

Programming Ubiquitous Things 2024/25 - Spring 2025 - UiO

Context-Awareness (part 2 of 2)

Prof. Paulo Ferreira
paulofe@ifi.uio.no
UiO/IFI/PT

© Paulo Ferreira

1



Context-Awareness (2/2)

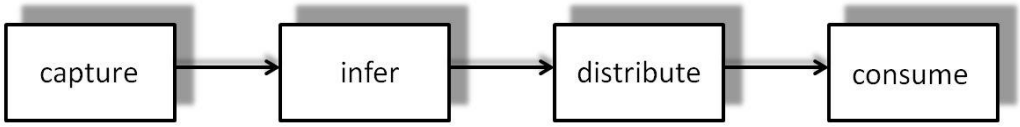
- Definition of Context — Addressed in the previous class
- Context-Aware Systems
- Distributed Context-Aware Systems
- Local vs. Distributed Context-Aware Systems
- Collaborative vs . Non-collaborative Distributed Context-Aware Systems
- Examples of Sensors and Types of Context
- Taxonomy (capture, infer, distribute, consume)

© Paulo Ferreira

2

Taxonomy

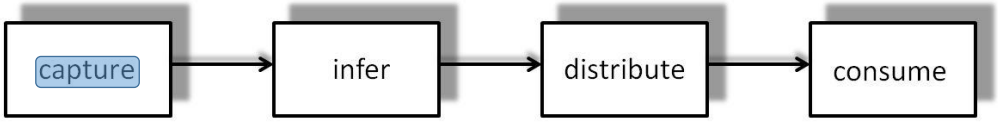
- Clearly **categorizes the architectural options** available to the application developer, explaining the **advantages and disadvantages** of each approach
- **Analyzes** the problems that are specific to distributed context-aware systems, whose (distributed) components have to communicate with each other, and how that affects **availability** and **scalability** of such systems
- Exemplifies each option with real applications that were deployed and evaluated by end users, instead of relying on generic toolkits, frameworks, and prototypes
- Four layers:



© Paulo Ferreira

3

Taxonomy (cont.)



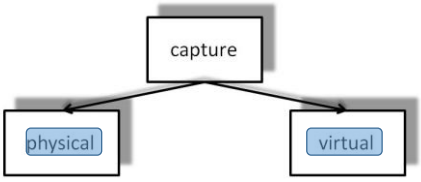
- Context data is **captured** from the environment using sensors
- A **context inference step** is needed to obtain higher-level aggregated data:
  - e.g., GPS device captures geographical coordinates from which a place (city, building, etc.) is inferred
  - this step is also known as *feature extraction*
- **Distribution of context data** among its components in an efficient and scalable manner
- Client applications **consume** this information in order to provide relevant services to their users.
- *storage layer is left out* because we focus on how context data is propagated

© Paulo Ferreira

4

Capture

- The Capture layer is responsible for **acquiring context data from the environment using sensors**
- The word "**sensor**" not only refers to sensing hardware but also to every data source which may provide usable context information:
  - physical, and virtual

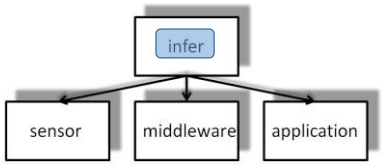


- **Physical sensors** are hardware sensors capable of capturing physical data such as light, audio, motion, and location:
  - location is, by far, the most researched type of physical sensor
- **Virtual sensors** acquire context data from software applications, operating systems, and networks:
  - detecting new appointments on an electronic calendar and watching the file system for changes are examples of virtual sensors

© Paulo Ferreira

Infer

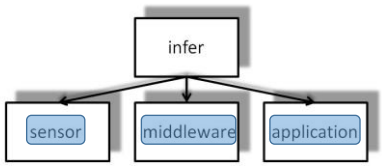
- The *Context-Inference* layer, is also known as the *Preprocessing Layer*
- It is responsible for **reasoning and interpreting** raw context information provided by sensors on the Capture Layer:
  - e.g., end-user applications do not need to know the exact latitude and longitude of an entity, being much more interested in knowing the place (city, street, etc.) in which the entity is located
- Context inference usually involves some kind of **reasoning**:
  - a transformation is needed to reach a higher level of abstraction
  - it is normally associated with classification techniques
  - e.g., inferring the user activity (walking, running) from an accelerometer
- We focus on ***where*** the inference occurs



© Paulo Ferreira

Infer (cont.)

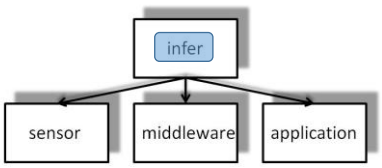
- The type of information inferred from sensorial data is specific to each application:
  - e.g., taking the data provided by an **accelerometer**, an application might use that information to know whether a user is **walking or running**, while another may be more interested in detecting **screw tightening**
  - it may make sense to **move the inferring task to the application** because of the specific semantic needs it tries to accomplish
  - transforming raw sensorial data into high-level context information can be a **resource-demanding task** (unsuitable for applications on constrained devices such as mobile phones)
  - some systems use a **middleware component** running in a machine with higher capability

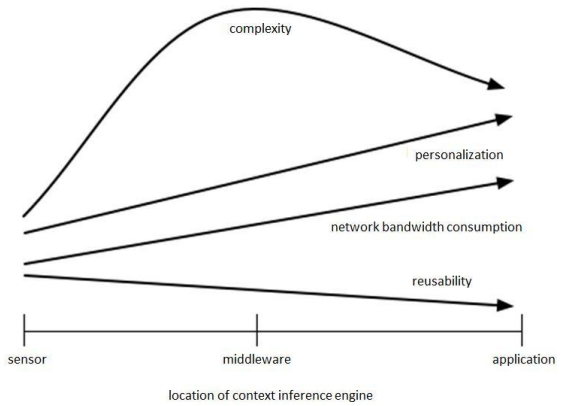


© Paulo Ferreira

Infer (cont.)

- The three **locations for inference** are intimately related to the following properties of a context-aware system:
  - Network bandwidth consumption
  - Complexity (CPU/memory consumption)
  - Reusability
  - Personalization

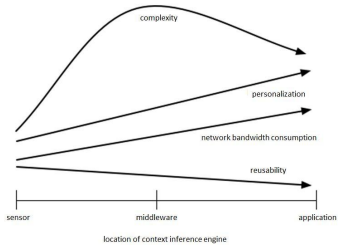




© Paulo Ferreira

Infer (cont.)

- Network bandwidth consumption
  - context inference transforms **fine grained sensorial** information into **coarse-grained high-level** data
  - it **reduces the amount of information** needed to represent a context message
  - moving the inference layer closer to the sensor results in **less network bandwidth consumption**

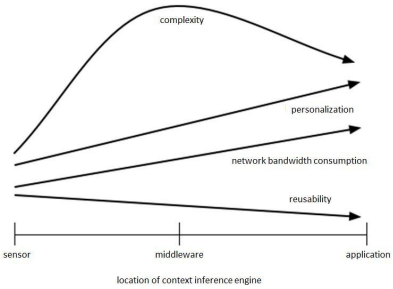


- Complexity (CPU/memory consumption)
  - context inference mechanisms can lead to **high CPU and RAM consumption**
  - even if the device is capable of computing the information, it can lead to excessive and unsustainable **battery consumption**
  - thus, the developer has to deploy the inference engine in a **resourceful machine** (e.g., a server)
  - most physical **sensors** are attached to **low-capability devices** (to increase mobility) and do not provid context-inference mechanisms
  - virtual sensors often run on servers or desktop computers (where they can access virtual context information from databases, shared calendars, etc.), so they are able to provide higher-level context information than their physical counterparts

© Paulo Ferreira

Infer (cont.)

- Reusability
  - makes sense for context types as location, where the output is sufficiently ***standard*** **to be used by multiple applications**
  - e.g., **inferring the street name** from geographical GPS coordinates is a recurring requirement in location-aware systems and does not make sense to (re)implement in every end-user application
  - moving the inference layer **closer to the sensor increases reusability**

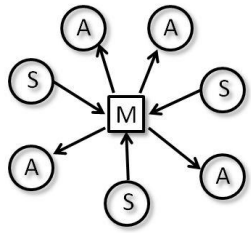


- Personalization
  - ability to **personalize** the inference engine to **better suit individual needs**
  - e.g., a phone application that allows its users to associate a certain movement (like drawing an imaginary circle with the phone) to some meaning or activity (e.g., going to lunch)
  - mediation of ambiguity - context inference includes dealing with **ambiguous data** and **explicit user mediation** is needed (thus adapting the inference mechanism to suit individual needs)

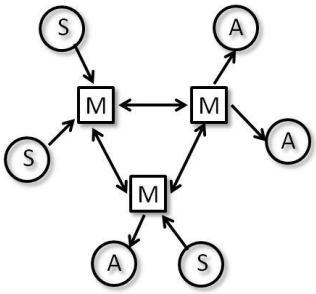
© Paulo Ferreira

Distribution

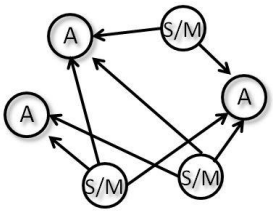
- Context has to be distributed among the system components (sensors, middleware, application): **centralized**, **partitioned**, **peer-to-peer**



a. centralized



b. partitioned



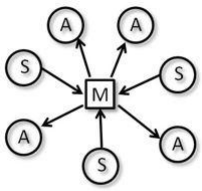
c. peer-to-peer

© Paulo Ferreira

11

11

Distribution (cont.)



a. centralized

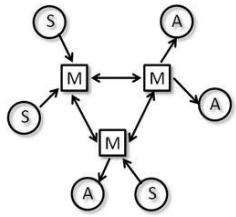
- Context has to be distributed among the system components (sensors, middleware, application):
  - centralized, partitioned, peer-to-peer
- Centralized:
  - **relieves resource-constrained devices** from CPU and memory demanding tasks
  - **simplifies the communication** since the devices (sensor or application) only need to establish a channel between them and a single component (the server)
  - it is a **single point of failure** and thereby **lacks robustness**
  - its **scalability is limited**—as more devices use the system, its performance will start degrading
  - it is not well fitted to scenarios where **network connection is intermittent** as it happens with mobile phones (it may stop working when the connection with the server drops)
  - solve some of the above mentioned problems by **using multiple servers**

© Paulo Ferreira

12

12

Distribution (cont.)

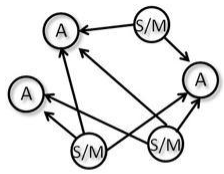


b. partitioned

- Partitioned:
  - each device still communicates with only one server, but **different devices may use different servers**
  - the **servers communicate directly with each other**, propagating information from devices in one partition to other partitions
  - this distributes the load among the servers, **increasing the system scalability**
  - **context transmission delay may increase** due to another hop in the communication path (the server-to-server communication)
  - as with the centralized approach, the **lack of redundancy** constitutes a limitation in assuring connectivity, since the failure of a server isolates all the devices in that server's partition

© Paulo Ferreira

Distribution (cont.)



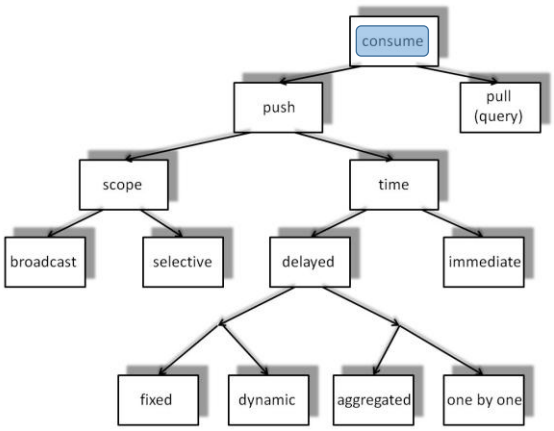
c. peer-to-peer

- Peer-to-peer:
  - each device communicates directly to the other devices
  - the middleware is usually embedded in the device itself
  - **do not suffer from the single point of failure**, since each device acts as a client and a server
  - it is **more resilient to network problems** as there are multiple paths to communicate
  - contrary to centralized and partitioned models, the components of these systems are in a **constant discovery process** of new sensors and applications that wish to communicate with them
  - depending on the routing algorithm, the **time necessary to propagate context may increase** when multiple hops are needed to connect the source and the destination nodes

© Paulo Ferreira

Consume

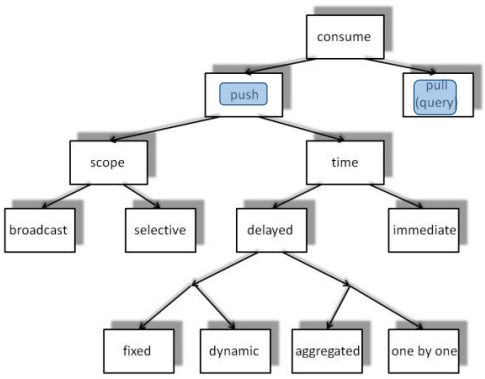
- After context information has been:
  - captured from sensors,
  - inferred into higher-level data,
  - distributed through the network,
  - it will be **consumed by client applications**



© Paulo Ferreira

Consume (cont.)

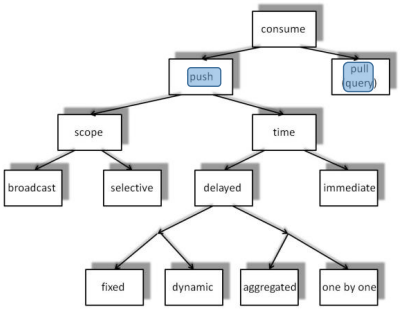
- **Pull-based** systems:
  - *pull-based systems* - the consumer (client application) queries the component that has the information (sensor or middleware) for updates
  - this can be done manually (e.g., the user clicks the "refresh" button) or periodically (e.g., checking for updates every 5 min)
- **Push-based** systems:
  - the component that has the information is responsible for delivering it to interested client applications
- The main distinction between these approaches lies on **who initiates the communication:**
  - if a client application initiates the communication, the system is pull-based
  - otherwise, it is push-based



© Paulo Ferreira

Consume (cont.)

- **Pull-based** mechanisms are **simple to implement**:
  - the client application only has to query a certain component with a certain **periodicity**
  - the application can simply **delegate the responsibility of refreshing** the information to the user
  - the **polling interval** can change according to the application current needs
  - e.g., an application running on a mobile phone can **increase the polling interval to save battery life**, when the battery power falls below a certain threshold

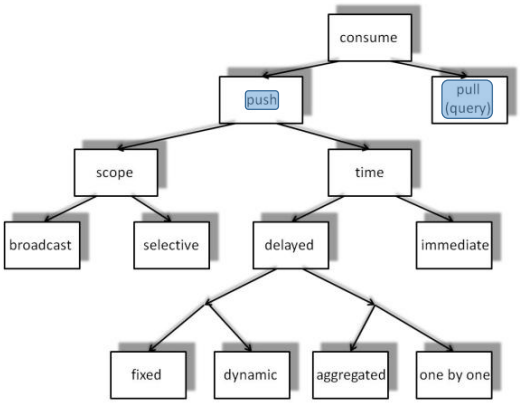


- However, this mechanism suffers from two problems that can be critical:
  - context information reaches the application with a delay (may equal the polling interval)
  - if polling interval is 5 min**, the application may receive context information from 5 min ago**, compromising its usefulness
  - the application can minimize this disadvantage by **reducing the polling interval**, but then, the second problem may start **occurring—most queries will be unnecessary** since there is no new context information available
  - since the application does not know when there will be updates, it has no alternative as to keep asking until there is new information, leading to **wasted network bandwidth, CPU and battery consumption**

© Paulo Ferreira

Consume (cont.)

- Push-based mechanisms are **more complex** than pull-based mechanisms.
- The component that has the information is also responsible for delivering it:
  - this component has to **know how to reach every possible consumer** that is interested in that information
- Usually, a permanent connection with all the consumers is necessary
- If the system has a large number of consumers, its **performance may degrade**



© Paulo Ferreira



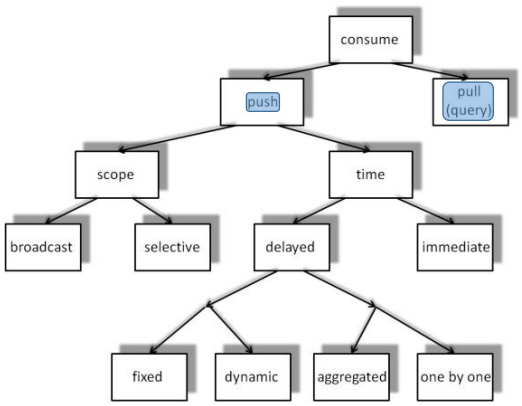
What happens of some client stops answering (ACK) ?

© Paulo Ferreira



Consume (cont.)

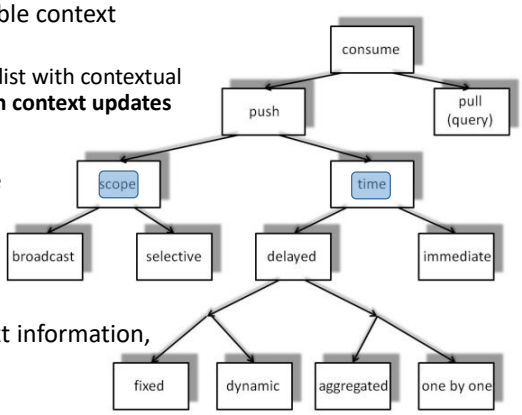
- Furthermore, these systems (both) have **poor scalability** as every new consumer degrades the performance even worse
- There are two measures to overcome the scalability problem:
  - *reduce scope* and/or
  - *relax delivery time*



© Paulo Ferreira

Consume (cont.)

- Users may be interested only in a subset of the system's available context information:
  - e.g., an application that enhances the traditional phone contact list with contextual information such as location and availability) is **only interested in context updates for the people in its contact list**
  - thus, if person A has person B in his list of contacts, A is notified if B moves from one location to another, but other people in the system without B in their contact list are not notified
- Selective scope systems propagate only a **subset of the available context information** based on user's selection
- Most context-aware systems propagate all the available context information, thus falling into the broadcast scope category

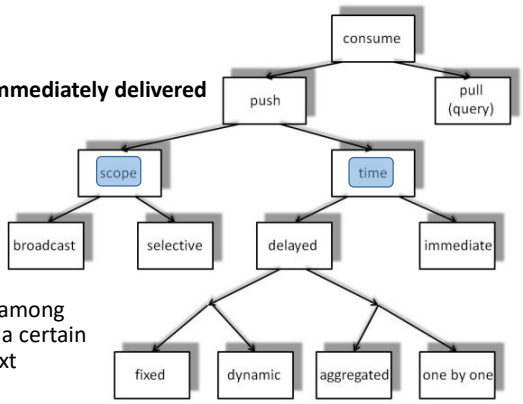


- Using scope is an effective technique to reduce the number of pushed messages in the system:
  - reducing the required **outbound network bandwidth** on the component that has the information and improving overall scalability
  - *quenching* - a mechanism that allows sensors to know whether client applications are interested in their information and only sending information in that case

© Paulo Ferreira

Consume (cont.)

- Time:
  - to improve push-based systems scalability,
  - assuming that certain context information **does not need to be immediately delivered**
- Users **tolerate some lag** as long as the information does not require urgent attention:
  - e.g., a context-aware system that shows friends near me is not required to be continuously up to date
  - on the other hand, an application that shares current availability among a group of friends (e.g., to help decide whether a person can call a certain friend without interrupting anything) loses its usefulness if context information is not propagated as soon as possible

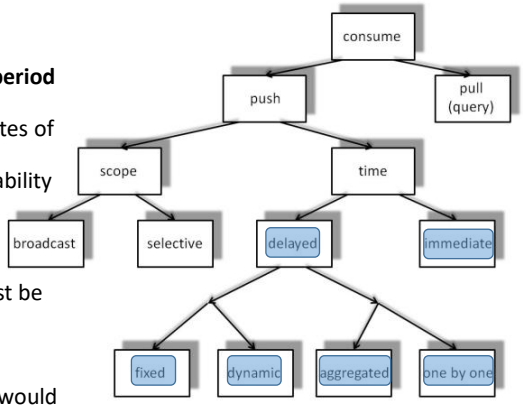


- Some of these issues have been studied in the context of *optimistic replication algorithms:*
  - increase availability and scalability of distributed data sharing systems by allowing replica contents to diverge in the short term

© Paulo Ferreira

Consume (cont.)

- Fixed vs. dynamic:
  - must decide when to actually push the updates
  - decision can be made based on a **fixed criteria such as time period** (push the update every x seconds) or **amount of retained information** (push the update when there is more than x Kbytes of context information)
  - obviously, both criteria can be used: e.g., to improve the scalability of a context-aware IM application
- Aggregated vs. one-by-one:
  - since there is a delay in context propagation, information must be retained and may start accumulating
  - when the system decides to push the update, there may be a considerable number of messages to transmit
  - simple approach is to **transmit the messages one-by-one**, as would be the case if there was not any delay (this can be extremely inefficient)
  - alternative solution is to **aggregate all delayed information in just one** (e.g., GPS locations during a certain amount of time)
  - even when all the retained messages are relevant, they can still be **aggregated in one big message**, achieving **higher levels of compression** and much **less round-trips** in the connection path



© Paulo Ferreira

Consume (cont.)

- Note that:
  - in context-aware related literature, the term "**aggregation**" is often associated with the **combination of information from different sensors**, along with **conflict resolution**
  - **do not confuse both** uses of the same term!
- More to read:
  - See several examples of context aware systems in the bibliography (Distributed Context-Aware Systems. Paulo Ferreira, Pedro Alves. SpringerBriefs in Computer Science, 2014)

© Paulo Ferreira