ECE452 / CS446 / SE464

Project Tutorial - Delivery 1

Monday, May 8, 2006

Sean Lau (sqlau@crete)
Course Website: swen.uwaterloo.ca/~se2

Overview

- Introduction to Delivery 1
- Baseline Functionality for Delivery 1
- System Variability Proposal / Negotiation
- Design Document
- Implementation
  - Phone Control Software
- Looking Ahead: Demo & Peer Review
- Summary

Introduction

- the first iteration of the project involves building a subset of the phone control software and a subset of the information management system
  - the output is a working interim build and a design document which reflects the build
- project is front loaded - a good design and implementation now should result in less work in the second iteration

Baseline Functionality (I)

- **baseline functionality**: the minimum functional requirements which must be present in all systems
- the baseline functionality for the first iteration represents enough to capture the architecture
- there will be a second set of baseline functionality for the second deliverable that complements this set

Baseline Functionality (II)

- normal successful call
  - extension mapping, permissions & system load
- add accounts and assign IP phone, extension & permissions through the administrator console
- define valid billing plans & assign to account
- manual hardware tests, system load
- administrator console login / logoff
- input validation

System Variability (I)

- different TAs in SE1 = different functional requirements between groups
- **value-added functionality**: features which exceed the baseline and documented in your SRS
- required to incorporate three pieces of functionality of **moderate complexity**
- examples include call display, call waiting, and call filtering

System Variability (II)

- the variable functionality will be negotiated with your TA during a short meeting (< 20 minutes)
  - meeting scheduling will be discussed later on
- put together a proposal (a list of value-added functionality, prioritized by your desire to incorporate them) and submit it to CourseBook 24 hours before the scheduled meeting
- bring hardcopies of your proposal and SRS to the meeting

Design Document

- documentation of the delivery 1 interim build
  - first half focuses on high level architectural decisions for planning development
  - second half focuses on detailed component design for understanding the implementation
- traceability between the code and the document is important
- submit electronically as a PDF file and physically as a bounded hardcopy

1.0 Introduction

- describe any functionality that has been included in the interim build that exceeds the baseline
- describe the value-added functionality that is / will be included in your system, as agreed upon by your TA
  - provide references to where the feature is defined in the SRS

2.0 Architecture Overview (I)

- captures a high-level view of the system and architectural decisions
- briefly describe design decisions and trade-offs
- summarize strengths and weaknesses of the design in a table

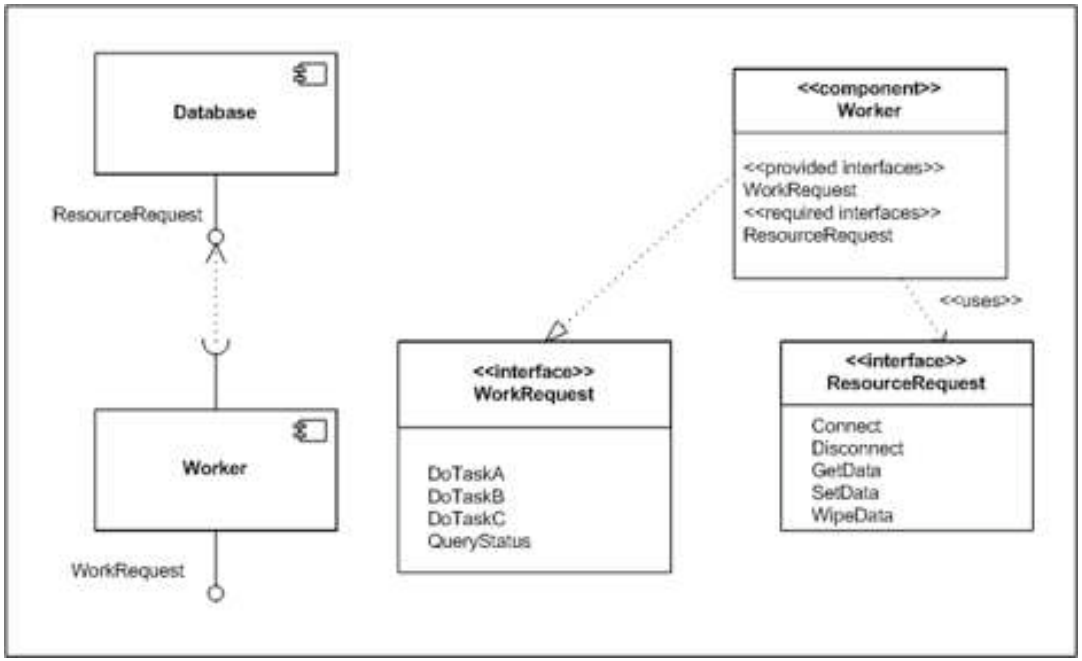
2.0 Architecture Overview (II)

- apply Kruchten's 4+1 approach to documenting the architecture
  - observation: one view is insufficient for addressing all concerns
  - main idea: four concurrent views to describe different concerns and one scenario view to tie the views together and validate the model
- this will be covered in greater detail during lectures

2.1 Logical View

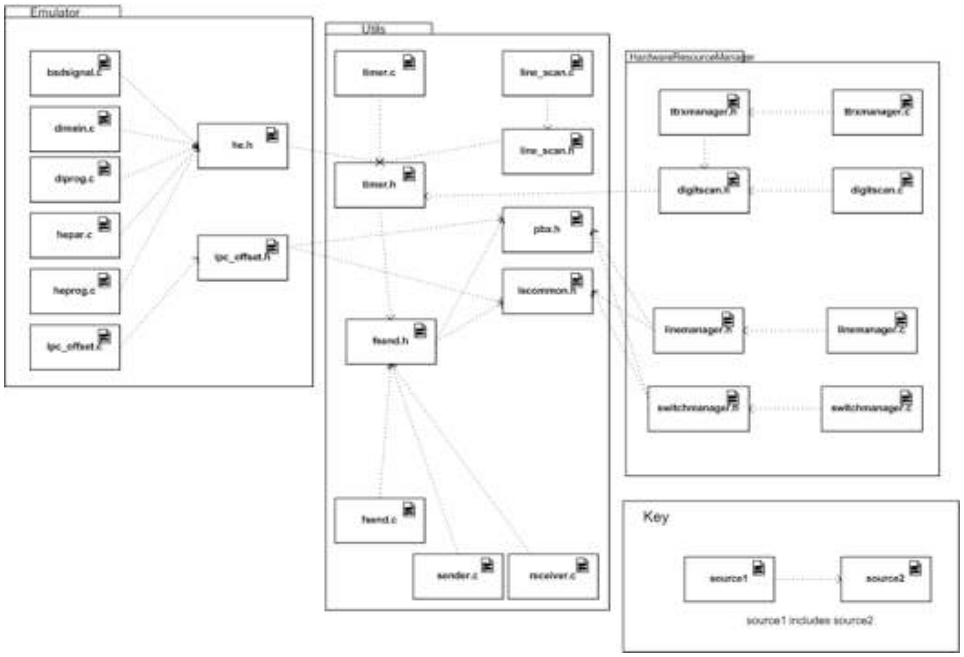
- view describes the domain and the relationships
- use this view to describe the decomposition of the system into components
- characteristics of a **software component**:
  - "a unit of composition with contractually specified interfaces..." (Szyperski)
  - encompasses a set of related functionality
  - possible implementation forms: a set of classes, libraries, or services
- create a subsection (e.g. 2.1.1) for each component and describe its responsibilities in terms of the functional requirements in the SRS

Example - Logical View



- a UML component diagram can be used to express the logical view
- note the use of the "ball and socket" notation for denoting interfaces and the named interfaces

2.2 Development View



- view describes the static organization of the software during development
- a UML package diagram can be used for this view
- indicate any package or library dependencies

2.3 Process View

- view describes run-time, concurrency and synchronization aspects
- use this view to illustrate information about UNIX processes, and their control and data dependencies
  - a UML class diagram can be used to represent this information and stereotypes can be used to classify dependencies

2.4 Physical View

- view describes the distribution of run-time artifacts to the physical hardware
- a UML deployment diagram can be used for this view
  - use specific machine names (e.g. cpu#, rees, etc.) to indicate where the artifacts are / can be executed
- should reflect the lab's **actual architecture**, not the **conceptual architecture**

3.0 Interfaces

- describe the communication protocol used for each named interface in section 2.1
- communication protocol defines the type of interface, the technology used and specific interface details
- types of interface:
  - programmatic interface (e.g. API)
  - message-based interface (e.g. XML messages)
  - database interface (e.g. SQL statements)
- examples of technology: RMI, JDBC, sockets, etc.

4.0 Data Schema

- provide a graphical representation of the database schema that indicates table structure, keys, relationships, and attribute types
  - can use an entity-relationship (ER) diagram or a third party tool to render the schema
- include a data dictionary which describes the type and purpose of each attribute
  - make sure that values with specific meanings are also defined (e.g. integer values, codes)

5.0 Technology Overview

- for each product / framework applied, briefly describe:
  - what the item is, how it is applied in your system, and why it was chosen

6.0 Component Details

- describes the structure and behaviour of the components mentioned in section 2.1
  - create a subsection for each component (e.g. 6.1)
  - further decomposition of a component into sub-components can be done in the subsection if necessary
    - the sub-components can be expressed using a nested component diagram

Subsection 1: Static Structure

- describe the component's classes using a UML design class diagram
- you can use reverse-engineering tools to derive the classes, but beware of an explosion in the number of classes if you are using a framework
  - in that case, edit the diagram for clarity and readability

Subsection 2: Run-time Structure
Subsection 3: Behaviour

- [2] describe the component's run-time artifacts (e.g. processes, threads) using a UML class diagram
  - clearly indicate any relationships (e.g. parent /child, data, control, etc.) between the artifacts
  - for components which are implemented as libraries, include a sentence indicating this
- [3] for each run-time artifact, describe its behaviour using UML activity diagrams or statecharts
  - consistency between the structural and behavioural models is important (e.g. actions correspond to class operations)

7.0 Scenarios

- scenarios exercise the interfaces between components and give a run-time view of the system
- use UML sequence diagrams created based on instances of the classes from the structural models and the interfaces which are defined
- scenarios will contain details which should be reflected in the diagram
- the required scenarios will be posted on the course web site a few weeks before the due date

Style Guidelines

- front matter: title page (with group id), table of contents, list of figures
- body: standard margins, 11 point font, 1.5 spacing
- diagrams: include a legend for unconventional notation and only explain things in the body which are not immediately obvious
- appendices: will not be marked
- **page limit**: 40 pages for the first design document

Implementation

- produce a working interim build which includes the baseline functionality (at minimum)
- stable and professional
- submit the code + a readme file which contains compilation and deployment instructions as a single zip file
  - do not submit binary files

Phone Control Software (I)

- recall the conceptual architecture:
  - designing embedded software to be run on the physical IP phones
- consequence: each phone must be represented as a separate operating system process
- issues:
  - what does it mean when a phone process is started / killed?
  - what does a phone process do exactly?

Phone Control Software (II)

- potential architectural styles
  - ***thin client***: no / minimal logic in process - all events are passed to a server for processing
  - ***P2P***: maximum logic in process - all events are handled by the phone, no server is required
  - ***client-server***: middle ground
    - how to decide the division of responsibility?

Looking Ahead

- Demo
  - will be held in the Nortel lab (MC3007) using both physical and emulated IP phones
  - TA will run test cases against your interim build to verify the baseline functionality
- Peer Review
  - will involve critiquing another group's design and code

Lab Notes

- the lab environment should be stabilized by Wednesday (e.g. the IP phones should work)
- don't forget to release resources by killing the phone process when you're done
- don't acquire IP phones if you are not in the lab
- don't run anything on **commando.student.cs**
- read the lab information on the course web page, especially the part about port assignments

Administration (I)

- group assignments:
  - groups 1 - 5: Peter
  - groups 6 - 16: Alfred
  - groups 17 - 28: Sean
- sign-ups for variability negotiation meeting
  - sign ups will be done through CourseBook
  - submit your proposal to CourseBook 24 hours before your meeting

Administration (II)

- due dates
  - variability must be negotiated before Monday May 15
  - electronic submission of design document and implementation are due on Monday June 12 by 8:30 am
  - physical submission of design document is due in lecture on Monday June 12

Summary

- try out the emulator and phone interface if you have not done so yet
- start sketching out the system architecture
- future lectures will contain more information about some of the design document sections
- direct your project questions to your assigned TA
- check the newsgroup regularly for announcements
- this week's tutorial: test-driven development using a unit testing framework