COMPONENT RUNTIME SELF-ADAPTATION IN ROBOTICS

Daniel Hernandez, Antonio Dominguez, Oscar Deniz and Jorge Cabrera

IUSIANI - ULPGC
Campus de Tafira, 350017, Las Palmas de Gran Canaria, Spain

Keywords: Robotic systems, self-adaptive software components.

Abstract: Mobile robotic applications have to deal with limited resources and variable execution conditions that must be appropriately managed in order to keep an acceptable system behavior. This requires the implementation of runtime adaptation mechanisms that monitor continuously system state and module the resulting performance as a function of the available resources. As we consider that these adaptation mechanisms should be offered as a facility to robotic application programmers, we have integrated them inside CoolBOT, a component oriented framework for programming robotic systems. CoolBOT contributes to reduce the programming effort, promoting code reuse, while the adaptation scheme allows for more robust applications with an extended range of operation. In this paper we also present a demonstrator that outlines the benefits of using the proposed approach in the development of real robotic applications.

1 INTRODUCTION

The management of shared resources is an important subject of research in robotic and sensor-effector systems. Some authors (Murphy, 2000)(Kortenkamp and Schultz, 1999), coincide in the necessity of including the adaptive aspect in order to build really robust systems. This is specially important in mobile robotic systems, configured as tactical multi-objective designs which are often affected by the shortage of shared resources (Jones, 1997). When strict guaranties of bounded reaction times and rigid operation frequencies are needed, hard real-time techniques are the most commonly adopted solution. However, there are many contexts of application in robotics where those strict guaranties can be relaxed to a certain extent, and a soft real-time adaptive control scheme may in our opinion represent a more convenient solution.

On the other hand, in the context of software development for robotic applications, the complexity of programming and maintenance (Kortenkamp and Schultz, 1999) has promoted the proposal of architectures (Coste-Maniere and Simmons, 2000) and frameworks (Fleury et al., 1997) (Schlegel and Wörz, 1999). Following this tendency we have presented **CoolBOT** (Domínguez-Brito et al., 2004), a component-based software framework aimed at facilitating the development of robotic systems without imposing any specific architecture.

The combination of the adaptive control and component-based software has been considered, outside mobile robotics, by some authors (Garlan et al., 2004) (Oreizy et al., 1999) with promising results. There are also some examples of adaptive robotic architectures (Musliner et al., 1999), but not for soft real-time frameworks. We consider that CoolBot could clearly benefit from the integration of run-time self-adaptive resources, constituting a more useful tool for robotic application developers.

This paper is organized in the following sections: brief outline of CoolBOT (2), adaptation mechanisms (3), experiments with a real mobile robot demonstrator (4) and conclusions (5).

2 CoolBOT

CoolBOT (Domínguez-Brito et al., 2004) is a component-oriented framework that allows designing software in terms of composition and integration of software components. In CoolBOT, components are active entities that act on their own initiative, carrying out their own specific tasks, running in parallel or concurrently, and are normally weakly coupled. More

Hernandez D., Dominguez A., Deniz O. and Cabrera J. (2005).
COMPONENT RUNTIME SELF-ADAPTATION IN ROBOTICS.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics - Robotics and Automation*, pages 399-402
DOI: 10.5220/0001166403990402

specifically, components are modelled as **Port Automata** (Steenstrup et al., 1983)(Stewart et al., 1997), a concept that establishes a clear distinction between the internal functionality of a component, an automaton, and its external interface, conformed by input and output port connections (*port connections*).

The framework introduces two kinds of facilities in order to support monitoring and control of components: *observable variables*, which represent features of components that might be of interest from outside; and *controllable variables*, that represent aspects of components which might be externally controlled so, through them, the internal behavior of a component can be modified. In addition, to guarantee external observation and control, CoolBOT components provide by default two important ports: the *control* port and the *monitoring* port. Fig. 1 illustrates these variables and ports in a typical control loop for a component using another component as an external supervisor.

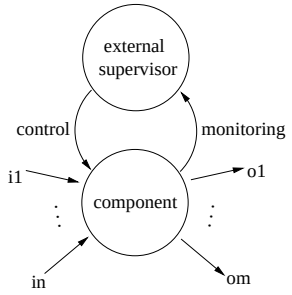

Figure 1: A typical component control loop.

Internally all components are modelled using the same default state automaton that contains all possible control paths a component may follow. This *default automaton* can be always brought externally in finite time by means of the *control* port to any of the controllable states of the automaton, which are: **ready**, **running**, **suspended** and **dead**. The rest of states are reachable only internally, and from them, a transition to one of the controllable states can be forced externally. The **running** state constitutes the part of the automaton that implements the specific functionality of the component, and it is called the *user automaton*. Furthermore, there are two pair of states conceived for handling faulty situations during execution. One of them devised to face errors during resource allocation (**starting error recovery** and **starting error** states), and the other one thought to deal with errors during task execution (**error recovery** and **running error** states). These states are part of the support CoolBOT provides for error and exception handling in components.

Components are not only data structures, but execution units as well. In fact, CoolBOT components are mapped as *threads* when they are in execution; Win32 threads in Windows, and POSIX threads in GNU/Linux.

CoolBOT components are classified into two kinds: *atomic* and *compound* components.

- *Atomic components* that have been mainly devised in order to abstract low level hardware layers to control sensors and/or effectors; to interface and/or to wrap third party software and libraries; and to implement generic algorithms. In this way they become isolated pieces of deployable software in the form of CoolBOT components. Thanks to the uniformity of external interface and internal structure the framework imposes on components, they may be used as building blocks that hide their internals behind a public external interface.
- *Compound components* are compositions of instances of several components which can be either atomic or compound. Compound components use the functionality of instances of another atomic or compound components to implement its own functionality, and in turn, can be integrated and composed hierarchically with other components to form new compound components.

Analogously to modern operating systems that provide IPC (**I**nter **P**rocess **C**ommunications) mechanisms to inter communicate processes, CoolBOT provides ***I**nter **C**omponent **C**ommunications* or *ICC* mechanisms to allow components to interact and communicate among them. CoolBOT *ICC* mechanisms are carried out by means of input ports, output ports, and ports connections. Communications are one of the most fragile aspects of distributed systems. In CoolBOT, the rationale for defining standard methods for data communications between components is to ease inter operation among components that have been developed independently, offering optimized and reliable communication abstractions.

3 ADAPTIVE CONTROL

A robotic application should be able to adjust its performance as a function of either the available resources or the resources assigned a priori. The objective is to force a smooth degradation when resources are not enough to meet application demands, and allow a controlled recovery when the system overload disappears. See (Hernández-Sosa, 2003) for a more complete description.

The CoolBOT framework provides mechanisms to support the adaptation of component consumption of computational resources during operation. CoolBOT adaptation mechanisms include a graceful degradation procedure when there are not enough resources available, and a performance status recovery procedure whenever possible. Additional objectives are

reactivity, stability and coordination to avoid system imbalances.

3.1 Elements of Runtime Adaptation

A component, whether atomic or compound, may be declared as *adaptive* or *non-adaptive*. If a component is declared as adaptive, it must publish the set of *performance levels* at which it can operate. A performance level represents a trade-of between resource consumption and quality of results (better quality demands higher requirements).

The integration of the dynamic adaptation of components inside CoolBOT is designed around two controllable variables: frequency of operation and quality level. On the frequency axis, a supervisor can modify the period associated to any of the components under its control, for example, increasing their values to face CPU saturation. On the quality axis, the supervisors can command lower qualities (sensor resolution, accuracy of computations, exhaustiveness, etc.) to reduce CPU load and latencies at the cost of increasing uncertainty or decreasing results quality.

The framework will monitor certain operation conditions, named as *adaptive observables*, at run time. These include component level measures such as period or elapsed/cpu times, and system level measures such as computational load, battery level or load profile. Some results from processing can also be used as elements in adaptive control, using the observable variable facility offered by the framework.

Depending on these measures and their reference values some elementary degradation/promotion adaptive commands can be triggered on adaptive components through their control ports.

3.2 Control Strategies

Several control policies have been designed to organize system adaptation. Their objectives include avoiding an unbalanced system degradation/promotion, reduce settling times and fostering stability.

3.2.1 Timeout control

Timeouts control adapts, on a hierarchical basis, the runtime demands of shared resources in the system in order to guarantee the specified frequencies of operation. Firstly, period violations are detected locally inside the time-pressured component, where the control thread generates the corresponding degradation order. To avoid systems unbalance, however, local control actions are limited to a scope defined by two homogeneity thresholds (minimum and maximum degradation values). If local adaptation resources are not enough, the component notifies the problem to upper levels, the supervisor component, where global actions can be executed.

3.2.2 CPU load control

The load control loop operates only at global level. The system load is estimated and compared with a certain reference level fixed externally. Promotion and degradation actions are generated accordingly to maintain the desired load level.

Candidate selection for targeting control actions plays an important role in adaptation performance. In general, an agreement between reactivity and stability must be reached. The most intense reactions are obtained when high frequency, CPU demanding, multiple destination and/or high-resolution sensor components are affected.

The supervisors evaluate these parameters as well as priority to select target components for adaptation commands. Several strategies have been implemented using different target selection criteria: priority-based, combined priority-degradation, frequency-based and topology-based.

The adaptive aspects are associated naturally in CoolBOT to control threads (port threads) inside each atomic component at low level, and supervisor components at higher levels (compound components, system level). The early separation of control, processing and communication areas permits the development of modular and integrable robotic applications.

4 DEMONSTRATOR

In order to illustrate the operation of the adaptation mechanisms on real-world applications, we have implemented a mobile robotic application as a demonstrator.

The application is based on a mechanical head (Directed Perception PTU and USB camera) mounted on a mobile robot (Pioneer). A notebook has been added for running the application, being connected via USB and serial ports to the head and the robot.

A minimal multi-purpose system has been designed combining two main objectives, line following and object detection, that can be prioritized alternatively. In the configuration used for this paper, the robot must follow, as tight as possible, a trajectory defined by a line traced on the floor. Secondarily, the robot can look at both sides of the route trying to detect some colored balls.

Four CoolBOT components have been used in the integration of the system corresponding to this demonstrator: one for controlling the Pioneer robot, other one for the PTU unit and the camera, another

one for line processing and following commands, and the last one for object detection.

As configured, on straight-line trajectory segments both tasks can perform alternatively at a pre-defined frequency. On curved segments, however, the risk of loosing the track increases. To avoid this, the movement amplitude and activation period of the object detection component is modulated according to and adaptive observable computed from the curvature of the line that the robot must follow. The modification of the scanning amplitude can be considered a quality-based adaptive control, as processing times are shortened at the cost of reducing the probability of finding color objects. The modification of the period, however, corresponds to a frequency-based adaptive control.

The Fig. 2 represents the executions of the color detection component task along a trajectory. On curved segments, both frequency and amplitude of scanning take lower values. On straight segments both parameters can increase their values.

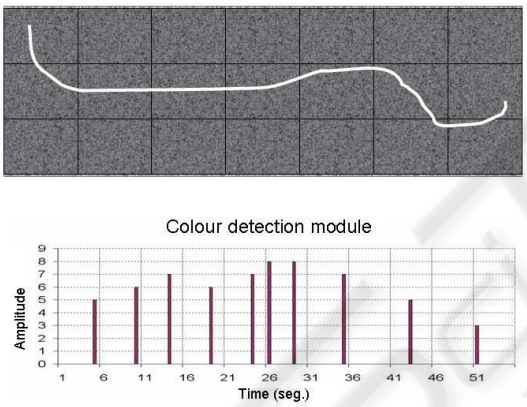

Figure 2: Color detection component execution chronogram.

5 CONCLUSION

In this paper, the runtime adaptation mechanisms available in CoolBOT have been presented. In CoolBOT, the control of shared resources has been integrated in the facilities offered by the integration framework. If this capacity, is to be used by the programmer, components must be declared adaptive and designed with adaptation capabilities. Adaptive components can coexist with non-adaptive components in the same application. These adaptation mechanisms allows the system to regulate the load that a computational context may provoke on the system or can be used to make room for new components when the computational context changes. The objective has been to introduce mechanisms that must avoid uncontrolled degradation of the system in high load situations, paving the road to achieve more robust systems.

REFERENCES

Coste-Maniere, E. and Simmons, R. (2000). Architecture, the Backbone of Robotic Systems. Proc. IEEE International Conference on Robotics and Automation (ICRA'00), San Francisco.

Domínguez-Brito, A. C., Hernández-Sosa, D., Isern-González, J., and Cabrera-Gámez, J. (2004). Integrating robotics software. IEEE International Conference on Robotics and Automation, New Orleans, USA.

Fleury, S., Herrb, M., and Chatila, R. (1997). GenoM: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 842–848, Grenoble, Francia.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., and Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 9162(10):46–54.

Hernández-Sosa, D. (2003). *Adaptación computacional en sistemas percepto-efectores. Propuesta de arquitectura y políticas de control*. PhD thesis, Universidad de Las Palmas de Gran Canaria.

Jones, S. D. (1997). *Robust Task Achievement*. PhD thesis, Institut National Polytechnique de Grenoble.

Kortenkamp, D. and Schultz, A. C. (1999). Integrating robotics research. *Autonomous Robots*, 6:243–245.

Murphy, R. R. (2000). *Introduction to AI Robotics*. The MIT Press.

Musliner, D. J., Goldman, R. P., Pelican, M. J., and Krebsbach, K. D. (1999). Self adaptive software for hard real-time environments. *IEEE Intelligent Systems*, 14(4):23–29.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., and Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62.

Schlegel, C. and Wörz, R. (1999). Interfacing Different Layers of a Multilayer Architecture for Sensorimotor Systems using the Object Oriented Framework SmartSoft. Third European Workshop on Advanced Mobile Robots - Eurobot99. Zürich, Switzerland.

Steenstrup, M., Arbib, M. A., and Manes, E. G. (1983). Port automata and the algebra of concurrent processes. *Journal of Computer and System Sciences*, 27:29–50.

Stewart, D. B., Volpe, R. A., and Khosla, P. (1997). Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transactions on Software Engineering*, 23(12):759–776.