Robots everywhere: the next step after PCs? with MSRS?

Bernhard Merkle
Central Unit Research & Development
Software-Engineering
SICK-AG Waldkirch

mailto: Bernhard.Merkle@sick.de

mailto: Bernhard.Merkle@gmail.com
Contact on linkedin.com or xing.com
Overview:

- **Robotics**
  - Definitions, Potentials, Challenges

- **Microsoft Robotics Studio (MSRS)**
  - MSRS Architecture
  - Runtime
    - Distributed System Services, DSS
    - Concurrency and Coordination Runtime, CCR
  - Authoring Tools
    - Development Tools,
    - Visual Programming Language VPL
  - Services and Samples
    - Robot Services and Models
    - Documentation, Community

- **Other Robotics Toolkits**
  - Open Source, Lego (RIS/NXT), FischerTechnik, etc

- **Demos**
Trial of Definition: A Robot is

- device,
- hard- and/or software
- with the capability of sensing
- and (re-)acting

- Has to react to the world →
  - Concurrent
  - Parallel
  - Have/Adopt some intelligence
Highly Diverse Market…
RoboCup (by 2050 we will lose against robots 😊)

RoboCup Soccer
- Simulation
- Small size
- Middle size
- Humanoid

RoboCup Rescue
- Simulation
- Rescue Robots
DARPA (05/07): autonomous vehicles…

http://pave.princeton.edu/main/
Robotics Market Potential / Challenge

Emerging service and consumer market
- Remote assistance/presence
- Facilities maintenance
- Security
- Education
- Entertainment

Emerging challenges
- Lack of reusability
- Reinventing
- Concurrency
- Complexity

*Source: Japan Robotics Association
M$ aims with MSRS (M$ Robotics Studio)

Usage

– Primary market:
  • Endconsumer / Personal-Robotics (2nd wave of PC)
– Secondary market:
  • concurrent programming (next “wave” of programming)
  • CCR (will be) distributed separately
  • Coordination with MS “Parallel Computing” Initiative
    (Parallel Extensions to .NET Framework CTP)

Team

– Tandy Trower
– George Chrysanthakopoulos
– Henrik F Nielsen
– Small team (approx. 15) of high educated staff
– Kind of Startup in Microsoft
Robotics: Shared Challenges…

Input from industry, hobbyists, academia, research,…

– Configuring sensors and actuators in running system
– Coordinating sensors and actuators asynchronously
– Starting and stopping components dynamically
– Monitoring/Interacting/Debugging running system
– Development when access to robot is limited
– Span multiple compute units
– Re-use of components across hardware platforms and devices
What Microsoft

learned…

– Too much complexity
– Too many resources required
– Lack of reusability, choice
– Limited tools and technologies
– Difficulties in sharing
– Transference of skills/experience

intends… ;-) 

– Next Wave after PCs will be robots !?
– “A Robot in Every Home”, Bill Gates in AS
– Be(come) a market leader for Robotics Solutions
– Establish .NET Technology also in Industry ?
Microsoft Robotics Studio

Addressing the Challenges

Runtime environment
- Execute, monitor, and interact with robotics applications

Authoring environment
- Write, orchestrate, and deploy robotics applications

Simulation environment
- Execute robotics applications using simulated hardware, physical entities, and terrain
Microsoft Robotics Studio: High-level Architecture

- **Authoring Environment**
  - Visual Studio
  - VPL
  - Other ...

- **Interaction**
  - Internet Explorer
  - Win Forms
  - Visualization & Simulation
  - Other ...

- **Services**
  - Orchestration
  - Devices
  - Simulation

- **Runtime Environment**
  - Decentralized Software Services (DSS)
  - Concurrency and Coordination Runtime (CCR)
Microsoft Robotics Studio

A lightweight concurrency and service oriented runtime
- Handling sensor input / controlling actuators
- Based on message passing
- DSS facilitating tasks and basic services

Authoring/development tools
- Visual programming editor (VPL)
- Visual Studio IDE (.NET languages)
- Message debugging
- Simulation

Libraries and basic algorithms
- Code samples
- Documentation
Microsoft Robotics Studio

A development platform for the robotics community, supporting a wide variety of users, hardware, and application scenarios
Microsoft Robotics Studio

A development platform for the robotics community, supporting a wide variety of users, hardware, and application scenarios
An application is a **composition** of loosely-coupled components **concurrently** executing
- Orchestration is a service
- Partnering and subscription (static, dynamic)
- Services are units of orchestrations (nested)
Service: simple abstraction for hard-/software

Service properties:
- Identity
- Structured State
- Operations and Handlers
- Support dynamic discovery
- Distributed and asynchronous
→ Service Contract

Uniform behaviour
- State retrieval / manipulation
- Service creation / termination
- Event notification
- Re-use (composition, aggregation)
Protocol based service interaction

- DSSP is a SOAP-base protocol for inter-service communication
  - Based on HTTP (existing web infrastructure)
  - Structured data manipulation,
  - Event notification
  - TCP or HTTP for Intra-node
  - open / performs
The CCR (Concurrent Coordination Runtime)

Asynchronous Programming model

- A concurrency model without manual threading, locks, semaphores, etc.
  - Based on message passing
  - Focus on coordination primitives
  - Sequential execution but with no thread blocking, with no need for callback

- Execution context for services
  - Isolation from infrastructure
  - Isolation from other services
Messages,Ports, and Arbiters

Messages are sent to Ports

```csharp
Port<int> myPort = new Port<int>();
myPort.Post(42);
```

Ports contain

- A FIFO data-structure holding values in a port
- A list of continuations that can be executed pending message arrival and arbitration
  - A continuation is represented by a C# delegate
  - Can either be named or anonymous

Arbiters

- Implement common concurrency constructs and patterns like
  - choice
  - join
  - interleaved calculations
  - (batch, persistent,…)
CCR Primitives

Single item receiver
- Executes code when a message arrives

Choice arbiter
- Chooses one receiver (join or single item) from many, across different ports, executes only first one with conditions met, discarding others

Join expressions
- Static join expressions
- Dynamic over a runtime specified number of ports and messages

Interleave arbiter
- Teardown group, Concurrent Group, Exclusive Group
Example: Receive

Port<string> port = new Port<string>();
stringPort.Post("StringA");

Arbiter.Activate(
    Arbiter.Receive(stringPort, delegate(string s)
    {
        Console.WriteLine("Received: " + s);
    }
);

//Multiple Items
Port<String> stringPort = new Port<String>();
for (int i = 0; i < 50; i++) stringPort.Post(i.ToString());

Arbiter.Activate(
    Arbiter.MultipleItemReceive(stringPort, 10,
        delegate(String[] strings)
        {
            Msg("Ten strings={0}", String.Join("", "", strings));
        }
    ));
Example: Choice

PortSet<int, string> port = new PortSet<int, string>();

Arbiter.Activate(
    Arbiter.Choice(port, MyIntHandler, MyStringHandler)
);

void MyIntHandler(int i)
{
    Console.WriteLine("Received: " + i);
}

void MyStringHandler(string s)
{
    Console.WriteLine("Received: " + s);
}
Example: Join

Port<double> balancePort = new Port<double>();
Port<int> depositPort = new Port<int>();

Arbiter.Activate(
    Arbiter.JoinedReceive<int,double>(true,
    depositPort, balancePort,
    delegate(int b, double d)
    {
        balance.post(b + d);
    }
);
Example: Dynamic Join

PortSet<Result, Exception> resultsPort =
    new PortSet<int>();

    // parallel computation by posting requests
For (int i=0; i<N; i++)
{
    computePort.Post(new DoWork(someData, resultsPort));
}

    // requests complete asynchronously with unknown number
    // of failures vs. successes
Arbiter.Activate(
    Arbiter.MultipleItemReceive(resultsPort,
        delegate (ICollection<Result> successes,
                     ICollection<Exception> failures)
        {
            foreach(Result r in results)
            {
                ......
            }
        });
Declarative Coordination for Services

[ServiceHandler(ServiceHandlerBehavior.Concurrent)]
public IEnumerator<ITask> GetHandler(Get get)
{
    get.ResponsePort.Post(_state);
    yield break;
}

[ServiceHandler(ServiceHandlerBehavior.Exclusive)]
public IEnumerator<ITask> UpdateHandler(Update update)
{
    update.ResponsePort.Post(new UpdateResponse());
    yield break;
}
Declarative DataContract for Services

```
[DataContract]
public class ServiceState
{
    private string _member = "This is my State!";

    [DataMember]
    public string Member
    {
        get { return _member; }  
        set { _member = value; }
    }

    private int _ticks;

    [DataMember]
    public int Ticks
    {
        get { return _ticks; }  
        set { _ticks = value; }
    }
}
```
Summary: Runtime Environment

Decentralized Software Services (DSS)
- Easy access: View, access component state
- Flexibility: discover, start, stop services
- Distribution: transparency of service location
- Reusable, composable, scalable

Concurrency and Coordination Runtime (CCR)
- Ease of use: small, simple library, that avoids the complexity of manual threading, locks, semaphores, etc
- Concurrency: coordination primitives
  Based on asynchronous message passing
- Execution context: isolated from infrastructure, other services
Microsoft Robotics Studio

A development platform for the robotics community, supporting a wide variety of users, hardware, and application scenarios
Authoring tools: develop/run a robot application

Web browser
- Inspect/change service state
- Run applications (e.g. JScript as language)

Visual Programming Language (VPL)
- Dataflow editing
- Model and generate
- Novice to expert

Visual Studio and .NET tools
- C# 
- C++/CLI
- Iron Python
- Visual Basic
<?xml version="1.0" encoding="utf-8" ?>

- <State xmlns:s="http://www.w3.org/
  xmlns:wse="http://schemas.xmlsoap.org/
  xmlns:d="http://schemas.microsoft.com/"
  xmlns:m="http://schemas.microsoft.com/"
  xmlns:image="http://schemas.microsoft.com/"
  xmlns=":">
  <DistanceMeasurements>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
    <int>8000</int>
void TimerHandler(DateTime signal)
{
    _mainPort.Post(new IncrementTick());

    Activate(
        arbiter.Receive(false, TimeoutPort(1000),
            delegate(DateTime time)
            {
                _timerPort.Post(time);
            }));

    [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
    public IEnumerator<ITask> GetHandler(ITask get)
    {
        get.ResponsePort.Post(_state);
        yield break;
    }

    [ServiceHandler(ServiceHandlerBehavior.Concurrent)]
    public IEnumerator<ITask> HttpGetHandler(HttpGet httpGet)
    {
        httpGet.ResponsePort.Post(new HttpResponseMessage(_state)).
        yield break;
    }
from Microsoft.Ccr.Core import *
from Microsoft.Dss.Hosting import *
import Microsoft.Dss.ServiceModel.Dssp as dssp
import W3c.Soap as w3c

manifestLocation = System.IO.Path.Combine(System.Environment.CurrentDirectory, ..\samples\config\LEGO.NXT.MotorTou
DssEnvironment.Initialize(50000, 50001, manifestLocation)

def Shutdown():
    global DssEnvironment
    DssEnvironment.Shutdown()
    System.Environment.Exit(0)

def DirectoryQueryFailure(failure):
    print "Could not find service"
    Shutdown()

def bumperUpdate(notification):
    if notification.Body.Pressed :
        print "Ouch - the bumper was pressed."

def DirectoryQuerySuccess(serviceInfo):
    global DssEnvironment
    try :
        bumperPort = DssEnvironment.ServiceForwarder[bumper.ContactSensorArrayOperations](System.Uri(serviceInfo.Serv
bumperNotificationPort = bumper.ContactSensorArrayOperations()
        bumperPort.Subscribe(bumperNotificationPort)
        print "Subscribing to bumpers..."
        Arbiter.Activate( DssEnvironment.TaskQueue, Arbiter.Receive[bumper.Update](True, bumperNotificationPort, bumper
        except :
            print "Could not subscribe to bumper:";
            sys.exc_info()[0]
    Shutdown()

    DirectoryQuerySuccess,DirectoryQueryFailure ))

    print "Wait a few seconds for bumpers or press 'Enter' anytime to exit"
    System.Console.ReadLine()
    Shutdown()
Reasons for simulator

Reasons:
- Robotics hardware is expensive
- Often a limited ressource (team)
- Difficult to debug and reproduce

Pros:
- Easy for prototyping
- Low barrier to entry
- Staged approach
- Useful for education

Cons:
- Incomplete model
- Lack of noisy data
- Accurate tuning?
Simulator
Simulator
Simulator Architecture: Engine Service

- Implemented as a service
- Maintains world state
- Manages input devices
- 3D rendering using XNA
- Ageia Physics Simulation
- Graphical User Interface
- Editor for modeling and debugging
Microsoft Robotics Studio

A development platform for the robotics community, supporting a wide variety of users, hardware, and application scenarios.

Services and Samples
- Tutorials
- Robot services
- Robot models
- Technology services
Supported Sensor: SICK LRF

SICK LRF
  – Driver in MSRS 1.0/1.5
  – Now improved version

Additional on roadmap
Supported Actuator: KUKA LBR3

KUKA LBR3
Samples: KUKA Educational Framework
Samples: Maze Simulator
Samples: Sumo
Samples: RoboCup
MSRS history:

First CTP June 2006,…

MSRS 1.0: December 2006
  – First release, include full runtime, parts of authoring, samples

MSRS 1.5: June 2007
  – Runtime
    • (DSS,CRR) ported to .NET CF
    • performance improvements
  – VPL
    • Code Generation, deployment, Manifest Editor
  – Simulator
    • UI, shadows, material editor
  – better documentation, new samples,
  – Updates/Bugfixes in Aug07+Nov07
Other Robotics Toolkits

- OpenSource
  - Open Dynamics Engine (ODE)
  - Simbad
  - TeamBots
  - Khepera II
  - Gazebo
  - CARMEN

- Lego Mindstorms and Lego NXT
  - Legos own development tool chain (RIS, NXGen)
  - Extensions: firmware, OS, programming languages (lejos, nqc/nqx, etc)
References

URL:

- Website
  http://www.microsoft.com/robotics

- Newsgroup/Forum
  http://msdn.microsoft.com/robotics/

- Channel9 wiki

- “A robot in every home” Bill Gates
  http://go.microsoft.com/?LinkID=5950849

- Parallel programming with .NET
  http://blogs.msdn.com/pfxteam/

- Blog
References

URL:
- Ben Axelrod
  http://www.benaxelrod.com/MSRS/
- Trevor Taylor
- KUKA Educational Framework
- DARPA Urban challenge, PAVE project
  http://pave.princeton.edu/main/
- IBM: Open source robotic toolkits, alphaworks
References

Books (in summer 2008)

– Programming Microsoft Robotics Studio, Sara Morgan
  Microsoft Press

– Professional Microsoft Robotics Studio
  Kyle Johns, Trevor Taylor, (Martin Calsyn)
  Wrox