We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

7

An Agent-Based Software Framework for Robotics and Automation Systems

Franco Guidi-Polanco and Claudio Cubillos
Pontificia Universidad Católica de Valparaíso
Chile

1. Introduction

The concept of "reuse" in software engineering is associated to well design, shorter development times, and easier maintenance of software applications. Today, the bigger reuse unit is given by software frameworks, which offer ready-to-use architectures and code implementations. Several frameworks have been proposed and adopted for a wide variety of traditional application domains (i.e. graphic interfaces, data persistence, web applications, etc.). The robotics and automation is a complex domain where the orientation to get reusable software architectural design has became a center of interest just in the last decade, once complex physiological functions of robots (i.e. sensing, walking, thinking, etc.) have reached certain degree of maturity.

In the field of robotics and automation, current application scenarios consider distributed autonomous cooperative systems, especially aimed to support integration of collaborative societies of devices. Thus, the paradigm is shifted from the single entity that establishes simple perception-planning-reaction interactions with its environment (i.e. detect signals, path planning, reach places), to a colony of autonomous members forced to interact among them in order to accomplish more complex tasks that are unable to be managed solely for each single one. The concept of "member" is used in this context to encapsulate each physical device or virtual process recognizable in the society, which pursues its own individual objectives.

In our vision such systems are conceptualized as a flat interconnection of autonomous and decentralized virtual and robotics agents, where no control hierarchy is enforced, and where each partner takes the initiative to reach to a decision. Agents interact in a peer-to-peer architectural model, and the global behaviour of the system becomes a synergic property of the interaction of their parts.

This work presents a software framework for building automation systems, which promotes different reuse levels. The framework offers a general layered architecture driven by the paradigm of *software agent*. The framework includes an agent platform that satisfies specific requirements in software development for communities of robots and automation devices.

In this paper the architecture is described with more detail, and an example of its application is provided.

Source: Convergence and Hybrid Information Technologies, Book edited by: Marius Crisan,
ISBN 978-953-307-068-1, pp. 426, March 2010, INTECH, Croatia, downloaded from SCIYO.COM

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 (Weiss 1999).

In recent years the literature offers several examples of multi-agent architectures and organizations created for domain-specific applications (see (Haibin, 2006), (Dioubate et al. 2008), (Lim et al., 2009), (Rogers et al., 2006) 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 (Sims et al., 2004) 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 (Valckenaers et al. 2008). This approach, offers an organizational model highly reusable, which can be applied at diverse abstraction levels and replicable in different domains (Jianhui et al., 2004). 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 (Weyns et al., 2005), 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 (Bellifemine et al. 1999)) 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 (Guidi_Polanco et al. 2004).

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 (Rimassa, 2004), 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

In the era of Internet, robotic and automation systems are conceived as flat interconnections of autonomous and decentralized decision making/control modules. In such a system, no hierarchy in the decision making is enforced, and each partner takes the initiative to reach a decision. Control modules have decision-making capabilities and coordinate their activities by exchanging data and events according to a peer-to-peer architectural model and common protocols (Brugali & Menga, 2002).

We envision the *agent paradigm* as the software engineering approach to model control modules in such 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 (Jennings, 2001): (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.

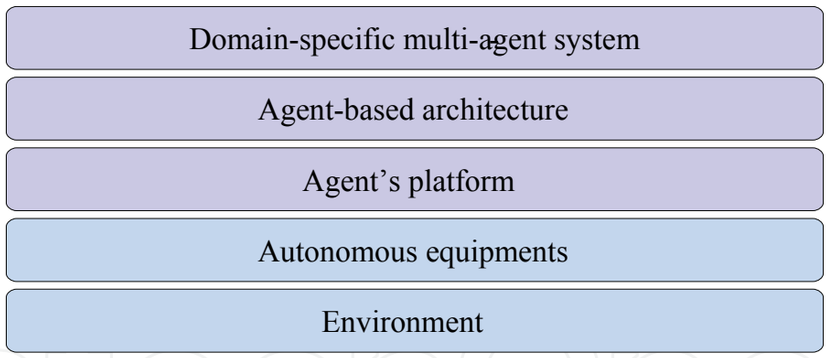

Fig. 1. The layers in a robotic society

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, computational 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. As it was stated above, agent platforms must be designed for their execution in devices with different hardware (e.g. PDAs, mobile phones, desktop computers) and software capabilities (in terms of operating systems and programming languages). A known example of agent platform is JADE (Bellifemine et al., 1999). A comprehensive list of agent platforms can be found in (AgentLink, 2004).
d. *Agent-based architecture*: represent 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 (Genesereth and Ketchpel, 1994), KQML (Finin et al., 1993), etc.) and interaction protocols, necessary to achieve interoperability among agents. The services offered by the architecture can be implemented by service agents (such as a yellow-pages agent), or as environment-dependent service (e.g access to some kind of physical device). 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 (FIPA, 2002)).
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. A reusable DSMAS architecture constitutes an agent-based framework for the development of systems within its domain. DSMAS are supported by the services offered by the agent-based architecture. Examples of DSMAS could be a colony of exploration robots, an automated work cell, or a domotic network of devices.

The model proposed above has three main characteristics:

- *Decouples design responsibilities*: the model presents the different aspects related to a multi-agent architecture in a separated way. Therefore, the design responsibilities can be clearly identified and assigned to different development projects or teams.
- - *Promotes high cohesion within each layer*: components within each layer are closely related from the functional and communicational point of view, in such a way that their interactions are optimized.

- *Clearly emphasizes the environment*: traditional agent architectures consider the environment implicitly, in most cases just as a mere communication supplier. In our model, the environment is distinguished as a physical and a virtual one.

4. The G++ Agent Platform

We have developed the G++ Agent Platform, our own software infrastructure for agents' implementation in robotic and automation societies (Guidi-Polanco et al., 2004). In this section, the architecture of the platform is described.

4.1 Design directions

Our work was motivated by the need for creating and integrating autonomous systems through geographical scale cooperation networks, so the following directions guided the design of the G++ Agent Platform:

- *Support for heterogeneous execution hosts:* the size and weight of computers have become considerably smaller, and mobile computers have reached the performance only seen before in desktop computing systems, increasing the range of devices that can be integrated in a distributed automation society. It can include robots, autonomous sensors, PDAs, mobile phones, among others.
- *Support for physical mobility*: some control modules in robotics and automation system are expected to be able to change its position at geographic scale. For example they can run in portable devices carried by its users (e.g. PDAs), or they can be part of inherently mobile systems (e.g. on-board computers in vehicles). This means that connectivity has to be implemented in most cases through wireless networks, and then associated problems such as limited bandwidth and continuity of communications, must be addressed.
- *Support for heterogeneous (wireless) networking*: wireless communication is supported currently by a variety of networking technologies, offering diverse conditions, such as area coverage, bandwidth, cost, or QoS. Even more, not all of the available technologies are present in all geographical places, or they are not always offered with the same configuration at the physical layer. The infrastructure for a global automation system does not have to bet to a convergence in a unique and global-wide technology, but instead it has to be able to manage heterogeneity.
- *Support for heterogeneous systems and resources*: the development of large-scale systems usually requires the integration of new and legacy enterprise resources, such as database systems or old applications. Two strategies are commonly applied to face this integration, if rewriting the application is not possible: wrapping the old application through an extension of its code that allows direct interaction with the external system, or implementing a transducer, which is an interface that translates the external messages in a form suitable for the legacy system, and vice-versa.
- *Support for geographical-scale distribution*: an automation system can integrate systems located in separate geographical places. Although in these days, such integration is possible due to the worldwide coverage of the Internet, it is important to deal with latency times in communications that can be significant in some applications (e.g. direct teleoperation of a robot).

Under such a scenario, the possibility of letting each agent with all the responsibility for its integration with the environment (and consequently, with other peers) implied the agent

overload and the replication of complex interaction functionalities. Existing platforms are not suitable to accomplish these requirements.
In the design of the G++ Agent Platform the above requirements are met. Reusability is a property we seek in this architecture, because it has to be applied in different context of robotics and automation, for example the operation of a colony of robots, the organization of virtual teams, or the integration of large-scale inter-factory logistics, among others.
The structure of our agent platform can be appreciated in Figure 2. It is important to state that the G++ Agent Platform is an agent infrastructure not committed to any standard agent architecture (e.g. FIPA), even if compatibility with standard specifications can be obtained adding compatibility modules.

4.2 The architecture of 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 offers two kinds of execution environment implementations, the Container and the Legs module.

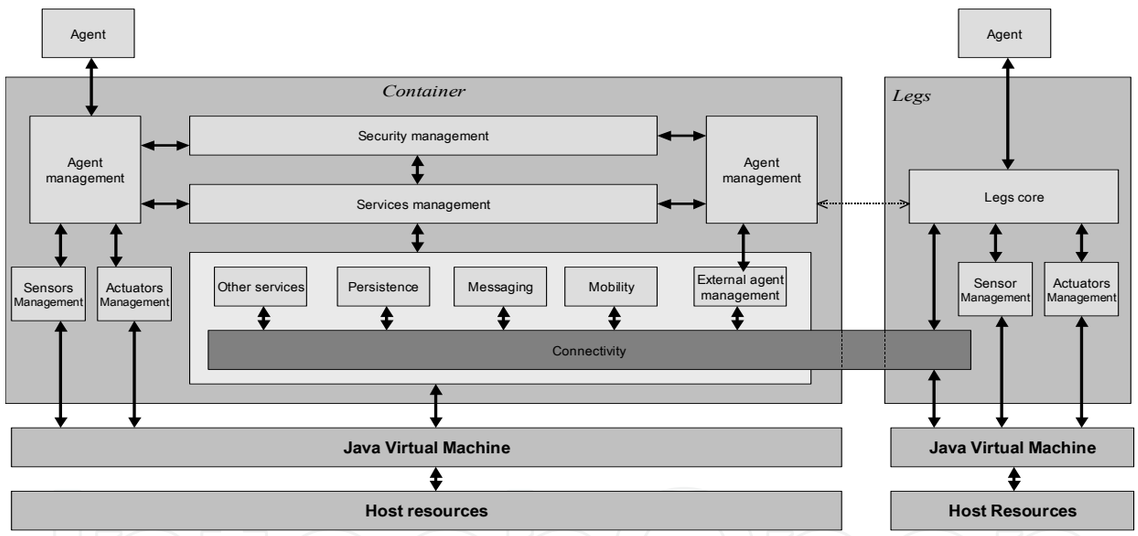

Fig. 2. The G++ platform's architecture.

a. Container

It is the environment for the execution of contained agents. A container runs over a Java Runtime Environment, which 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.

b. Legs module

The platform can integrate agents running outside the container. These agents are called external agents or stand-alone agents.

The execution of external agents is allowed by Legs (Local External aGent Support) modules, which are limited execution environments able to host and execute one agent at time. They provide connectivity to a container, and then, to the entire platform. As contained agents, external agents can access all the services provided by the underlying JVM, and some of the communication services offered by the container, but they cannot access other services, such as the agent mobility. External agents can be useful, for example, for the implementation of control systems running on-board of mobile devices with limited capabilities.
The implementation of external agents follows the same structure given for the implementation of contained agents. In fact, if a contained agent does not use resources restricted to contained agents, or host's special resources, it can be transformed in an external agent just launching it from a LEGS module.
In the following subsections, main aspects of the platform are described.

4.2.1 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.
The timestamp is a key data to support the quality of the messaging service, but its generation is not easy because requires the adoption of a global time, shared among containers.

4.2.2 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.

4.2.3 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. In fact, as it is stated in (Arsten et. Al, 1996) complex systems often require development of prototypes and the simulation of the execution. 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.

4.2.4 Security

The platform security is focused in the protection of the hosting platform (the container and its agents) from the attacks of potentially malicious visiting agents. The protection of the host is mainly based on trust, and this allows the adoption of a partially open distributed platform. It is *open* because it is possible to add new containers to the network, and each container can admit external agents, coming from other containers of the network. However, this openness is *partial*, because only authorized containers are accepted to join the network, and, potentially, only authorized agents are allowed to visit each host. This approach supposes the container to provide at least two security services, authentication and authorization. Both services are supported by a public key infrastructure (PKI).

5. The Agent-Based architecture for automation and robotics

In this section is described the agent-based software architecture for automation and robotics systems. The architecture follows the abstract model described in Section 3. It provides a set of agent-based meta-services to support advanced communication of the domain-specific systems built on top of it.

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.

5.1 Messaging service

The messaging service implements reliable messaging delivery among agents, based on *Messenger agents* that extend basic communication capabilities of the G++ Agent Platform. The implementation of the messaging services requires providing each container of the agent society with a messenger agent that interfaces communicating agents with the service. Communicating agents that require to be supported by this service are requested to register themselves with the messenger agent, which maintains a list of agents that are subscribed for the service. Thus, the messenger agent only accepts messages having as target a registered agent, and rejects other messages. Each registration has as parameter the duration of the registration, which represents for how long the agent is interested in being supported by the messaging service. Therefore, the messaging service will be active for each specific communicating agent accordingly to the duration indicated in the registration, but in any moment registrations can be renewed for new periods. The messenger agent accepts messages sent by local agents (i.e. agents pertaining to the same container of the messenger), and delivers them to other agents residing in remote containers. It offers two modalities for delivery: normal delivery, that means the sender only receives an acknowledgement from the local messenger agent indicating that the message has been received by the messaging

service, and notified delivery, that allows the sender to receive a notification when the message has finally reached the receiver.

The messaging service is performed through different interaction protocols that regulate the possible conversations between senders and receivers of messages and the messenger agents. Such protocols are:

a. *Subscription request*: performs the registration of an agent with the local messenger for a given period of time.
b. *Subscription renewal*: allows an agent the renewal of a subscription with the messenger for a new period.
c. *Subscription cancel*: an agent subscribed with a messenger can cancel its subscription in any moment, sending a cancel request message.
d. *Activate delivery*: after registration, an agent can request the activation of the message delivery, sending an activate delivery message to the messenger, so queued messages will be delivered to the requesting agent, and further messages received by the messenger will be delivered instantaneously.
e. *Suspend delivery*: an agent can request suspension of delivery of incoming messages at any moment, which means that the messenger agent will stop delivering messages addressed to that agent through it.
f. *Send message*: it is used to transmit a message to a receiver agent using the messaging service supported by messenger agents.
g. *Message Transfer*: it is used by two messengers when exchanging queues of messages.
h. *Message delivery*: this protocol regulates the conversation between a regular agent subscribed to the messaging service and the related messenger agent that has messages to deliver to the former.

5.2 Event distribution service

The communication among components is one of the important problems faced in the development of distributed systems. This is a characteristic of the architectural style of an application, and it can be implemented adopting two approaches (Moro & Natali, 2002): *request/response* and *publish/subscribe*.

The request/response paradigm, is widely adopted in traditional client/server distributed systems such as Web-based applications. However, it results not always adequate, particularly when applications need to continuously collect data generated from large-scale distributed sources, because the network could be overloaded with a high traffic of request and responses. Moreover, if the application includes components running on mobile systems, implementing complete cycles of polling could spend unnecessarily the limited power resource of the device, or could increase the expenses associated to communication traffic.

On the other hand, the publish/subscribe paradigm is claimed to provide the loosely coupled form of interaction required in large-scale systems (Eugster et al., 2003). In this model, components acting as subscribers have the ability to express their interest in some typologies of messages. Thus, they are subsequently notified when publishers generate the messages that match their interest. Using this approach, the communication between publishers and subscribers becomes loosely coupled because both participants do not need to know anything about each other. Communication services implemented over this architecture are usually known as event services.

The core of the event management in our architecture is the *EventBroker* agent, responsible for collecting subscriptions and sending events to the registered subscribers. Subscribers register their interest on events sending a *subscribe* message to the EventBroker agent, without the need to know the effective sources of these events. This subscription information remains stored in the EventBroker, and it is not forwarded to publishers. The event service also provides an *unsubscribe* operation that terminates a subscription. The subscription contains the following information: (1) typology of event of interest; (2) optionally the source of interest; and (3) duration of the subscription.

5.3 Service brokering service

Collaboration among distributed and autonomous control modules is the final objective of our robotics and automation platform. Thus, the whole structure of the framework is built around the idea of a reliable infrastructure for service integration. In part, this can be understood as the objective of classical networking infrastructures, such as DCOM, RMI or CORBA, which is the problem of finding and invoking remote services. However, our architecture does not oversimplify the relationships between the network and the applications, as the cited technologies do. The latter means that the network is seen as a not completely transparent environment, that is, mainly subject to a lack of reliability, with latency and limited bandwidth, in a mutable topology. Our framework explicitly considers that control modules can crash and the system should be aware of such services that become no longer available.

The services brokering service is aimed to establish relationships among the three entities called *client, server* and *service*. The service represents the object of interest that justifies the interaction between the other participants. Services are offered by agents or systems that play the role of servers, which are also responsible for providing, and if it is necessary, for scheduling the accesses to the resources need to perform the service. For example, in an image capture and transmission service, the robotic telecamera agent (server) is responsible for providing and managing the access to the telecamera requested to give the service. In our model, a server is completely free to offer whatever configuration of services, this means that a server can offer only one typology of service, or a wide variety of them. For a specific typology of service, a server could offer concurrently only one instance of service, that is, it can serve just client at time, or it can perform multiples executions simultaneously, depending on the specific implementation and the possibility to share the involved resources (such as external devices, CPU, memory, etc). Any server that joins the system and intends to let clients use its services, must clearly declare this intention by making a long term commitment to taking on a well-defined class of future requests. This declaration is called an *advertisement* and contains a specification of the server capability with respect to the type of request it can accept.

The *ServiceBroker* is the agent of our architectural model that manages a database of advertisements, i.e. it knows the name and location of registered servers, their capabilities, the service interfaces, and the supported communication protocols. When a client agent needs a service that implements a specific interface with specific capabilities, it queries the ServiceBroker for the name of all the available servers in the system that provides that capability. Clients are not supposed to have knowledge about the location of the services

they require, they only have to know the name of the ServiceBroker, and direct their requests to it.

The GAP framework adopts the Extensible Markup Language (XML) as the content language to specify capabilities and preferences and to exchange information among distributed control modules over the platform. Description of services expressed in XML documents are exchanged during interactions among interacting agents. The messages that interchange the ServiceBroker with other agents are:

a. *Service advertisement*: it is sent by a server to the ServiceBroker describing the service it want to publish and how long it will be available.
b. *Service request*: contains a description that the client sends to the broker, regarding the characteristics of a requested service.
c. *Broker response*: it corresponds to the answer that the broker returns when dealing with a request coming from a client, indicating references to the services that can satisfy it.
d. *Service advertisement renewal*: allows a server to renew for another period its registration in the ServiceBroker.
e. *Advertisement cancellation*: it is sent by a server to the ServiceBroker that wants to cancel a previous service advertisement.

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.

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.

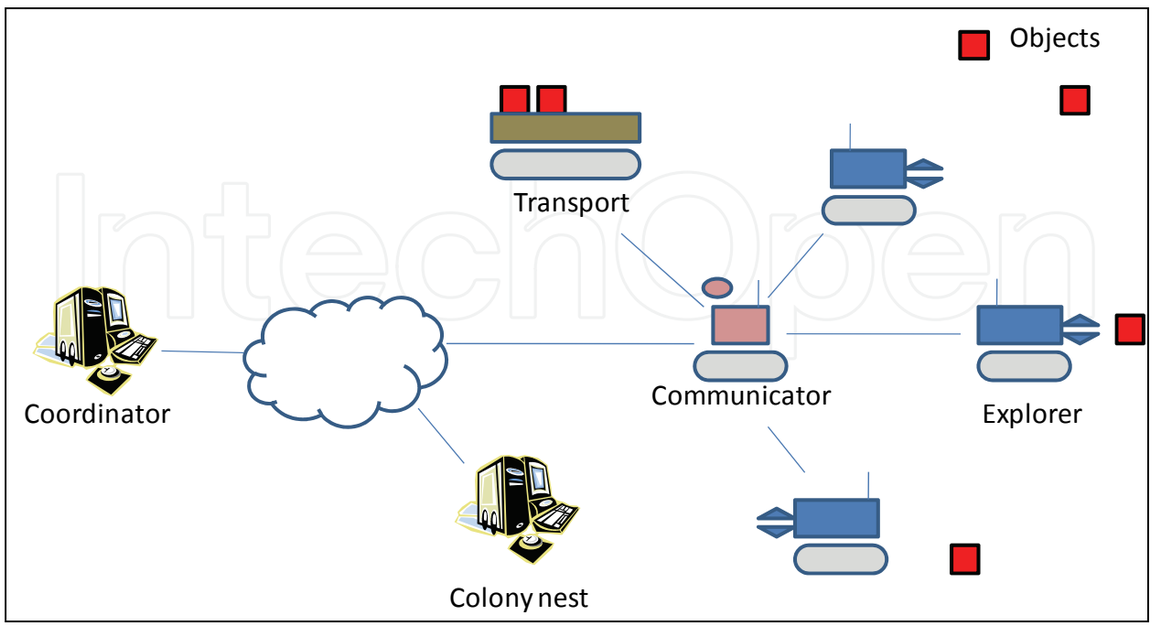

Fig. 2. A colony of explorers.

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. References

Aarsten, A.; Brugali, D. & Menga G. (1996). Designing Concurrent and Distributed Control Systems: an Approach Based on Design Patterns. *Communications of the ACM*, Vol.39, No. 10 (October 1996), pp. 50-58.

AgentLink (2002). Software Products for MultiAgent Systems. Technical Report. *Europe's Network of Excellence for Agent-Based Computing*.

Bellifemine, F.; Poggi, A. & Rimassa, G. (1999). Jade, a FIPA-Compliant Agent Framework. *Proceedings of the 4th Int. Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology*, 1999.

Eugster, P.Th.; Felber, P.A., Guerraoui R. & Kermarrec A.M. (2003). "The Many Faces of Publish/Subscribe". *ACM Computing Surveys,* Vol. 35, No. 2 (June 2003), p. 114-131.

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

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

Genesereth M.R. & Ketchpel, S.P. Software Agents. *Communications of the ACM,* Vol 37, No. 7, 1994, p. 48-53.

Haibin Z. (2006). 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.

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

Jianhui, L.; Kesheng, W.; Hang, G. & Ligang, Q. (2004). 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.

Lim, C.S.; Mamat, R. & Braunl, T. (2009) Market-based approach for multi-team robot cooperation. *4th International Conference on Autonomous Robots and Agents, ICARA 2009*, pp.62-67, Feb. 2009.

Mamady, D.; Tan, G.; & Toure M. L. (2008). 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

Moro G. & Natali A. (2002). On the Event Coordination in Multi-Component Systems. *Proceedings of SEKE 2002*, Ischia, Italy.

Odell J.; Van Dyke Parunak H. & Bauer B. (2000) Extending UML for Agents. *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence 2000.*

Rogers, T.E.; Sekmen, A.S. & Peng, J. (2006) 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.

Sims, M.; Corkill, D. & Lesser, V. (2004). Separating Domain and Coordination in Multi-Agent Organizational Design and Instantiation, *Proceedings of the International Conference on Intelligent Agent Technology, IAT 2004.*

Valckenaers, P.; Van Brussel, H. & Holvoet, T. (2008). 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

Weiss, G. (1999) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, 1999

Weyns, D.; Schumacher, M.; Ricci, A., Viroli M., & Holvoet T. 'Environments for multiagent systems, State-of-the-art and research challenges'. In Lecture Notes in Computer Science, vol 3374 (2005), Berlin, Heidelberg, Germany.

Convergence and Hybrid Information Technologies

Edited by Marius Crisan

ISBN 978-953-307-068-1
Hard cover, 426 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

Starting a journey on the new path of converging information technologies is the aim of the present book. Extended on 27 chapters, the book provides the reader with some leading-edge research results regarding algorithms and information models, software frameworks, multimedia, information security, communication networks, and applications. Information technologies are only at the dawn of a massive transformation and adaptation to the complex demands of the new upcoming information society. It is not possible to achieve a thorough view of the field in one book. Nonetheless, the editor hopes that the book can at least offer the first step into the convergence domain of information technologies, and the reader will find it instructive and stimulating.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen