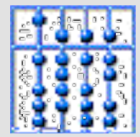
DWARF UI Tutorial

Christian Sandor, Otmar Hilliges
Lehrstuhl für Angewandte Softwaretechnik
Institut für Informatik
Technische Universität München
(sandor | hilliges)@in.tum.de

1.7.2003



Contents

I. About this Tutorial
II. Introduction to DWARF UI Framework
III. The User Interface Controller
IV. The Viewer
V. Input Devices

I. Objectives

In this tutorial you are going to learn:

- How the DWARF UI framework is designed and how it relates to DWARF itself.
- How the User Interface Controller (UIC) is used for Multimodal Input Fusion and Dialog Control.
- How the 3D Viewer can be used to display AR scenes.
- How input devices are integrated into the framework.
- How advanced UIs can be built.

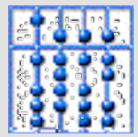
I. Overall Structure

Introduction to DWARF UI Framework

The User Interface Controller

The Viewer

Input Devices



I. Structure of Chapters

The chapters 2-6 all follow the same schema:

- Objectives: What are you going to learn?
- Sources of Information:How to get more detailed information?
- The Problem: What is the basic problem?
- Our Solution: How did we solve it?
- Implementation: How does the implementation look like?
- Future Work: What are the next steps planned?
- Discussion: Is it possible to solve this even better?
- Exercises: Run sample programs and do modifications yourself.

I. Prerequisites

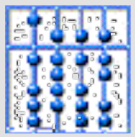
We recommend the following setup:

- SuSE Linux 8.1 with gcc 3.2
- KDE with konsole (very convenient)
- vi/emacs and bash shell

(more detailed requirements can be found at http://www.augmentedreality.de)

Other setups have been tested as well:

- Windows with cygwin: works quite well. Was used for parts of SHEEP.
- Mac OS X: works ok, but not fully tested



Contents

I. About this Tutorial
II. Introduction to DWARF UI Framework
III. The User Interface Controller
IV. The Viewer
V. Input Devices

II. Objectives

- What are the main problems implementing AR UIs?
- How does DWARF fit into the picture?
- What are the conceptual parts of an AR UI?
- How did we implement these parts?

II. Sources of Information

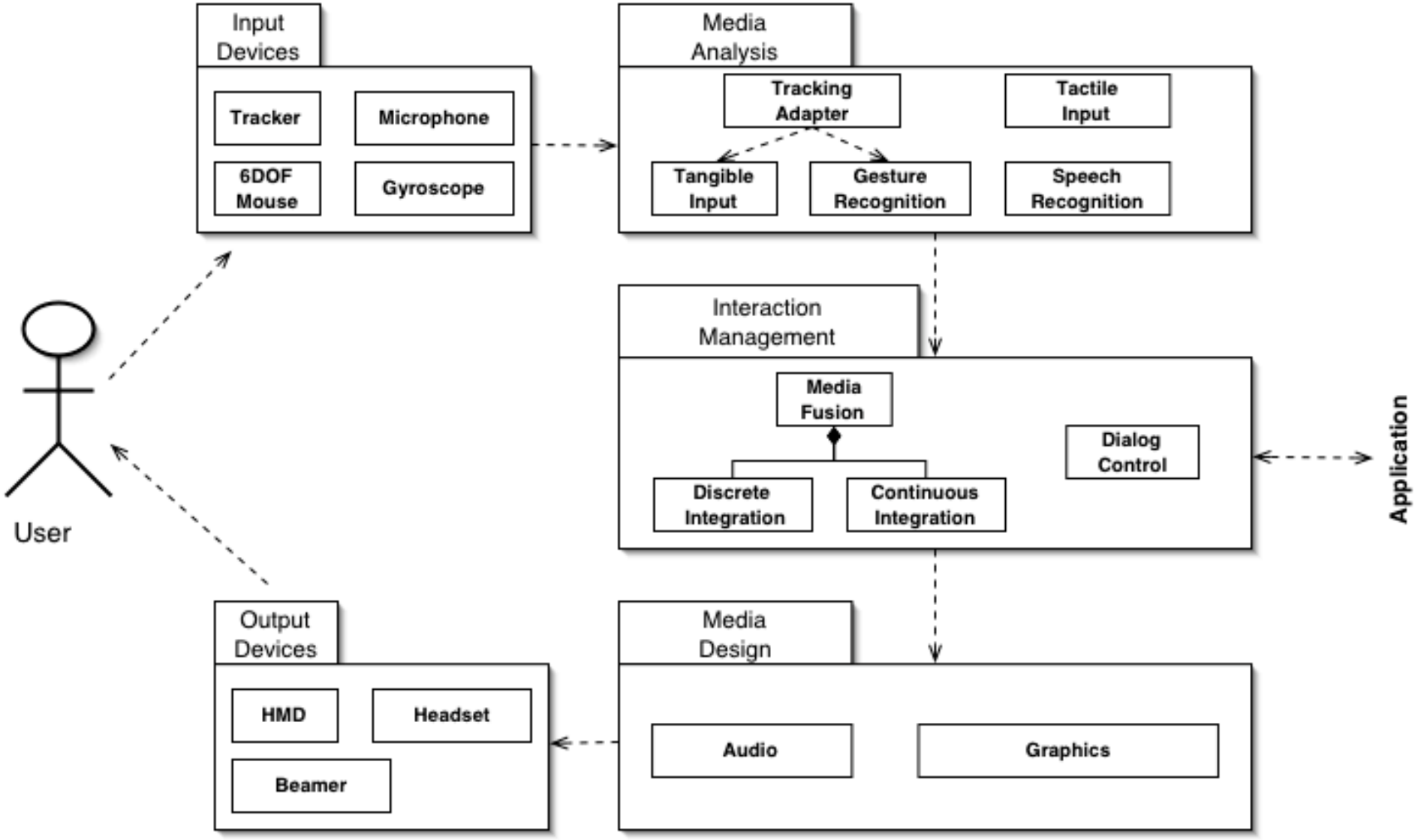
- General Information:
  - www.augmentedreality.de: post your questions there!
- DWARF framework:
  - ISMAR2001 paper
  - ISWC2002 paper
- UI framework
  - Rejected paper for Computer Graphics and Applications: http://janus.informatik.tu-muenchen.de/~sandor/cga03.pdf
  - Upcoming publications :-)

II. The Problem - Requirements

Typical Requirements for AR UIs :

- Multimodal: Semantic unification of several input channels. Also several presentation components have to be coordinated.
- Rapid prototyping: To start the usability lifecycle for finding and evaluating new UI metaphors.
- Spatial: Tracking data linked to objects in 3D views.
- Flexible: I/O devices can be added/removed at runtime.
- Distributed: Software components run on several machines.

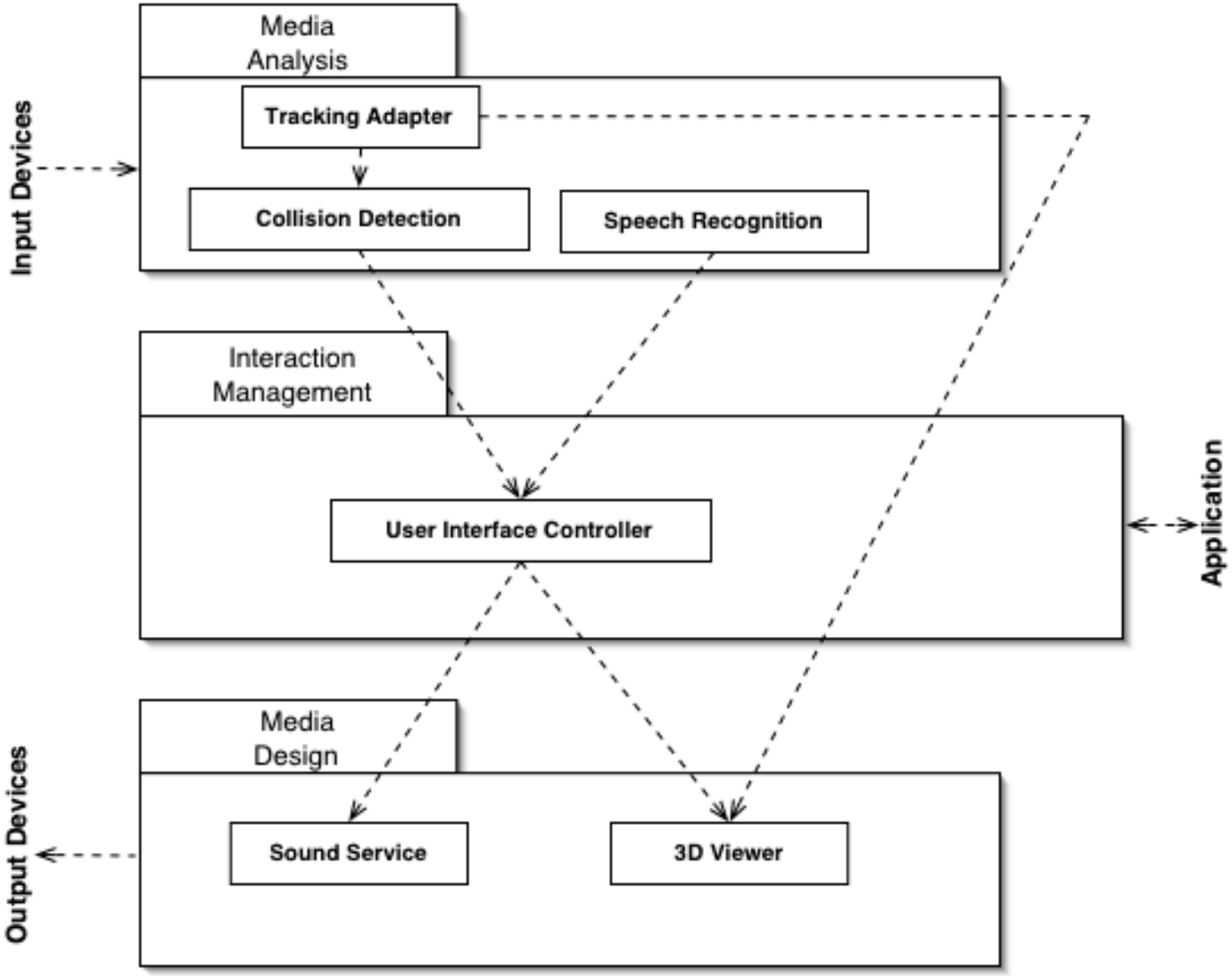
II. The Problem - Functional Decomposition



II. The Solution - Overview

- DWARF:
  - Operating system for AR systems.
  - Solves: Flexibility, distribution.
- UI Framework:
  - Everything is built with DWARF components.
  - Solves remaining requirements:
    - Multimodality -> User Interface Controller component
    - Spatial -> Viewer component
    - Rapid Prototyping: Architecture recommendations for efficient combination of above components.
    - Flexibility revisited: Input device taxonomy.

II. Implementation - Architecture



II. Implementation - Component Summary

- User Interface Controller:
  - Semantic fusion
  - Dialog control
- Viewer:
  - 3D presentation
- Input services:
  - Deliver discrete input tokens to User Interface Controller
  - Continous tokens are most often passed directly to Viewer (e.g. viewpoint of user coupled to viewpoint in scene)

II. Future Work

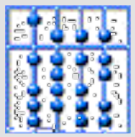
- Models should influence the dialog control and presentation (-> Knowledge-based AR, Feiner)
  - Context
  - User
  - Task
  - Hardware
  - Discourse
- Simplify everything and build more systems

II. Discussion

- How does this architecture relate to STUDIERSTUBE?
- Can parts of DWARF and STUDIERSTUBE be used together?

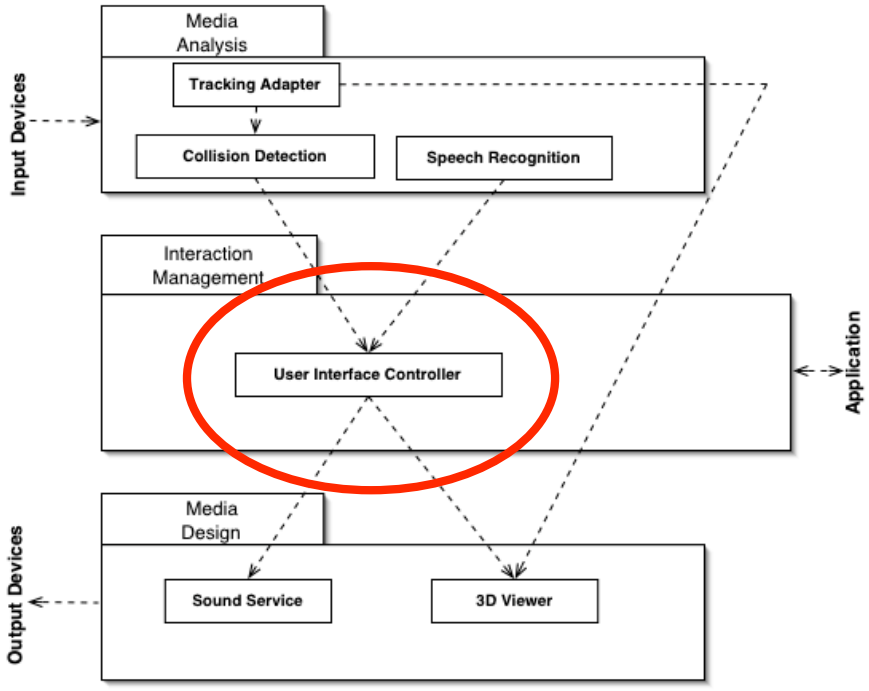
Contents

I. About this Tutorial
II. Introduction to DWARF UI Framework
III. The User Interface Controller
IV. The Viewer
V. Input Devices



III. Objectives

- What are the requirements that led to the UIC?
- How can Petri-Nets be used to model interactions?
- How does the UIC-API look like?



III. Sources of Information

- Interaction chapter in: Herding Sheep - Live Development of a Distributed Augmented Reality System. ISMAR 2003
- Jfern Homepage: http://sourceforge.net/projects/jfern
- Rejected paper for Computer Graphics and Applications: http://janus.informatik.tu-muenchen.de/~sandor/cga03.pdf

III. The Problem

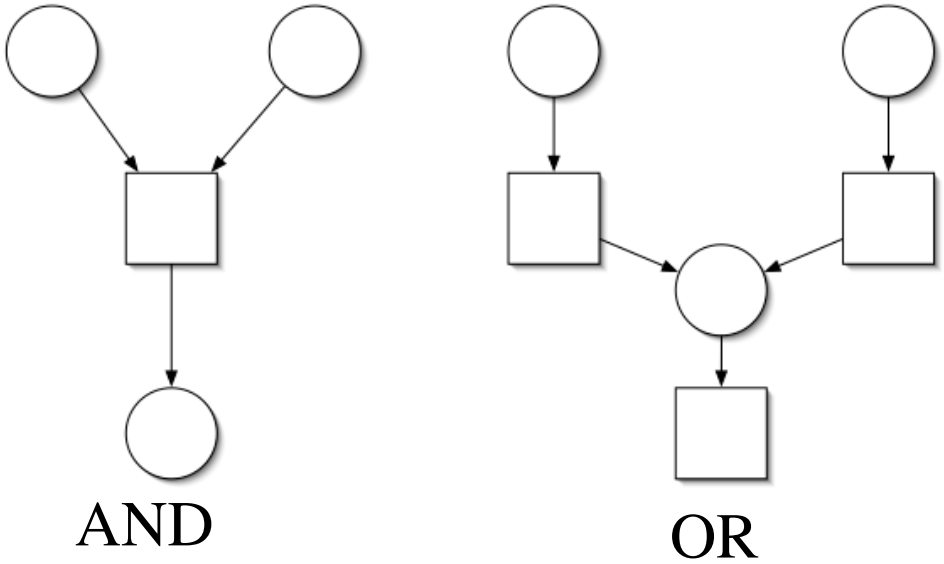
- Several stateless input services emit tokens.
- User intention has to be extracted.
- Application and Presentation services have to be controlled accordingly.
- State of UI should be stored centralized.
- During runtime intenal state of dialog control should be visible (user, developer, usability engineer)

III. The Solution

- User Interface Controller DWARF service
  - Holds state of UI
  - Controls I/O services
  - Merges Discrete input tokens into user intention
  - Displays internal state of dialog control and input fusion
  - Rapid prototyping:
    - Reuse of interaction patterns
    - Surrogate for application logic
    - Jam sessions

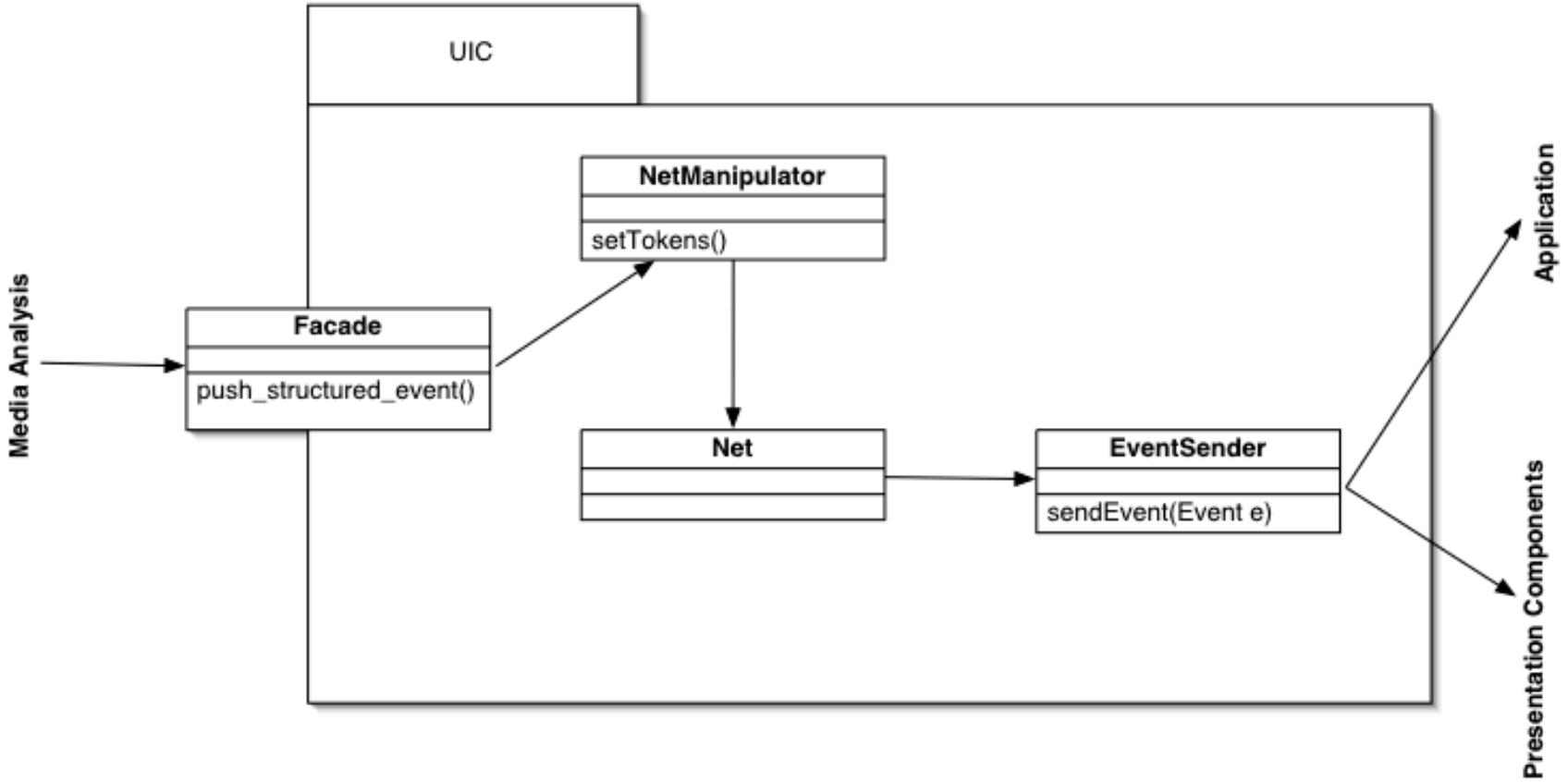
III. Implementation 1/4

- Based on Petri-Net framework Jfern:
  - XML with Java snippets for guards and actions.
  - Generates class files
  - Only a very small subset of Jfern is used.
  - Input is modelled as tokens that are placed on places.
  - Actions are triggered in transitions.



III. Implementation 2/4

- Object Design



III. Implementation 3/4

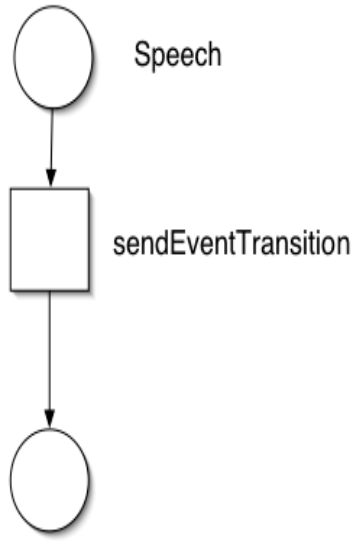
- Code Snippets

1. Input service sets:

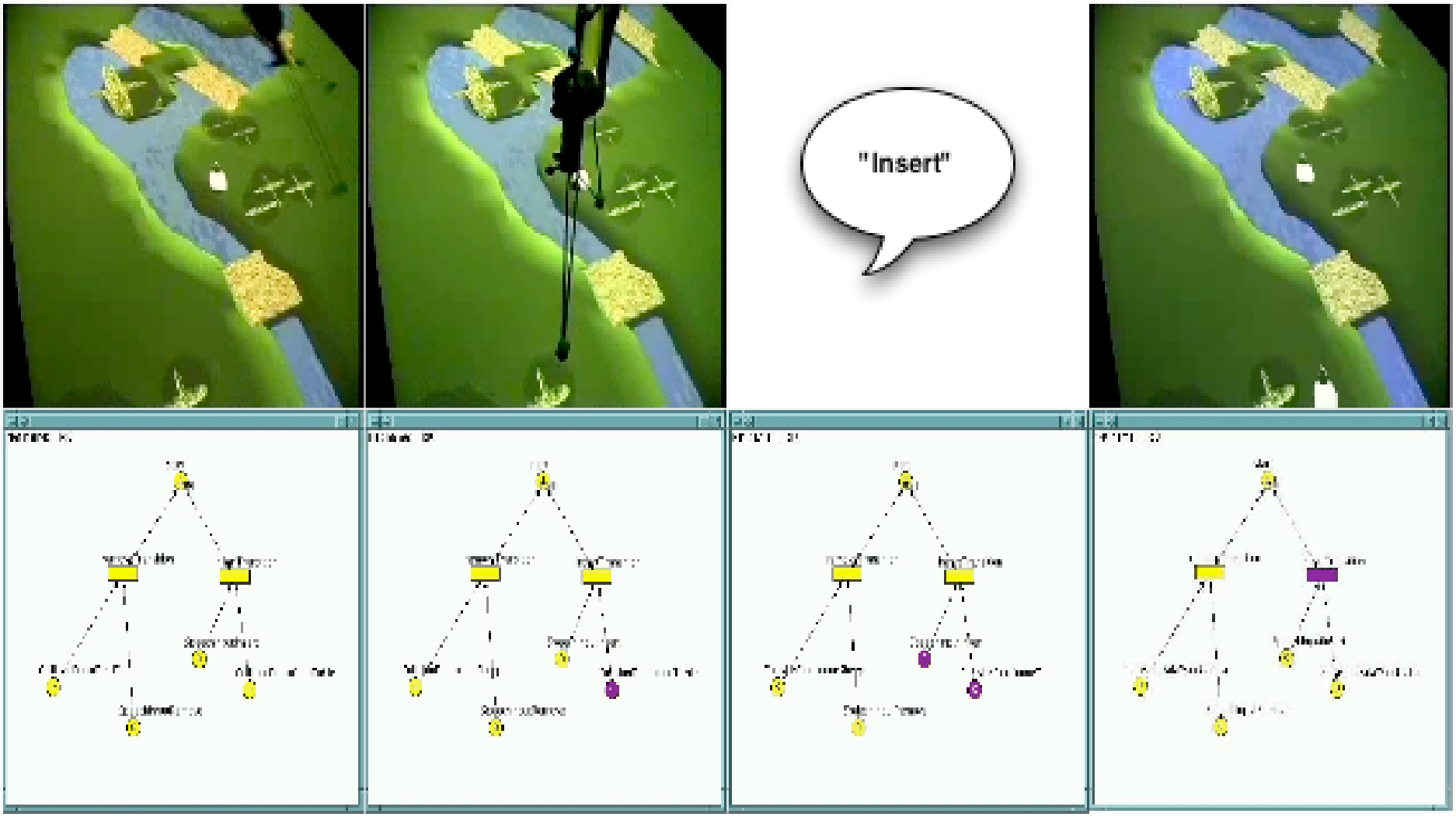
```
UserInputEvent.fixedHeader.event_type.name="Speech"
```

2. Petri-Net XML:

```
<place id="Speech"/>
<transition id="sendEventTransition">
    .... // assemble event to be sent
    UICEventSender d =
    (UICEventSender)UIC.getInstance("UIC").
    UICEventSenderHash.get("SceneData");
    d.sendEvent(event);
</transition>
```



III. Implementation 4/4



III. Future Work

- Current implementation already sufficient for everything we want to do.
- Fancy features:
  - Collect interaction patterns in library, e.g. Point-and-speak
  - Action templates („put-that-there“)
  - Even higher-level description language

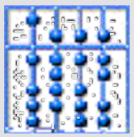
III. Discussion

- How does the UIC relate to OpenTracker?
- How is multimodality solved in Studierstube?
- UIC as future OpenTracker node?

III. Exercises

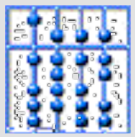
Connect TestStringSender, UIC and TestStringReceiver to form a basic data flow network:

http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/DWARF/DwarfUserInterfaceTutorial#Lesson_1_Learn_to_use_the_Servic



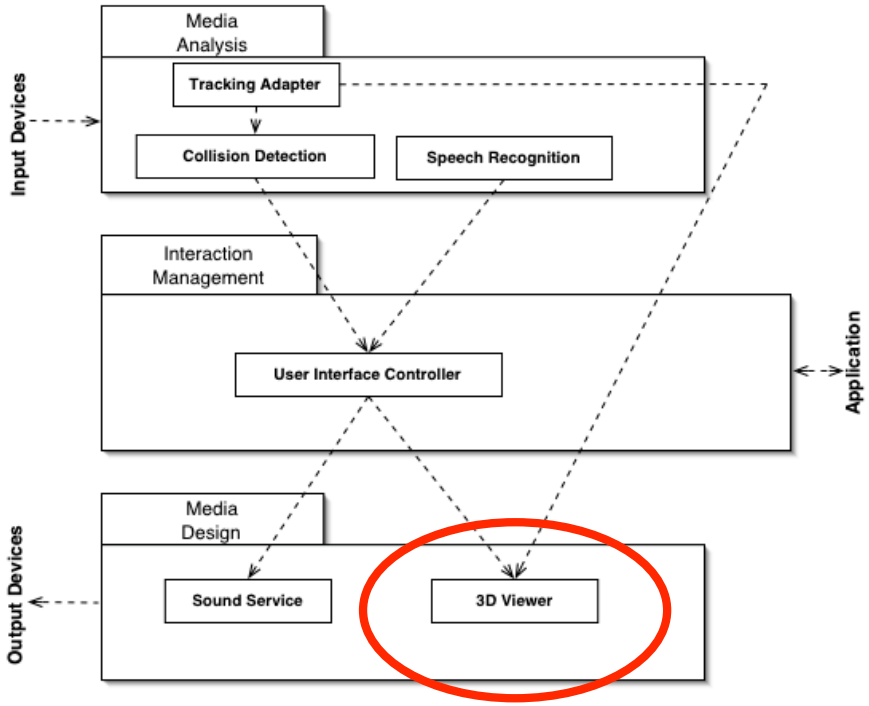
Contents

I. About this Tutorial
II. Introduction to DWARF UI Framework
III. The User Interface Controller
IV. The Viewer
V. Input Devices
VI. Advanced Topics



IV. Objectives

- Get to know the Viewer‘s capabilities
- Learn about the interaction with UIC or Model service



IV. Sources of Information

- The code foundation: www.studierstube.org
- Otmar Hilliges‘ SEP documentation (upcoming)
- Wiki-site about ServiceViewer: http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/DWARF/ServiceViewer
- Main 3D library: http://www.coin3d.org

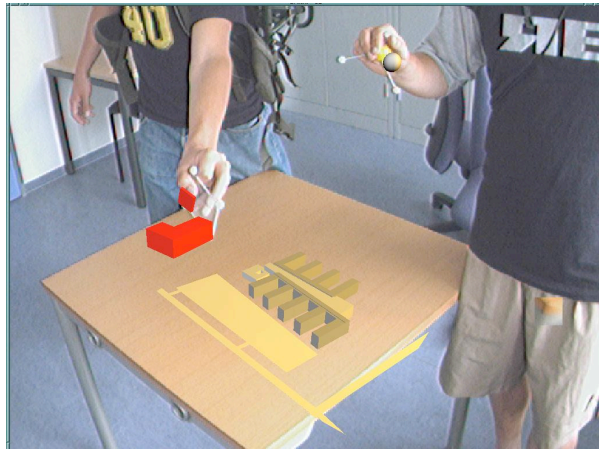
IV. The Problem

- Realtime 3D Rendering of VRML/IV scenes in different modes:
  - Video background
  - Stereo: anaglyphic, line-inteleaved ...
- Connection of SceneGraph (=SG) objects to trackers
- Control of SG objects:
  - Addition/removal
  - Change of properties
- Display of HUD-style information
- Synchronization with data storage for multi-view applications

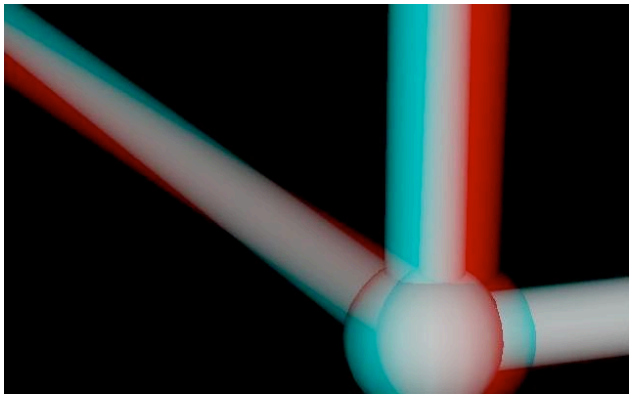
IV. The Solution

- Very lightweight OpenInventor-based viewer for DWARF
- Interfaces to
  - UIC:
    - Addition/Removal of SG objects
    - change of properties(not implemented, yet :-)
    - Overlay SG for HUD information
  - Model
    - Same interface as UIC
  - Trackers:
    - PoseData is used to set transforms of SG objects

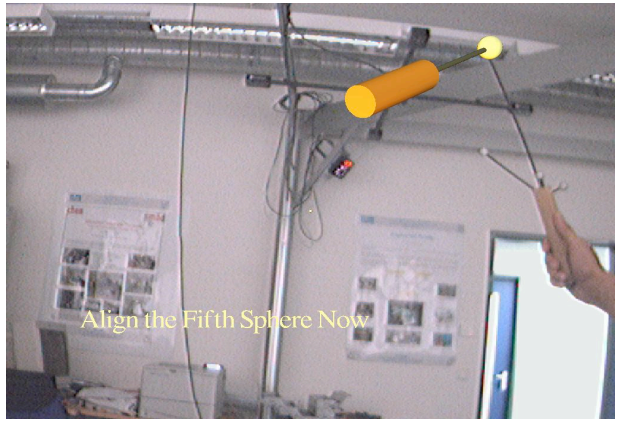
IV. Implementation 0/4



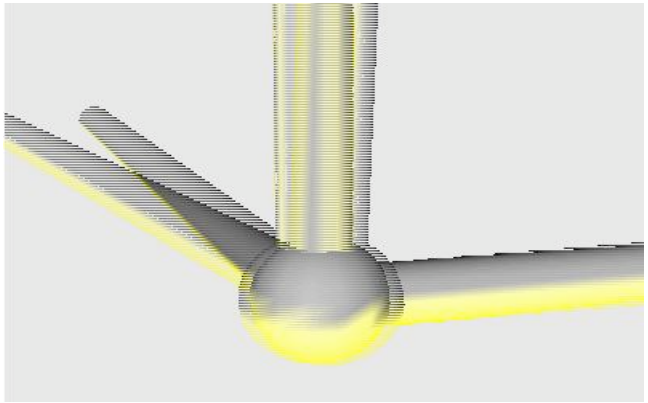
Video see-through



Anaglyphic stereo



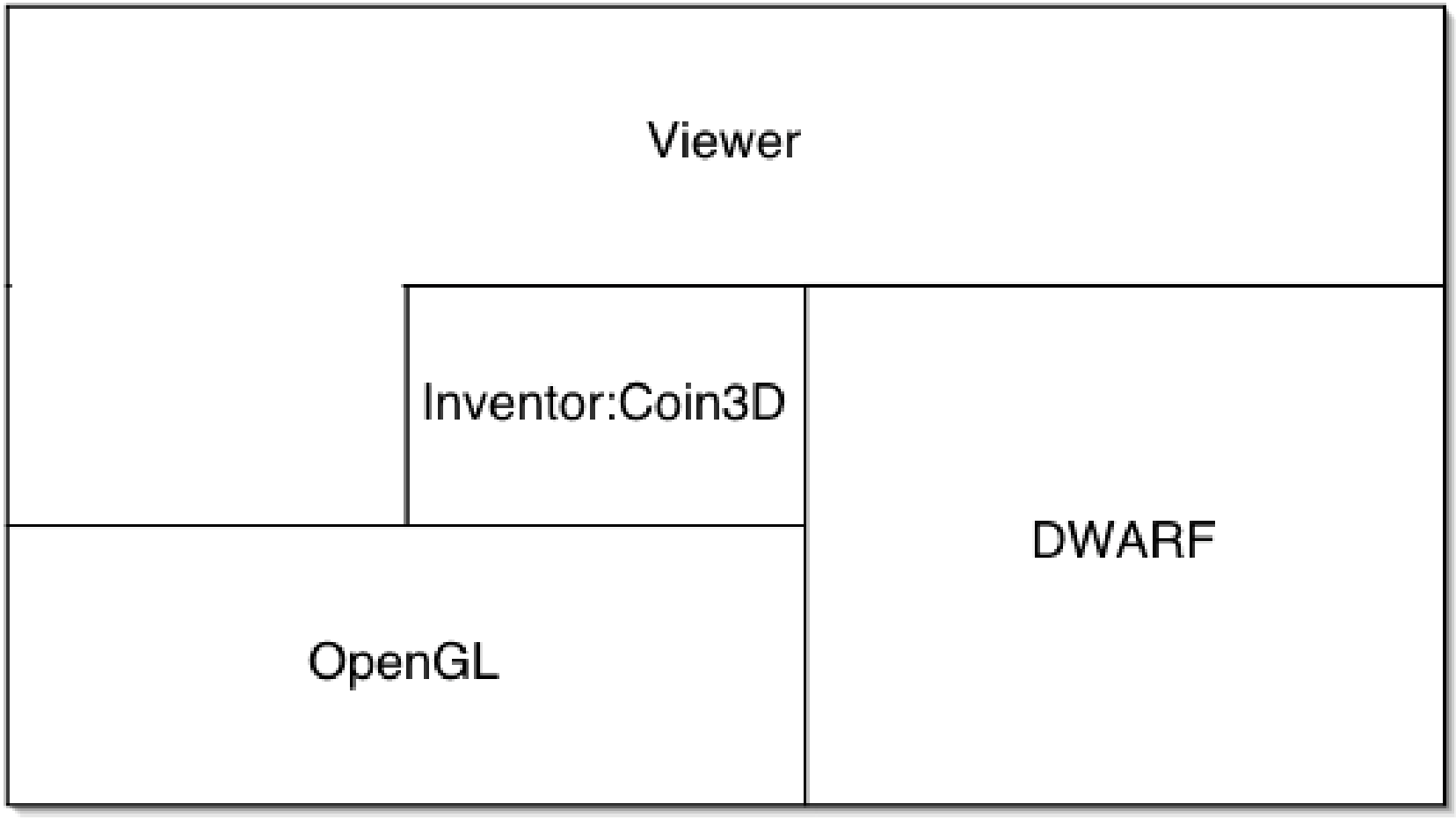
HUD-style overlay



Line-interleaved stereo

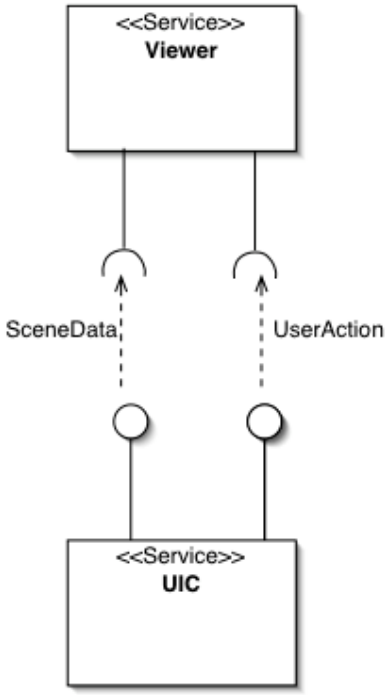
IV. Implementation 1/4

- Layering

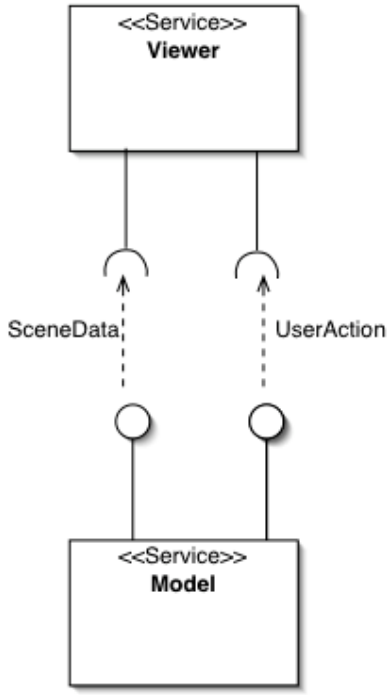


IV. Implementation 2/4

- UIC
  - Controls one view
  - Rapid prototyping



- Model
  - Controls many views
  - Persistency
  - Consistency



IV. Implementation 3/4

- Typical Tasks:
  1. Load initial scene
  2. Add objects to scene
  3. Add overlay (HUD-style)
  4. Remove objects
  5. Connect objects to trackers
  6. Disconnect objects from trackers
- All tasks are realized with two DWARF events:
  - SceneData: 1-4
  - UserAction: 5-6

IV. Implementation 4/4

- SceneData

```
module DWARF {
  //IV scene
  typedef string Scene;

  //Different Actions
  enum SceneAction {CreateObject,
  DeleteObject,ReplaceScene,SuperImpose};

  struct SceneData {
    SceneAction action;
    VirtualObjectId id;
    VirtualObjectId parent;
    Scene newScene;
  };
};
```

- UserAction

```
module DWARF {
  enum UserActionType
  //different Actions
  {SelectVirtualObject,
  DeselectVirtualObject};

  struct UserAction {
    UserActionType action;
    VirtualObjectId id;
    ThingID realObjectId;
  };
};
```

IV. Future Work

- Allow change of properties of SG objects
- Add more stereo modes
- Integration of 3D layout engine from Columbia CG lab
- Knowledge-based AR: Adaption of views according to:
  - Context
  - User
  - Task
  - Hardware
  - Discourse

IV. Discussion

- Interoperability between DWARF and Studierstube Viewer?
- Python binding would be interesting for us.
- 3D Layout engine could be interesting for you.

IV. Exercises

1. Output from TestStringSender is displayed as overlay in Viewer:
   http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/DWARF/DwarfUserInterfaceTutorial#Lesson_2_Learn_to_use_the_Servic

2. Tangible sun for illuminating a virtual scene:
   http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/DWARF/DwarfUserInterfaceTutorial#Lesson_3_the_ServiceViewer_Viewe

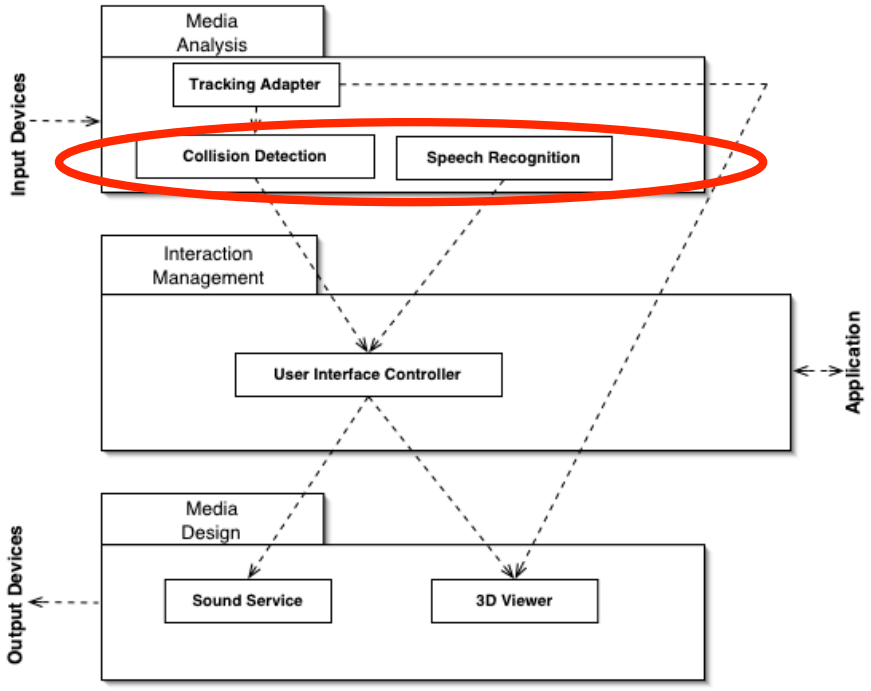
3. Load a scene by a speech command:
   http://wwwbruegge.in.tum.de/projects/lehrstuhl/twiki/bin/view/DWARF/DwarfUserInterfaceTutorial#Lesson_4_the_ServiceViewer_Viewe

Contents

I. About this Tutorial
II. Introduction to DWARF UI Framework
III. The User Interface Controller
IV. The Viewer
V. Input Devices

V. Objectives

- What are the requirements for AR input services?
- How did we solve them?
- How can DWARF‘s Selector service be used to solve ambigiuos situations?



V. Sources of Information

- Johannes Woehler‘s SEP documentation

	Unit	VRIB	VRJuggler	OpenTracker
Dynamic Configuration	no	yes	yes	no
Device Fusion	yes	no	no	yes
All kinds of Devices	yes	yes	no	no
Easy Device Misuse	yes	no	no	yes

•**Unit**: Alex Owal and Steven Feiner, *Unit – A Modular Framework for Interaction Technique Design, Development and Implementation,* http://www1.cs.columbia.edu/~aolwal/abstract.htm

•**VRIB**: *Virtual Reality Interaction Toolbox*, www.vrib.de

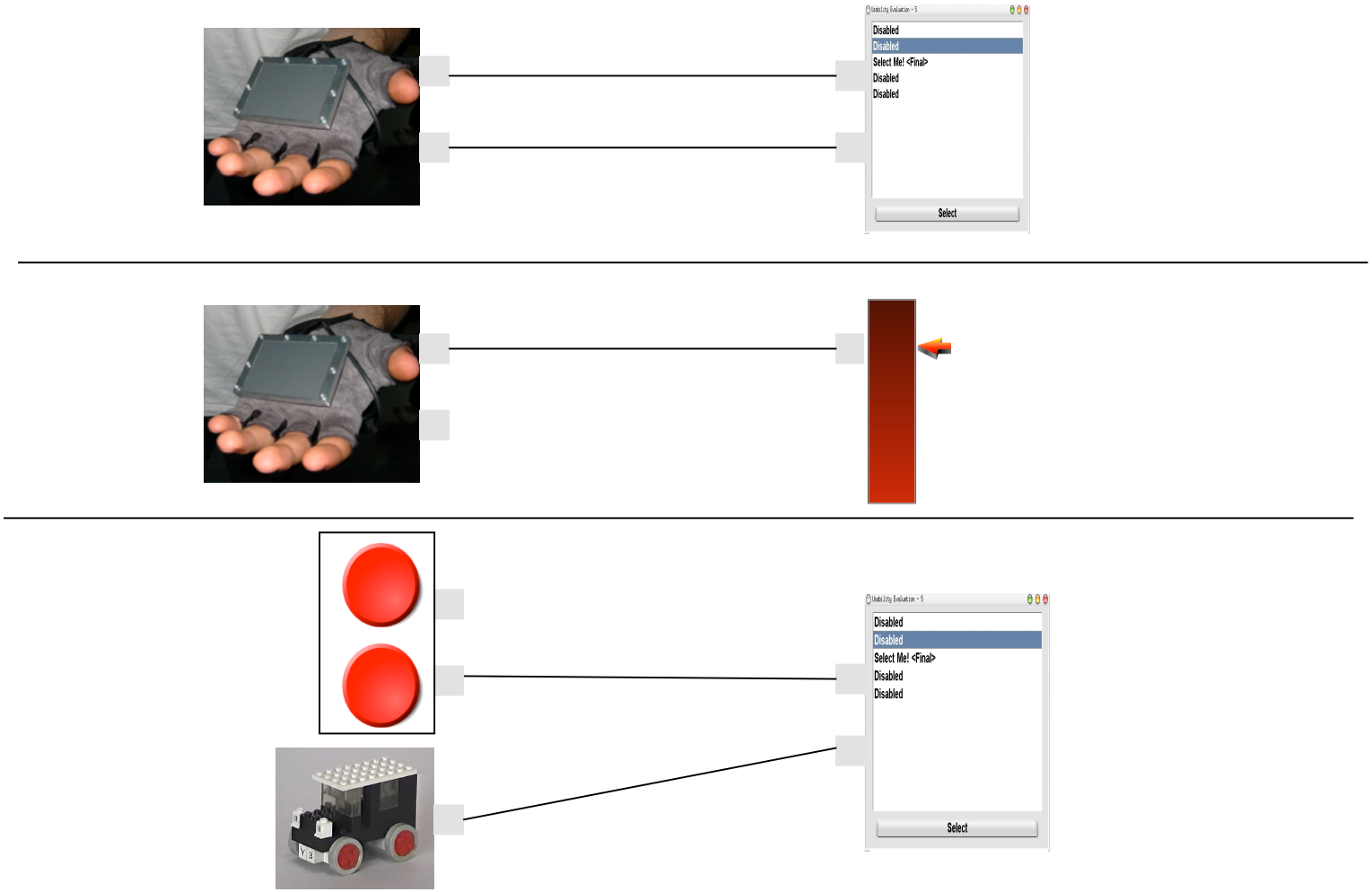
•**VRJuggler**: C. Just et al., *VRJuggler: A Framework for Virtual Reality Development,* www.vrjuggler.org

•**OpenTracker**: Studierstube, *An Open Architecture for Reconfigurable Tracking based on XML,* http://www.studierstube.org/opentracker/

V. The Problem 1/2

- Input devices for AR are often customly-built. Difficult to develop a generic protocol supporting all of them.
- Highly dynamical environments require exchange of input devices at runtime.
- Which input channel is mapped to which DWARF ability?

V. The Problem 2/2

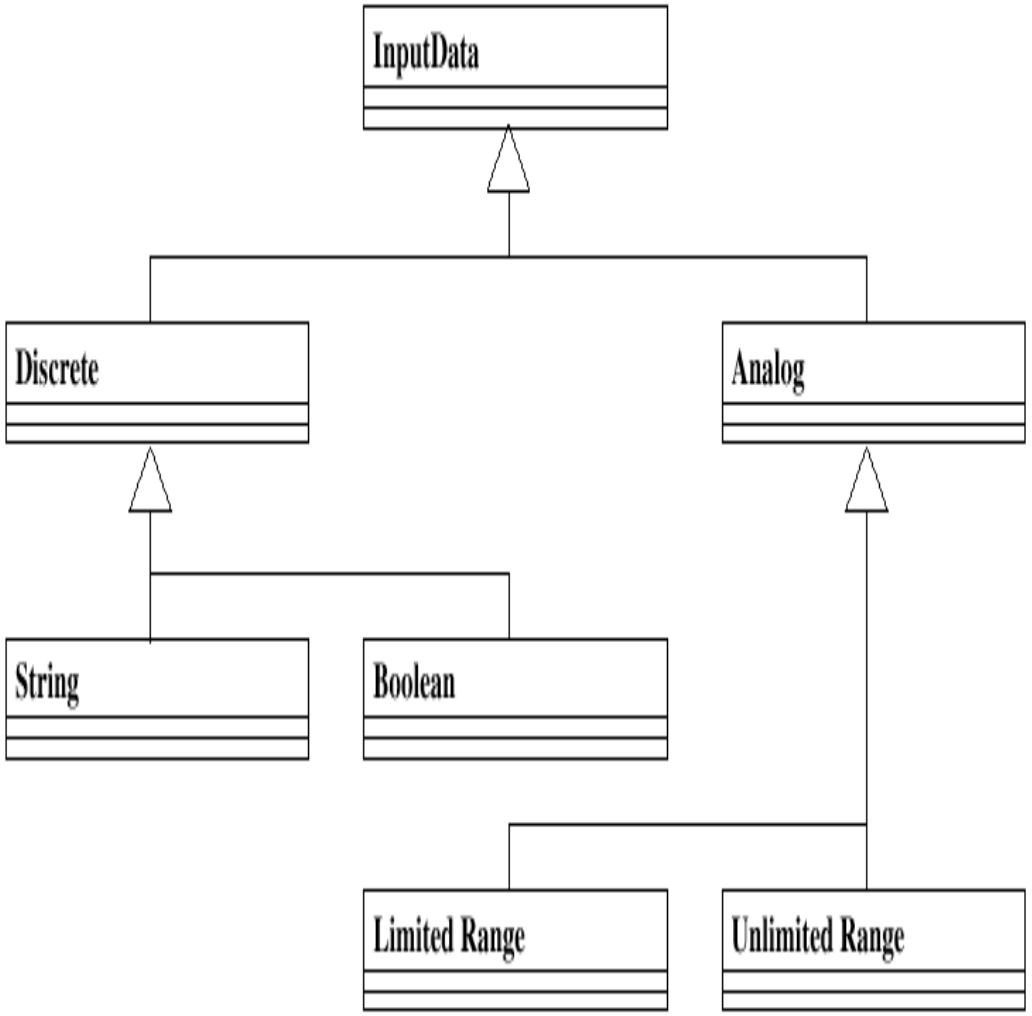


V. The Solution 1/2

- Dynamics for exchange of input devices: DWARF
- Matching of input services and UIC: DWARF‘s Needs and Abilities
- Two steps in setting up connection UIC-input services:
  - Syntactical matching: strictly according to token formats
  - Semantical matching: which token covers which functionality?

V. The Solution 2/2

- UserInput can be decomposed into four token types
- Different syntactical matching possibilities:
  - Equality
  - Subset
  - Intersection
- Semantic matching: For ambiguities user has to give decisive input: Selector service



V. Implementation 1/2

DWARF realization:

- CORBA IDLs for syntax hierarchy leaf nodes

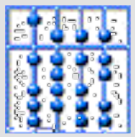
InputDataBool
eventType: InputDataEventType boolValue: boolean

InputDataString
eventType: InputDataEventType stringValue: string

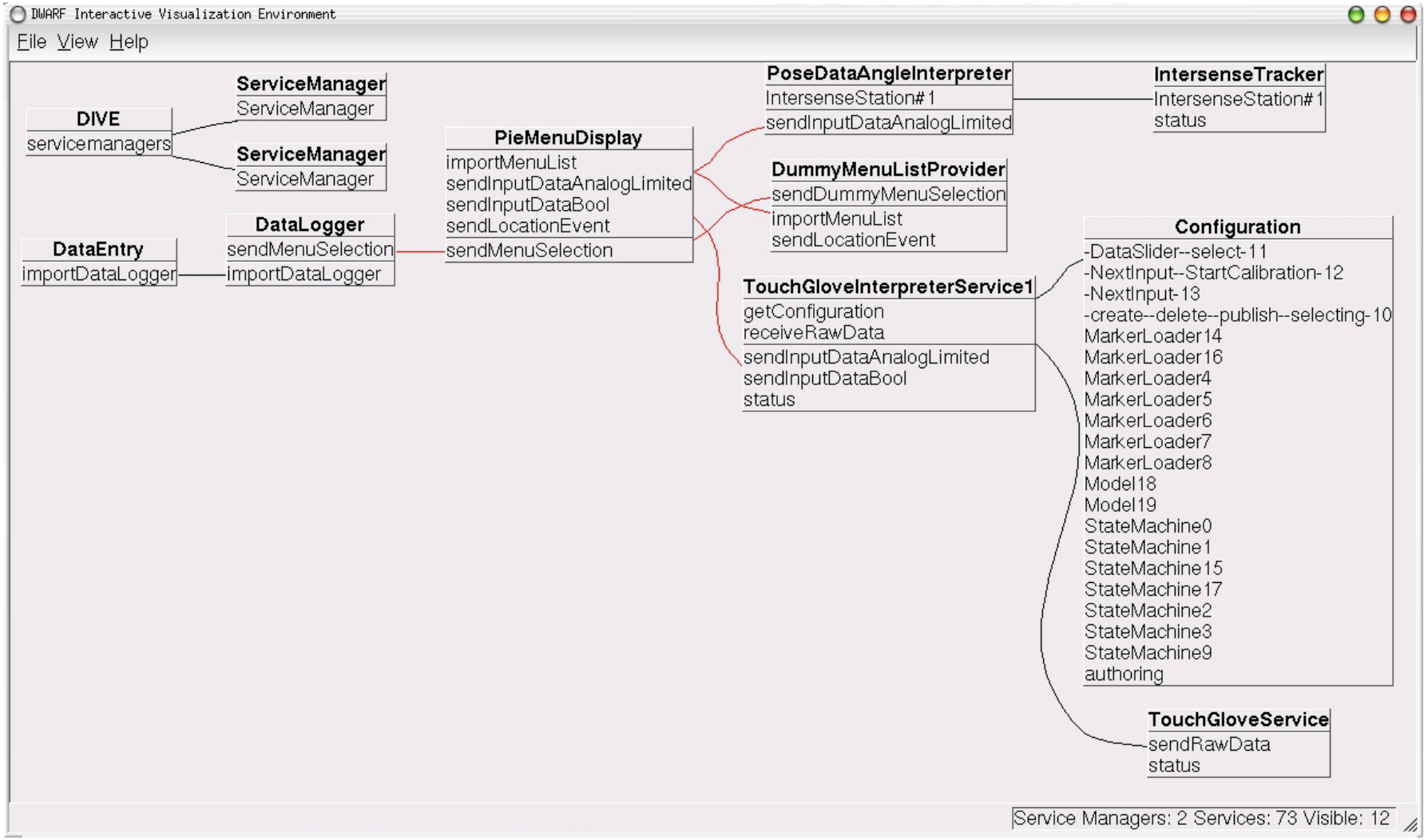
InputDataAnalogLimited.idl
eventType: InputDataEventType limitedAnalogValue: double

InputDataAnalogUnlimited.idl
eventType: InputDataEventType analogValue: double

InputDataEventType = {InputDataState | InputDataChange}



V. Implementation 2/2



V. Future Work

- Refine taxonomy to support advanced input paradigms (drag‘n drop)
- Work on a more usable selection of input devices

V. Discussion

- Interoperability between DWARF and Studierstube input tokens?