Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 1 (March), pp. 81-89

Architecting Robotics and Automation Societies over Reusable Software Frameworks: the Case of the G++ Agent Platform

F. Guidi-Polanco, C. Cubillos

Franco Guidi-Polanco, Claudio Cubillos
Pontificia Universidad Católica de Valparaíso
Av. Brasil 2241, Valparaíso, Chile.
E-mail: {franco.guidi,claudio.cubillos}@ucv.cl

Abstract: This work presents a software framework that allows the implementation of societies made-up by autonomous collaborative devices. The framework is structured as a multi-layer reusable architecture, adopting the software agent as the design paradigm. In this paper, an overview of its main features is offered, and an example of its adoption in the design of a robot colony is provided.
Keywords: Global Automation, Robots colony, Agent Architecture, Agent-Based Framework.

1 Introduction

In de last decades, robotics and automation systems have evolved both in "body" and in "soul": physiological functions of robots (like sensing, walking, etc.) have reached a degree of maturity, and control systems allow programming robots as entities that can perform autonomous and proactive behaviours. Today, the research is focused in architecting models of societies made-up by autonomous mobile cooperating devices, performing common goal-seeking tasks. Building software for such kind of systems, however, encompasses several challenges, such as how to increase efficiency in the development of control modules (avoid building always from scratch or unnecessary replicating code), how to the deploy software updates in geographically distributed devices, or how to provide data interaction between mobile robots over unreliable communication channels, among other common development problems.

In our vision, the agent paradigm is envisioned as the software abstraction for development of societies of distributed autonomous collaborative systems. Under such paradigm we have designed an agent-based framework providing a reusable architecture that addresses common issues in robotics and automation systems.

This paper offers a description of the state of the art in software architectures for robotics and automation systems, and describes our agent-based framework architecture (the G++ Agent Platform).

2 Related Work

The multi-agent system (MAS) paradigm is being adopted to implement control and communication in distributed automation and robotic societies. In such systems, the modelling paradigm is centred in the concept of agent. An agent is a software entity capable to perceive its environment, to evaluate these perceptions against some given design objectives, and to perform some activity in order to reach them, interacting with other similar entities, and acting over its environment. Agents should be designed to exhibit robust operation, even if they are immersed in an open or unpredictably changing environment [18].

In recent years the literature offers several examples of multi-agent architectures and organizations created for domain-specific applications (see [8], [11], [15] for some examples). These architectures accent the identification of agent's roles and responsibilities, and the description of their interactions and communications. As expected, due to their ad-hoc nature, these architectures are hardly reusable outside their original domains.

In order to improve the reuse of design, some studies establish the convenience of identifying and separating domain-specific aspects from those generic aspects that are common in families of systems. One example is the orientation followed in [16] that proposes the reuse of organizational coordination mechanisms across different problem domains and environmental situations. Nevertheless, their work just emphasizes organization and distribution of tasks and goals, while the system's structure is not deeply treated. An important contribution, in accordance with the latter approach, is the *holonic* paradigm [17]. This approach, offers an organizational model highly reusable, which can be applied at diverse abstraction levels and replicable in different domains [10]. However, it is a conceptual model that does not specify implementation of concrete services that can be required and reused when developing such systems. On the other hand, models of agent societies and agent platforms implementations play insufficient attention to the agent's environment, which is an essential part in robotic system's structure. In practice agent architectures fail to adequately identify and consider its role. As indicated in [19], popular frameworks minimize the environment reducing it just to a message transport system or to a brokering infrastructure. In terms of structure and services, the development of generic agent platforms (e.g. Jade [2]) presents concrete architectures with high degree of reusability, but made-up by low-granularity components (commonly, basic communication and directory services), that implement commonly agreed abstract models (e.g. FIPA). Also, these platforms are not designed to satisfy security, connectivity, and scalability requirements originated in the robotic and automation domain [7]. The adoption of agent systems as enabling technologies for the development of distributed organizations' infrastructures is currently matter of research. In particular, the agent technology seems not only to satisfy the demand for high flexibility requested by enterprise-wide integration, but also to provide approaches to support autonomous self-configuration and self-adaptability of their activities in their operational environment.

3 An abstract model for agent-based robotics societies

We envision the *agent paradigm* as the software engineering approach to model entities (control modules) in robotic architectures. The arguments in favour of an agent-oriented approach in software engineering for modelling a system can be summarized in the three ideas indicated in [9]: (1) Agent oriented decompositions are an effective way of partitioning the problem space of a complex system; (2) The key abstractions of the agent-oriented mindset are natural means of modelling complex systems; and (3) The agent-oriented philosophy for modelling and managing organizational relationships is appropriate for dealing with the dependencies and interactions that exist in complex systems. Our approach introduces a layered model that identifies and classifies system's components (i.e. agents and services) accordingly with different granularities. Those components and services that share similar levels of reuse from both, the structural and the organizational point of view, are grouped together. The model is build recognizing at its basis the physical environment, which is virtualized in superior levels, making explicit the way in which agents will interact with it.

This vision is constructed as the abstract model depicted in Figure 1. The abstract model is divided by the following five layers:

a) *Environment*: it is composed by physical objects pertaining or observed in the real world (e.g. objects in mobile robot's environment, wired or wireless communication networks, computa-

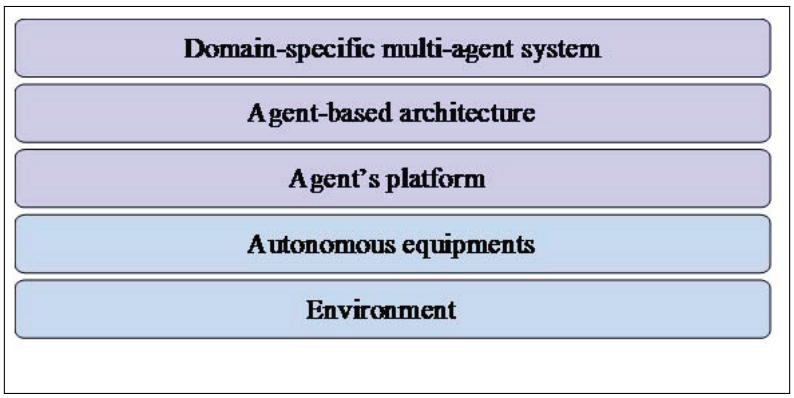

Figure 1: The layers in a robotic society

tional systems in organizations, human operators, etc.), and concepts conventionally adopted for its characterization (e.g. geographical coordinates obtained from a GPS service, temperatures, data transmission latency, water flows measurements, etc.). The physical world is conceptualized as a multidimensional space surrounding agents accomplishing physical-related tasks.

b) *Autonomous Equipments*: represent computing-enabled platforms, such as mobile robots, automated factory machines, or computing devices, which has to be programmed in order to act proactively in the robotic community. Such systems, can offer a wide range of capabilities expressed in terms of CPU, runtime memory, data storage, data communication, or operating system. These equipments are usually provided with sensors that allow the perception of surrounding relevant variables, actuators to interact with the environment (changing their own position, taking objects, etc.), and communications devices to interchange messages with other equipments. Also, the autonomous equipments provide the runtime environment for the agents, so they must satisfy a set of minimum hardware/software requirements imposed by the agent platform's software (or in an opposite point of view, the agent platforms must be designed to be executed in specific categories of devices).

c) *Agent platform*: corresponds to the software that offers the base classes to build agents, and to virtualize environment-dependent services (e.g. interfaces to peripheral devices, motors, databases, network communication, etc.). It also offers the execution environment that controls the entire agent's life-cycle, and regulates its interactions with other agents and resources. A known example of agent platform is JADE [2]. A comprehensive list of agent platforms can be found in [1].

d) *Agent-based architecture*: represents a reusable architecture to support the development of different kinds of agent-based systems. The architecture specifies a set of common services (e.g. directory facilitator, yellow pages, etc.), and a framework of communication/content languages (e.g. ACL [6], KQML [4], etc.) and interaction protocols, necessary to achieve interoperability among agents. An example of a particular agent-based architecture is specified by FIPA standards, which was conceived to obtain interoperability between different and generic agent systems (e.g. FIPA Request Interaction Protocol [5]).

e) *Domain-specific multi agent system* (DSMAS): corresponds to a concrete instance of a multi-agent system, where domain-dependent agents are designed to represent real-world services and systems, and interactions among them are well defined. At this level, agents are often abstractions of real entities pertaining to the application domain. DSMAS architectures can be reused within the scope of the context they were created for. Examples of DSMAS could be a colony of exploration robots, an automated work cell, or a domotic network of devices.

4 The G++ Agent Platform

The G++ Agent Platform runs over a Java Virtual Machine (JVM) hosted in a computational device. The execution environment of the G++ agent platform provides connectivity services, being responsible for the interactions among all agents. It is also responsible for the virtualization of the physical environment, through the implementation of sensors and actuators interfaces that agents can access. The platform provides a Container, which is the environment for the execution of agents. A container runs over a Java Runtime Environment that allows the access to the resources offered by the host. The container presents to the contained agents common services, such as messaging transport, local event communication, and support for access to external data repositories. Containers implement connectivity services among them for message interchange, and for agent and services migration. They also provide connectivity and state monitoring of external agents, and they instantiate proxies to make transparent the communication between external and internal agents.

4.1 The communication infrastructure

Since early stages of the design, this agent platform has been envisioned as the cornerstone of the distributed architecture for automation systems. In particular, under our conception this environment not only corresponds to the space where agents can perform their duties (as all platforms do), it is also aimed to provide a reliable communication infrastructure that agents can (and should) exploit to interact among themselves in a distributed application. As result, the G++ Agent Platform is able to offer an implementation of a robotic and automation system that will delegate to the own agent's container the conduction of the major communication traffic. So agents can communicate among themselves asking their own container to deliver the message to its destination. Messages are delivered following the best effort policy (i.e. no unnecessary delays are introduced in their expedition), but it is not guaranteed their reception in the right order. This can happen for two main reasons: (1) the latency of the Internet, plus costs incurred in retransmissions of packets naturally tends to increase the time required to transmit a message over long distances, and (2) the interconnections between containers define the paths that messages have to follow from the source to the target, each node acting as a router (the processing time on each container has to be added to the network delays described above). The platform, however, can guarantee the delivery of messages, detecting and informing the sender when they are not arrived within the pre-established time. A time window and a timestamp message field are used in the message for this scope. The time window value can also be infinite, which means no time window is specified. The message timestamp can also be useful to the message receiver, to determine the exact sequence of messages.

4.2 Virtual mobility

Virtual Mobility allows agents to be suspended, transported and restored in diverse containers. Mobility can be decided autonomously by the agent, in terms of the moment and destination in which it will be done, or can be enforced by the agent's owner, or by another agent. Mobility is implemented through the serialization of the state of the agent, the transport together with the code (if necessary), and the de-serialization at the destination container. The platform does not provides support for the serialization of the stack of calling methods, so when this procedure is activated, the agent has to be suspended. When an agent is moved from its home container to a foreign container, its original agent management system together with the mobility service is responsible to keep trace of the new position of the agent. In such a way, it is possible to implement automatic roaming in the communication to the agent.

4.3 Interaction with the environment

The structure of an agent considers a subsystem responsible for achieving information from its environment, where the environment can be virtual, composed by software processes or systems running in a computing device, or physical, as the real world is. For example, a virtual agent can be able to listen to keystrokes, listen to messages sent by other agents, receive network information, or perceive events from the operating system; a robotic agent can be enabled with sonars, infrared range sensors, accelerometers or gyroscopes to perceive the physical environment and its own relationship with it. On the other hand, agents must be able to act over its environment in order to achieve their goals. The actions can result in a virtual effect, such as the creation of files, the communication of messages to other agents, or physical, as commands over the engine in a wheel-enabled robot. Sensors and actuators are closely related to the environment because their functionality depends directly on the aspects that they have to detect. In this way, sensors and actuators are device-dependent. However, enabling software agents with specific sensors and actuators can limit their mobility in virtual spaces. The G++ Agent Platform manages sensors and actuators through interface objects that can be attached to agents in runtime. This allows a migrating agent to get access to the specific sensors and actuators offered in each container/platform. This flexibility is obtained providing a common interface for all sensors and actuators that the agent must use to interact with. Also is supported the definition of descriptors to recognize sensors and actuators that the agent could access. The independence between the agent implementation and its environment makes it possible to follow an evolutionary approach in the development of software agents. Portability of agents allows new agents to be tested in simulation environments before they are deployed in the real world (e.g. a mobile robot controller). On the other hand, the model well suits for agents based on learning architectures or requiring initial training (such as those based on neural networks), because they can be conditioned for final execution in a simulated environment.

5 The Agent-Based architecture for automation and robotics

This agent-based architecture introduces a communication standard and a set of services to build global automation systems in different domains. The former defines the languages that will be used for exchange of information between entities participating in automation systems. The latter, the set of services that are available for supporting their activity. Three services are offered at this level:

a) *Messaging*: it provides persistence and reliability in direct messaging between senders and well-defined receivers. It is based on based on persistent messages queues, which allows time-decoupled communications among participants b) *Event distribution*: it implements the asynchronous publish/subscribe communication model. Each container provides local event publication and notification services. The architecture for global automation systems includes agents for the management of distributed subscriptions and notifications. c) *Service brokering*: it supports dynamic reconfiguration of the relationships between service providers and consumers. Each container provides local event publication and notification services. The architecture for global automation systems includes agents for the management of distributed subscriptions and notifications (that is, among different service points). The design of the services has explicitly considered the problem of distribution, particularly the unreliability of network connections, which makes indistinguishable crashed components from slow components. This problem, common to all implemented services, was addresses through a mechanism of registration and renewal of the registration with the service provider, that interested users must perform during their lifecycle.

6 An example of domain-specific multi agent system

For the domain-specific application example, let us consider a colony of mobile robots that have to explore a surface, and cooperate synchronously collecting certain objects, that must be carried to the robots' base. This colony requires the participation of different kinds of autonomous devices:

- Explorers, which are autonomous mobile robots provided with telecameras and grippers, that recognize objects to be collected, and carry them to the nearest transport robot. - Transport robots: they are autonomous mobile robots that receive the objects collected by explorers and transport them in batches to the colony nest. - Coordinator: is the autonomous system that receives communication from carriers and coordinates the operation of explorer and transport robots. It operates in a fixed computing device, which is located outside the exploring area, and connected through satellite to the Communicator robot. - Communicator: is a mobile device that acts as a gateway between the Coordinator and the robots located in the surface to explore. - Colony Nest: is the computing device that runs messaging and brokering services of the nest.

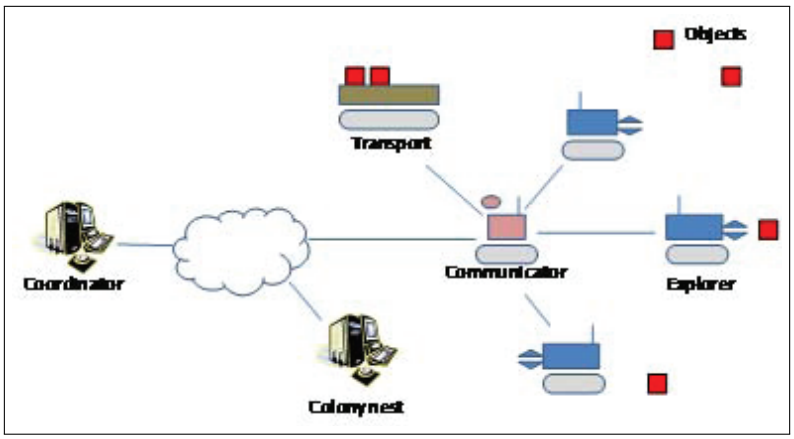

Figure 2: A colony of explorers.

In this system all the members run a container where operates the agent that implements the own control module. At startup, the containers pertaining to all devices in the surface to explore establish communication among themselves, using a peer-to-peer discovering protocol. Then they register their service capabilities in the ServiceBroker running in the Colony Nest device. The coordinator, remotely located, creates a data channel with the Colony Nest, using a known IP direction to locate it. Thus, the coordinator asks the service broker located in the Colony Nest for the suitable robotics devices to accomplish the task. From the answer provided by the service broker, the coordinator selects a Communicator device, some explorers and transport robots. The Coordinator transmits the mission to each robotic device. The explorers move across the surface detecting objects to be collected, and transmit their positions to the Coordinator. With this information, the coordinator assigns routes to the transport robots, in order to pick up the objects collected by the explorers. If an explorer or transport robot goes out of the communication range, it starts a "turn back home" procedure in order to reestablish communication. If messages had been sent to that robot during the "blackout", they are kept in the messages queue of the MessagingService and transmitted when the device is "visible" again. The Coordinator could be subscribed in the event broker to listen for certain events, such as mechanical failures. Thus, if a failure message is received, the coordinator can take measures like a replacement for the defective robot. In the following subsections, the system is described using the model presented in this chapter.

6.1 Environment and autonomous equipments (levels 1 and 2)

The environment is mainly composed by the physical surface that have to be explored, the objects that have to be collected, the obstacles in the route of each robot, etc. Also, available communication networks or global positioning systems that can be accessed by devices are considered part of the environment. In terms of autonomous equipments we have all the devices participating in the colony, including the Coordinator and the Colony Nest devices. Robots are mobile vehicles enabled with sensors (cameras, GPS, encoders, etc.), actuators (engines, grippers, etc.) and communication interfaces (Wi-Fi, satellite, etc.) that must be available for local control modules (robot controllers).

6.2 The agent platform (level 3)

The G++ Agent Platform is used to provide the underlying software infrastructure for the implementation control modules. Each device must provide a Java Runtime Environment where a G++ Container will run. Also, in each robotic device the agent platform must have access to the API (application programming interface) of the available sensors and actuators, in order to build the access to physical components of the robots. The access to the communication stack is obtained, in general cases, through the standard TCP/IP interface provided by the device's operating system.

6.3 Agent-based architecture (level 4)

The agent based-architecture is made-up by all the components required to achieve interoperability among the different participants. For example, the ServiceBroker agent, which is responsible for maintaining the network location (IP address) of each robotic device, and the list of services that them are capable to provide. Another agent that participates in the architecture is the MessengerAgent that supports message queues for reliable delivery of message to mobile robots.

6.4 The domain-specific multi-agent system (level 5)

Each participant in the colony is conceptualized as a software agent, programmed to accomplish its own mission. The implementation requires providing every one of the devices with an agent/control. The control modules can be implemented using different artificial intelligence model. At this level must be also programmed the standard interfaces to the sensors and actuators, and registered locally as service objects in the device's container.

7 Conclusion

In this work we have described our framework for the implementation of distributed robotics and automation systems. Its design was driven by the interest to obtain a decoupled and scalable infrastructure in different application scenarios. This approach emphasizes software engineering aspects of agency, which is a differentiating point when comparing it with other architectures, whose functionalities are more focused in distributed artificial intelligence. Currently we are developing some case of studies that can help us to test the framework in systems offering different complexity levels.

8 Acknowledgement

This work has been partially funded by CONICYT through Fondecyt Project No. 11080284 and the Pontificia Universidad Católica de Valparaíso (www.pucv.cl), through Nucleus Project No. 037.115/2008 "Collaborative Systems".

Bibliography

[1] AgentLink. *Software Products for MultiAgent Systems*. Technical Report. Europe's Network of Excellence for Agent-Based Computing, 2002.

[2] F. Bellifemine, A. Poggi, G. Rimassa. Jade, a FIPA-Compliant Agent Framework. Proceedings of the 4th Int. Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology, 1999.

[3] P. Eugster, P. Felber, R. Guerraoui, A. Kermarrec. The Many Faces of Publish/Subscribe. ACM Computing Surveys, Vol. 35, No. 2 (June 2003), p. 114-131.

[4] T. Finin, D. McKay, R. Fritzson, R. McEntire. KQML: An Information and Knowledge Exchange Protocol.Proceedings of the Int. Conference on Building and Sharing of Very Large-Scale Kniowledge Bases, December 1993.

[5] FIPA (2002). FIPA ACL Message Structure Specification. Standard N. SC00061G, December 2002.

[6] M. Genesereth, S. Ketchpel. Software Agents. Communications of the ACM, Vol 37, No. 7, 1994, p. 48-53.

[7] F. Guidi-Polanco, C. Cubillos, G. Menga. The Agent-Based GAP: A Framework for Global Automation Systems, Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, p.53-58, June 14-16, 2004

[8] Z. Haibin. A Role-Based Approach to Robot Agent Team Design. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics SMC, 2006. Vol.6 pp.4861-4866, October 2006.

[9] N. Jennings. An Agent-Based Approach for Building Complex Software Systems. Communications of the ACM, Vol 55 No. 4 (April 2001), p. 35-41.

[10] L. Jianhui, W. Kesheng, G. Hang, Q. Ligang. An OOT-supported migration approach to holonic robot assembly cell. Proceedings of the 8th Int. Conference on Computer Supported Cooperative Work in Design, 2004. Vol.2 pp. 498-501.

[11] C. Lim, R. Mamat, T. Braunl. Market-based approach for multi-team robot cooperation. 4th International Conference on Autonomous Robots and Agents, ICARA 2009, pp.62-67, Feb. 2009.

[12] D. Mamady, G. Tan, M. Toure. An artificial immune system based multi-agent model and its application to robot cooperation problem. Proceedings of the 7th World Congress on Intelligent Control and Automation, WCICA 2008, pp.3033-3039, June 2008.

[13] G. Moro, A. Natali. On the Event Coordination in Multi-Component Systems. Proceedings of SEKE 2002, Ischia, Italy.

[14] J. Odell, H. Van Dyke Parunak, B. Bauer. Extending UML for Agents. Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence 2000.

[15] T. Rogers, A. Sekmen, J. Peng. Attention Mechanisms for Social Engagements of Robots with Multiple People. The 15th IEEE International Symposium on Robot and Human Interactive Communication, ROMAN 2006, pp.605-610, Sept. 2006.

[16] M. Sims, D. Corkill, V. Lesser. Separating Domain and Coordination in Multi-Agent Organizational Design and Instantiation, Proceedings of the International Conference on Intelligent Agent Technology, IAT 2004.

[17] P. Valckenaers, H. Van Brussel, T. Holvoet. Fundamentals of Holonic Systems and Their Implications for Self-Adaptive and Self-Organizing Systems. Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2008., pp.168-173, Oct. 2008.

[18] G. Weiss. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge, Massachusetts, 1999

[19] D. Weyns, M. Schumacher, A. Ricci, M. Viroli, T. Holvoet. Environments for multiagent systems, State-of-the-art and research challenges. In Lecture Notes in Computer Science, vol 3374 (2005), Berlin, Heidelberg, Germany.