Architecture of distributed systems

FLUENT framework

B. Orlić, I. David, R. Mak, J.J. Lukkien
What is FLUENT?

**Component framework** that provides:
- Resource management
- Application lifecycle management
  - composition
  - deployment
  - communication
  - hot insertion / removal of component
  - runtime parameterization

**Target area:**
- Complex distributed systems
  (e.g. video processing applications, cloud computing...)

**Work in progress**
- lot of opportunities for MSc projects
Component framework services

blue = current focus
red = near future
orange = further future

* Not aimed towards platforms with minimalistic resources such as wireless sensor nodes, but lightweight compared to frameworks such as CORBA, COM, OpenDDS

Classification based on picture by Johan Muskens

8-Dec-11

B. Orlic, I. David, R. Mak, Johan J. Lukkien, b.orlic@tue.nl
TU/e Computer Science, System Architecture and Networking
Requested features - functionality

Composing applications
- application composed out of services, services mapped to components that provide them
  - 2 ways: 1) add-hoc compositions on the fly,
    2) saving scripts with application descriptions

Deployment
- 2 ways: 1) open scripts with predefined application descriptions
  2) deploy on selected nodes interactively from GUI
- future: autonomous deployment based on some predefined policy

Operation
- communication between binded components, should support variety of interaction types (also depends on application domain e.g. video streaming is a must for video processing applications)
- runtime adjustment of control parameters (operator induced & autonomous) e.g. PTZ cameras, sensor settings, algorithm parameters, QoS levels...
Requested features - QAs

Resource management
- multiple applications competing for resources => arbitrate resource usage
- goal is to achieve efficient resource usage
- handle highly fluctuating resource demands (e.g. due to node addition/removal, or due to inherent properties of the components as in video processing)

Performance & interaction
- decouple communication mechanism from component
- reduce communication overhead

Portability & flexibility
- cross-platform solution (Windows & Linux need to be supported)
- used open source libraries should have permissive licences so that framework code can be extended without need to open up derived source code
Stakeholders

- **Application designers**
  - compose applications (ad-hoc at runtime or predefine and save)
  - specifies application as set of possible configurations (plus reconfiguration rules)

- **Application operators**
  - start the application
  - selects/approves the actual configuration (possible configurations determined at runtime through negotiation between framework entities)
  - monitor operation and requests reconfiguration

- **Plugin developers**
  - need easy & intuitive ways to create plugins from their code

- **Framework developers & maintainers**
  - work on improving and extending framework features and properties
5 Degrees of freedom in runtime res. mgt.

Resource management = balancing of:
- resource demands of applications
- resource supply provided by platform

- done through managing resource budgets
Application lifecycle

- application description = set of configurations + reconfiguration rules
- defining configuration = fixing parameters in 5 stages (from A to E)
Component lifecycle

- development, repository, deployment
- deployment cycles: with installation, only instantiation, only binding (using plugin as a service, or same plugin used in multiple configurations)
Used architectural styles and concepts
Resource management

- Address highly fluctuating resource demands
  - runtime resource management

- QoS levels of components in same app are not independent
  - application layer in resource management
  - dedicate one entity for managing every application (Orchestrator)

- Only on system level there is enough insight for good optimisation
  - system layer in resource management
  - one central entity that grants resource budget requests (Resource Manager)

- Resources are used on nodes
  - resource usage monitoring & control must be on nodes
  - one entity on every platform node, in charge of resources for that node (Device Manager)
Architecture – key framework entities

B. Orlic, I. David, R. Mak, Johan J. Lukkien, b.orlic@tue.nl
TU/e Computer Science, System Architecture and Networking
System overview
What is deployment unit?

Device Manager

simplest to implement

component decoupled from communication mechanism

comm. overhead minimized

exe1 exe2 exe3
dll1 dll2 dll3

dock (exe) dock (exe) dock (exe)

Complex component

dll1 dll2 dll3
dock (exe) dock (exe)

Device Manager

B. Orlic, I. David, R. Mak, Johan J. Lukkien, b.orlic@tue.nl
TU/e Computer Science, System Architecture and Networking

8-Dec-11
Architecture – key framework entities

B. Orlic, I. David, R. Mak, Johan J. Lukkien, b.orlic@tue.nl
TU/e Computer Science, System Architecture and Networking
Components and services

- **Framework as a whole offers set of services** *(discovery, composing, deployment, monitoring..)* **to the user**

- **Repository contains:** 1) **service types** and 2) **component types**

- A **service type** is defined via interface(s) and can be implemented by various **component types** *(binaries, packed in dll libraries)*

- **Component** = instance of component type *(running on some node)*

- **Component offers services** *(instances of service types, defined via interfaces)*

- **Future:** **Dock** will eventually allow connections/translations between native component plugins and **UPnP / DPWS services**

- **Future:** instantiate component instances as standalone entities offering services
Architecture – key framework entities

B. Orlic, I. David, R. Mak, Johan J. Lukkien, b.orlic@tue.nl
TU/e Computer Science, System Architecture and Networking
System overview

B. Orlic, I. David, R. Mak, Johan J. Lukkien, b.orlic@tue.nl
TU/e Computer Science, System Architecture and Networking
Predefined macros for: declaring component descriptions, interfaces & channels, channel communication, logging from plugin to remote screen (separate executable associated with Dock)

Macros result in creation of appropriate structures =>
component declaration generated on demand from plugin & wrapper mechanism for binding interfaces is generic
Component Model - summary

Component model:
- component is an entity that exchange messages with other components & whose lifecycle is controlled by cooperating set of framework entities (dock, orchestrator, device manager, resource manager)
- provided / required interfaces containing channels of different types (now FIFO buffers, in future also video streaming, events & RPC)

Implementation
- interface/component definitions embedded in code using simple to use MACRO directives => no compiler support needed, reduced possibility for errors
- instances of components hosted / executed at runtime by docks

Packaging
- dll library can contain one or more plugins

Deployment
- orchestrator requests deployment, docks load dlls & instantiate components (plug-ins)
Remember: Framework

- **‘static’ part**
  - programming model, data model (see slides 19, 25-30)
  - life cycle model (see slides 8 & 9)
  - methods or tooling for development (see slides 19, 32, 25-30)

- **a ‘dynamic’ part**
  - a run-time system, or platform (slides 11-18)
    - In this case set of framework entities: Docks, Device managers, Orchestrators, Resource manager, Repository
  - a set of services (slides 10, 22-31)
    - Composing applications, saving app. description, opening saved application description, deploying application, monitoring application, dynamic reconfiguration as a mean of resource management

- **a process model**
  Dock (executable) hosts one or more plugins (loaded from dynamic libraries), each plugin in separate thread and can start more threads
Framework services – now & future

Discovery
- now: Device manager searches for dlls with plugins in local repository
- future: Central Repository that will allow downloading components to nodes
- future: plug & play discovery of new nodes

Composition
- using GUI tool
- save/load application descriptions scripts to/from files

Distribution
- now: plugin can be started on any node where a dll library that contains it exist
- future: manage dlls by introducing central repository and mechanisms for synchronizing local repository with contents of central one on demand of applications

Deployment
- start: control lies with an Orchestrator
- binding interfaces: binding UDP server on provided side with UDP client on required side
Discovery

Central repository
- OpenCv1comp
- OpenCv2Comp
- Video source
- Detect trespassing
- Video display

Platform node 1
- Device manager
- Local repository
  - OpenCv1comp
  - OpenCv2Comp

Platform node 2
- Local repository
- Video display

Orchestrator

CR | N1 | N2
---|----|--
?  | ?  | ?
?  | ?  | ?
?  | ?  | ?

Operator
Discovery – GUI tool perspective

<table>
<thead>
<tr>
<th>available plugins</th>
<th>ID</th>
<th>name</th>
<th>lib. path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>OpenCVComp1 V1.0</td>
<td>OpenCVcomps</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>OpenCVComp2 V1.0</td>
<td>OpenCVcomps</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**interfaces**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>lib. path</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>VideoProcessingParamsProv</td>
<td>provided</td>
</tr>
</tbody>
</table>

**channels**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>lib. path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sigmaOutputChannel</td>
<td>DATA_PCDL</td>
</tr>
<tr>
<td>1</td>
<td>smoothTypeOutputChannel</td>
<td>DATA_PCDL</td>
</tr>
</tbody>
</table>

description: 'This interface allows supplying video processing parameters to an outer provider of video processing service.'

description: 'sigma is input parameter for laplacian algorithm.'
Discovering and managing components in a distributed system using a central repository and platform nodes.
Entity Descriptors

Communicated between executables to describe status of an entity
Entity proxies

- Entity descriptors are just data structures, Entity Proxies have behavior
- Represent entities existing in remote framework parts, use entity descriptors data
Composition

Orchestrator

Save GUI tool

node 1

node 2

Detect trespassing

Platform node 2

Detect trespassing

Video display

Platform node 1

Video source

Application Description

xml

GUI tool

Orchestrator

Save
Application & platform models

- Model of application and platform entities (used for describing composition, deployment & status during operation)
- Navigation through model (used in MVC architecture for GUI tool)
Distribution of components

Future work

Central repository

Video source
Detect trespassing
Video display

Platform node 1

Device manager
Local repository

Video source
OpenCv1comp
OpenCv2Comp

Platform node 2

Local repository

Detect trespassing
Video display

Device manager
Deployment

Master server node

Orchestrator

Platform node 1
- Video source
- Dock

Component Instance Desc

Interface Binding

Platform node 2
- Video display
- Dock

Detect trespassing

Detect trespassing

GUI tool

Application Description

Load

TU/e
Technische Universiteit
Eindhoven
University of Technology
Portable data types, hierarchical data types possible, serialization/deserialization to/from xml strings
GUI tool – MVC architecture

- Model controller is ‘singleton’
- Basic GUI controls supported and instantiated for wxWidgets library
Current status

Provided generic mechanisms for:
- discovery of plug-ins available on nodes
- composition of applications from discovered plugins
- automated deployment of composed applications
- load applications from files (entering app. Descriptions & saving not complete yet)
- GUI tool (not finished)
- extensive logging capabilities

Not provided yet:
- Central repository of component types
- Resource management
- Video streaming support
- SOA approach for user interacting with system
FLUENT framework
Demo of initial development