Zerberus System
A tool chain for the development of augmented reality applications
Personal Background

- Ph.D. at the University of Munich
- Chair VI: Embedded Systems and Robotics
- Interests: safety critical applications, real-time systems, development models
Content

• Introduction
  – Example application
  – Requirements elicitation
  – Demands on development models

• Zerberus System
  – Development process
  – Zerberus language

• Applicability of Zerberus regarding AR
  – Discrepancies
  – Solutions

• Summary and Discussion
Introduction
Example application: Medical assistance

**Task:** Assistance during an minimal-invasive operation

**Sensors:**
- Ultrasound
- Video
- magnetic resonance imaging
- Computed tomography scans

**Requirements:**
- Safety
- Real-time

**Further Applications:**
Telemedicine -> Reliability
Example application: Robotics

Task:
- Remote robot control
- Simulation of robot movements

Sensors:
- Video
- Servos

Requirements:
- Safety
- Reliability
Example Application: Visual Annotation

**Task:**
- Assistance in maintenance

**Sensors:**
- Video
- User Input (buttons, voice)

**Requirements:**
- Real-time
Summary of Requirements:

- Safety
- Reliability
- Real-time
- Different sensors
- Distributed computing

- Time consuming computations
  ⇒ Portability to new more powerful hardware
Requests on development models

- Simplicity
- Cost-efficiency
- Development process acceleration
- Tool support
- Automatic code generation
- Usability by application domain expert
- Inherent safety
Zerberus System
Background

• Current development process for safety critical applications:
  – Development is based on internal experiences of the company
  – Many domain experts are involved (real-time systems, fault-tolerance, application domain)
  – Certification authority can only check if standards are met
  – Long-lasting, error prone development process
  – Time and cost-intensive
Zerberus Approach

• Suggestion of a development model
• Provision of a tool chain
• Support of automatic code generation
• Based on commercial-of-the-shelf hardware
• Platform independent specification of the functional model
• Timing restrictions can be specified directly in the functional model
• Automatic realization of fault-tolerance mechanisms
• Only application-dependent code has to be implemented by the developer
Fault-tolerance mechanisms

- Active structural redundancy
- Diversity in hardware and software is supported
- Developer can choose the appropriate level of fault-tolerance
Development process

Step 1: Design of the functional model

Step 2: Analysis of the requirements on the system dependability

Step 3: Implementation of application dependent code

Step 4: Selection of Zerberus Runtime System

Step 5: Code Generation

Executable Code
Requirements on the specification language

- Division of the application into the functional parts
- Determinism
- Specification of temporal constraints

For fault-tolerance mechanisms:
- Replica determinism
- State synchronization
Zerberus language introduction

Attributes:

- Time-triggered
- Platform independent
- Intuitive and simple (only 7 object types)
- Extendable
- Few restrictions on IO
Introductory example

Task:

• Robot should move in the laboratory without collisions

• Robot should pick up objects and move them to targets

• Jobs are send to the robot via wireless LAN

• Via radio the robot can determine its position
Separation of inner state: Ports

- Ports are distinct places in memory
- Specified size
- Representation of values is unique on all platforms
- Read- and write accesses are performed time-triggered -> determinism
- Voting algorithms are based on the port values
- Integration of units is based on port values
Ports in the application:

- Object coordinates
- Target coordinates
- Joint settings
- Motor settings
- Camera data
- Job Queue
- Robot path plan
- Path plan for robot arm

```
Robot coordinates
Job Queue
Camera data
Robot path plan
Robot arm path plan
Motor settings
Joint settings
```
Functional Elements: Tasks

- Periodically called functions
- No inner synchronization points
- All communication between tasks is performed via ports
- Tasks start logically at the begin of each period and finish at the end of the assigned period
- The physical execution on the CPU is transparent to the user
Application Tasks:

- Path planner
- Robot arm path planner
Problem: Bumpy curves

⇒ Long execution times
⇒ Bumpy curves
⇒ Introduction of steering tasks:

• Motor control task
• Joint control task
Realization of IO: Sensors and Actors

- Simple IO functions
- Execution occurs time-triggered
- IO functions are executed within the run-time systems context
- Synchrony assumption: execution is performed instantaneous
Application IO

- **Sensors:**
  - Camera
  - WLAN
  - Radio

- **Actors:**
  - Motor
  - Joints
Supporting different application modes: Mode

General:
- Set of tasks, sensors and actors with specified execution frequencies
- Specified mode cycle duration

Application Modes:
- Robot movement towards objects or goals
- Robot arm movement
Switching the context: mode changes and guards

- Binary functions based on port values

- Executed in run-time system's context (synchrony assumption)

- Mode changes: way to change modes (only at the end of one mode cycle possible)

- Guards: more precise possibility to control the application
The whole application:
Execution Model:

Steps performed by the system in each round:

1. Termination of tasks
2. Voting and synchronization
3. Actor execution
4. Modechange evaluation
5. Sensor execution
6. Task start
7. Advance time
Logical execution

mode mode1
{
  task= t1 1, t2 2;
  actor= a 2;
  sensor= s 1;
  duration= 500000000 ns;
}

Simple mode declaration
Actual execution
Further steps in the development process
Step 2: Implementation of application dependant functions

Functions that have to be implemented:

- task functions
- sensor functions
- actor functions
- modechange functions
- guard functions
Step 3: Selection of fault-tolerance mechanisms

Currently supported:
- Active structural redundancy
- Hardware diversity
- Software diversity (N-Version programming)
- others will be supported
Step 4: Selection of platform

- Zerberus run-time systems are provided for different platforms

- Run-time systems realize the execution of Zerberus applications, the synchronization, integration and voting

- Extensibility: User can implement own run-time systems

- Reusability: Zerberus tags to support application independent implementation of run-time systems
Step 5: Code Generation

- Zerberus File
- AD-Checker
- RTF-Parser
- RTF-Checker
- Code-Generator
- Executable Code
- Application Data
- ZF-Parser
- Runtime Files
- Fault-tolerance mechanisms
- Application dependent code
Augmented Reality: Using the Zerberus System
Requirements:

- Safety **ok**
- Reliability **ok**
- Real-time **ok**
- Different sensors **ok**
- Distributed computing **ok**
- Portability to new more powerful hardware **ok**

But:
- Current need for redundancy
- Destructive voting
Demands:

More flexible layout of the applications:

- limiting the needs for redundancy:
  - single unit applications
  - single application processes

- using results of different units in a more constructive way
  - merge of redundant results (e.g.: in case of different sensors)
More flexibility:

- Abdication of need for redundancy
  - Introduction of a possibility to spread the different tasks on the units
  - relocation of tasks in case of a unit failure
  - Tool supported task assignment

- Solution for augmented reality:
  - Use of redundancy only if safety standards have to be met
Constructive voting

• Separation of functional design and fault-tolerance
  – Events: Deterministic points in time to execute fault tolerance mechanisms (voting, plausibility tests)
  – Exception: Occurrence of faults
  – Handling: Fault reaction (separation of faulty unit, process rollback, constructive methods, application dependant reactions)

• Solution for augmented reality:
  – introduction of an interface for application dependant mechanisms
Literature:


Summary:

• Requirements regarding typical augmented reality applications were gathered

• The Zerberus System can fulfill these requirements

• Solutions for remaining problems were suggested
Let's discuss