UNIVERSITY OF WATERLOO
CENTER FOR PATTERN ANALYSIS AND MACHINE INTELLIGENCE
CPAMI

Developer Manual for Axon v.0.1.0

prepared by

Michael Morckos
August 28, 2012

Table of Contents

1.0 Introduction **1**
1.1 Middleware Architecture 1

2.0 Installation **3**
2.1 Prerequisites 3
2.2 Server 3
2.3 Entity Component 5
2.3.1 Entity Manager 5
2.3.2 Entity Interfacer 6

3.0 Adding an Entity to the Environment **7**
3.1 Basic Profile 7
3.2 Services Profile 8
3.3 Entity Component 10
3.3.1 EntityInterfacer 10
3.3.2 Communicator 12

4.0 Tested Robots **12**
4.1 PeopleBot Mobile Robot 12
4.2 Cyton Alpha Arm 12

References **14**

1.0 Introduction

Axon is a middleware for robotics designed to accommodate different types of robots in a lab environment. Axon provides developers with a very abstracted high level representation of robots. It recognizes each robot as a single entity in the environment that possess certain properties and has a number of services to offer to a human operator, or other robots. The main goals of Axon are:

- Assist in designing and developing high-level multi-robot applications and experiments where new algorithms and techniques involving robotic behavior, collaboration, reasoning, and so on can be rapidly prototyped and tested.
- Enable full usage of vendor-provided development frameworks, while providing a generic-enough protocol that could accommodate different robots with their heterogeneous frameworks.

A typical flow of events in using the middleware can be as follows: When a client program is introduced to the system, the user can browse through a list of all connected robots. The user can then "check out" one or more robots, provided that they are available, and issue various tasks to them. After running some experiments and scenarios, for instance, the user can then release previously "checked-out" robots so that they become available to other users.

1.1 Middleware Architecture

Axon is based on a component-based design. Moreover, the use of software design patterns, rigorous developing disciplines, and documentation ensures that the middleware can be easily upgraded in the future.

Figure 1 illustrates the high level architecture of the middleware. The server component is the central hub of the middleware. Its main functions are to keep track of all connected entities and clients, as well as providing efficient and reliable message routing between different clients and entities.

The entity component is the part of the middleware that resides on the robot's onboard machine. This vital component is responsible for providing complete transparency to the developer by hiding all networking and concurrency details while presenting a simple and extensible interface that can easily interface with custom-developed robotic applications. The entity component is compose of two modules: the Entity Manger (EM) and the Entity Interfacer (EI). The EM is the interface of a custom-developed application to the environment. The EI module, associated with the developed application, interfaces the application with the EM.

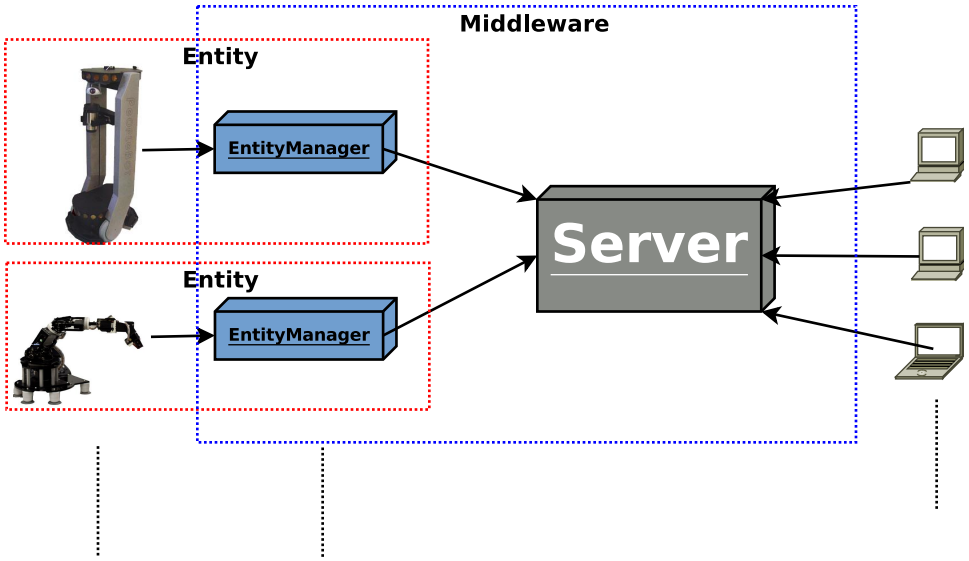

Figure 1: Middleware Abstracted Overview.

Axon relies on the Internet Communications Engine (ICE) [1] for communication and data exchange between different entities and clients.

2.0 Installation

This section describes the procedure that the developer needs to follow in order to install the components of the middleware, namely the entity component and the server. The middleware source code, documentation, and samples are provided in the `axon_backend.tar` archive. The current version of the middleware supports x86 GNU/Linux platforms.

2.1 Prerequisites

There are two software packages that need to be installed prior to installing any component of the middleware on a machine. They are listed as follows:

- The ICE software package v.3.4.2 or later. This package can be downloaded from this link: http://zeroc.com/ice.html, or from the package manager of the Linux distribution on the target machine.
- The GPB package v.2.4.1 or later. This package can be downloaded from this link: https://developers.google.com/protocol-buffers/, or from the package manager of the Linux distribution on the target machine.

2.2 Server

The server component is composed of two modules: the entity server and the frontend server. The installation instructions are as follows:

1. Open a terminal.
2. Untar the `axon_backend.tar` archive.
3. Go into the "`server/`" directory by typing: "`cd axon_backend/server/`".
4. The default installation directory is "`/opt/axon_backend/server/`". This can be changed as follows:
   (a) Go to the "`config`" directory.
   (b) Open the `Make.rules` file using a text editor.
   (c) Edit the *prefix* parameter by typing the desired installation directory.
   (d) Save and close.
5. Inside the "`server/`" directory, type "`make`", followed by "`make install`". This will compile and install the server into the target installation directory.

After installation, the installation directory should contain the following subdirectories:

- "`bin`": contains the executables for the two server: `entityserver` and `frontendserver`. This directory also contains a simple running script, `run_server.sh`.
- "`certs`": contains certificate files used in communications with the EMs of different entities. These files must not be modified.
- "`config`": contains configuration files for assigning the servers to ports in order for EMs and external client programs to be able to locate the servers.
- "`globalservices`": contains a sample global services XML-based profile. This profile is used to define collaboration scenarios between different robots, by stating the participating entities and the order of execution of services. A developer must use the profiles of participating entities to compile a global services profile.
- "`slice`": contains the ICE IDL definitions used for communications with EM and client programs. These IDL definitions can be used to develop custom client applications. (The current version of the middleware does not include a fully-fledged client application.)
- "`doc`": source code documentation.
- "`db`": used for inter-server communications. This directory must not be deleted.

Before running, the server must be assigned to port numbers and the host machine IP address in order for external entities and clients to be able to connect to it. Two configuration files in the "`config/`" subdirectory of the installation directory must be edited as follows:

- `glacier2router_entity.cfg`: for the entity server. In this file, the parameter *EntityServer.Endpoints* takes a port number and two IP address values for SSL and TCP types of connections. The IP addresses must be the same for both types. On the other hand, port numbers must be distinct.
- `glacier2router_frontend.cfg`: for the front-end server. In this file, the parameter *FrontEndServer.Endpoints* must be edited in the same manner as the *EntityServer.Endpoints* of `glacier2router_entity.cfg`.

In order to run the server, go to directory to the "`bin/`" directory and execute the `run_server.sh` script by typing in the terminal: "`sh run_server.sh`". If initialization is success, the server will display success messages. It is now ready to receive connections.

2.3 Entity Component

A developer must perform the following steps in order to properly compile and install the entity component.

2.3.1 Entity Manager

1. Open a terminal.
2. Untar the `axon_backend.tar` archive.
3. Go into the "`entitymanager/`" directory by typing: "`cd axon_backend/entitymanager/`".
4. The default installation directory is "`/opt/axon_backend/entitymanager/`". This can be changed as follows:
   (a) Go to the "`config/`" directory.
   (b) Open the `Make.rules` file using a text editor.
   (c) Edit the *prefix* parameter by typing the desired installation directory.
   (d) Save and close.
5. Inside the "`entitymanager/`" directory, type "`make`", followed by "`make install`". This will compile and install the EM into the target installation directory respectively.

After installation, the installation directory should contain the following subdirectories:

- "`bin`": contains the EM executable. The EM must be run alongside the custom-developed robotic application.
- "`certs`": contains certificates files used in communications with the server. These files must not be modified.
- "`config`": contains configuration files for configuring the connection to the server, and the connection to the robotic application.
- "`sample_profiles`": contains samples of robotic entities profiles (explained later).
- "`doc`": source code documentation.

The EM must be configured before running. Two configuration files in the "`config`" subdirectory must be edited as follows:

- The "`entitygl2client.cfg`" Glacier2 client configurations file. The *Ice.Default.Router* parameter takes two IP address and port values. To ensure that the EM can locate the server, the two IP addresses must match the IP address of the machine the entity server is residing on. Moreover, the two port numbers must match those the entity server is listening on.
- The "`ports.cfg`" configuration file. The *PORTa0* and *PORTa1* are used to specify a port range. This will be used to connect to the EI. The range must fall between `5500` and `8000` inclusive.

2.3.2 Entity Interfacer

1. Open a terminal.
2. Untar the `axon_backend.tar` archive.
3. Go into the "`entityinterfacer/`" directory by typing:
   "`cd axon_backend/entityinterfacer/`".
4. The default installation directory is "`/opt/axon_backend/entityinterfacer/`". This can be changed as follows:
   (a) Go to the "`config/`" directory.
   (b) Open the `Make.rules` file using a text editor.
   (c) Edit the *prefix* parameter by typing the desired installation directory.
   (d) Save and close.
5. Inside the "`entityinterfacer/`" directory, type "`make`", followed by "`make install`". This will compile and install the EI into the target installation directory.

After installation, the installation directory should contain the following subdirectories:

- "`config`": contains configuration files for configuring the connection to the EM.
- "`include`": contains the "`entityinterfacer.h`" header file. This header file must be included in the application source code.
- "`lib`": contains the "`libentityinterfacer.so`" shared library. The application must be linked against this library.
- "`doc`": source code documentation.

The EI and the EM communicate through a TCP socket. The socket port number can be configured by editing the "`ports.cfg`" configuration files found in the "`config/`" subfolder. A range of port numbers can be entered. This range must be between `5500` and `8000` inclusive, and it must match that specified for the EM.

3.0 Adding an Entity to the Environment

In the middleware, each entity in the environment is recognized by a unique *entity profile*. The entity profile contains a comprehensive set of declarative information, such as the entity's type, location, and services. This information uniquely defines an entity as an active actor in a multi-robot, multi-user environment. The entity profile is composed of two sub-profiles: the *basic profile* and the *services profile*. These XML-based profiles must be defined and created prior to plugging the entity in to the environment. They are discussed in detail in the following subsections. Sample entity profiles can be found in the "`sample_profiles`" subdirectory in the install directory of the entity manager.

3.1 Basic Profile

The basic profile provides basic information about an entity that can uniquely identify it in the environment. The basic profile for an entity is made available to all connected clients, and it should help a user decide which entity best suits his needs. Figure 2 illustrates the basic profile for a mobile robot.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<BasicProfile name = "PeopleBot"
              category = "ROBOT"
              type = "MOBILE"
              descr = "Mobile robot with an array of range finding sensors, basic gripper and camera."
              xpos = "1000"
              ypos = "1500"
              zpos = "0" >
</BasicProfile>
```

Figure 2: A Sample Basic Profile XML File.

A typical basic profile should contain the following attributes:

- ***Name***: the name of the entity.
- ***Category***: describes the nature of the entity. In the current version of the middleware, two categories are supported: `ROBOT` and `MACHINE`.
- ***Type***: the type of the entity. Two types are supported: `MOBILE` and `STATIONARY`.
- ***Description***: an optional brief description of the entity.
- ***Xpos***, ***Ypos***, and ***Zpos***: these three parameters define the current position of the entity in an environment.

3.2 Services Profile

In addition to the basic profile, the developer must also provide the services profile. This profile contains all the details about the services offered by the entity. In the context of the middleware, a service is defined as an executable task that can be performed by an entity upon request from a human operator, or from another entity (such as in collaboration). For each service, the following attributes must be provided:

- ***Name***: the name of the service.
- ***Global ID***: a global name for the service. This can be used in case the same service is provided by more than one entity, or the service is a sub-service of a composite service. It is the responsibility of the developer to ensure the uniqueness of the global ID.
- ***Type***: the type of the service. There are two supported values: `SMPL` and `CMPLX`. `SMPL` signifies a simple atomic service, while `CMPLX` signifies a composite service that can be composed of other services that are executed in a specific order.
- ***Active***: an optional flag attribute indicating whether the service is enabled or disabled. An enabled service means that it can be requested. A robotic application may enable/disable a service. This can be done, for example, to preserve battery power or resources. However, this choice is left to the developer.
- ***Blocking***: an optional flag attribute indicating whether the service can be executed simultaneously with another service or not. A blocking service could mean that it makes use of resources/components that cannot be shared, or that the simultaneous execution of a second service could interfere with the former.
- ***Pausable***: an optional flag attribute indicating whether the service can be paused in-mid execution or not.
- ***Description***: an optional simple description for the service.
- ***Notes***: optional short notes related to the service.
- ***Service list***: a list of other services. This list should be populated only if the service is `CMPLX`.

Figure 3 illustrates a portion of the services profile for a mobile robot.

In addition to its attributes, each service can house two optional sub-components:

- ***Control Parameter***: a control parameter defines how external actors can interact with a service. Control parameters are used to configure a service by providing input and runtime configurations, as well as communicating results and feedback to the requester.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<ServiceProfile count = "7">

<!-- Service #0 -->
<Service parameter_count = "1"
         name = "Simple motion"
         type = "SMPL"
         active = "true"
         blocking = "true"
         pausable = "false"
         global_id = "Walk" >

  <Parameters>
    <Param name = "Distance"
           type = "INPUT"
           data_type = "INT"
           units = "meters"
           value = "1000"
           required = "true"
           set = "true"
           modifiable = "true"
           descr = "Distance to travel" />
  </Parameters>
</Service>

                          <!-- continued -->

</ServiceProfile>
```

Figure 3: A Sample Services Profile XML File.

- ***Resource***: a resource represents external data or files that a service might require for execution. Examples of resources are map files, images, or a batch of sensor readings. (Resources are not fully supported in the current version.)

Each parameter is defined by the following attributes:

- ***Name***: the name of the parameter.
- ***Type***: the type of the parameter. A parameter can be an input, output, configuration, or feedback parameter. The allowed values for the type are `INPUT`, `OUTPUT`, `CONFIG`, and `SMPL`.
- ***DataType***: the data type of the parameter. A parameter can be an integer, double, string, or boolean. The allowed values are `INT`, `DBL`, `STR`, and `BOOL`
- ***Units***: an optional attribute signifying the units of the parameter (like millimeters).
- ***Ready***: an optional flag attribute indicating whether the value of the parameter is set or not.
- ***Required***: an optional flag attribute indicating whether the parameter is a mandatory dependency for the owning service or not.
- ***Savable***: an optional flag attribute indicating whether the parameter can be saved or not. If the flag is not set, it means that the parameter can change multiple times, or that it must be set each time its owning service is requested.

- ***Multiple***: this optional attribute is used to indicate that the parameter can store a list of values of the same data type. This can be useful for streaming data or having multiple choices for a single parameter.
- ***Value***: the value of the parameter.
- ***Value list***: list of values of the parameter.

3.3 Entity Component

As mentioned earlier, the entity component is the part of the middleware that resides on the robot's onboard machine. For development, the EM can be treated as a black box that will interface with the robotic application. The EI acts as a middleman between the application and the EM. The rest of the section will mainly focus on the EI API, and how to use it in custom-developed applications.

The current version of the EI provides a TCP socket-based communications link with the EM, as well as a simple and extensible API that the robotic application must interface with to be able to be seamlessly plugged in to the environment. The design goals of the EI are as follows:

- Providing complete transparency to the developer who is implementing the robotic application. The developer does not need to worry about details such as queueing, concurrency, multithreading issues, or networking.
- Providing reliability and safety features. The main implemented feature at the moment is a stop signal that is dispatched instantly in cases of client requests, disconnections, or improper behavior on the client side. The stop signal is guaranteed to reach the application in a minimal amount of time. However, it is the responsibility of the developer to make meaningful use of the signal.
- Maximizing the automation of the process. All a developer needs to do is to run his own application and the EM. Zero intervention is required after that, except in cases of severe errors.

In the current version, a `C++` implementation of the EI exists. The main classes of the EI are explained below. More details can be found in the source code documentation in the project package.

3.3.1 EntityInterfacer

This class provides the main API for robotic applications developers. A robotic application can interact with the environment by calling functions in this class. Moreover, this class

interacts with the robotic application through a set of callbacks. The class functions are explained as follows:

- `init`: used to initialize the entity interface.
- `start`: used to start the entity interface. The interfacer will then establish connection with the EM.
- `stop`: used to stop the entity interface.
- `uploadBasicProfile`: used to upload the basic profile.
- `uploadServiceProfile`: used to upload the services profile.
- `setStatus`: used to set the current status of the entity, either busy or available.
- `updatePosition`: used to update the current location of the entity in the environment.
- `sendInfoMsg`: used to send a simple informative message to the client.
- `sendTaskFeedback`: used to send the feedback of a recently finished tasked.
- `sendIntParam`: used to send an double parameter to the client.
- `sendDoubleParam`: used to send an double parameter to the client.
- `sendStringParam`: used to send an string parameter to the client.
- `sendBooleanParam`: used to send an bool parameter to the client.
- `setCallee`: used to set the object in the entity application that will be interacting with the entity interfacer.
- `setTaskCallback`: used to set the callback function for receiving a new task.
- `setIntParamCallback`: used to set the callback function for receiving an integer parameter.
- `setDoubleParamCallback`: used to set the callback function for receiving an double parameter.
- `setStringParamCallback`: used to set the callback function for receiving an string parameter.
- `setBooleanParamCallback`: used to set the callback function for receiving an bool parameter.
- `setResourceCallback`: used to set the callback function for receiving a resource.
- `setStopCallback`: used to set the callback for receiving stop signals.
- `testCallbacks`: used to test that all callback functions have been set.

3.3.2 Communicator

This class represents the communication port of the EI, and is responsible for establishing and maintaining the communications link with the EM. It handles all operations related to the transmission, packing and unpacking of data. This provides complete transparency to the developer. Applications cannot interact directly with this class.

4.0 Tested Robots

The middleware's ability to accommodate heterogeneous robot units by using two different types of robots in a real lab environment: a multi-purpose mobile robot and a stationary platform mounted robotic arm. Each robot comes from a different vendor and has its own software libraries and APIs.

4.1 PeopleBot Mobile Robot

PeopleBot [2], shown in Figure 4, is a multi-purpose differential-drive robot developed by MobileRobots, Inc. [3], for research and applications involving HRI, cooperative robotics, and much more. PeopleBot comes with a set of sensors, as well as sonar and laser based range-finding devices for navigation and obstacle avoidance purposes. Moreover, PeopleBot is able to recognize objects and people, as well as track colors using a 120-degrees pan/tilt/zoom (PTZ) camera. In addition, PeopleBot is equipped with infrared table sensors, as well as a 2-DOF gripper with pressure sensors to sense tables and grab objects respectively.

4.2 Cyton Alpha Arm

Cyton Alpha, shown in Figure 5, is a 7-DOF 1G robotic arm mimicking the human arm, and is developed by Robai for the purposes of performing lightweight tasks, prototyping, and research [4]. Cyton Alpha features 7-DOF movement capability plus a single end effector gripper.

Figure 4: The PeopleBot Mobile Robot [2].

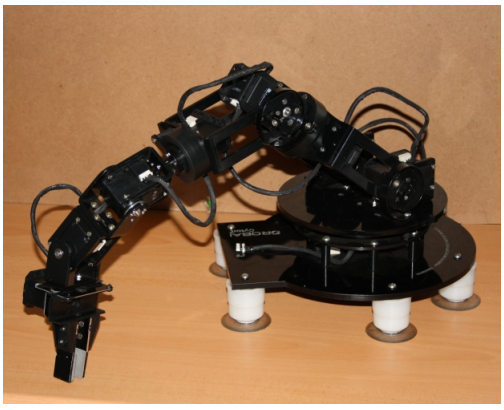

Figure 5: The Cyton Alpha Robotic Arm [4].

References

[1] ZeroC - the Internet Communications Engine. http://zeroc.com/ice.html. Accessed: May 24, 2012.

[2] Peoplebot Robot Makes Human-Robot Interaction Research Affordable. http://www.mobilerobots.com/ResearchRobots/PeopleBot.aspx. Accessed: June 18, 2012.

[3] Intelligent Mobile Robotic Platforms for Service pobots, Research and Rapid Prototyping. http://www.mobilerobots.com/Mobile_Robots.aspx. Accessed: June 18, 2012.

[4] Robai - Powerful Affordable Robots. http://www.robai.com/. Accessed: June 19, 2012.