchmarking in robotics. They distinguish between two categories:

1) benchmarking by proof which they refer to as analytical benchmarking.

2) benchmarking by experimentation which they refer to as functional benchmarking.

Each of these categories is further related with the system level and the component level, thus producing four types of benchmarks which are mostly concerned with the hardware components and algorithms of a robotic system. Similar aspects are considered in [2] where authors present benchmarks for path and trajectory following, static planning and dynamic planning.

Other comparable efforts which are concerned with evaluating the performance of navigation algorithms are presented in [3] and [4]. In the first paper, the authors establish a set of trial runs for a museum guide robot to test the “reliability” and “robustness” (as defined by authors) of the “NaviGates” navigation system [3]. In the second paper, authors assess the performance of four functional components of the Mars rover navigation system, i.e. goal designation, rover localization, hazard detection and path selection [4].

Unlike the mentioned research, [5] describes issues in assessing the performance of social robots. The authors...
state that approaches for performance measurement in autonomous mobile robotics can generally be categorized as:

- theoretical predictions of performance
- computer simulations
- real-world experiments

Their approach mostly emphasizes anthropomorphic characteristics of robots.

Unfortunately, in most of the research presented, there is no clear common view on what benchmarking is and how one should evaluate a system against a particular benchmark. This led to the creation of a considerable corpus of terminology related to the topic which is often used with different meanings by the community.

In the domain of robotic software system evaluation, one can list such initiatives as [6], [7], [8], [9], [10], [11], [12]. In [6] authors primarily aim to identify common characteristics of software integration systems (SIS) and come up with suggestions for designing new and better robotic software systems. They present the results of a comparative evaluation of three SIS - Saphira, TeamBots and ISR Berra. The evaluation is carried on with respect to seven categories as:

- robot hardware abstraction
- extendibility and scalability
- limited run-time overhead
- actuator control model
- software characteristics
- tools and methods
- the level of documentation

The similar approach to evaluation is presented in [7]. Here, the authors propose a conceptual evaluation framework based on four categories of criteria:

- specification, which includes formalisms, methodologies, and design tools
- platform support, which is related to the hardware and its low-level interface (e.g., the operating system)
- infrastructure, which refers to components and capabilities that are part of the software systems
- implementation, which includes aspects of application development

The authors also emphasize that these conceptual framework is not enough to provide complete system evaluation. Therefore, they introduce two additional factors which influence this process:

- system’s practical usability, which includes architectural design, implementation and execution with an emphasis on ease of use and performance.
- system impact on the field of robotics, which considers the amount of published work on the system.

Other initiatives, [8], [9], [10], [11], [12], also suggest methods to evaluate robotic software systems, but most of them lack systematic structure and often are not concerned with possible operational profile of a robotic application.

B. Software Architecture Evaluation (SAE)

Unlike benchmarking techniques, which often require presence of at least some system components implemented, SAE methods are usually involved during the design/development phase of a system. This approach usually makes it possible to carefully analyze a system in advance. One can mainly distinguish three evaluation techniques:

1) Questioning techniques - ARID [13], SAAM [13], checklists, scenarios based methods. This category of techniques uses checklists, questionnaires to evaluate a system.

2) Measurement based techniques - Rate Monotonic Analysis, automated tools and Architecture Description Languages(ADL), prototypes, experiments [13]. This category often uses experiments and metrics to evaluate a system.

3) Hybrid techniques - ATAM, SPE, PASA, etc [14], [13]. This category of techniques are considered more efficient with respect to the other two because they are often based on both measuring and questioning techniques.

We will constrain our discussion on SAE to ATAM approach which serves as the basis for our evaluation framework presented in section 4. For further details on SAE approaches and their comparison we confer to [15], [16], and [17].

Attribute Tradeoff Analysis Method(ATAM) [13] is an architecture evaluation method the techniques and concepts of which emphasize the notion of quality attributes. Its main aim is not only to reveal whether adopted architectural styles do meet the quality requirements but also to identify relations between different quality attributes of one system and point out necessary tradeoffs in order to achieve optimum results.

One of the main tasks during the evaluation process in ATAM is the generation of scenarios. This is performed by constructing a quality attribute utility tree. The main purpose in generating the utility tree and scenarios is to determine the priorities of the systems quality attributes so that later on this can be taken into account while identifying sensitivity and tradeoff points. The utility tree makes it possible to directly translate system attributes/business drives into specific scenarios.

Though ATAM is the efficient evaluation framework, it is too general to be directly applied to robotic software evaluation. One needs to adapt it to the constraints of robotic software systems and support it with quantitative evaluation data from benchmarking experiments. The section 4 presents the framework which has these features.

III. SETUP FOR THE EVALUATION

A. Software systems explored

1) GenoM:

- General Architectural Concepts - Genom is developed by the RIA group, LAAS CNRS, France. It is a part of the LAAS robotic software architecture where it serves as an implementation of a functional layer. It supports the generation of software components and comes with

---

1SAE can also be applied to already existent systems, e.g. legacy systems which need to be improved.
a set of utilities, i.e. communication libraries, application server, and build tools, which allow to construct robotic software applications [18]. In GenoM, functional layer abstractions are represented in the form of software components called “module”s [figure 1], [18]. To implement a robot specific application, these modules are combined into a network of communicating nodes which may use services provided by each other [figure 6]. The functionality useful to the client of a module comes from its services, i.e. the requests it provides which can be accessed through a requests/replies library interface. The running services are referred to as activities. There can be several activities running simultaneously. To be able to monitor and manage the execution of the requests the GenoM module organizes code in the form of “codels”. The codel is the smallest entity that the module can handle and thus the codel can be considered as atomic or to finish within a known small amount of time. They are implemented as standard C functions which encapsulate the user code.

- **Communication Mechanisms** - there are two methods to establish inter-module communication:
  - RPC based approach - this method relies on a set of methods defined in posterLib and csLib libraries to write/read data in posters and perform messaging between modules.
  - TCL script based approach - this method, though it may lose in performance to the former one, tends to be simpler and more scalable. It uses an application server to communicate with modules and the TCL interpreter.

2) **ORCA2**:
- **General Architectural Concepts** - ORCA2 is developed at the department of Field Robotics, University of Sydney, Australia. ORCA2 is a robotics software integration framework [19]. It makes use of concepts of the Component Oriented Software Engineering paradigm and is based on ZeroC’s Internet Communication Engine (ICE) middleware system. At the core of the ORCA2 lies the idea of representing a robotic software application as a network of communicating independent software components [figure 2]. ORCA2 enables a flexible approach to an application development while still directing this process by defining some guidelines one should follow to achieve efficient results, e.g. communication interfaces all components should implement.

- **Communication Mechanisms** - ORCA2 uses ICE’s communication mechanisms. ICE uses both synchronous and asynchronous communication methods, where both server and client side operations have opportunity to process and communicate in both modes. ORCA2 components usually communicate based on publisher-subscriber model using notification service [19], [figure 6].

3) **GO**:
- **General Architectural Concepts** - GO is developed at the department of Robotics, Fraunhofer IPA, Stuttgart Germany. It is completely implemented in Python programming language and is distributed as a Python module. Additionally to the GO module the package includes a set of supportive modules for communication, visualization and application runtime tracing. GO is more suitable as a sequencer, most of the constructs it defines work on activities, which are the methods of the component class. GO defines two types of activities: synchronous activities and asynchronous activities. GO also provides features for state-based monitoring. Though it does not define as strict and elaborate a control path as in GenoM, it still provides useful information to perform necessary cleanup or recovery actions in some situations [20], [figure 6].

- **Communication Mechanisms** - depending on how the application is distributed GO takes on two communication approaches. This also follows from the fact that GO applications run in the Python interpreter: **Python sockets API using TCP** and **Python object references**.

The table in figure 3 summarizes distinguishing features of each system. Figure 6 show structural/interaction view of the sample application implemented on each system.

### B. Hardware and Software Settings

It is usually difficult to perform benchmarking directly on a system of interest. That is why experimental scenarios are often conducted with the help of supporting tools, i.e. system under benchmarking (SUB). SUB are most of the time composed of a hardware platform, an operating system...
and other supportive facilities. In experiments conducted, SUB is composed of:

- **Hardware platform** - IBM Thinkpad X31 laptop with 1.4 GHz Pentium M processor and 512 Mb of RAM, four Schunk power cube modules [figure 4], 24 volt power supply, CAN bus and CAN-to-USB converter.
- **Software platform** - Ubuntu Linux 6.1 with linux kernel 6.2.27, CAN-to-USB drivers, Schunk power cube programming API.

In all experiments, sample applications were organized in terms of client and server components. In particular, applications run on the robotic manipulator were organized as four client and one server components. Here each client represented a component for controlling an arm joint and the server component was responsible for the generation of positions for each joint. The task composed of moving all the joints to the same position [figure 6].

IV. METHODOLOGY

A. Method for Comparative Evaluation

The approach combines several features from the benchmarking and software architecture evaluation methods. By combining them we cover the evaluation process from both practical and theoretical perspectives. Figure 5 provides a simple view of the evaluation method. This procedure can be considered as top-down approach, where one moves from general to more specific system aspects. We identified four stages for this procedure. The output of each stage serves as an input to the proceeding one, thus narrowing the problem to several specific operational situations. For instance, consider that one needs to assess system robustness to various external faults or events (e.g. a robot should not go havoc when a patient at the hospital wants to draw a window curtain which may change light conditions). How can one measure this attribute? It would be difficult to do this directly because there can be several factors influencing the robustness. The option is to identify often arising operational conditions and perform the required measurements with respect to them.

Below we briefly summarize each of these stages for more detailed discussion we confer to [21], [22]:

1) Stage of identification of initial quality attribute requirements for a software system - at this stage the developer is required to list the most important quality attributes. The framework includes the list of commonly questioned non-functional attributes in robotic software applications. As of current state of the research, they are grouped into four classes according to the similarities of issues they address.

- **Performance** - is usually concerned with how long it takes the system to respond to various internal and external events.
- **Fault tolerance and robustness** - is concerned with the ability of a system to face various threats and associated consequences.
- **Usability** - is concerned with how easy it is for a system user to achieve a desired task and what kind of support does the system provide for this purpose.
- **Flexibility, scalability and modularity** - is concerned with how well does a system adapt to external needs, how well does the system scale or how difficult is it to add new features to the system etc.

2) Stage of quality attribute refinements - after having identified a set of quality attributes, one has to elaborate on concepts which are tightly coupled with these attributes. These refinements represent the concepts which influence quality attributes the most. For instance a cycle time of a server component will affect the performance of the application it is used in.
3) **Stage of experimental scenarios** - at this stage one has to devise experimental scenarios which elicit each of the attribute refinements proposed. These are short descriptions of real experiments to be conducted and results of these experiments determine whether a system meets its required quality attributes.

4) **Stage of experimental scenario dissection** - this last stage is concerned with the detailed dissection of experiments based on their short descriptions. We identify seven components to fully describe an experiment:

- **stimuli and their sources** - stimuli source is an entity (actor on a system which triggers something) that generated the stimulus. Stimuli is a condition/event which needs to be taken into account when it arrives at a system.

- **environment/operation mode** - a set of conditions under which the system works. Stimuli are caused by these conditions. It can be a normal mode of operation, a dangerous mode or some other mode of operation.

- **responses and measures** - response is an activity undertaken by the system at the arrival of the stimulus. For a system to be assessed against its requirements the responses it produces should be measured in some manner.

- **benchmarking target** - a software system which is being evaluated in our case it is either one of ORCA2, GenoM or GO.

- **system under benchmarking** - a collection of supportive tools that are used to conduct evaluation experiments.

- **workload** - describes the operational profile of a software system in terms of the loads it undergoes. It is usually the data transmitted between different parts of the system.

- **faultload** - describes a set of stresses (i.e. faults) a software system may undergo in the real world counterpart of the given experimental scenario. This is also directly linked to systems operation mode.

To clarify the procedure, let us consider the following example. **An industrial robot is to handle parts arriving on a conveyor belt at a fixed rate of 50 parts/min. In case of a malfunction or a jam, the conveyor+robot system stops for 3-4 seconds allowing maintenance. Under normal operational conditions a software running on the robot allows it to reposition itself in 400 ms after a part is processed. Considering this application, we would like to check whether the robot+software system meets the requirements specified.**

In this case, one of the quality attributes needs to be verified is performance (from the example we can also verify for the fault tolerance). The aspects/refinements which could directly influence performance are a cycle time of the robot software, a communication bandwidth between different parts of the software, a cycle time of the conveyor belt, etc. Identifying these refinements concludes stage two of the evaluation. In stage three, we devise experiments which assess each of the refinements. For instance, developing a software to interface robot joints, the conveyor belt and the central controller. The software could simulate the normal operational profile of the robot under which the measurements of the communication bandwidth could be conducted. After this, the proposed scenario is decomposed into subparts to structure the process of experimentation. In the example it looks as follows:

- **stimuli/sources** - periodical commands generated by the controller.
- **environment/operation mode** - normal mode
- **responses and measures** - commands are handled by respective components, i.e. arm moves to a specified position and belt delivers parts. A measure is data throughput and it is measured in bytes/sec.
- **benchmarking target** - a software application running on the system, i.e. in our case implementation using either of GenoM, ORCA2 or GO.
- **system under benchmarking** - the robotic arm, the conveyor belt, the controller PC and OS etc.
- **workload** - data from sensors to the controller, commands from the controller to actuators
- **faultload** - none

The following section shows part of the experimental results obtained in two categories of experiments: performance and robustness.

**V. EXPERIMENTAL RESULTS**

All the values presented in this section are averaged over 25-30 experimental runs.

**A. Performance Experiments**

There were three types of experiments conducted in this category. Each of the experiments was targeted at articulating a particular performance refinement. These are
1) **Inter component communication** - in this subclass such quantities as round trip time and data throughput are measured for each system. Figure 13 summarizes this results. As it can be seen these results are presented as intervals of values rather than a unique number. This is related to the fact that often these quantities have fluctuating nature. Among the systems, ORCA2 often preserved stable communication which can be crucial when implementing a safety mechanism which may require timely responses. On the other hand, GenoM system using two different protocols, i.e. shared memory based and TCP based, often performs in big range of values from \(2 \times 10^{-4}\) to \(5 \times 10^{-3}\) seconds.

2) **System execution times** - in this subclass such quantities as task execution time and system response time are measured for each system. Measuring the task execution time allows to infer, to some extent, the delay between software function call and hardware response(to measure the delay directly one will need external reference clock). Here the task is moving a specified joint from one position coordinate to the other with specific speed and acceleration settings. The software system response time is a time delay between a method invocation by a client and the actual start of the invocation on the server. Figures 10 and 11 summarize the results.

3) **System runtime resource consumption** - this subclass of experiments is concerned with such quantities as system runtime memory usage and system processes. Here we have to emphasize that such platform dependent metrics are only comparable and reproducible when conducted under same environmental conditions and on the same platform, which is the case in these experiments. Figure 12 summarizes these results.

**B. Robustness Experiments**

In this category of experimental scenarios, we are concerned with runtime fault handling capabilities and system behavior in the presence of faults. At the current stage we have identified the following refinements for this quality attribute:

1) **System behavior in the presence of hardware faults** - In this experiment hardware fault was simulated by disconnecting the hardware component while the application was still running. In the experiment, the whole arm was disabled by unplugging and plugging the USB-to-CAN connection for one second. This was meant to test the so called OS-USB stick effect, where OS would recognize plugging and unplugging of the stick. Figure 14 summarizes the results.

2) **System behavior in the presence of software faults** - Here software fault is simulated by forcefully stopping one of the running components.

We also conducted experiments to evaluate the other two quality attribute categories, i.e. usability and flexibility, scalability and modularity, for the detailed information on the results we confer reader to [21], [22].

The analyses show that among the systems considered the GO package is the best in the performance domain, ease of use, learning curve, portability and is handy for the quick and simple solutions. But GO is meant to be a sequencer rather than a functional layer software. That is why, as applications grow, it may eventually not meet requirements for the functional layer software. This is in particular true when concerned with fault handling mechanisms, resource handling, scalability and flexibility aspects. In such cases, GenoM and ORCA2 seem to be more appropriate choices. But there are some notable differences between these two systems, as well. Specifically, ORCA2 has a more stable inter-component communication. This can be a crucial point when a robotic system requires steady timely communication rather than being fast and variable as it is in case of GenoM. But on the other hand, on a system with limited hardware resources and higher safety requirements GenoM will make a better use. GenoM has got several mechanisms to handle resources used and to take necessary actions in the presence of dangerous failures. It should be emphasized, though, that dependability, especially safety, is very dependent on stable communication mechanisms. Another aspect which can be important while choosing a software system is the availability of ready to use code base/components. In that case, ORCA2 is the favorite system [21], [22].

VI. **Robotics Standards - Evaluation Methodology**

A Standard, as defined in [23] is something established by authority, custom, or general consent as a model or example, it can also be viewed as a rule for the measure of quantity, weight, extent, value, or quality. The standardization is the process of complying/developing a standard to achieve the compatibility, interchangeability and interoperability of different entities in a particular domain. For computer hardware domain these entities can be hardware interfaces (USB, PCI etc), whereas for software it can be APIs, data formats and component interfaces (POSIX, JAXB) [24]. The development of standards allows to constrain and direct the technological progress. But for a standard to be efficient it should not only be adopted by a group of people but also meet requirements of most users. As indicated in [25] some of such requirements are that it

- should cover all(most) entities that are present in the domain considered
- creates no ambiguities in interpretation of represented information
- allows efficient implementation for all(most) scenarios in the domain considered

But meeting these requirements is often difficult, in particular in robotics domain where entities are of different nature (hardware and software). One solution is to divide the domain into subdomains as it is done in [25] and define the ways these subdomains map representations between each other. A layered structuring can be one option where the semantics of entities in particular level becomes more abstract as one moves from the lowest to the highest level.
For instance on native hardware level one will be talking about bit order, float, int and how they should be represented, whereas on ontology level the entities will be a robot, its position and orientation. A similar approach can be observed in operating system (OS) domain [26], where these layers are Digital logic level, Microarchitecture level, Instruction set architecture level, OS level and so on till Problem oriented language level. In this situation, one also moves from specific to more abstract entities. But unlike situation in robotics there have already been developed many different standards for each of these layers (RISC, CISC, POSIX, OSI, HDF5 etc.). Another aspect of OS domain similar to that in robotics is computer/OS - environment - task [fig. 7] triangle (but unlike in OS field, in robotics environment plays a more active role). Therefore one actually can infer that the four layers [fig. 8], [25] are vertical to [fig. 7] and that each dimensions in [fig. 9] may have their own standards based on [fig. 8].

So where do evaluation and comparison methods come into play in [fig. 9] and why one would need them? Again we come back OS field for assistance. One can count more than ten types of Linux or FreeBSD distributions. Each distribution can have different kernel implementations, utilities, applications and target deployment domains. How does one choose the right distribution for the desktop, server, watch or even airplane among the many available? Surely, one will need/devise a set of system requirements to be met and use this list as the criterion for evaluation and to make a final choice. This evaluation can be something as simple as comparison of type of licenses or as sophisticated as measuring speed of interprocess communication. The same situation is currently observed in the field robotics software. But this does not stop here. Imagine airplane company manufactures a new type of plane and requirements for it go up and company decides to go for a new type of OS but wants to port the software from the old plane because most of the hardware is the same/similar or because it is simply too costly to do everything anew. To be able to port software from one system to the other two should have sth in common, for instance driver or application programming API. To identify this commonalities and discrepancies, the company will have to compare/evaluate both systems.

The point we want to make here is that evaluation/comparison do help to identify points of interest in a system which may need to be standardized. These points of interest can be a set of quality and functional requirements, APIs, data formats, models etc. To conclude, one can say that the evaluation and comparison are the first steps towards standards.

VII. CONCLUSIONS AND FUTURE WORK

This paper provided preliminary results on a practical and conceptual framework for robotic software system evaluation. The work also provides some arguments on why robotics needs evaluation methodologies and the relation to future standards. The efficiency of the approach has been validated on the evaluation of three software systems, GenoM, ORCA2 and GO. The methodology shows the basic concepts behind the evaluation process, the method itself and allows a comparison of the attributes of each system. Currently, we are working on further improvements of the framework to make its results even more fair and elaborate. Additionally, we plan to broaden the applicability of the framework to the evaluation of different algorithms for robotics and provide an automated evaluation procedure.

VIII. ACKNOWLEDGMENTS

We express our gratitude to the staff at the Department of Robotics at Fraunhofer Institute for Manufacturing Engineering and Automation IPA Stuttgart Germany, Master Autonomous Systems at the University of Applied Sciences Bonn-Rhein-Sieg, Computer Science department at the University of Lund, Sweden and Mechanical Engineering department at the Catholic University of Leuven, Belgium.
REFERENCES

[23] “Meriam-webster dictionary http://www.m-w.com/.”