
Design toolset for realising robotic systems

Yiannis Gatsoulis¹, Ioannis Chochlidakis², and Gurvinder S. Virk³

Intelligent Systems Group, School of Mechanical Engineering, University of Leeds,
Woodhouse Lane, Leeds, West Yorkshire, U.K. LS2 9JT

(¹menig, ²men2ic, ³g.s.virk)@leeds.ac.uk

Summary. This paper describes the important criteria for selecting design software tools for realising specific robotic solutions quickly and efficiently. Possible quantitative and qualitative methods for the proposed selection framework are discussed. A case study on formulating a design toolset for the research and development of search and rescue robotic systems is presented.

1 Introduction

Research and development (R&D) of robotic systems is a complicated process and the produced systems are usually expensive. Robotic systems designers prefer to use a modular approach in order to reduce the R&D and maintenance costs as well as make their systems more flexible. However, these modules are task oriented and restrained within the scope of the particular platform. This kind of modularity is called *in-house modularity*. As a result research teams are wasting considerable amounts of time and resources in developing their own modules.

On the other hand if standard components were available in the market, developers would be able to build their prototypes faster as there will be a reduced need for individual development of all the components. Furthermore, the whole process is more cost effective due to mass production and reuse of the modules and the technologies for a wide range of applications. This kind of modularity is called *open modularity* [15; 16].

Figure 1 shows an approach to open modular R&D where the overall problem is broken down into different stages. Firstly, potential application areas and the operational environments should be identified. A generic list of task requirements is then formed for each. The general capabilities required by the robot are then formed, followed by a design of the specific capabilities. All components are grouped under four generic categories, namely input, processing, output and infrastructure modules. Super modules can be constructed from basic ones. All modules (basic and super) integrate with each other using

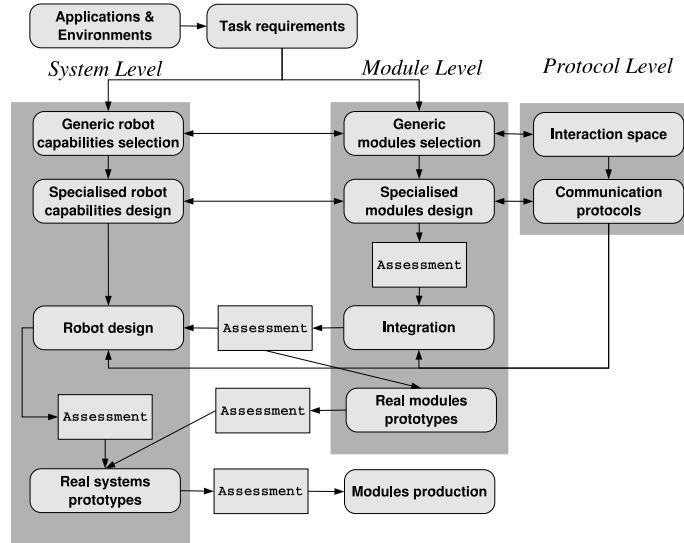


Fig. 1. Open modularity process

the interaction space as defined by the CLAWAR community [14]; this defines six ways in which interactions can take place, namely mechanics, power, digital, analogue, databus and environment. This methodology was inspired by Brooks' subsumption architecture [2] with emphasis given to the modular level [4]. Normally we would expect system level and module level designs to be carried out in a parallel way as shown in Figure 1. Standard protocols and software tools must be developed for that. Assessments must be conducted at different levels of the R&D before any real components are built and the modules are finalised for mass production. This is achieved by extensive testing and refining of the solutions until all aspects are achieved.

However, the increasing use of design tools to assist the developers has resulted in a vast number of software packages. The selection of an appropriate tool is a difficult and time consuming task, especially if all the available packages have to be investigated to find the best one to use. This is the reason for the development of suitable simulation studies which try to set an evaluation framework for the selection of appropriate software tools according to the specific user requirements [11].

Several evaluation frameworks have been proposed. A hierarchical arrangement with high level characteristics that are decomposed into sub-characteristics and attributes is defined by the ISO/IEC 9126 standard [10]. The high level characteristics include reliability, usability, efficiency, maintainability and portability. An alternative hierarchical framework considers the user, the vendor, the model and the input, the execution, the animation, the testing and the efficiency and the output of the software as the highest

levels in the hierarchy [13]. Another one uses input, processing, output, support and cost as the generic categories [1]. Relative evaluation methods where the candidate software tools are compared in pairs with each other have also been used [5]. Furthermore, case studies have been produced for fields such as aerospace engineering [6], mail transfer issues [7] and structural engineering [12]. Expert systems that implement selection frameworks have also been written [9].

As mentioned previously building a robot prototype is a complex, time consuming and expensive task. Hence, tools are needed to simulate the electromechanical and behavioural aspects to assess the designs before they are actually built. The number of available software tools is vast and each has different capabilities and comes at a different cost, ranging from free to thousands of pounds.

This paper describes an evaluation framework that can aid the selection of appropriate software tools as well as formulate a toolset for robots to perform system level designs. The remainder of the paper is organised as follows. Section 2 describes the evaluation criteria, qualitative and quantitative methods for making the assessments and discusses the various different significances in the overall framework. Section 3 discusses whether the framework is consistent and reliable and a case study example illustrating the principles for a search and rescue robot is presented in Section 4.

2 Design framework

In this Section the assessment criteria of the software is presented, how the different elements can be measured and the effect of weighting in the evaluation.

2.1 Criteria

Cost: The cost of the simulator is the first criterion for its selection. It includes user training costs, maintenance costs, expenses for hardware requirements and development time costs.

Usability: This measures how well a design tool meets the users' requirements. It is one of the most important criteria as well as the most difficult one to measure. It depends upon the users' requirements and the task needed to be simulated. The difficulty in measuring it lies on the users' vague idea initially of what needs to be designed and assessed.

Expandability: This measures the likelihood and the time needed for the developers to improve the software, as well as if there are facilities that allow the users to expand it on their own and include their own modules.

Reusability: This measures if a software tool can be used for the design as well as the assessment of a model; if the produced models can be used by

other software tools; and if the already written control programs can be reused within real robots.

Development time: This measures how fast new designs can be developed.

Existence of predefined blocks and components as well as other tools can greatly speed up the implementation of it.

Efficiency: This measures the performance and other execution facilitation of the software. The performance is determined by the compilation and run time speed. Facilitation include speed control, off-line run, multiple and automatic batch run, reset capability, interaction, start in a non-empty state and debugging tools.

Visualisation: This measures how realistic the implemented designs and environments look like. For example if a chair looks like it is, or if it is represented just as a simple shape.

Portability: This considers if a software can be run in multiple operating systems. Depending on the user requirements this may be omitted.

User friendly: This measures how easy it is for the users to learn to use the software. Manuals, command references, general documentation, illustrative examples, training seminars and a user friendly interface are all features that help in learning.

Technical support: This measures how likely it is to get assistance. It includes technical assistant from the vendor or the developers, from help forums, FAQ lists and user groups.

Analysis facilitation: This measures if the software provides any facilitation for analysing and visualising the data such as business graphs and charts, structured output of the data or exportation into a spreadsheet, analysis functions and video capture or screenshots.

2.2 Qualitative and quantitative measurement methods

Assessment criteria naturally need some kind of qualitative or quantitative measurement. One simple way is to strictly examine if the candidate software tools cover each of the subcriteria or not. The preferred simulator would be the one that covers most of them in as complete manner as possible. However, it is difficult to measure the criteria in all aspects as probably they would be partially covered. Even if we overcome this problem by skipping the high level layer and we consider only the lower one as our list of criteria, we still have to face the same dilemma when a simulator covers a subcriterion to some partial extent.

A more flexible approach is to measure the criteria quantitatively depending on the subcriteria. It can be done either descriptively or numerically. An example of this is a scale of possible values from 0–1, covering from “not at all” to “full”, summed to give an overall criterion score, which in turn can be summed up for all the different criteria (with different weights if necessary as discussed in Section 2.3) to give the total assessment of the software.

All these are absolute evaluation methods. This means that each simulator is individually evaluated by assigning a value which signifies how well it meets the criteria. An alternative method is relative evaluation [5], where each simulator is compared with every other simulator under consideration in pairs. Hence, a given simulator will have relative scores that signify how better or worse it is from each of its “competitors”. This method has shown to give satisfactory results, although there have been sometimes some surprising rankings in the comparisons [5].

2.3 Weighting of the criteria

In the methods described in Section 2.2 there has been no implication of whether the criteria have different importance. In fact it was assumed that they all have the same weight. Obviously, this is not rational as some criteria are more significant than others depending on the user requirements and priorities. Each *score* can be multiplied by a weight w signifying the contribution of a criterion in the overall assessment and each subcriterion can also have its own weight within the scope of higher level criterion. All scores can be normalised to values in the range 0–1. This is algebraically described by equations 1 and 2.

$$score_{tool} = \frac{\sum_{i=0}^{i=n} (w_i \times score_{crit_i})}{\max(score_{tool})} \quad (1)$$

$$\text{where, } score_{crit_i} = \frac{\sum_{j=0}^{j=n} (w_j \times score_{subcrit_j})}{\max(score_{crit_i})} \quad (2)$$

3 Discussion

A list of criteria that can be used as an evaluation framework for the formation of an appropriate design tool-based environment for robotic systems was described in Section 2. From the proposed software evaluation frameworks presented in Section 1 we could say that there is an attempt to categorise the large number of criteria in generic groups. Features can be easily added or removed from a hierarchical framework depending on the simulation domain and the user requirements. Furthermore, the features can be better visualised in a hierarchical organisation.

On the other hand, it can be seen that each study proposes its own hierarchical structure. Even if there are a lot of similarities especially in the low level attributes between the various hierarchical frameworks, this uniformity causes confusion. Moreover, hierarchical models tend to be static within their scope. This means that the features should be independent from each other, otherwise the hierarchical structure is compromised. One of the strengths of hierarchical frameworks is that they are flexible to minor changes. However, “major

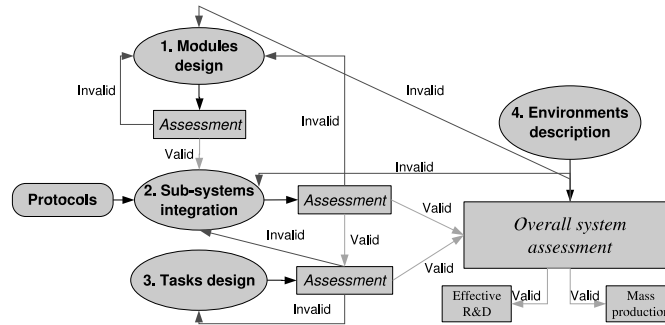


Fig. 2. Simulation cycle

changes, although their occurrences are rare, may cause a re-organisation of the hierarchy” [13].

Our list of criteria is concentrated on the selection of robot system simulators. The main criteria include various subcriteria and this gives the impression that this framework has a strict hierarchy as well. However, in this structure generic criteria can be added or deleted according to the user requirements and sub-features can have multiple instances when required. Furthermore, the criteria can be affected by other ones outside their generic groups. The pay-off of a dynamic structure is its computational complexity when interactions between the criteria occur. It must be noted that this is rare as the generic criteria are distinct from each other.

4 Case study: Urban search and rescue applications

A design and assessment cycle is shown in Figure 2. At the module level the basic and the super modules are designed and assessed both individually and together. At the system level the complete robotic systems are considered in their operational environments. Suitable design tools for the module level design are described in [4]. To highlight the system level design toolset, a specific case study on search and rescue robots is presented upon.

It is obvious that a system level simulator should provide the capability to implement a virtual environment realistic to the real one. The modularity concept adds to this list of requirements that the simulator should not be restricted to a particular robotic system or to a specific set of predefined modules, rather than it should allow the user to expand the library of modules and allow any desirable integration of them.

The operational environment in urban search and rescue (USAR) applications can be characterised as dynamic, hostile and rough. Entry points are often narrow and difficult to reach. The terrain the robots have to navigate can be extremely uneven, making even the simplest movements difficult without getting stuck. There is always the danger of a further collapse and the light

conditions are normally very poor. Even worse, due to the complete disorder of the environment the readings of individual sensors can be noisy and unreliable. Robots have been used for the first time in the World Trade Centre disaster and very useful information has been collected [3]. However, USAR robots are far from achieving their missions mainly due to their incapability of traversing through large piles of debris that are often found in urban disaster sites, teleoperation difficulties and communication losses in these environments and lack of reliable robot autonomy. Therefore researchers frequently simplify the operational environment to partially collapsed and unstable structures for investigation to be allowed.

A semi-autonomous USAR robot in its minimal form must carry a video camera, be able to be teleoperated through partially collapsed buildings and transmit the video either wirelessly or through a communication cable back to the operator. The following additional modules are also useful to have:

- Input modules: thermal camera, directional microphones, laser range finder, etc.
- Processing modules: hardware such as better microcontrollers, more memory, etc. and software modules like internal monitoring, auto navigation to a given position, recovery from communication dropouts, etc.
- Output modules: grippers, drills, suction pipe, etc.
- Infrastructure modules: more power, more powerful motors, better locomotion system, waterproof structure, etc.

More details about these as well as a more detailed presentation of a modular design of a USAR robot is found in [8].

About 150 software tools that can be used in the R&D design of robotic systems have been identified. They are classified in various categories such as graphics-environment modelling, image processing, programming libraries, physics libraries, planners, crossover tools, robot control libraries, robot dynamics and statics and system simulators as defined in our context. About 40 of them can be considered as system simulators where a robot can be simulated in a virtual environment. In order to make an evaluation of these, the assessment framework described in Section 2 was used.

The most important criteria from the requirements of the software tools are usability, reusability and expandability. Cost and technical support are also considered important. These five criteria contribute nearly 80% of the total evaluation. Table 1 shows the most suitable ones that cover to some extent the requirements described previously and Table 2 shows their scores based on our evaluations.

From all these the open source package of Player/Stage/Gazebo (PSG) seems to be the most appropriate. Stage is a 2D simulator capable of simulating large numbers of robots. Gazebo is a full physics 3D simulator which is quite processing demanding allowing a small number of robots to be simulated simultaneously. Player is a robot interface providing a network interface to a variety of robot and sensor hardware. The server/client architecture allows

Table 1. Suitable system level simulators

Name	License	URL (http://)
Player/Stage/Gazebo	Open source	playerstage.sourceforge.net
Simulation Studio	Commercial	eyewyre.com/studio
Webots	Commercial	www.cyberbotics.com/products/webots
Easybot	Commercial	iwaps1.informatik.htw-dresden.de :8080/Robotics/Easybot
Dynamechs	Open source	sourceforge.net/projects/dynamechs
Missionlab	Open source	www.cc.gatech.edu/aimosaic/ robot-lab/research/MissionLab
Rossum's Playhouse	Open source	sourceforge.net/projects/rossum

Table 2. Scores of system level simulators in a range 1–5. The weight of each criterion is shown in the parentheses.

Name	Overall Score	Usability (20%)	Reusability (15%)	Expandability (15%)	Cost (20%)	Technical Support (10%)	Rest (20%)
Player/Stage Gazebo	3.9	4.7	4.7	4.7	5.0	3.9	0.9
Simulation Studio	3.2	4.1	3.8	3.1	4.0	4.1	1.0
Webots	3.2	4.5	4.6	4.7	1.3	4.1	1.1
Easybot	3.0	3.8	3.4	3.8	4.0	2.0	0.7
Dynamechs	2.9	3.3	2.5	3.3	5.0	2.6	0.7
Missionlab	2.8	1.6	3.8	2.9	5.0	3.0	0.6
Rossum's Playhouse	2.0	1.2	1.3	1.3	5.0	3.1	0.5

the control programs to be written in any language. These can be used for simulating virtual robots in Stage, Gazebo as well as real ones with no or little modification. PSG can be extended as new interfaces can be written for the hardware modules. As far as technical support is concerned there is helpful documentation as well as an active email list. One of the disadvantages of PSG is that the learning time is much longer than for other tools.

From the rest of the freeware simulators the following are worth a mention:

- Dynamechs and its graphical front end RobotBuilder is a good 3D simulator in which both the environment and a robot system can be modelled. It also has a long learning time like PSG. It is used for simulation and not for controlling real robots.
- MissionLab is a set of tool that allows the simulation in 2D environments and control of a single or a group of robots.
- Rossum's Playhouse is a simple 2D simulator and although it has a score below the average (2.5), it is mentioned here as it is the first software release of the ROSSUM project which “will grow into a resource where

developers can download reliable, highly capable modules to incorporate into their robots”.

The following simulators are the commercial ones:

- Simulation StudioTM is a 3D interactive simulator which allows the control of simulated and real robots with a BASIC Stamp microcontroller. It can be extended with new modules provided by the developers.
- WebotsTM is also a powerful simulator with similar characteristics to PSG with the only difference that it is more expensive than most of the others. Both simulated and real robots can be controlled in full physics 3D environments. It can also be extended with new user’s modules.
- Easybot is also a commercial tool which depends on the cost of the used 3D modeller LightVision3DTM. Unlike PSG and Webots, Easybot does not allow the control of real robots. Another drawback is the lack of a physics engine. The simulator is discontinued for the favour of JRoboSim which is the successor of Easybot.

For the requirements of a USAR robot simulation and those of modularity it seems that the most suitable tools are PSG and Webots. Both have capabilities to implement virtual USAR environments, modelling of various modules and systems and teleoperation of the semi-autonomous systems. The development time of Webots should be faster as it seems to provide more graphical facilities for the implementation. Webots is a commercial package and in fact it is expensive compared to the free PSG. This is the main reason why PSG is preferred. Furthermore, both have interconnections with other third party software; Webots with MatlabTM and LabViewTM which are also commercial, while PSG with the free SourceForge¹ projects MARIE and Robot-Flow/FlowDesigner.

5 Conclusions

The design of modular robotic systems can be decomposed into system and module level designs which are expected to be carried out in parallel. Software tools are needed for these. However, the vast number of available packages makes the selection of appropriate ones difficult and time consuming.

This paper described a guidance evaluation framework for the selection of appropriate design software tools. It consists of high level criteria which can be decomposed into particular features/subcriteria. A software toolset focusing on the system level design is formalised with the aid of the evaluation framework. This was illustrated through a case study on search and rescue robot applications.

¹<http://sourceforge.net>

References

- [1] J. Banks. Selecting simulation software. In *Simulation Conference Proceedings*, pages 15–20. IEEE, 8–11 Dec. 1991.
- [2] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, Mar. 1986.
- [3] J. Casper. Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center. Master’s thesis, Dept. of Computer Science and Engineering, University of South Florida, May 2002.
- [4] I. Chochlidakis, Y. Gatsoulis, and G.S. Virk. Open modular design for robotic systems. In *Proc. of CLAWAR 2004, 7th Int. Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*. Springer, 22–24 Sep. 2004.
- [5] L. Davis and G. Williams. Evaluating and selecting simulation software using the analytic hierarchy process. *Integrated Manufacturing Systems*, 5(1):23–32, 1994.
- [6] D.S. Eccles. Building simulators for aerospace applications: processes, techniques, choices and pitfalls. In *Aerospace Conference Proceedings*, volume 1, pages 517–527. IEEE, 18–25 March 2000.
- [7] X. Franch and J.P. Carvallo. A quality-model-based approach for describing and evaluating software packages. In *Joint Int. Conference on Requirements Engineering Proceedings*, pages 104–111. IEEE, 2002.
- [8] Y. Gatsoulis, I. Chochlidakis, and G.S. Virk. A software framework for the design and support of mass market clawar machines. In *Proc. of IEEE Mechatronics and Robotics Int. Conference*, 13–15 Sep. 2004.
- [9] V. Hlupic and A.S. Mann. Simselect: a system for simulation software selection. In *Simulation Software Proceedings*, pages 720–727. IEEE, 1995.
- [10] ISO/IEC. Standards 9126 (information technology – software product evaluation – quality characteristics and guidelines for their use), 1991.
- [11] A.M. Law. How to conduct a successful simulation study. In *Winter Simulation Conference*, volume 1, pages 66–70. IEEE, 7–10 Dec. 2003.
- [12] N.A. Maiden and C. Ncube. Acquiring COTS software selection requirements. *IEEE Software*, 15(2):46–56, March–April 1998.
- [13] J. Nikoukaran, J. Hlupic, and R.J. Paul. Criteria for simulation software evaluation. In *Simulation Conference Proceedings*, pages 399–406, 1998.
- [14] G.S. Virk. Technical Task 1: Modularity for CLAWAR machines – specifications and possible solutions. In G.S. Virk, M. Randall, and D. Howard, editors, *2nd Int. Conference on Climbing and Walking Robots*, 1999.
- [15] G.S. Virk. CLAWAR: Modular robots for the future. In *Proc. of the 3rd Int. Workshop on Robot Motion and Control, RoMoCo02*, pages 73–76. IEEE, 9–11 Nov. 2002.
- [16] G.S. Virk. CLAWAR modularity for robotic systems. *The International Journal of Robotics Research*, 22(3–4):265–277, March–April 2003.