Integration of Google Glass into a Choreographed Service Oriented Software Architecture

By: **Hanqing Zhao**

Introduction

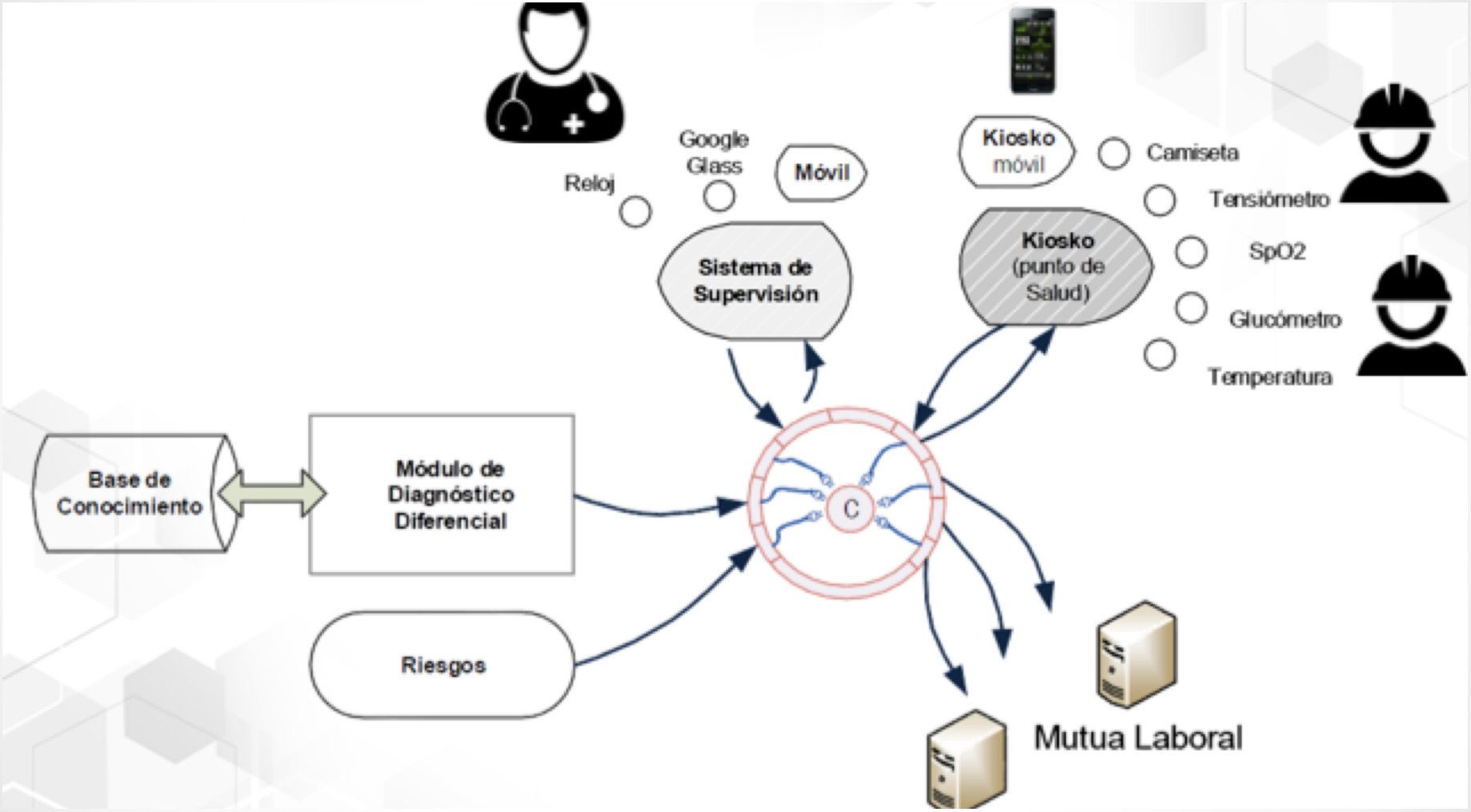

Figure 1. Integration of Health Sensors and and Wearable Devices [1]

Requirements for Glassware

Software is developed according to requirements
In this project.

- Usability (Glassware with good user interaction experience)
- Reliability (Stable Glassware addresses development and runtime needs [functional and non-functional])
- *Confidentiality (Ensuring communications channel between Choreographer and Glassware is more secure)
- Connectivity (Communications between Glass and Choreographer)

Usability

User gestures: tap (one or two fingers), swipe (forward or backward or downward on touchpad)

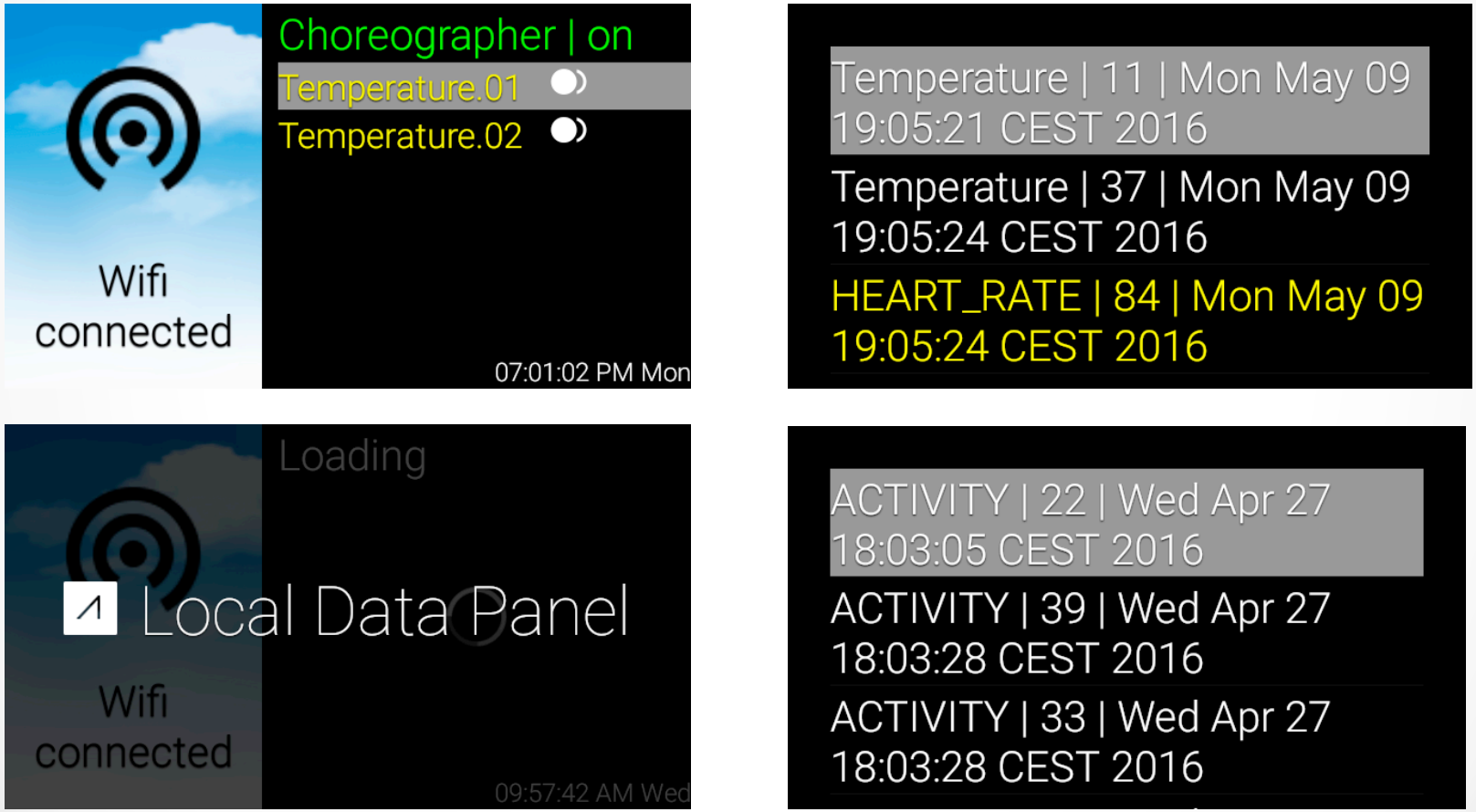

Figure 2. User Interfaces [2]

Reliability

- Software Design Patterns? Modified MVP

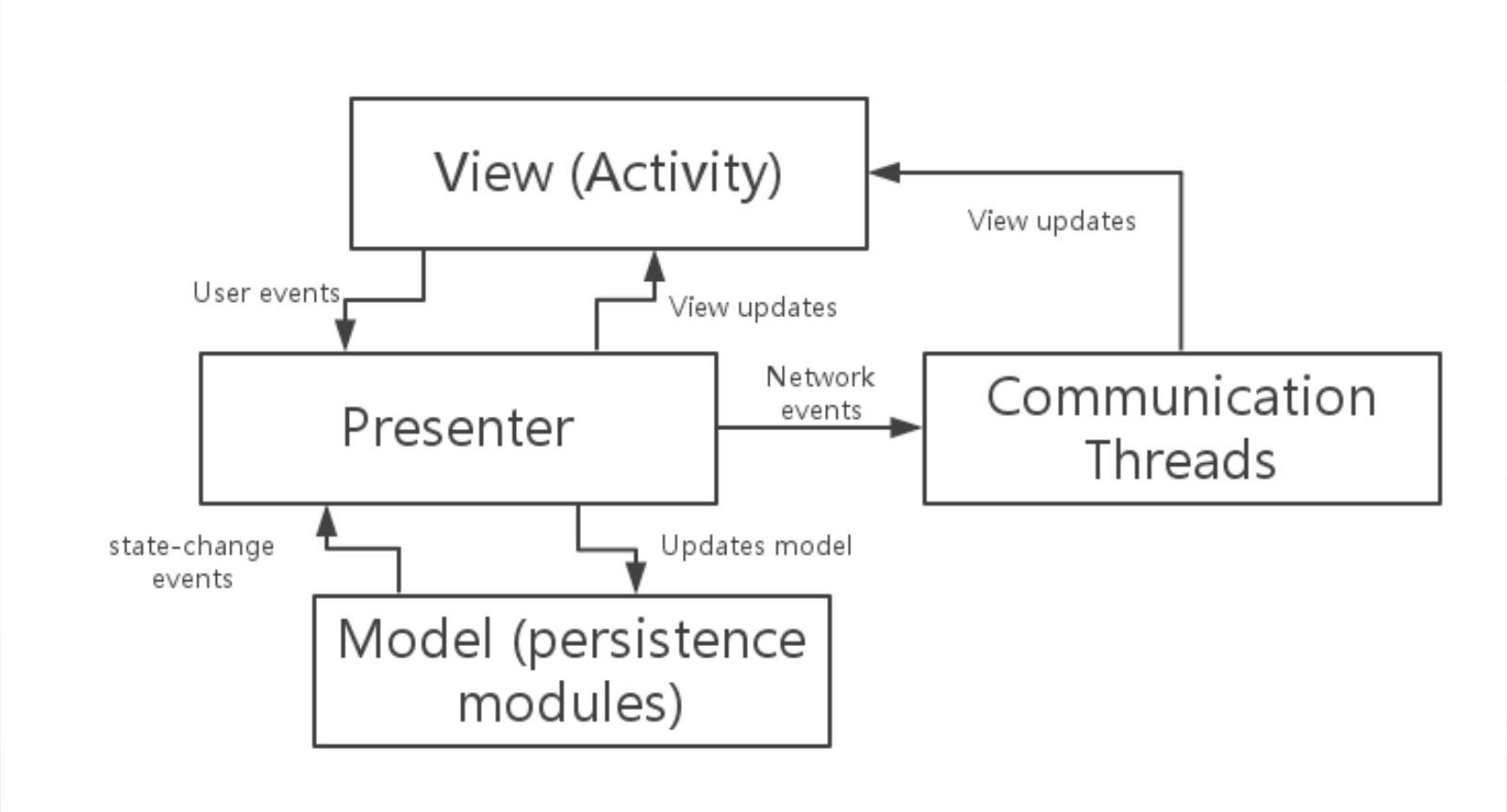

Figure 3. Modified MVP Pattern [2]

Reliability Cont'd

- Structure codes for better management? Dependency Injection (DI)

```
public CoffeeMaker(){
        this.heater = new Heater(); this.pump = new Pump;
        …}
```

- What if you want to reuse those objects outside the scope?

```
public CoffeeMaker(Heater h, Pump p, Beans b){
        this.heater = h;
        …}
```

- Then you can reuse those objects outside the scope, passing value to another class "ElectricCoffeeMaker" to function. [2]

Reliability Cont'd

- Threading Models

Glassware ← auth (later), pull, push → Choreographer

Data on Glassware from "pull" and "push"

Glassware Background WIFI, Choreographer connectivity monitor

Network Behaviors which might be time-consuming

*Confidentiality

- Authentication is an optional but necessary task

Tell the Choreographer which is the "right" Glassware to send messages (pull and push operations)

IPv4 public address of Glassware wrapped in authentication packet

HTTPS website: https://ifcfg.me/ipv4

Note: W3C sponsored, stable and reliable

Connectivity

- WFI

Android System does not validate WIFI with available data

Linux shell command: “ping –c 1 www.google.com”

- “pull” and “push” operations

[Choreographer with Glassware]

Initiated after auth operation;

Connectivity Cont'd

- REST and TCP connection

REST built on WEB HTTP:

when you request, I respond back (auth and pull)

TCP built on transport layer of networking protocols:

Reliable (triple hand-shake), but functions rely on implementations (push)

Choreographer Services

- Services (registered on Choreographer's xml file)

AuthService ← RESTfulService ← Glassware auth

…… → ..... → auth response

SensorManager (Sensor) ←RESTfulService ← Glassware pull

..... → .....→ pull response

Sensor → TCPConnector → Glassware push operation

Tests

- Communications Testing

Choreographer's logs and Glassware's logs

e.g. getSensorList pull request delay pattern:

0.02, 0.7, 0.15, 0.16, 0.16, 0.15, 0, 0.01, 0.015, 0.01 … (converges to 0.01s or 10ms)

- Glassware UI Testing (manually)

  Periodic refreshing behaviors

- Integration Tests

  Finite State Machines, transition inspection

Conclusion

- Usable Glassware with good interaction designs.
- Reliable Glassware with modified MVP pattern, DI and Threading Models.
- Integration of Glassware and Choreographer with well-designed REST, TCP connections on auth, pull and push operations for Connectivity.
- The authentication operation ensures the optional but necessary Confidentiality.
- Services designed on Choreographer to meet needs for integration with Glassware
- Final Tests on Glassware, on Integration with Choreographer

References

- [1] ITACA-Sabien, UPV. (2016). *Integración de sensores de salud y dispositivos wearables*. Retrieved May 15, 2016, from ITACA-Sabien Laboratory website*:* *http: //www.sabien.upv.es*
- [2] Hanqing, Z. (2016). Final Report. Retrieved May 15, 2016, from QMPlus Module “Project” uploading system: http://qmplus.qmul.ac.uk/mod/assign/view.php?id=524119

THE END

- Thank you for listening!