Microsoft Robotics Developer Studio

Sathyanarayanan Kuppusami & Paul Hanzal
Universität Hamburg
M.Sc Intelligent Adaptive Systems
TABLE OF CONTENTS

Introduction
Reference Platform Design
Software Components of Robotics Studio
Illustration
Visual Programming Language
Visual Simulation Environment
Product Comparison
Discussion
Introduction

RDS is Windows based and provides an integrated programming environment built on .NET framework making it possible to build, test and debug robotics applications, without having to make any assumptions about the underlying hardware.

Major Objectives

Ease of development

Parallel and Multithreading (Asynchronous)

Code portability
REFERENCE PLATFORM DESIGN

Reference document provides guidance on the basic requirements for developing a “real” robot for use with services provided by RDS.

The Reference Platform Design takes advantage of the Microsoft Kinect™ sensor and also referred to as MARK – Mobile Autonomous Robot using Kinect.

Basic Components are:
- Base and all mounting hardware
- Upper deck for mounting a PC
- Kinect mounting support and cable connection
- Motors and Wheels with Encoders
- Caster(s) for stability
- Battery and Charger
- Proximity sensors (combination of Infrared and Sonar)
- Voltage regulators (at least one 12V 2A regulator, or better)
- H-Bridges for Motor Control
- Robot IO Controller Board (for motors and sensors) capable of communicating with a PC
- Laptop or Netbook computer
- Kinect sensor
- Wireless Xbox Controller

MARK Robot
Other 3rd party Supported Robots

- Eddie (Parallax)
- Boe-Bot (Parallax)
- CoroBot (CoroWare)
- Create (iRobot)
- Lynx 6 Robotic Arm (Lynx Motion)
- Mindstorms NXT (Lego)
SOFTWARE COMPONENTS

High-Level Services

Low-Level Services

CCR / DSS

.NET / CLR

Windows 7

Custom Firmware

Micro Framework

Windows Devices
Concurrency and Coordination Runtime (CCR) is a managed code library, a Dynamically Linked Library (DLL), accessible from any language targeting the .NET Common Language Runtime.

CCR facilitates to meet:

**Asynchronism**: The fact that different sections of the program can work independently and in parallel, without any one part causing the whole program to fail in the event of a problem.

**Concurrency**: If the program is organised into several independent sections, the fact that more than one section may want to access the same resource at the same time needs to be dealt with.

**Coordination**: Coordinating the various sections of the program and handling any errors that may arise.
DSS

DSS is a runtime environment that sits on top of CCR, and which is used to run services that have been built using Microsoft Robotics Studio.

- DSS is used to manage all parts of a robot and the related software in the form of services.
- DSS exposes the services that can be used (or “consumed”) by another program, another service or a user interface in a standardized manner.
- DSS makes use of CCR to manage parallel processing. The service represent a hardware component such as a sensor or actuator.
- Services are run within the context of a DSS node.
- Two protocols are used to access services, i.e. HTTP (making it is possible to view a service’s status from a browser) and DSSP (Decentralized Software Services Protocol), a protocol similar to SOAP, which is the protocol used to interact with web services.
Benefits of Services

**ROBUSTNESS**: Each service runs independently, making it possible to offer a loosely-coupled architecture. Consequently, if the service running a robot’s arm fails, the rest of the robot can continue to work.

**COMPOSABILITY**: Services are standardized, which means that the way they are used, the way to communicate with them and the way to query their state are standardized. This simplifies code development enormously, because the developer does not have to reinvent the wheel for these different operations. Furthermore, services built in this way can be ported from one DSS node to another.

**OBSERVABILITY**: DSS can be used to observe service states, thereby making it possible to determine the status of a system as a whole. This is known as observability. To do this, DSS attributes a URI to each service and exposes the state of those services. Furthermore, each service is accessible via a web browser, thereby increasing the system’s monitoring options.
ORCHESTRATION DSS

Robot Dashboard

Obstacle Avoidance Service
Generic Differential Drive

Robot Service
Generic Differential Drive
IR Sensor Array
Sonar Sensor Array

MCU Service
Differential Drive
Sensor Array

Serial Port Driver

IO Controller Firmware

Kinect Service
Depth Camera
RGB Camera
Tilt Servo/Sensor

Kinect Driver

Kinect Sensor Firmware
A service is a programming model involving the following elements:

- Contract
- An identifier
- State
- Communication ports
- Service handlers
- Notifications
- Partner services
# DSS MONITOR

## Service Instance Directory

**Description:** This service provides a directory of service instances currently running. In the "Service Instance" column, you can see each instance represented as a link pointing to the current state of the actual service. In the "Partners" column, you can see the other service instances that each service instance has partnered with. Partners are also represented as links pointing to the current state of the actual services.

<table>
<thead>
<tr>
<th>Service Instance</th>
<th>Partners</th>
</tr>
</thead>
<tbody>
<tr>
<td>/console/output</td>
<td>SubscriptionManager</td>
</tr>
<tr>
<td>/constructor</td>
<td></td>
</tr>
<tr>
<td>/contractdirectory</td>
<td></td>
</tr>
<tr>
<td>/controlpanel</td>
<td>ManifestLoaderClient</td>
</tr>
<tr>
<td>/defaulttarget</td>
<td></td>
</tr>
<tr>
<td>/manifestloader/0a3dca14586-433d-d825-eb978f14dbf2</td>
<td></td>
</tr>
<tr>
<td>/manifestloaderclient</td>
<td>ContractDirectoryService</td>
</tr>
<tr>
<td>/mountpoint</td>
<td>CreatorService</td>
</tr>
<tr>
<td>/resources</td>
<td>TargetService</td>
</tr>
<tr>
<td>/security/manager</td>
<td>SubscriptionManager</td>
</tr>
<tr>
<td>/servicetutorial4</td>
<td>CreatorService</td>
</tr>
<tr>
<td>/servicetutorial5</td>
<td>ConstructorService</td>
</tr>
<tr>
<td></td>
<td>PartnerListService</td>
</tr>
<tr>
<td></td>
<td>SubscriptionManager</td>
</tr>
<tr>
<td></td>
<td>StateService</td>
</tr>
</tbody>
</table>
SERVICE PROGRAM IN VS2012
ROBOTICS PROGRAM IN VS2012
VISUAL PROGRAMMING LANGUAGE

It is a graphics tool can be used to design a robotics program by using drag and drop to move the specialized “boxes”, which are the different entities (services) representing the program’s effectors, sensors, bricks, controls and all other third-party programs.

- Easy-to-use graphical programming
- Based on a robust scalable infrastructure: CCR and DSS
- Uses the .NET software platform (given that VPL can generate code that can then be modified and improved, this makes it a very professional, comprehensive platform)
- Debug functions
- A data flow language (in contrast to imperative languages such as C++ or Java) that is better suited to creating programs relying heavily on parallelism and multi-tasking
VPL Interface
The powerful Visual Simulation Environment enables users to develop robots in a rich virtual environment with realistic physics and state of the art rendering.

Components of Simulation environment are

- The **Simulation Engine Service** - is responsible for rendering entities and progressing the simulation time for the physics engine. It tracks of the entire simulation world state and provides the service/distributed front end to the simulation.

- The **Managed Physics Engine Wrapper** - abstracts the user from the low level physics engine API, provides a more concise, managed interface to the physics simulation.

- The **NVIDIA™ PhysX™ Technology** - enables hardware acceleration through the NVIDIA™ PhysX™ capable NVIDIA™ graphics processing units (GPUs).

- **Entities** - represent hardware and physical objects in the simulation world. A number of entities come predefined with the RDS and enable users to quickly assemble them and build rich simulated robot platforms in various virtual environments.
REAL-TIME PROGRAM EXPERIENCE

Video Demonstration of programming CoroBot
# COMPARISON OF PLAYER AND MRDS

Examining the documented feature and Usability criteria of RDE on 914 PC-BOT with the tasks of wandering & foraging by David S. Michal, Dr. Letha Etzkorn (University of Alabama)

## Qualitative Analysis

<table>
<thead>
<tr>
<th>Experience</th>
<th>Player</th>
<th>MRDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation</td>
<td>Installing Gazebo is harder</td>
<td>Installation easier</td>
</tr>
<tr>
<td>Creating simulated robot and environment</td>
<td>Creating Simulated robot easier with gazebo</td>
<td>Creating simulated environment easier</td>
</tr>
<tr>
<td>Creating service for a robot</td>
<td>Possible to write or extend service</td>
<td>Possible to write or extend service</td>
</tr>
<tr>
<td>Implementing wandering</td>
<td>Programmed in c++, simulator was robust, transition was smoother</td>
<td>Programmed in c#, simulator was robust, transition was smoother</td>
</tr>
<tr>
<td>Implementing foraging</td>
<td>Easy integration of opencv &amp; cvblobslib for vision processing, Map library</td>
<td>Steep learning curve on c#</td>
</tr>
</tbody>
</table>
## QUANTITATIVE ANALYSIS

### Systematic Criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Player</th>
<th>MRDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Engineering &amp; Architecture neutrality</td>
<td>Good</td>
<td>Average</td>
</tr>
<tr>
<td>OS, Hardware Support, Simulator</td>
<td>Good</td>
<td>Average</td>
</tr>
<tr>
<td>Distributed, scalability, security, fault tolerant</td>
<td>Average</td>
<td>Good</td>
</tr>
<tr>
<td>Languages, software integration</td>
<td>Good</td>
<td>Average</td>
</tr>
<tr>
<td>Map Making, Route planning, Speech recognition, vision processing</td>
<td>Good</td>
<td>Average</td>
</tr>
</tbody>
</table>

### Usability Criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Player</th>
<th>MRDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational Overhead</td>
<td>AV</td>
<td>BA</td>
</tr>
<tr>
<td>Low level usability</td>
<td>AV</td>
<td>AV</td>
</tr>
<tr>
<td>High level usability</td>
<td>AV</td>
<td>AV</td>
</tr>
<tr>
<td>Task implementation</td>
<td>AV</td>
<td>BA</td>
</tr>
</tbody>
</table>
REFERENCES

http://www.microsoft.com/robotics

http://www.generationrobots.com/microsoft-robotics-studio,us,8,17.cfm

A Comparison of Player / Stage / Gazebo and Microsoft Robotics Developer Studio