EML ROS-Environment
Basic Ideas
emerging-mind.org eJournal ISSN 2567-6466

Gerd Doeben-Henisch
info@emerging-mind.org
gerd@doeben-henisch.de

Oct-5, 2017

1 Introduction

This text deals with the software ROS (Ros Operating System) as tool for the emerging-mind lab software framework. One main application will be for the Actor-Actor Interaction (AAI) paradigm of the uffmm.org project.

Taking the AAI theory as a first source for requirements we can infer the following requirements for useful simulation tools:

AS: Built a mathematical graph for the actor story (AS). This will be called *actor-story graph (ASG)*

ASSim: Built an actor story simulator which can read an actor-story graph (ASG) as input. This simulator will be called *AS-simulator (ASSim)*

AM: Built *actor models (AM)* for selected actors.

AMSim: Built a actor *model simulator (AMSim)* which can take actor models as inputs.

ASWP: Built an *actor software platform (ASWP)* which can manage an actor story simulator with n-many distributed actor model simulators.

TPs: Enable the software platform to allow defined *test protocols (TPs)* to check the behavior of AMSims with regard to an ASG.

INTr: Enable the ASWP to allow *interactions* of human actors with AMSims

Depending from the problem to be solved there is a great variety of actor stories and actor models possible.

The same holds for possible hardware and programming tools.

In this text the selection of the software has a clear bias: we want to use software which fits to real world requirements too, not only to theoretical considerations. With this bias there it is a good working hypothesis to select the *robot operating system (ROS)* based on ubuntu 14.04 as a software platform (SWP). This software platform enables from the beginning distributed actor models (AMs) and test protocols (TPs). Depending from the way of programming one can state further that the ubuntu+ROS SWP offers built-in simulation power for AMSims as well as ASSims.

What has to be done from scratch that is the definition and construction of ASGs as well as AMs.

To learn all these concepts we will stepwise examine the robot operating system running on ubuntu-linux.

Helpful websites are the following ones:

ubuntu: http://wiki.ros.org/indigo/Installation/Ubuntu or http://howtoubuntu.org/how-to-install-ubuntu-14-04-trusty-tahr

ros: http://wiki.ros.org/indigo

linux tutorial(s): http://www.ee.surrey.ac.uk/Teaching/Unix/index.html

python: https://docs.python.org/3.4/

screen recorder: http://www.maartenbaert.be/simplescreenrecorder/

A helpful book for ROS is that by the authors of the ROS Quigley et.al.(2015) [QGS15].

2 Concepts meet Concepts

Having the platform ubuntu and ros under your fingertips we want to explore, whether this platform really can do the job we want to do.

2.1 Simulated Robots

Before we go into the details of the platform and the concepts there is an interesting idea presented in the book of Quigley et al., which offers a very general perspective.

Although the final goal of ROS is to operate *real robots* connected to human actors and different kinds of real systems the ROS can also support *simulated robots*. It is a strong point of ROS that it makes not too much a difference whether you are using only a software robot or a real robot. For simulations you need simulation environments presented to the expert in a *graphical* format. Generally you can connect many different such simulation systems with ROS. One powerful simulator with the name 'gazebo' comes with the ros distribution. It is contained in the gazebo_ros package. In the book it is described as follows: "This package provides a Gazebo plugin module that allows bidirectional communication between Gazebo and ROS. Simulated sensor and physics data can stream from Gazebo to ROS, and actuator commands can stream from ROS back to Gazebo. In fact, by choosing consistent names and data types for these data streams, it is possible for Gazebo to exactly match the ROS API of a robot. When this is achieved, all of the robot software above the device-driver level can be run identically both on the real robot, and (after parameter tuning) in the simulator."([QGS15]:p.94)

Thus for the goals of the AAI-theory this offers the perspective that one can translate the concepts of the AAI-theory into software concepts realized as simulations, which then can also be used for real world implementations.

2.2 AAI Graph

Within the AAI theory two basic concepts are the *actor story (AS)* and the *actor model (AM)*.

The actor story describes the complete process of practicing a task within the space which is given by the problem statement of some stakeholder. There can be several tasks which have to be considered and there can be tasks in parallel or interrelated.

This set of tasks can be described with the aid of a mathematical graph with extended properties (for details see the AAI-theory at `uffmm.org`). The nodes of such an AAI-graph represent the state of affairs at a certain moment t of time. These *states of affairs* are called *situations* or *scenes*. A scene is a collection of properties organized as subsets, which can be embedded representing hierarchies. Different kinds of actors are then different

kinds of subsets of a scene. What can change a scene is either an *event* caused by the *context* of actors or by some *action* of an actor. Events or actions trigger therefor a *change* in the scene at time t and produce thereby a new state of affairs at some time t+x on a time line. Thus the connections between two scenes is an *edge* representing such an event or action. Every scene can have more than one edge going out and coming in. In the special case of a *zero event* or *zero action* no property is changing and the scene stays 'unchanged'. In this case the outgoing edge can represent an incoming edge too.

Before we discuss the concept of an actor model we will examine whether and how the concept of an AAI-graph can be supported by the ROS.

2.2.1 ROS Graph

The main conceptual property of ROS is the *network of processes* which can exchange *messages*.

Every such process is in ROS interpreted as a *node in a graph*, where the messages are seen as *edges* which can travel from every node to every other node. This is often called a *peer to peer* network. Typically a node is a POSIX conform process and the messages are representing TCP connections (cf. [QGS15]:pp.9ff). Furthermore a node is organized as a kind of a framework called *workspace*. Inside of such a workspace you will find two different things: (i) a section with *source code* defining the behavior of this node and (ii) several kinds of other sections needed for the organization of the node within the network as well as for the software build-processes.

To realize the the communication between the nodes there exists a special process – a kind of a meta-node – called *roscore*, which allows the different 'normal' nodes to contact the roscore and become registered in the roscore for communication. Being registered a node can receive message from other nodes or send messages.

2.2.2 AAI Graph meets ROS Graph

The interesting question now is, whether and how the AAI-graph and the ROS-graph do match.

The answer is: they do match, but perhaps in a different way the reader may expect.

The main point is that the network of nodes, communicating with each other represent on a time line a sequence of different states of affairs.

At some time t on the time line we have a collection of nodes with certain properties. When a communication happens this can trigger some change in one of the nodes. In that case we have some time t+x where the properties in the collection of nodes have changed (or not, a special case).

Seen with the eyes of an AAI-theory the collection of nodes at some time t in the ROS network all together represent a situation, a scene, which represents a *node in the AAI-graph*. The different ROS-nodes are in the AAI-graph subsets of a AAI-scene representing possible actors. Every *AAI-actor* can have more subsets, representing a kind of a hierarchy. The ROS-messages are in the AAI-graph possible events or actions caused by AAI-actors. If such AAI-events or -actions *change* at least one property then an AAI-scene changes and generates by this a new AAI-node. Otherwise the AAI-scene continues. Because the properties of AAI-nodes can change the whole AAI-Graph can change. This corresponds well to the ROS-graph where nodes can be introduced during the process or can disappear; clearly ROS-nodes can also change properties while they are acting.

To be continued .... 'Building a simple ROS-graph'....

References

[QGS15] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1 edition, 2015.