# Middleware for Robotics in Assisted living: A case study

**Tizar Rizano**
**Luca Abeni**
**Luigi Palopoli**
DISI - Università di Trento
Via Sommarive 5 38123 - Trento
tizar.rizano@disi.unitn.it, luca.abeni@unitn.it, luigi.palopoli@unitn.it

**Abstract**

In the FP7 E.C. DALi project, we pursue a mobility aid device – the cWalker – that supports navigation in crowded and unstructured spaces by acquiring sensory information by anticipating the intent of human agents and by deciding the path that minimises the risk of accidents.

To achieve its ambitious goals the cWalker relies on a multi-disciplinary effort tapping different areas of expertise. The resulting complexity of the system engineering motivates the use of a middleware combining a usable and intuitive API with acceptable latencies to permit its use in a system with challenging real–time constraints. In this paper, we report a performance evaluation of three interesting technological middleware solutions available off-the shelf that could potentially meet the demanding requirements of the cWalker and, more in general, of a new generations of robotic applications.

## 1 Introduction

The recent developments in sensing and battery technologies and in embedded computing devices are creating the premises for the development of low cost robotic applications for a consumer market. The ever-increasing presence of robot vacuum cleaners in our homes, of robotic toys amusing our children, of robotic drones shooting impressive pictures from surprising points of view are witnesses of a clear market trend. At the forefront of this movement are robots created to assist older adults or people with different disabilities. One of the basic needs that can effectively be addressed by assistive robots is personal mobility. In this class of application we find the the cognitive walker (cWalker), designed to assist adults with non-severe cognitive abilities in the navigation of complex and crowded environments (e.g., an airport or a mall), which challenge the sense of direction and generate anxiety. This is a paradigm of a much wider class of complex robotic applications that are called to operate in real–time with the environment and interact with humans or with different devices.

The c-Walker integrates several modules and relies on different types of sensors that convey information on the surrounding environment. In particular, the position and the velocity of human by-standers in the environment is detected by video sensors, while gyroscopes encoders, 3D cameras and RFID readers are used to localise the c-Walker within a map. The same level of complexity is on the software architecture, that is comprised of modules for video-analyis, mission planning, short term planning and control. These services interact with a geo spatial database that store relevant information about the environment. The geo spatial database maintains a consistent description of the environment, where each model inserts additional information layers.

The integration of this complex network of modules calls for a middleware solution striking a good tradeoff between conflicting needs such as: modularity, architecture independence, re-use, easy access to the limited hardware resources and real–time constraints.

Three middleware architectures proposed in the last years (each one with a well maintained binding on the Linux Kernel and on the most used network protocols) offer reliable and easy to use abstractions

and intuitive publish-subscribe mechanism that can simplify the development of complex robotic application like DALi to a good degree. The first one is Open Data Distribution Service (OpenDDS) [9], which implements a standard proposed by the Object Management Group[10]. The second alternative that we have considered is ZeroMQ [7], which implements a publish-subscribe paradigm to support concurrent programming over socket connections using a publish-subscribe paradigm and is freely available from its website [13]. The third middleware is Open Real-Time Ethernet (ORTE) [12] and implements a publish-subscribe mechanism over a real–time ethernet connection (in particular, it is compliant with the RTPS - Real-Time Publish-Subscribe - 1.0 protocol). The three solutions have different reasons of interest. OpenDDS builds on top of the decennial experience made by the CORBA community and offers powerful abstractions. ZeroMQ is extremely lightweight and potentially interesting for its easy adaptation to embedded architectures. ORTE is a product has been developed for a special care for its real–time performance.

Putting aside the different historical background of these solutions, their compliance with the different requirements of complex robotics application (first and foremost real–time constraints) remains to be tested on the field, and is the objective of this paper. In particular, we evaluate the performance of the three middlewares in terms of latency, maximum connections, and processor utilisation. This comparison is used a a cornerstone for the development of a reliable software architecture for the cWalker and can pave the way for the introduction of real–time middleware in a large class of robotic applications.

The paper is organised as follow. In Section 2, we offer an overview of our case study. In Section 3, we shortly describe the three middleware analysed in the paper and compare their features. In Section 4, we report our results on the performance comparison between the three different alternatives. In Section 5, we state our conclusions and discuss future work directions.

## 2 The Case–Study

An important motivational example for this work has been offered us by a cooperative European project [3] coordinated by the University of Trento. The objective of the project is the development of a robotic assistant to help older adults with emerging cognitive impairments navigate large and challenging environments (e.g., a shopping mall, or an airport). Because the main focus of the project is to compensate for cognitive deficiencies, the assistant is called cWalker (cognitive walker) A simplified scheme of the most important functionalities of the cWalker is shown in Figure 1. The cWalker prompts the user for a sequence of target points in the environment that he/she wants to visit through a visual interface. The Long Term Planner finds the most convenient path using the map of the environment and the real–time information on the state of the place, which is acquired querying remote sensors (e.g., the surveillance cameras). When the users starts to move following, the walker guides her/him along the path using electro-actuated brakes [4], haptic interfaces and audio/video interfaces. The guidance requires a real–time localisation system which tracks the position of the cWalker while it moves. Along the way, the cWalker localises the user in the environment, detects anomalies and the motion of people in the surroundings and plans deviation from the planned path when required (e.g., to avoid accidents or such behaviours as could violate the social rules). This is done by a Short–Term planner.

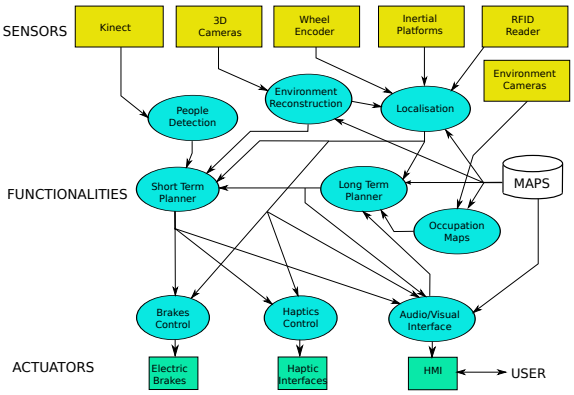


**FIGURE 1:** *Simplified functional scheme of the DALi cWalker*

A complete description of the different functionalities is beyond the goals of the present paper. We offer some additional details for a few in order to illustrate the requirements of the middleware.

**Localisation.** The ability of the cWalker to localise itself in the environment is key to most of its goals and is used by the motion planner (both short term and long term) and by the guidance systems (brake control, haptic control and audio visual interfaces). This localisation module uses a combination of a relative positioning system, which utilises encoder and inertial sensors to track the motion of the cWalker from a specified position, with two absolute positioning systems, which respectively utilise an RFID reader and a 3D camera to construct a model of the surroundings and match it with a-priori knowledge

of the place (e.g., the presence of landmarks). The relative positioning system fuses the information of two position encoders mounted on the wheels and of two inertial platforms (embedding accelerometers and gyroscopes). This system produces a new estimate of the position every $5ms$. The absolute positioning system is activated on the occurrence of events (e.g., the presence of a RFID tag underneath the carpet), or with a fixed periodicity (in the order of seconds). The interested reader is referred to a recent paper describing the system [8].

**Long Term Planner.** The long term planner produces a sequence of via points from a set of desired locations, a map of the environment (coded in a GIS database) and the occupation maps, which relate the different location of the map with an estimate on the density of people. This plan is produced on the activation of the system (in response to the user input) and with a fixed periodicity during the operation of the system (in the order of ten seconds). The long term plan is used by the short term planner and by the user interface.

**Short term planner.** The short term planner finetunes the plan considering the presence of obstacles along the way. In particular, it receives information from a Kinect based people tracker on position and velocity of human agents in the proximity of the cWalker. This information is propagated ahead in time to predict the future positions of the agents in the horizon of a few seconds using a stochastic model based on the social force model [6]. The planner seeks the minimum change to the path than meets a specification on the probability of accidents [2]. The tracker produces a new estimate every 100ms. A new short term plan is produced every 0.5s.

**Brakes Control.** The brakes control system guides the user along the specified path by operating on the brakes [5]. The idea is to minimise the jerk and to leave the maximum possible level of freedom to the user. Only when he/she deviates significantly from the trajectory does the system come into play. In order to ensure a good level of comfort and an effective driving action the activation frequency of this component has to be very high (in the order of 50ms).

## 2.1 Architectural design

The functional architecture outlined above suggests the following considerations:

1. Many of the components are re-usable across a wide family of applications and systems (e.g., the localisation module and the people tracker);
2. The computational demand and the physical constraints call for a distributed hardware implementation, in which the functionalities could be deployed in different nodes in different implementations or operating conditions (e.g., in response to a system failure);
3. The different components require varied expertise; the resulting development team is large and heterogeneous.

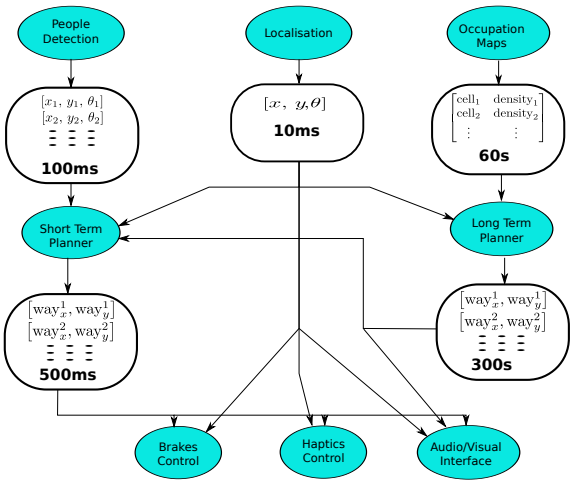


**FIGURE 2:** *Publish-Subscribe architecture for some of DALi's components*

The adoption of a middleware infrastructure providing publish-subscribe functionalities allows us to meet most of these requirements and decouple the development of the different modules. We report in Figure 2 a possible implementation scheme for the communication between some of the modules. As an example, the people tracker publishes a sequence of positions and velocity of the people within the reach of the sensors with a periodicity of 100ms and this topic is subscribed to by the short term planner. The localisation module publishes a new position of the cWalker every 10ms and this information is used by various subscribers (at least those shown in the figure). Similarly in the graph one can read the topics published and subscribed to by other modules.

# 3 Publish-Subscribe Middlewares

As explained in Section 2, the various modules composing the DALi software architecture communicate according to the publish-subscribe paradigm,

and this functionality is provided by a middleware. Since the DALi modules are characterised by some real-time constraints to be respected, the middleware has to provide predictable communication latencies (without compromising the throughput). Hence, while there are several middlewares providing publish-subscribe functionalities, in this paper we focused on those that aim at providing support for real-time communications.

We also tried to evaluate the various middlewares in the light of some existing standards. In particular, we focused on the Data Distribution Service (DDS) and the Real-Time Publish-Subscribe (RTPS) protocol, which are two standards published by the Object Management Group (OMG).

The DDS specification describes two levels of interfaces:

1. **Data-Centric Publish-Subscribe (DCPS)**, a lower layer that is targeted towards the efficient delivery of the proper information to the proper recipients. It introduce a virtual global data space.
2. **Data-Local Reconstruction Layer (DLRL)**, an optional higher-level layer which allows for a simpler integration into the application layer. It provides an object-oriented layer of the data

DDS keeps the wire protocol open which allows different vendor to have their own protocol. RTPS is one of possible wire protocol standard for DDS.

The RTPS protocol is designed for real time communication on top of unreliable and connectionless transport protocols such as UDP. RTPS introduces publication and subscription timing parameters and properties to achieve some performance and reliability goals. A publisher and subscriber are connected via two parameters: topic, type. The topic is the label that identifies each data flow while the type describes the data format.

In this work, we considered three different middlewares: OpenDDS, ORTE, and ZeroMQ. While the first one is fully compliant with the DDS standard, the second one just implements the RTPS protocol, and the third one is not compliant with any specific standard. Hence, comparing the three middlewares allows to evaluate the cost and the benefits of the various standards and the overhead they might introduce.

**OpenDDS** is an implementation of OMG Data DDS v1.2 and the Real-time Publish-Subscribe Wire Protocol DDS (DDS-RTPS) v2.1. It is an open source C++ implementation of DDS standard that uses Adaptive Communication Environment (ACE) to provide cross platform portability and The ACE Orb (TAO) for its DCPS layer. OpenDDS uses CORBA only to administer the discovery service. OpenDDS architecture borrows from TAO's framework.

**The Open Real-Time Ethernet (ORTE)** is open source implementation of OMG RTPS communication protocol v 1.0 (OMGdocument formal/06-08-02).

**ZeroMQ** is an open source socket based messaging library that include support for publish-subscribe messaging pattern. It provides a set of API with multiple language binding e.g. C, C++, python and java.

Table 3 shows a comparison of features available in these middlewares.

## 3.1 The Middleware Abstraction Layer

Since the specific Middleware that will be used in the DALi walker has not been decided yet (but only the needed features have been identified), an abstraction layer providing the needed publish-subscribe functionalities has been developed. Such an abstraction layer exports a simplified API that allow to create publishers and subscribers, publish and receive topics, etc...

In particular, the abstraction layer is written in C++ and its API is composed by:

- A class modelling a DDS "domain" (representing the set of applications that can communicate each other);
- A class modelling a Publisher. This class can be instantiated once a domain has been defined, and can publish a topic on such a domain;
- A class modelling a Subscriber. Similarly to the publisher class, this class can be instantiated only once a domain has been defined, and receives messages concerning a specified topic on such a domain.

The domain class only provide a constructor, a destructor, and two methods to create a Publisher or a Subscriber in this domain. When creating a Publisher, it is possible to specify a name for the topic it publishes; the Publisher class then

| features | ZeroMQ | OpenDDS | ORTE |
|---|---|---|---|
| Standard | - | DDS-RTPS v1.2 | RTPS v1.17 |
| Notification | wait | listener/wait | listener/wait |
| Transport | TCP | TCP/UDP | UDP |
| Message type | raw buffer | typed message with IDL | raw / typed message with IDL |
| Naming service | - | global via DCPS layer | - |
| Serialisation | self managed | DLRL layer | self managed |

provides a `publish()` method that allows to send messages for this topic. When creating a Subscriber, it is possible to specify the name of the topic to subscribe to; the Subscriber class then provides a `register_callback()` method that allows to specify a callback to be invoked when a message for the specified topic is received.

The C++ classes then hide all of the implementation details (and the middleware API), allowing to write code using the publish-subscribe paradigm without relying on a specific middleware. The abstraction layer currently supports the three middlewares considered in this paper, but extending it to other middlewares based on the publish-subscribe paradigm should be simple.

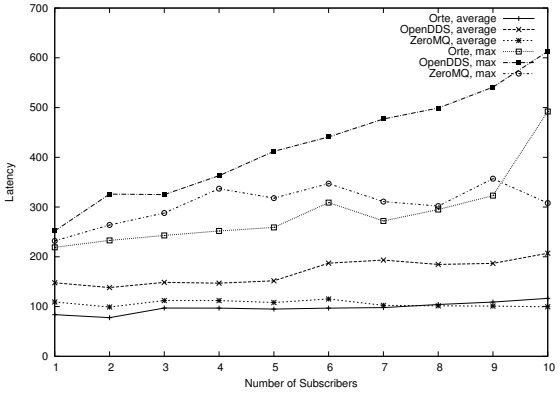


**FIGURE 3:** *Publisher/Subscriber latency for small messages as a function of the number of subscribers.*

# 4 Middleware Comparison

In order to check the usability of the three considered middlewares in the DALi software architecture, their performance has been compared considering both the (worst case) real-time latencies and the (average) communication throughput. This evaluation has been performed by using some test programs implementing publish-subscribe communication, and using a setup similar to the one described in Figure 2. The results obtained in the two cases are consistent, and this section reports the ones based on the simple test programs.

In a first set of experiments, the real-time performance of the three middlewares has been compared, by measuring the latency between the generation of a message (from the publisher) and its arrival to the subscribers. Two small test programs (one for the publisher and one for the subscribers) have been developed using the abstraction layer presented in Section 3.1 and have been used to measure the message latency for various configurations (changing the message size, the number of subscribers, etc...). Using the abstraction layer allowed to repeat the experiments with ZeroMQ, OpenDDS, and ORTE without having to write dedicated tests for each middleware.

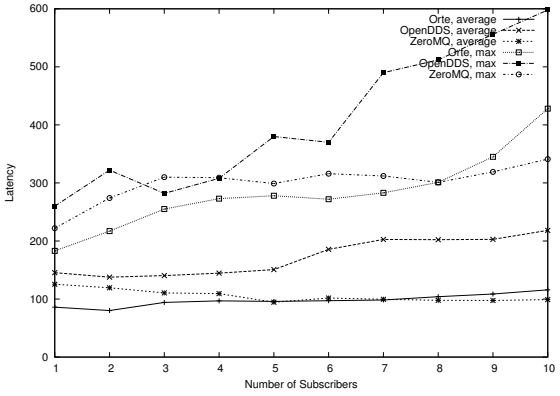


**FIGURE 4:** *Publisher/Subscriber latency for large messages as a function of the number of subscribers.*

Figures 3 and 4 present the measured publisher/subscriber latency (the delay between the generation of a message and its arrival to subscribers) measured for small messages (payload of 50 bytes) and large messages (payload of 1500 bytes) and a number of subscribers ranging from 1 to 10. Notice that the figures present both the average latency and the worst-case latency measured over 100000 samples (which is probably more interesting to evaluate

the real-time performance of the middleware). Notice that while the average latency does not show a strong dependence from the number of subscribers, the worst-case latency is more affected by this parameter. In particular, the worst-case latency for OpenDDS shows a noticeable increase (almost linear), while the worst-case latency for ORTE increases less dramatically for small numbers of subscribers, showing larger increasesonly for more than 7 or 8 subscribers. On the other hand, ZeroMQ performs worse than ORTE when there are few subscribers, but seems to scale better.

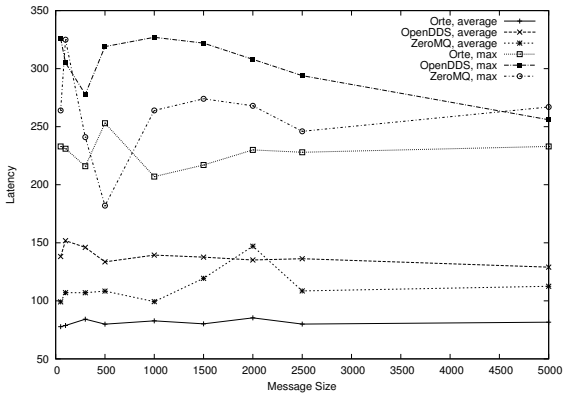


**FIGURE 5:** *Publisher/Subscriber latency for 2 subscribers a function of the message size.*

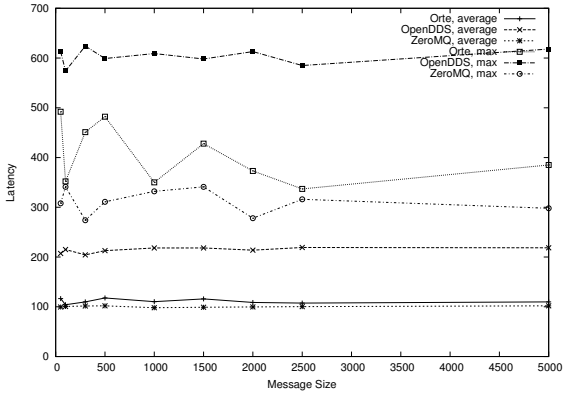


**FIGURE 6:** *Publisher/Subscriber latency for 10 subscribers as a function of the message size.*

Figures 5 and 6 show how the measured latency depends on the message size, when there are few subscribers (2) or a larger number of subscribers (10).

The next set of experiments compared the maximum throughput that can be achieved using the various middlewares. This throughput has been measured by measuring the minimum amount of time needed by a subscriber to receive 10000 messages, with a publisher generating messages at the maximum possible rate that does not cause losses.

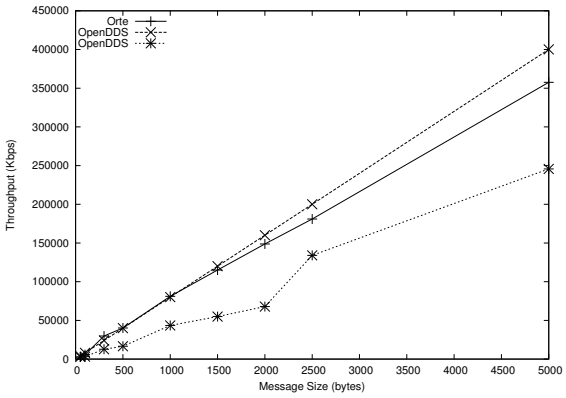


**FIGURE 7:** *Publisher/Subscriber throughput as a function of the message size.*

Figure 7 shows the maximum throughput measured with the various middlewares, as a function of the message size.

Notice that since producer and consumers executed on the same machine, the loopback network device has been used for communication, hence the throughput is quite high. ZeroMQ performs better than the other 2 middlewares, but all of the middlewares can achieve a throughput larger than $10Mbps$, and are thus suitable for being used in the DALi architecture.

Based on the results of the experiments reported above, ORTE seems to offer the best tradeoff between expressive power of the supported abstraction, latency and throughput.

# 5 Conclusions

Robotic applications are one of the most promising areas of innovation for the application of Information and Communication Technologies. Their increasing complexity, the integration of third party components and the involvement of teams with heterogeneous expertise requires middleware infrastructure to speed up the development time and facilitate re-use. One of the most important requirements that a middleware for robotic applications must satisfy is the ability to support real–time computation and communication. We have tested three middlewares available of the shelf to identify the solution that best suits the needs of robotic applications, with a particular focus on the real–time requirements. The experimental set-up was designed taking inspiration

from an existing robotic application that our group is developing. Based on our result, ORTE seems to strike an adequate tradeoff between the conflicting requirements of robotic applications.

The goals of our future investigations are manifold. One of the most important is to extend the analysis to other middleware solutions explicitly developed for robot applications such as ROS [11] and OROCOS [1].

# References


[1] Herman Bruyninckx. Open robot control software: the orocos project. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2523–2528. IEEE, 2001.

[2] Alessio Colombo, Daniele Fontanelli, Axel Legay, Luigi Palopoli, and Sean Sedwards. Motion planning in crowds using statistical model checking to enhance the social force model. In *Decision and Control (cdc2013), 2013 Proc. of 53nd IEEE Conference on*, Firenze, Italy, Dec. 2013.

[3] `http://www.ict-dali.eu.` Website.

[4] D. Fontanelli, A. Giannitrapani, L. Palopoli, and D. Prattichizzo. Unicycle steering by brakes: a passive guidance support for an assistive cart. In *2013 IEEE 52st Conference on Decision and Control (CDC)*, Florence, Italy, 2013.

[5] Daniele Fontanelli, Antonello Giannitrapani, Luigi Palopoli, and Domenico Prattichizzo. Unicycle steering by brakes: a passive guidance support for an assistive cart. In *Decision and Control (cdc2013), 2013 Proc. of 53nd IEEE Conference on*, Firenze, Italy, Dec. 2013.

[6] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[7] Pieter Hintjens. *ZeroMQ: Messaging for Many Applications*. O'Reilly, 2013.

[8] Payam Nazemzadeh, Daniele Fontanelli, David Macii, Tizar Rizano, and Luigi Palopoli. Design and performance analysis of an indoor position tracking technique for smart rollators. In *Proc. of 4th International Conference on Indoor Positioning and Indoor Navigation, IPIN 2013*, Monbeliard, Belfrot, France, October 2013. IEEE.

[9] `http://www.opendds.org`. Website.

[10] OMG. Data distribution service for real-time systems – version 1.2. Technical report, The Object Management Group, 2007.

[11] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[12] Petr Smolik, Zdenek Sebek, and Zdenek Hanzalek. Orte–open source implementation of real-time publish-subscribe protocol. In *Proc. 2nd International Workshop on Real-Time LANs in the Internet Age*, pages 68–72, 2003.

[13] `http://zeromq.org.` Website.