

System Design, Part 2

©Andrey Kruglyak, 2010

Today

- ◀ Design of distributed systems
- ◀ Design of embedded systems
 - ◀ using Timber
- ◀ Component-based design
 - ◀ of embedded systems
 - ◀ using Timber

Distributed systems

Distributed systems

- ◀ A distributed system consists of multiple autonomous systems united by a common goal and communicating through a network.
 - ◀ Homogenous computing systems programmed as a single system
 - computer clusters (parallel programming model)
 - multiplayer games (central server + X clients / peer-to-peer)
 - distributed databases
 - ◀ Heterogenous computing systems where nodes are programmed independently and have different functions (e.g. control systems in cars, airplanes, GSM networks)

Distributed systems

- A single distributed system or several cooperating systems?
 - Often a question of perspective
 - Often determined by external factors (a customer is commissioning a system that should conform to a certain specification, this system may need to be distributed)
 - If the communication protocol is well-defined in advance, there is a good chance we are working with individual systems
 - If the membership of the system is static, there is a good chance we are working with a distributed system

Friday, May 28, 2010

5

Design of distributed systems

- Homogeneous: parallel programming model or duplication of systems + communication infrastructure
- Heterogeneous: much more challenging
 - Approach 1 : explicit division into nodes (e.g. TTA)
 - Approach 2 : middleware masking problems of heterogeneity and network communication (e.g. CORBA, Common Object Request Broker Architecture, AMI, Asynchronous Method Invocation, .NET)

Friday, May 28, 2010

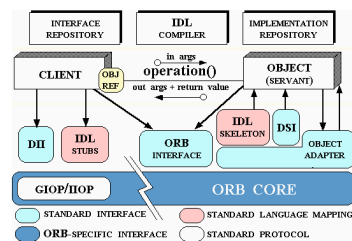
6

CORBA

The Object Request Broker

- provides a mechanism for transparently communicating client requests to target object implementations
- simplifies distributed programming by decoupling the client from the details of the method invocations, client requests appear to be local procedure calls

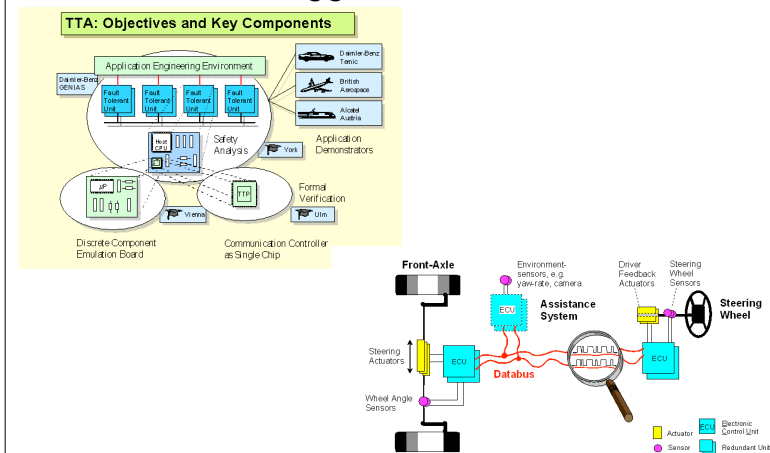
When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller



Friday, May 28, 2010

7

Time-Triggered Architecture



Friday, May 28, 2010

8

Sensor networks

- ◁ A special case of a distributed system
- ◁ Additional challenges:
 - ◁ can be massively parallel => requires data aggregation
 - ◁ stationary or mobile, with dynamic membership
 - ◁ context awareness may be required (different behavior in different surroundings, incl. presence of other nodes)
 - ◁ central or distributed decision making
 - ◁ QoS requirements (system must be more robust than individual nodes)

Friday, May 28, 2010

9

Embedded systems

Friday, May 28, 2010

10

Embedded systems

- ◁ Unlike general-purpose computing systems that are designed to perform a wide range of computations, an embedded computing system is a part of a larger device and is designed to perform specific functions supporting the functionality of the device
- ◁ A customer often specifies the requirements for the whole system, including mechanical parts, hardware parts and software, and the exact specification of the software is left to the developer
- ◁ Examples include various control systems, consumer electronics, etc.

Friday, May 28, 2010

11

Properties of ES

- ◁ Resource-constrained
 - ◁ CPU, memory, battery life, size
- ◁ A lot of interaction between software and hardware
- ◁ Real-time: the time when the result of a computation is delivered is as important for correct functioning of the systems as is the computed value
 - ◁ The source of real-time requirements is interaction with surrounding hardware or communication protocols for radio or network communication
- ◁ Safety-critical

Friday, May 28, 2010

12

Real-time systems

- ◀ Most embedded systems are real-time
- ◀ Timing requirements: meeting reaction deadlines & limiting jitter
- ◀ Hard RTS: missing a deadline is equivalent to system failure
- ◀ Soft RTS: the value of the computed result diminishes with time (QoS metric)
- ◀ Mixed RTS: some tasks are hard RT, others have no timing requirements or are soft RT

Friday, May 28, 2010

13

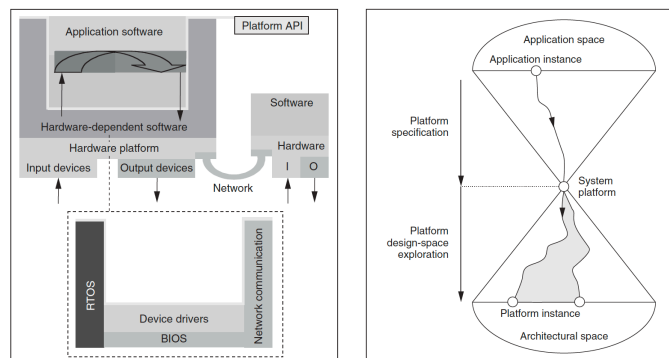
ES design methodologies

- ◀ Time-Triggered Architecture (distributed systems, does not describe how nodes should be developed; bandwidth reservation)
- ◀ Targeting QoS (\Leftarrow network latency & bandwidth)
 - ◀ RT-CORBA (added timing specifications and QoS enforcement)
 - ◀ Adaptive RTS for QoS management (dynamic priorities to improve utilization of network bandwidth)
- ◀ Platform-based design
- ◀ Component-based design (later today)
- ◀ ES design using Timber (hard RT only, no support for QoS)

Friday, May 28, 2010

14

Platform-based design of ES



Friday, May 28, 2010

15

ES design methodologies

- ◀ Time-Triggered Architecture (distributed systems, does not describe how nodes should be developed; bandwidth reservation)
- ◀ Targeting QoS (\Leftarrow network latency & bandwidth)
 - ◀ RT-CORBA (added timing specifications and QoS enforcement)
 - ◀ Adaptive RTS for QoS management (dynamic priorities to improve utilization of network bandwidth)
- ◀ Platform-based design
- ◀ Component-based design (later today)
- ◀ ES design using Timber (hard RT only, no support for QoS)

Friday, May 28, 2010

16

Using Timber for modeling and implementation of ES

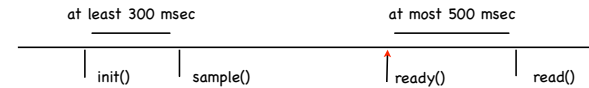
- ◀ Modeling hardware in Timber is possible due to reactivity and massive parallelism (this is also true of high-level synchronous languages)
- ◀ Consistent modeling of software and hardware is an advantage since it is often easier to model the whole embedded system than to model only its software part
- ◀ Timber is sufficiently abstract to be used for modeling and at the same time Timber code can be executed (no translation from model to implementation) [there are performance issues that need to be addressed]
- ◀ Timber allows to specify timing in the code, for each method invocation; this timing specification is preserved in the code and is used by Timber run-time system to guide scheduling

Friday, May 28, 2010

17

Timing specification using Timber

- ◀ Hardware interaction protocols:



Friday, May 28, 2010

18

Timing specification using Timber

- ◀ Hardware interaction protocols:



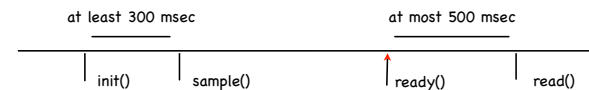
```
m1 = after (sec 0) before (millisec 50) action
  portA.init -- request
  after (millisec 350) before (millisec 50) m2
m2 = action
  portA.sample -- request
ready = before (millisec 500) action
  v <- portA.read --request
  ...
```

Friday, May 28, 2010

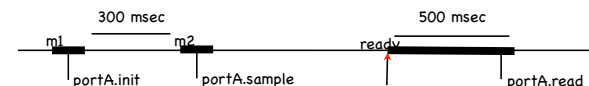
19

Timing specification using Timber

- ◀ Hardware interaction protocols:



```
m1 = after (sec 0) before (millisec 50) action
  portA.init -- request
  after (millisec 350) before (millisec 50) m2
m2 = action
  portA.sample -- request
ready = before (millisec 500) action
  v <- portA.read --request
  ...
```



Friday, May 28, 2010

20

Timing specification using Timber

- Periodic processes with period Y and maximum jitter $2 * X$

```
p = before X action
...
after Y p
```

- Y can depend on arguments or other input:

```
p t = before X action
if (t > 5) then
  after (sec 5) p
else
  after (sec t) p
```

Timing specification using Timber

- Another important case is inheritance of the baseline & deadline (written without `after/before`)

- To disallow setting new baseline that has already passed we define

`after X => new baseline is "max(current_baseline + X, now)"`

which gives us

`after 0 => new baseline is "now"`



Verification of ES

- As ES are often safety-critical, they need to be verified by more than testing
- Verification of implementation
 - compiler certification
- Verification of the model: correctness of computation algorithms, memory consumption, liveness, deadlines
 - formal verification, incl. model-checking (FSM, timed automata)
 - schedulability analysis (Processor Demand Analysis)
- Verification is possible for Timber (ongoing work)

Component-Based Design

CBD of software systems

- ◁ CBD has proven to be an effective approach to development of general-purpose software systems
- ◁ It is essentially an extension of OO design, developed to maximize software re-use, improve modularity and maintainability of software
- ◁ Some CBD methodologies also improve robustness by making components into fault containment regions

Friday, May 28, 2010

25

Component definition

- ◁ There are many conflicting definitions, below is one of the most widely accepted (by Clemens Szyperski):

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition."
- ◁ Independent deployment is important for distributed systems and for large on-line software systems (typically achieved by dynamic linking)

Friday, May 28, 2010

26

Goals/advantages of CBD

- ◁ Giving structure to systems under development
- ◁ Reuse of development effort
- ◁ Supporting system maintenance and evolution
- ◁ Enabling a market for software parts

Friday, May 28, 2010

27

Component-based design: standards and tools

- ◁ JavaBeans/Enterprise JavaBeans (Sun)
- ◁ .NET/COM (Microsoft)
- ◁ CORBA (OMG group)
- ◁ lots of other models in research

Friday, May 28, 2010

28

Component-Based Design of ES

Friday, May 28, 2010

29

Component-Based Design of ES

- ◀ CBD of ES has been an active research topic but with little success in industry so far, mainly hampered by
 - ◀ a lack of widely adopted standards for component technology (with different sectors of industry having different priorities)
 - ◀ absence of solutions for specification & predicting of timing and QoS properties
 - ◀ difficulty in separating hardware from software in system specification (and hence the model)

Friday, May 28, 2010

30

Using Timber for CBD of RTS

- ◀ Advantages:
 - ◀ consistent modeling of hardware and software
 - ◀ timing specification (at object/method level) integrated into the structural/functional model
 - ◀ straightforward mapping from model to implementation

Friday, May 28, 2010

31

Using Timber for CBD of RTS

- ◀ Challenges:
 - ◀ no support for QoS (soft RT requirements) in the language
 - ◀ the language for expressing timing specification (baseline + deadline) is probably not sufficient for specifying timing behavior of a component
 - ◀ no formal verification of consistency of timing specification at component (or system) and object level, no algorithm for deduction of timing specification
 - ◀ the ongoing work on verification of timing properties of an implementation (on a specific hardware platform) has not been completed

Friday, May 28, 2010

32