Agent-Oriented Programming
One Step Beyond OOP

Jürg Gutknecht
ETH Zürich
October 2004

Overview

- Some goals
- The standard OOP computing model
- A new model unifying OOP and concurrency
- The experimental language *Active C#*
- Some archetypal use cases
- Remote user interfaces: a vision
- Conclusion
Some Goals …

- Create a better (more „realistic“) model
- Integrate concurrency with OOP
- Puts concurrency „behind the scenes“ (language constructs replace library calls)
- Abstract from deployment details (central or distributed)
- Introduce new kind of programming-language independent interfaces or contracts
- Present active objects as self-contained units with programming-language independent interfaces
- Allow 1:1 mapping of active objects to devices

The Standard OOP Computing Model

„2-Class Society“
Active Objects …

Object a

Caller (Client)

Intrinsic activity

Active object

Object b

Callee (Server)

Active object

… and Active Links

Object a

Caller (Client)

Dialog activity

Dialog

Object b

Callee (Server)

Active link
Dialogs

Object a

Caller (Client)

Dialog activity

Object b

Callee (Server)

Active link

Dialog Syntax

Protocol = \{ Token \{ Token \} \{ Token \}\).

dialog definition in EBNF

Dialog Instance
Comparison

<table>
<thead>
<tr>
<th>methods</th>
<th>dialogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocking</td>
<td>non blocking</td>
</tr>
<tr>
<td>stateless</td>
<td>stateful</td>
</tr>
<tr>
<td>parameters</td>
<td>syntax</td>
</tr>
<tr>
<td>language dependent</td>
<td>language independent</td>
</tr>
</tbody>
</table>

Experiment Active C#
Project within MSR ROTOR initiative

- **activity** new class member
  - anonymous: started implicitly after constructor
  - named: started explicitly
- **dialog** interface entities (contracts)
  - Specify dialog syntax and keywords
  - Implemented by correspondingly named activities
- "!" and "?" send and receive operators
- **lock** mutual exclusion relative to object scope
- **await cond** waiting for condition
- **passivate dt**
Experiment Active C#
Project within MSR Rotor initiative

- **activity** new class member
  - Anonymous: started implicitly after constructor
  - Named: started explicitly

- **dialog** interface entities (contracts)
  - Specify dialog syntax and keywords
  - Implemented by correspondingly named activities

- "!" and "?" send and receive operators

- **lock** mutual exclusion relative to object scope

- **await cond** waiting for condition

- **passivate dt**

---

Dialogs in Active C#

**Definition and Implementation**

```csharp
namespace N {
    dialog D { a, b, c } // definition with keywords
    class S {
        int i; ...
        public S (...) { ... }
        private bool f (...) { ... }
        public activity A: D {
            // implementation with parser
            D kw;
            ?kw; ... !2004;
        }
    }
}
```

- **Contract**
- **Server Code**
Dialogs in Active C#

- Use

```csharp
namespace N {
    dialog D { a, b, c } // definition with keywords
    class C {
        ...
        activity Behavior {
            S s = new S(...);
            dialog D d = new s.A;
            int i
            ... d!D.b; ... d?i;
        }
    }
}
```

Experiment Active C#
Project within MSR ROTOR initiative

- **activity** new class member
  - anonymous: started implicitly after constructor
  - named: started explicitly
- **dialog** interface entities (contracts)
  - Specify dialog syntax and keywords
  - Implemented by correspondingly named activities
- "!" and "?" send and receive operators
- **lock** mutual exclusion relative to object scope
- **await** cond waiting for condition
- **passivate** dt
Example: Finite Buffers

- **public void Put (object x)** {
  
  lock(this) {
  
  if (m == 0) { Monitor.Wait(this); }
  
  m--; buf[tail] = x;
  
  tail = (tail + 1) % size;
  
  n++; Monitor.Pulse(this);
  
  }
  
  }

- **public object Get ()** {
  
  lock(this) {
  
  if (n == 0) { Monitor.Wait(this); }
  
  n--; object x = buf[head];
  
  head = (head + 1) % size;
  
  m++; Monitor.Pulse(this); return x;
  
  }
  
  }

Finite Buffers

- **public void Put (object x)** {
  
  lock(this) {
  
  if (m == 0) { Monitor.Wait(this); }
  
  m--; buf[tail] = x;
  
  tail = (tail + 1) % size;
  
  n++; Monitor.Pulse(this);
  
  }
  
  }

- **public object Get ()** {
  
  lock(this) {
  
  if (n == 0) { Monitor.Wait(this); }
  
  n--; object x = buf[head];
  
  head = (head + 1) % size;
  
  m++; Monitor.Pulse(this); return x;
  
  }

  Correct?
Analysis

- Error scenario
  - Arrival Get₁ Get₂ Put
  - Pulse Put → Get₁ → Get₂
- Correction: Replace if with while

.NET model (without eggshell)
- Mixed queue (containing newly entering producers and consumers)
  - Correction: Replace Pulse with PulseAll

Eggshell model
- Producers only or consumers only in (inner) waiting list
  - Pulse correct

Finite Buffer with Signals

- public void Put (object x) {
  lock(this) {
    while (m == 0) { Monitor.Wait(this); }
    m--; buf[tail] = x;
    tail = (tail + 1) % size;
    n++; Monitor.PulseAll(this);
  }
}
- public object Get () {
  lock(this) {
    while (n == 0) { Monitor.Wait(this); }
    n--; object x = buf[head];
    head = (head + 1) % size;
    m++; Monitor.PulseAll(this); return x;
  }
}
Finite Buffers in Active C#

- `public void Put (object x) {`
  - `lock {`
    - `await (m != 0);`
    - `m--; buf[tail] = x;`
    - `tail = (tail + 1) % size;`
    - `n++;`
  - `}`
- `public object Get () {`
  - `lock {`
    - `await (n != 0);`
    - `n--; object x = buf[head];`
    - `head = (head + 1) % size;`
    - `m++; return x;`
  - `}`
  - `}`

Use Case 1: Monkey Rock Problem

The northern and southern monkeys have to eat and drink! There is a small rope between the two rocks. The rope can carry up to m>=1 monkeys. Concurrent crossing in both direction is not possible.

Note: There are some nasty hungry crocodiles below the rope.
Monkey Rocks in Active C#

dialog CoordMonkeyDialog { claim, release }
class Rope {
    const NofMonkeys = 25;
    const MaxOnRope = 4;
    const CrossingTime = 100;
    static int curOnRope = 0;
    // > 0, < 0 South-North, North-South traversal
    activity CoordMonkey: CoordMonkeyDialog {
        object msg;
        while (true) { ... }
    }
    static void Main() {
        for (int i = 0; i < NofMonkeys; i++) new Monkey();
    }
}

while (true) {
    ?msg; // wants South-North traversal
    lock {
        await (0 <= curOnRope) & (curOnRope < MaxOnRope);
        curOnRope++;
    }
    passivate(CrossingTime);
    lock { curOnRope--; }
    !CoordMonkeyDialog.release;
    ?msg; // wants North-South traversal
    lock {
        await (0 >= curOnRope) & (curOnRope > -MaxOnRope);
        curOnRope--;
    }
    passivate(CrossingTime);
    lock { curOnRope++; }
    !CoordMonkeyDialog.release;
}
Monkey Rocks in Active C#

- `dialog CoordMonkeyDialog { claim, release }
- `class Monkey {
  `static Random rnd = new Random();
  `activity {
    `dialog CoordMonkeyDialog d =
    `new Rope.CoordMonkey;
    `object msg;
    `while (true) {
      `passivate(rnd.Next(1000));
      // eat/drink for a random time
      `d!CoordMonkeyDialog.claim;
      // send keyword claim
      `d?msg; // receive whatever is sent
    }
  }
}

Use Case 2: Next Meeting Time

Jay Misra

[Diagram of a use case involving Manager and Coord, with actions like proposal and next avail]
Next Meeting Time in Active C#

dialog MeetingDialog {} // no keywords

class Coordinator {
    const int NofManagers = 20;
    static int proposal = 0;
    activity AllocMeeting: MeetingDialog {
        int t;
        while (true) {
            !proposal; ?t;
            lock {
                if (t > proposal) proposal = t;
                await(proposal > t);
            }
        }
    }
    static void Main() {
        for (int i = 0; i < NofManagers; i++)
            new Manager();
    }
}

class Manager {
    private int NextPossibleTime (int t) {
        // check agenda
    }
    activity {
        dialog MeetingDialog d = new Coordinator.AllocMeeting;
        int proposal, t;
        while (true) {
            d?proposal; d!NextPossibleTime(proposal);
        }
    }
}
Use Case 3: Frisbee Fun

```csharp
dialog FrisbeeDialog { request, take }

class Player {
    dialog FrisbeeDialog d = null;
    int nofFrisbees = 0;
    public Init (Player next; int nofFrisbees) { ... }
    activity ManageFrisbees: FrisbeeDialog { ... }
    activity { ... }
    static Random rnd = new Random();
    static void Main() { ... }
}
```
Frisbee Fun in Active C#

```csharp
activity ManageFrisbees: FrisbeeDialog {
    object msg;
    while (true) {
        lock { await (nofFrisbees == 0); }
        !FrisbeeDialog.request; ?msg;
        lock { nofFrisbees = 1; }
    }
}

activity {
    object msg;
    while (true) {
        lock { await (nofFrisbees != 0); }
        d?msg; d!FrisbeeDialog.take;
        lock { nofFrisbees = 0; }
    }
}

public Init (Player next; int nofFrisbees) {
    lock {
        d = new next.ManageFrisbees;
        this.nofFrisbees = nofFrisbees;
    }
}

static void Main() {
    const int NofPlayers = 11;
    Player last = new Player (), q = last;
    for (int i = 1; i < NofPlayers; i++) {
        Player p = new Player ();
        p.Init(q, rnd.next(2)); q = p;
    }
    last.Init(q, 0);
}
```
Use Case 4: Santa Claus

- Santa Claus sleeps at the North pole until awakened by either all of the nine reindeer, or by a group of three out of ten elves. He performs one of two indivisible actions:
  - If awakened by the group of reindeer, Santa harnesses them to a sleigh, delivers toys, and finally unharnesses the reindeer who then go on vacation.
  - If awakened by a group of elves, Santa shows them into his office, consults with them on toy R&D, and finally shows them out so they can return to work constructing toys.
- A waiting group of reindeer must be served by Santa before a waiting group of elves. Since Santa's time is extremely valuable, marshalling the reindeer or elves into a group must not be done by Santa.

Santa Claus

- Discussed and solved by Ben-Ari with Rendez-Vous (in Ada95) and monitors (in Java) in "How to Solve the Santa Claus Problem, M. Ben-Ari, Wiley & Sons, 1997".
Problem Extension: Negotiation

- Before joining, elves should be informed about the expected waiting time and be given the opportunity to withdraw
- Extension of dialog syntax

CoordElf = join ( Negotiation | reject ).
Negotiation = [ wait join ] release | wait release.
Reindeers as Active Objects

- `dialog Contact { join, release }
- `class Reindeer {
  `activity {
    object t;
    Contact c = new Coordinator.CoordReindeer;
    while (true) {
      passivate (Christmas.Rnd());
      c!Contact.join;
      c?t;
    }
  }
}

Elves as Active Objects

- `dialog XContact { join, reject, wait, release }
- `class Elf {
  `activity {
    object t;
    XContact c = new Coordinator.CoordElves;
    while (true) {
      Thread.Sleep(Christmas.Rnd());
      c!XContact.join;
      c?t;
      if (XContact)t == XContact.wait)
        if (Christmas.Rnd % 3) == 0)
          c!XContact.release;
        else { c!XContact.join; c?t; }
    }
  }
}
Santa as Server

- **dialog Service** { deliver, consult, done }
- **class Santa**
  
  ```csharp
  const int consultTime = 10, deliverTime = 20;
  public static activity Work: Service {
    object t;
    while (true) {
      ?t;
      if (Service)t == Service.deliver) {
        Console.WriteLine("Santa delivering");
        passivate (deliverTime); } 
      else {
        Console.WriteLine("Santa consulting");
        passivate (consultTime); }
      !Service.done
    }
  }
  }
```

Coordinator as Active Server

- **class Coordinator**
  
  ```csharp
  static int rGo = 0, rBuild = 0, rSize = 0;
  static int eGo = 0, eBuild = 0, eSize = 0;
  public static activity CoordReindeer: Contact {
  }
  public static activity CoordElves: XContact {
  static activity {
    lock {
      object t;
      Service s = new Santa.Work;
      while (true) {
        await ((rBuild > rGo) || (eBuild > eGo));
        if (rBuild > rGo) {
          s!Service.deliver; s?t; rGo++;
        } else {
          s!Service.consult; s?t; eGo++;
        }
      }
    }
  }
  ```
Reindeer Coordination

public static activity CoordReindeer: Contact {
    lock {
        object t;
        int groupNo;
        while (true) {
            ?t;
            groupNo = rBuild; rSize++;
            if (rSize == Christmas.reqReindeer) {
                rSize = 0; rBuild++;
            };
            await (rGo > groupNo);
            !Contact.release;
        }
    }
}

Elf Coordination

public static activity CoordElves: XContact {
    lock {
        object t; int groupNo = -9999;
        while (true) {
            ?t;
            if (eBuild <= groupNo + 2) !XCoord.reject;
            else {
                if (eGo < eBuild) { !XContact.wait; ?t; };
                if (XCoord)t == XCoord.join) {
                    groupNo = eBuild; eSize++;
                    if (eSize == Christmas.reqElves) {
                        eSize = 0; eBuild++;
                    };
                    await (eGo > groupNo);
                    !XContact.release;
                }
            }
        }
    }
}
Christmas Scenario

- **public class Christmas**
  - public const int nofReindeer = 9;
  - public const reqReindeer = 9;
  - public const int nofElves = 10;
  - public const int reqElves = 3;
  - static Random rnd = new Random();
  - public static int Rnd () {
      return rnd.Next(1000); }
  - static void Main() {
      for (int i = 0; i < nofReindeer; i++)<
          new Reindeer ();
      for (int i = 0; i < nofElves; i++)>
          new Elf ();
  }
}

Automated Rental System: Server

dialog Negotiate { Accept, Return };

using System, System.Dialog;
public class RentalStation {
  static int nofFree;
  static bool [] free;
  static Rental () { // constructor }
  static activity RentalService: Negotiate { // dialog implementation
  }
  private static int Next (int obj) { }
  private static void Free (int obj) { }
  static void Main() { // main activity }
}
Automated Rental System: Server

```csharp
static RentalStation () {
    free = new bool [100];
    for (int i = 0; i < free.Length; i++) {
        free[i] = true;
    }
    nofFree = 100;
}

static void Main() {
    DialogManager.Start(typeof(Rental),
        new TCPTransportManager());
}
```

Automated Rental System: Server

```csharp
static activity RentalService: Negotiate {
    Negotiate msg;
    int obj = -1;
    do {
        obj = Next(obj); !obj; ?msg;
        if (msg != Negotiate.Accept) Free(obj);
    } while (msg != Negotiate.Accept);
    ?msg; // return
    Free(obj)
}
```
Automated Rental System: Server

```java
private static void Next (int obj) {
    lock {
        await (nofFree > 0);
        while (!free[obj++ % 100]) {};
        free[obj] = false; nofFree--;
        return obj;
    }
}

private static void Free (int obj) {
    lock {
        free[obj] = true; nofFree++;
    }
}
```

Automated Rental System: Client

dialog Negotiate { Accept, Return };

```java
using System, System.Dialog;
class Client {
    static void Main(string[] args) {
        DialogManager.Start(new TCPTransportManager(args[0]));
        dialog Dialog d = DialogManager.Open("RentalStation", "RentalService", typeof(Negotiate));
        do { d?obj;
            bool suitable = Check(obj);
            if (!suitable) d!Negotiate.Return;
        } while (!suitable)
        d!Negotiate.Accept;
        // now use rental object
        d!Negotiate.Return;
    }
    DialogManager.Stop(true);
}
```
Automatic Parser Generator

- Use attributed syntax
- Treat it as C# attributes of a dialog declaration

```csharp
[ syntax
    "Negotiate = int @Eval {↓Return int @Eval}
    ↓Accept @Use ↓Return."
]
dialog Negotiate { Accept, Return };
```

A Vision: From Local UI ...

[Image of an ATM and a group of people]
A Vision: ... To Remote UI

Contract

ATMService = CustomerID Number
("withdraw"
   [ { Destination | Choice } | "pay"].
CustomerID = CharString "." CharString.
Destination = "cash" | "chip" | "account".
Choice = Number [ AccountID ].

dialog

Summary

- The New Model
  - Meets the goals stated at the beginning
  - Has proved its suitability in numerous case studies
  - Suggests new modes of interoperability
  - Is implemented in the form of Active C#,
    available from
    http://www.avocado.ethz.ch/ActiveCSharp/