INTERNATIONAL JOURNAL OF RESEARCH IN MECHANICAL, MECHATRONICS AND AUTOMOBILE ENGINEERING (IJRMMAE)
ISSN: 2454-1435 (Print) | 2454-1443 (online)
Volume 5 Issue 2, July – September 2019 - www.ijrmmae.in – Pages 121-126

ROBOTIC SYSTEM ARCHITECTURE AND PROGRAMMING

D. Shameer[1], R. Deepak[2], T.Ravichandra Babu[3]

[1,2]Department of Mechatronics Engineering, Agni College of Technology, Chennai.
[3]Department of Electronics and Communication Engineering

ABSTRACT - Robot software systems tend to be complex. This complexity is due, in large part, to the need to control diverse sensors and actuators in real time, in the face of significant uncertainty and noise. Robot systems must work to achieve tasks while monitoring for, and reacting to, unexpected situations. Doing all this concurrently and asynchronously adds immensely to system complexity. The use of a well-conceived architecture, together with programming tools that support the architecture, can often help to manage that complexity. Currently, there is no single architecture that is best for all applications – different architectures have different advantages and disadvantages. It is important to understand those strengths and weaknesses when choosing an architectural approach for a given application. This chapter presents various approaches to architecting robotic systems. It starts by defining terms and setting the context, including a recounting of the historical developments in the area of robot architectures.

OVERVIEW

The term robot architecture is often used to refer to two related, but distinct, concepts. Architectural structure refers to how a system is divided into subsystems and how those subsystems interact. The structure of a robot system is often represented informally using traditional boxes and arrows diagrams ormore formally using techniques such as unified modeling language (UML) [8.1]. In contrast, architectural style refers to the computational concepts that underlie a given system. For instance, one robot system might use a publish–subscribe message passing style of communication, while another may usea more synchronous client–server approach. All robotic systems use some architectural structure and style. However, in many existing robot systems it is difficult to pin down precisely the architecture being used. In fact, a single robot system will often use several styles together. In part, this is because the system implementation may not have clean subsystem bound- aries, blurring the architectural structure. similarly, the architecture and the domain-specific implementation are often intimately tied together, blurring the architectural style(s). This is unfortunate, as a well-conceived, clean architecture can have significant advantages in the specification, execution, and validation of robot systems. In general, robot architectures facilitate development by providing beneficial constraints on the design and implementation of robotic systems, without being overly restrictive. For instance, separating behaviors into modular units helps to increase understandability and reusability, and can facilitate unit testing and validation.

SPECIAL NEEDS OF ROBOT ARCHITECTURES

In some sense, one may consider robot architectures as software engineering. However, robot architectures are distinguished from other software architectures because of the special needs

ISSN: 2454-1435 (Print) | 2454-1443 (online)
Volume 5 Issue 2, July – September 2019 - www.ijrmmae.in – Pages 121-126

of robot systems. The most important of these, from the architectural perspective, are that robot systems need to interact asynchronously, in real time, with an uncertain, often dynamic, environment. In addition, many robot systems need to respond at varying temporal scopes – from millisecond feedback control to minutes, or hours, for complex tasks. To handle these requirements, many robot architectures include capabilities for acting in real time, controlling actuators and sensors, supporting concurrency, detecting and reacting to exceptional situations, dealing with uncertainty, and integrating high-level (symbolic) planning with low-level (numerical) control. While the same capability can often be implemented using different architectural styles, there may be advantages of using one particular style over another. As an example, consider how a robot system's style of communications can impact on its reliability. Many robot systems are designed as asynchronous processes that communicate using message passing. One popular communication style is client–server, in which a message request from the client is paired with a response from the server. An alternate communication paradigm is publish–subscribe, in which messages are broadcast asynchronously and all modules that have previously indicated an interest in such messages receive a copy. With client–server-style message passing, modules typically send a request and then block, waiting for the response. If the response never comes (e.g., the server module crashes) then deadlock can occur. Even if the module does not block, the control flow still typically expects a response, which may lead to unexpected results if the response never arrives or if a response to some other request happens to arrive first. In contrast, systems that use publish–subscribe tend to be more reliable:

MODULARITY AND HIERARCHY

One common feature of robot architectures is modular decomposition of systems into simpler, largely independent pieces. As mentioned above, robot systems are often designed as communicating processes, where the communications interface is typically small and relatively low bandwidth. This design enables the processes/modules to handle interactions with the environment asynchronously, while minimizing interactions with one another. Typically, this decreases overall system complexity and increases overall reliability. Often, system decomposition is hierarchical –modular components are themselves built on top of other modular components. Architectures that explicitly support this type of layered decomposition reduce system complexity through abstraction. However, while hierarchical decomposition of robotic systems is generally regarded as a desirable quality, debate continues over the dimensions along which to decompose. Some architectures [2] decompose along a temporal dimension – each layer in the hierarchy operates at a characteristic frequency an order of magnitude slower than the layer below. In other architectures [3–6], the hierarchy is based on task abstraction – tasks at one layer are achieved by invoking a set of tasks at lower levels. In some situations, decomposition based on spatial abstraction may be more useful, such as when dealing with both local and global navigation [7]. The main point is that different applications need to decompose problems in different ways, and architectural styles can often be
found to accommodate those different needs.

SOFTWARE DEVELOPMENT TOOLS

While significant benefit accrues from designing systems using well-defined architectural styles, many architectural styles also have associated software tools that facilitate adhering to

ISSN: 2454-1435 (Print) | 2454-1443 (online)
Volume 5 Issue 2, July – September 2019 - www.ijrmmae.in – Pages 121-126

that style during implementation. These tools can take the form of libraries of functions calls, specialized programming languages, or graphical editors. The tools make the constraints of the architectural style explicit, while hiding the complexity of the underlying concepts.

LAYERED ROBOT CONTROL ARCHITECTURES

One of the first steps towards the integration of reactivity and deliberation was the reactive action packages (RAPs) system created by Firby. In his thesis, we see the first outline of an integrated, three-layer architecture. The middle layer of that architecture, and the subject of the thesis, was the RAPs system. Firby also speculated on the form and function of the other two tiers, specifically with the idea of integrating classic deliberative methods with the ideas of the emerging situated reasoning community, but those layers were never implemented.

Later, Firby would integrate RAPs with a continuous low-level control layer. Independently and concurrently, Bonasso at MITRE devised an architecture that began at the bottom layer with robot behaviors programmed in the Rex language as synchronous circuits. These Rex machines guaranteed consistent semantics between the agent's internal states and those of the world. The middle layer was a conditional sequencer implemented in the Gapps language, which would continuously activate and deactivate the Rex skills until the robot's task was complete. This sequencer based on Gapps was appealing because it could be synthesized through more traditional planning techniques. This work culminatedin the 3T architecture (named after its three tiers of interacting control processes – planning, sequencing, and real-time control), which has been used on many generations of robots. Architectures similar to 3T have been developed subsequently. One example is ATLANTIS, which leaves much more control at the sequencing tier. In ATLANTIS, the deliberative tier must be specifically called by the sequencing tier. A third example is Saridis' intelligent control architecture.

The architecture begins with the servo systems available on a given robot and augments them to integrate the execution algorithms of the next level, using VxWorks and the VME bus. The next level consists of a set of coordinating routines for each lower subsystem, e.g., vision, arm motion, and navigation. These are implemented in Petri net transducers (PNTs), a type of scheduling mechanism, and activated by a dispatcher connected to the organizational level. The organizational level is a planner implemented as a Boltzmann neural network. Essentially the neural network finds a sequence of actions that will match the required command received as text input, and then the dispatcher executes each of these steps via the network of PNT coordinators.

The LAAS architecture for autonomous systems (LAAS) is a three-layered architecture that includes software tools to support development/programming at each layer. The lowest layer (functional) consists of a network of modules, which are dynamically parameterized control and perceptual algorithms. Modules are written in the generator of modules (GenoM) language, which produces standardized templates that facilitate the integration of modules with one another. Unlike most other three-layered architectures, the executive layer is fairly simple – it is purely reactive and does no task decomposition. It serves mainly as a bridge – receiving task sequences from the highest layer and selecting and parameterizing tasks to send to the functional layer. The executive is written in the Kheops language, which automatically generates decision networks that can be formally verified.

INTERNATIONAL JOURNAL OF RESEARCH IN MECHANICAL, MECHATRONICS AND AUTOMOBILE ENGINEERING (IJRMMAE)
ISSN: 2454-1435 (Print) | 2454-1443 (online)
Volume 5 Issue 2, July – September 2019 - www.ijrmmae.in – Pages 121-126

At the top, the decision layer consists of a planner, implemented using the indexed time table (IxTeT) temporal planner, and a supervisor, implemented using procedural reasoning system (PRS). The supervisor is similar to the executive layer of other three-layered architectures – it decomposes tasks, chooses alternative methods for achieving tasks, and monitors execution. By combining the planner and supervisor in one layer, LAAS achieves a tighter connection between the two, enabling more flexibility in when, and how, replanning occurs.

The LAAS architecture actually allows for multiple decisional layers at increasingly higher levels of abstraction, such as a high-level mission layer and a lower-level task layer. Remote agent is an architecture for the autonomous control of spacecraft. It actually consists of four layers – a control (behavioral) layer, an executive, a planner/ scheduler, and mode identification and recovery (MIR) that combines fault detection and recovery. The control layer is the traditional spacecraft real-time control system. The executive is the core of the architecture – it decomposes, selects, and monitors task execution, performs fault recovery, and does resource management, turning devices on and off at appropriate times to conserve limited spacecraft power.

The lanner/scheduler is a batch process that takes goals, an initial (projected) state, and currently scheduled activities, and produces plans that include flexible ranges on start and end times of tasks. The plan also includes a task to reinvoke the planner to produce the next plan segment. An important part of the remote agent is configuration management– configuring hardware to support tasks and monitoring that the hardware remains in known, stable states. The role of configuration management is split between the executive, which uses reactive procedures, and MIR, which uses declarative models of the spacecraft and deliberative algorithms to determine how to reconfigure the hardware in response to detected faults.

The Syndicate architecture extends the 3T model to multirobot coordination. In this architecture, each layer interfaces not only with the layers above and below, as usual, but also with the layers of the other robots at the same level. In this way, distributed control loops can be designed at multiple levels of abstraction. The version of Syndicate in used a distributed market-based approach for task allocation at the planning layer. Other noteworthy multitiered architectures have appeared in the literature. The National Bureau of Standards (NBS) developed for the Aeronautics and Space Agency (NASA) the NASA/NBS standard reference model (NASREM), later named real-time control system (RCS), was an early reference model for telerobotic control.

It is a many-tiered model in which each layer has the same general structure, but operates at increasingly lower frequency as it moves from the servo level to the reasoning levels. With the exception of maintaining a global world model, NASREM, in its original inception, provided for all the data and control paths that are present in architectures such as 3T, but NASREM was a reference model, not an implementation. The subsequent implementations of NASREM followed primarily the traditional sense–plan–act approach and were mainly applied to telerobotic applications, as opposed to autonomousrobots. A notable exception was the early work of Blidberg. While three-layered robot architectures are very popular, various two-layered architectures have been investigated by researchers. The coupled layered architecture for robot autonomy (CLARAty) was designed to provide reusable software for

NASA's space robots, especially planetary rovers. CLARAty consists of a functional and a decision layer. The functional layer is a hierarchy of object-oriented algorithms that provide more and more abstract interfaces to the robot, such as motor control, vehicle control, sensor-based navigation, and mobile manipulation. Each object provides a generic interface that is hardware independent, so that the same algorithms can run on different hardware. The decision layer combines planning and executive capabilities. Similar to the LAAS architecture, this is done to provide for tighter coordination between planning and execution, enabling continual replanning in response to dynamic contingencies.

CONCLUSION

Robot architectures are designed to facilitate the concurrent execution of task-achieving behaviors. They enable systems to control actuators, interpret sensors, plan, monitor execution, and deal with unexpected contingencies and opportunities. They provide the conceptual framework within which domain-dependent software development can take place, and they often provide programming tools that facilitate that development. While no single architecture has proven to be best for all applications, researchers have developed a variety of approaches that can be applied in different situations. While there is not yet a specific formula for determining which architecture will be best suited for a given application

REFERENCES

[1] Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process (Addison Wesley Longman, Reading 1998)
[2] J.S. Albus: RCS: A reference model architecture for intelligent systems, Working Notes: AAAI 1995 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents (1995)
[3] R.A. Brooks: A robust layered control system for a mobile robot, IEEE J. Robot. Autom. **2**(1), 14–23 (1986)
[4] R.J. Firby: An Investigation into Reactive Planning in Complex Domains, Proc. of the Fifth National Conference on Artificial Intelligence (1987)
[5] R. Simmons: Structured control for autonomous robots, IEEE Trans. Robot. Autom. **10**(1), 34–43 (1994)
[6] J.J. Borrelly, E. Coste-Maniere, B. Espiau, K. Kapelos, R. Pissard-Gibollet, D. Simon, N. Turro: The ORCCAD architecture, Int. J. Robot. Res. **17**(4), 338–359 (1998)
[7] B. Kuipers: The spatial semantic hierarchy, Artif. Intell. **119**, 191–233 (2000)
[8] R. Orfali, D. Harkey: Client/Server Programming with JAVA and CORBA (Wiley, New York 1997)
[9] R. Simmons, G. Whelan: Visualization Tools for Validating Software of Autonomous Spacecraft, Proc. Of International Symposium on Artificial Intelligence, Robotics and Automation in Space (Tokyo 1997)
[10] R. A. Brooks: The Behavior Language: User's Guide, Technical Report AIM-1227, MIT Artificial Intelligence Lab (1990)
[11] R.J. Firby, M.G. Slack: Task execution: Interfacing to reactive skill networks, Working Notes: AAAI Spring Symposium on Lessons Learned from Implemented Architecture for Physical Agents (Stanford 1995)

ISSN: 2454-1435 (Print) | 2454-1443 (online)
Volume 5 Issue 2, July – September 2019 - www.ijrmmae.in – Pages 121-126

[12] E. Gat: ESL: A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents, Proc. of the IEEE Aerospace Conference (1997)
[13] V. Verma, T. Estlin, A. Jónsson, C. Pasareanu, R. Simmons, K. Tso: Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences, Proc. 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space (Munich 2005)
[14] S.A. Schneider, V.W. Chen, G. Pardo-Castellote, H.H. Wang: ControlShell: A Software Architecture for Complex Electromechanical Systems, Int. J. Robot. Res. **17**(4), 360–380 (1998)