SOFTWARE CHALLENGES

PHILIPPE LALANDA

KOBE UNIVERSITY – AUGUST 2017

Purpose of this lecture

- Show that software plays a major role in pervasive applications
- Describe the major challenges
- Show that much progress is still needed

Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

More non functional properties

Conclusion

Pervasive computing

Pervasive computing promotes the integration of smart, networked devices in our living environments in order to provide us services.

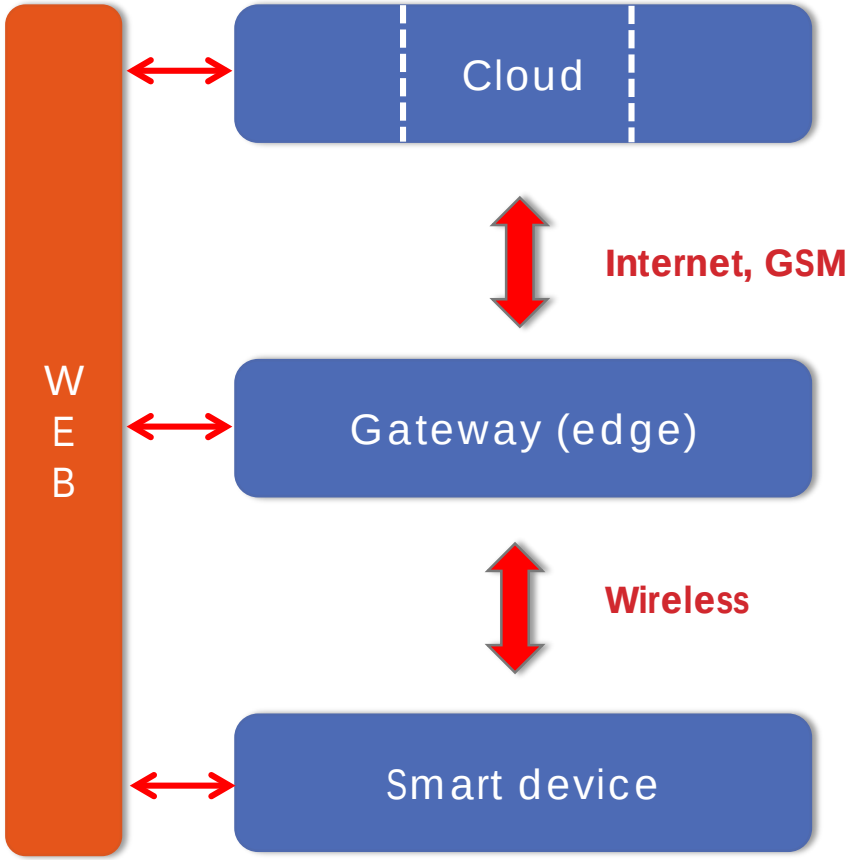
Those services are

- are context aware
- require minimal and natural interaction
- bring real added value
- are easy to administrate by end-users

Implications

Context-aware	Capture and model contextual information Adapt services accordingly
Minimal and natural interfaces	Develop new interaction mechanisms (voice, virtual reality) Put Human in the loop
Added-value services	Real-time and deliberative services Powerful infrastructure (processing, storage)
Easy administration for end-users	Automated mechanisms User care (devices, applications)

This leads to complex architectures...



Deliberative services - big data, learning, advanced processing.

Real-time services and data redirection.

Sensing and streaming data.

… where software is key

For a given service, code is needed in

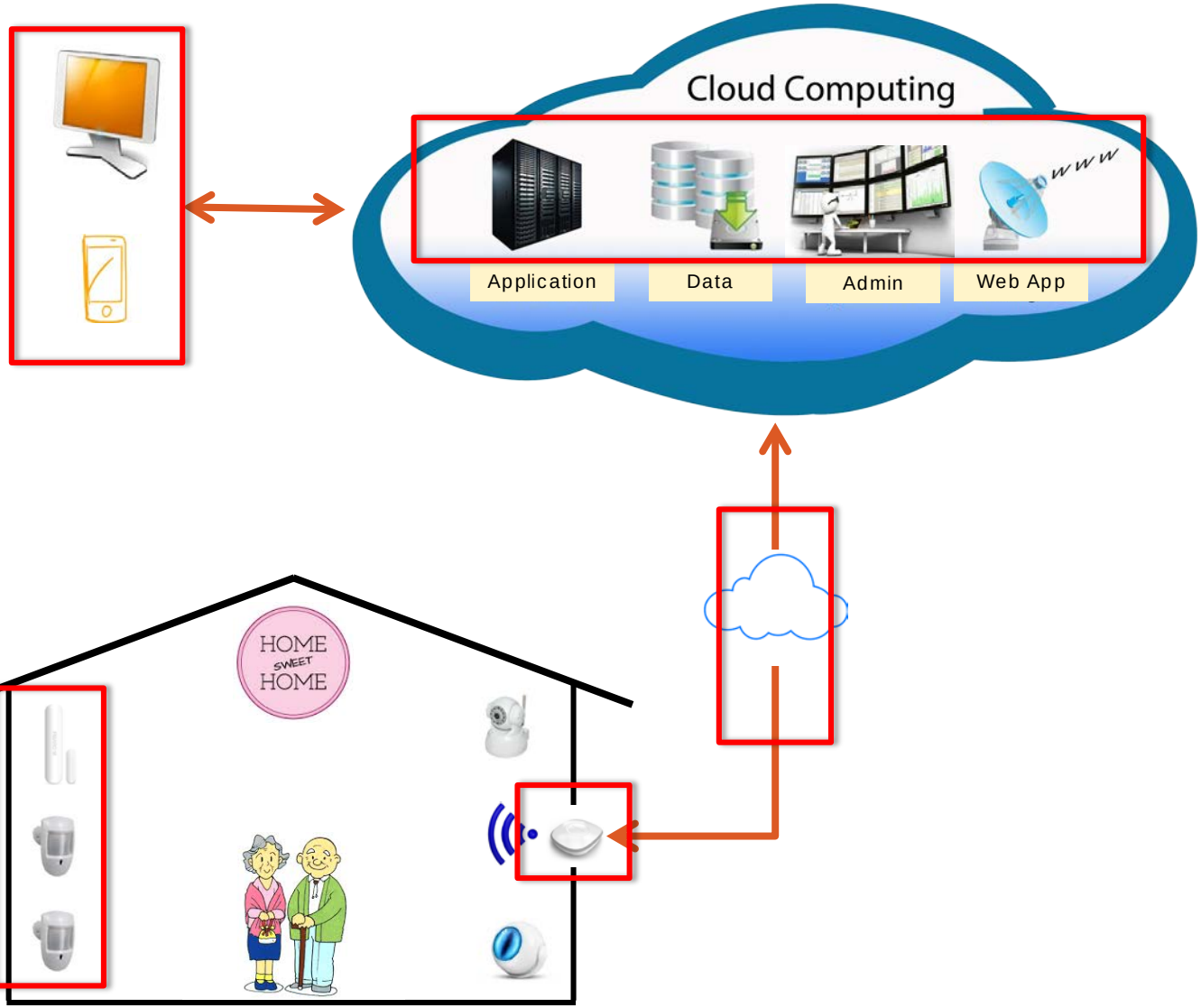
- devices
- gateways
- communication
- multi-modal interaction
- mobiles
- cloud

In addition, developing this code is very challenging.

Note: there is also complex code for the infrastructure!

Example – where is the code of "activity tracking"?

Activity tracker



Code complexity

All this code is for a single application

different requirements

different languages

different Operating System (if any)

Very different from usual applications. It often implies multiple parties (for Internet transport for instance).

The cost for a newcomer is very high (Bosch example)

Major requirements

Device level	Hard real-time Limited resources (CPU, code size, memory) Consumption Security
Gateway (hub) level	Soft real-time Flexibility (networks, devices, clients) Autonomy Security
Mobile level	Responsiveness Usability Flexibility (to change layout/functions) Security
Cloud level	Advanced data management and analysis Scalability (machines, storage, ...) Security

Impact n° 1

Such architecture cannot be entirely developed by a single person or team

- too different domains
- too many technologies

Collaboration between individuals and companies are needed

- hard to put in place
- this explains why many products are not good or limited

Impact n° 2

Such architecture cannot be entirely developed without tools

- middleware
- programming languages
- Integrated Development Tools

iCasa is one of them

This lecture

The purpose of this lecture is to highlight the non functional requirements of the code and see their impacts on software

- context awareness
- distribution
- heterogeneity
- dynamicity

Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

More non functional properties

Conclusion

Context awareness

A service is context-aware if it uses information about the situation to perform its tasks. This includes information about the environment, the users, the supporting machines, etc.

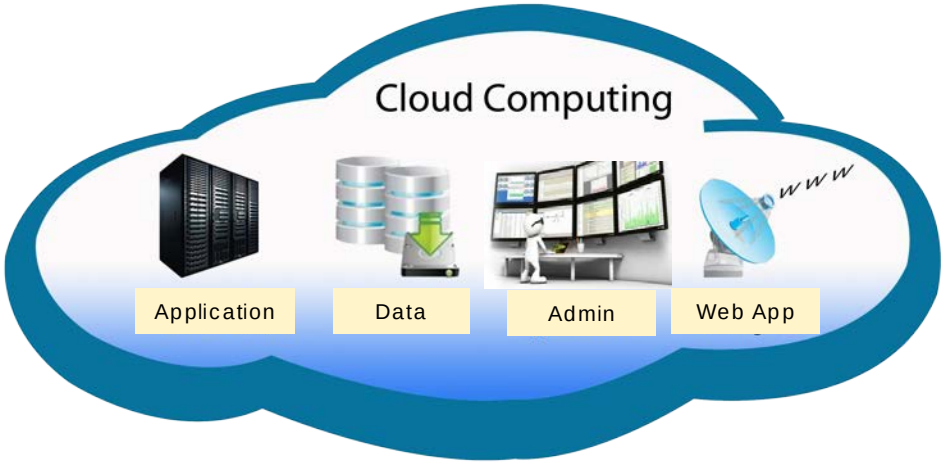
The service collects information in the environment.

It uses this information for its execution.

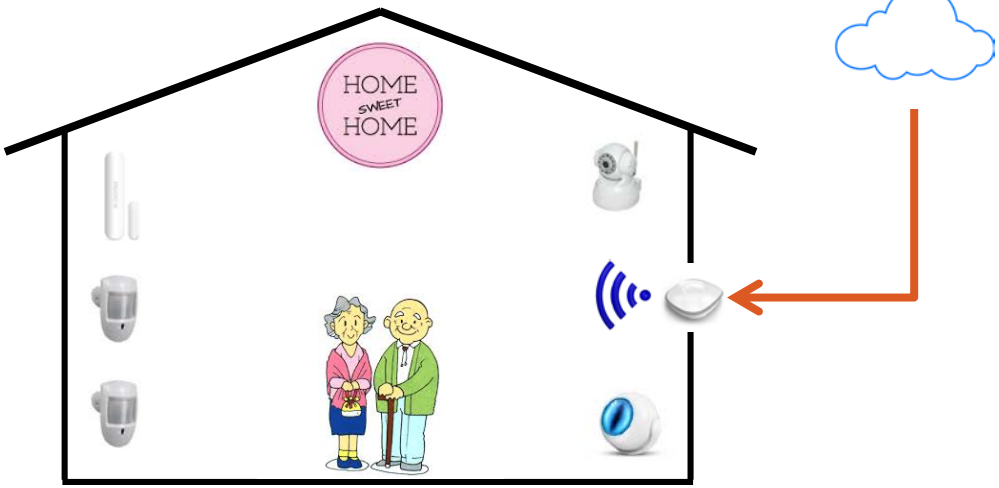
It may act upon the environment.

Activity tracking – context awareness

Activity tracker

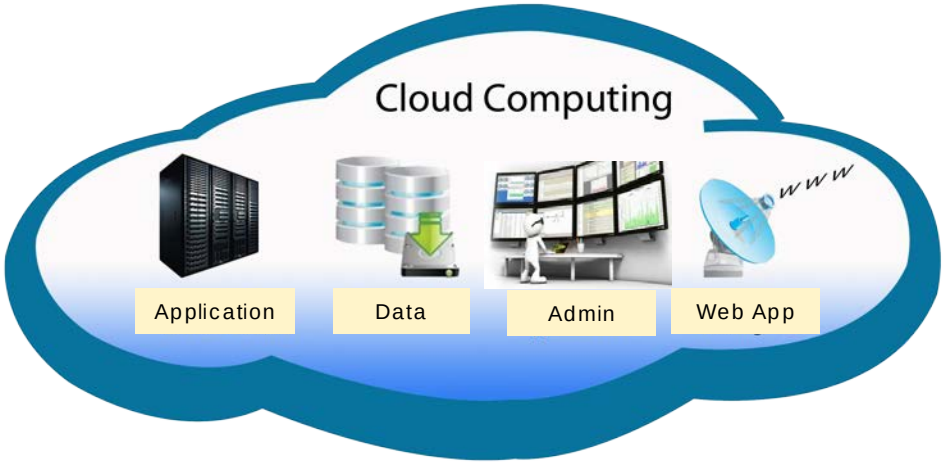


Use of the available devices and networks

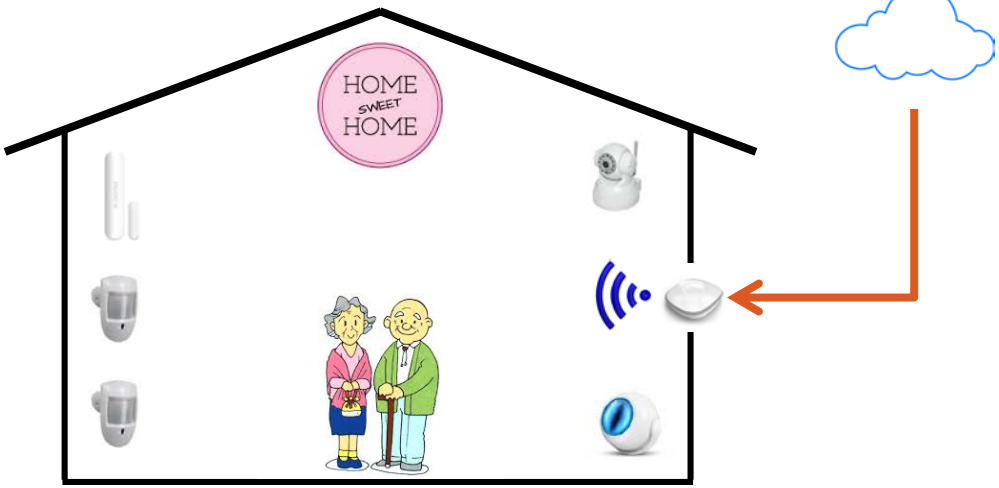


Activity tracking – context awareness

Activity tracker

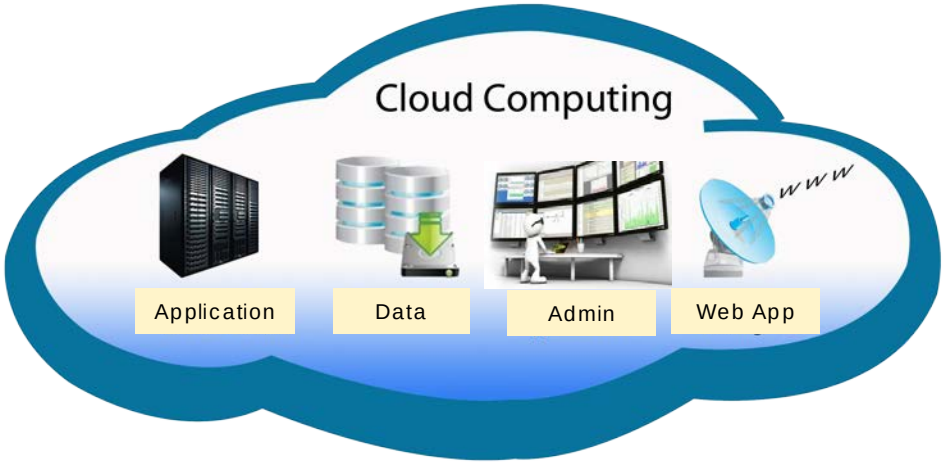


Consider the persons' situation (health...)

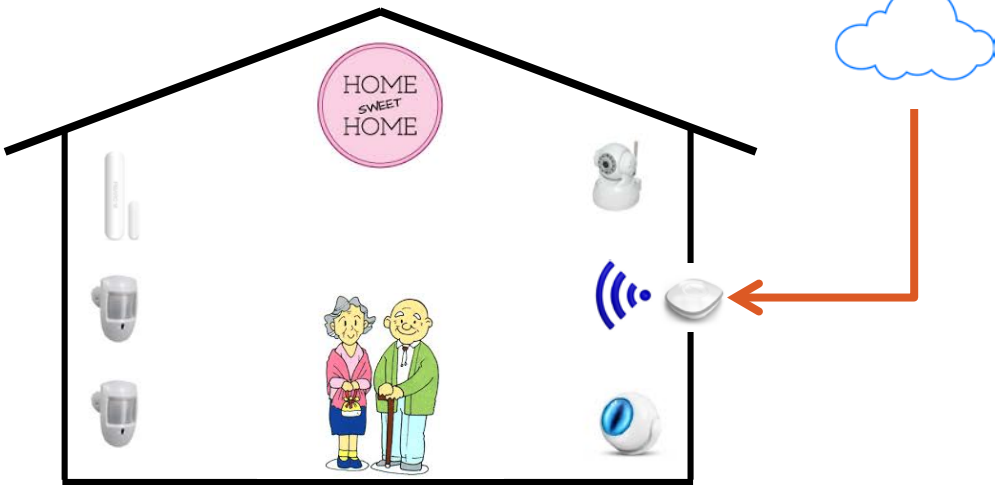


Activity tracking – context awareness

Activity tracker



Use of the available data to derive activity



Service adaptation

Software wise, context awareness raises major difficulties

- Observing the world
- Building a model of the world that is in line with applications needs
- Adapting software at runtime

Observing the world

The world is dynamic, stochastic and not fully observable

Dynamic: the world is evolving

Stochastic: evolutions depend on actions
but also on some unknown factors
that cannot easily be predicted

Not fully observable: some pieces of information
cannot be captured

Observing the world

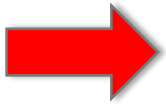
Pervasive application have then to build some form of representation of the world. This is called context.

How to model contextual information?

How to update it at the right pace?

Which information should be sought?

Should contextual information shared?



A lecture on that subject

World model

The model representing the world is

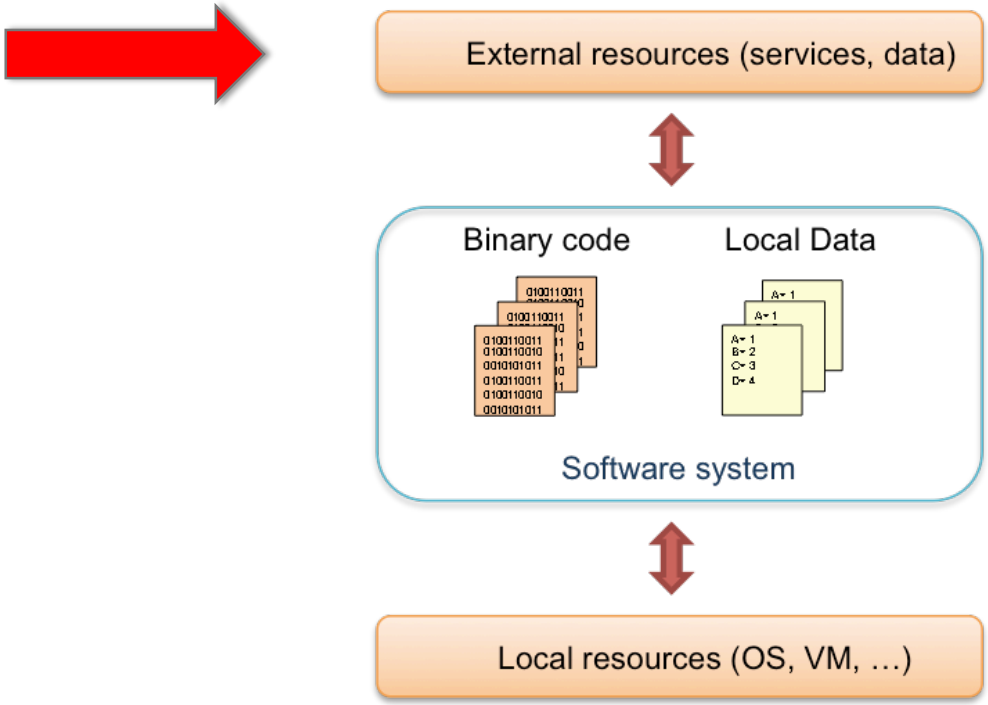
- incomplete
- uncertain (possibly with probabilities)
- late (image of the past)

This has to be considered in the application code

- use of Bayesian methods
- use of Markov model
- check causality
- ...

Adapting application code

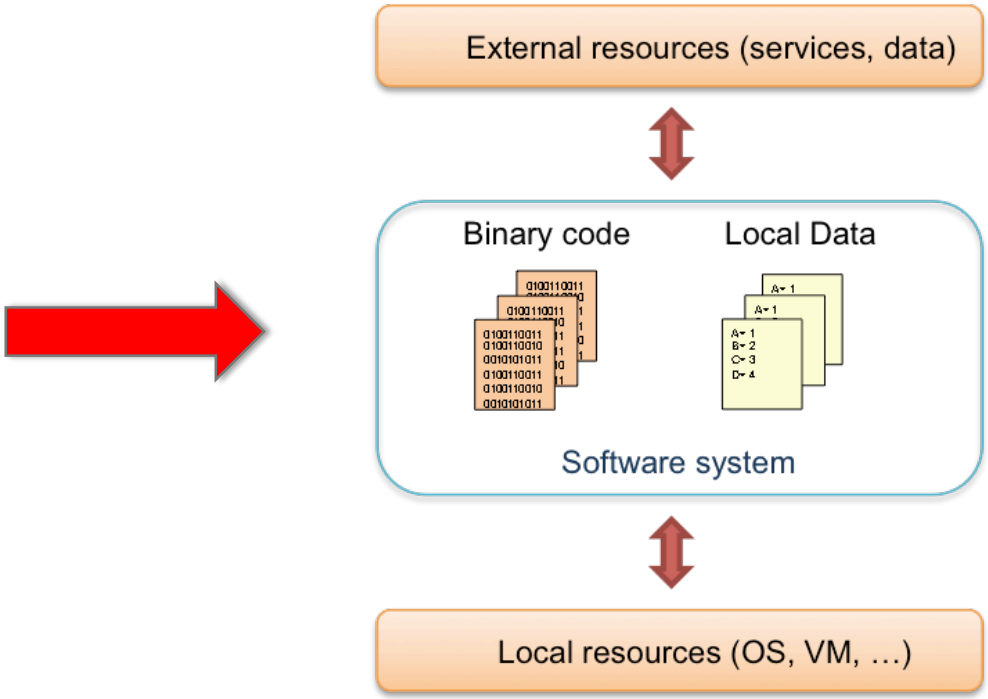
Services must constantly adapt to unpredictable changing conditions to meet their requirements.



Adaptation may result in using different resources ...

Adapting application code

Services must constantly adapt to unpredictable changing conditions to meet their requirements.



... Or, more complex, in changes in the code/data

Adapting application code

Code adaptation may concern every aspect of an application , including

- configuration
- algorithms
- placement
- display
- quality of service

Adaptation can be done

- at compile time
- at design time
- at run time

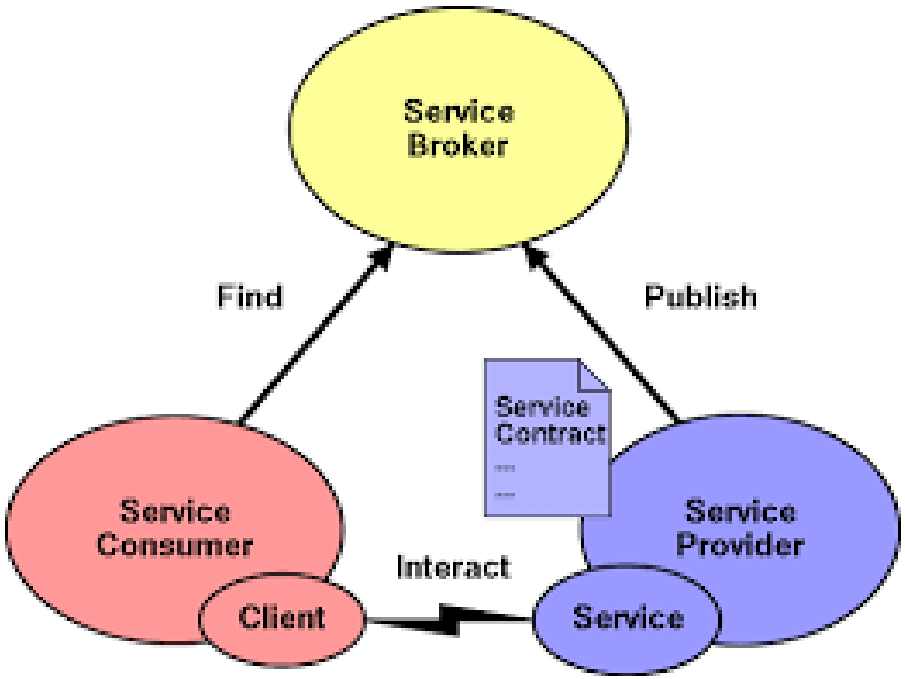
Approaches

Runtime adaptation

Separated configuration files and directives

Dynamic loading and binding (OSGi, Erlang, etc.)

service-orientation



Context-awareness and pervasive platform

A pervasive platform has to provide

- a programming model favoring adaptation
- a context service allowing the creation and management of contextual information

Ideally, adaptation is partially autonomic and based on generic mechanisms

Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

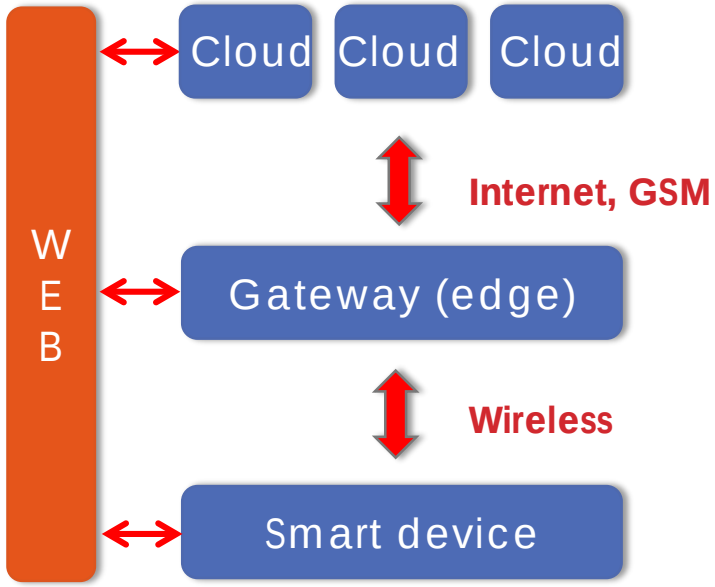
More non functional properties

Conclusion

Distribution

Pervasive applications are inherently distributed among mobile and stationary devices

- devices integrated in the physical environment
- mobile devices (smartphones, smart watches, ...)
- gateways
- edge computers
- cloud computers
- Web resources



Distribution - issues

Distributed software is complex to build and to manage

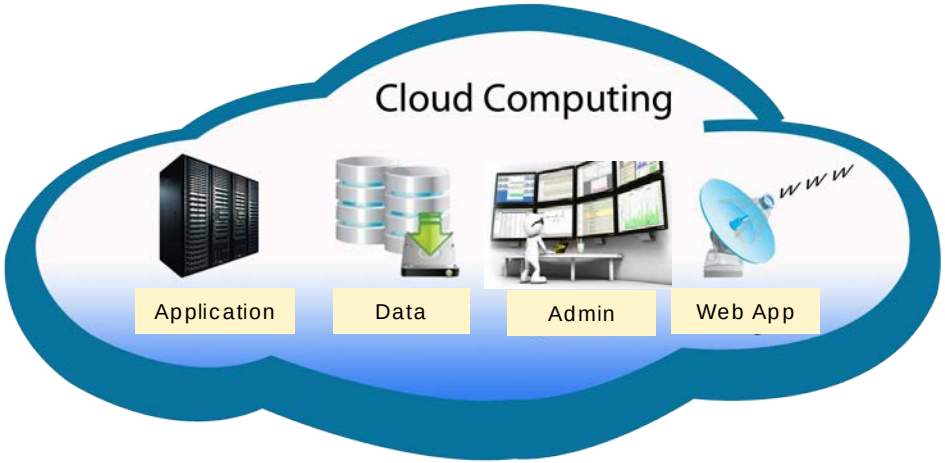
Important errors about distribution

The network is reliable	→ error handling
Latency is zero	→ take time into account
Bandwidth is infinite	→ prune data
The network is secure	→ protect data
Topology doesn't change	→ automate configuration
Transport cost is zero	→ select data to be sent

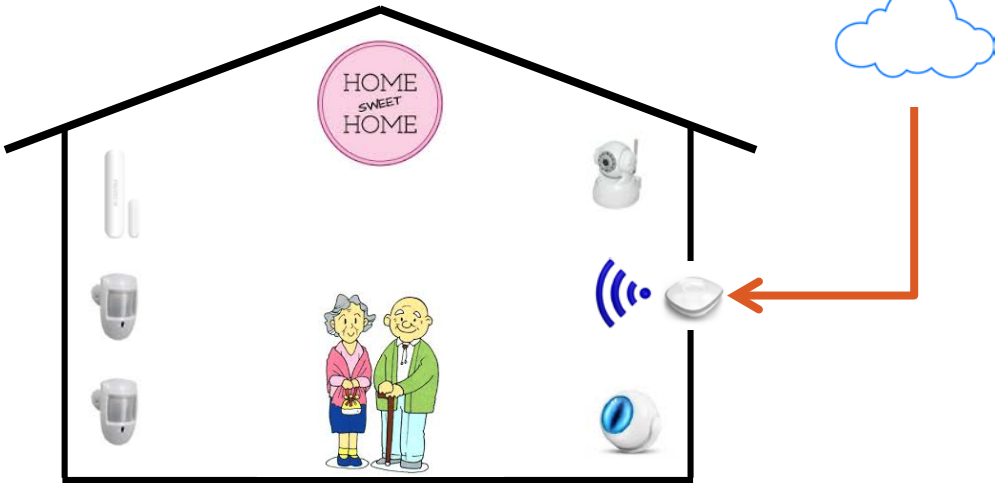
Distributed architectures require extra care at design time

Activity recognition (part of tracker)

Activity tracker

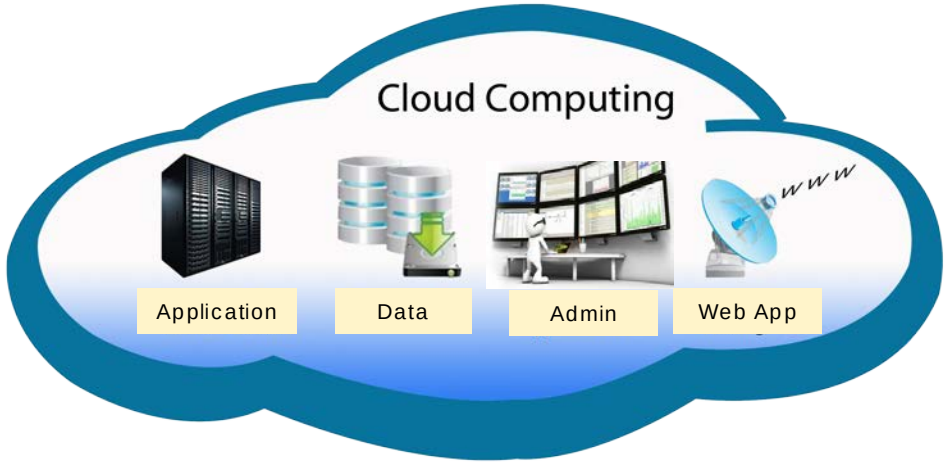


Good repartition real-time and deliberative services

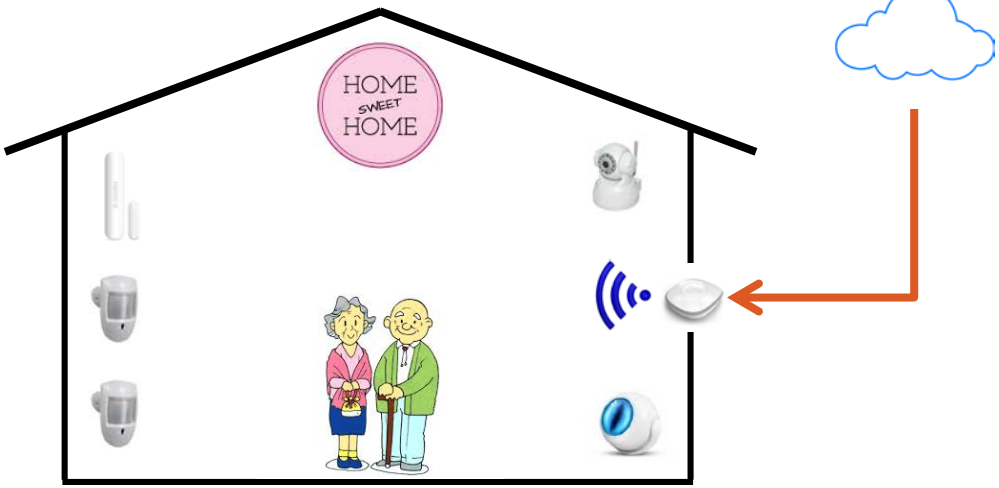


Activity recognition (part of tracker)

Activity tracker

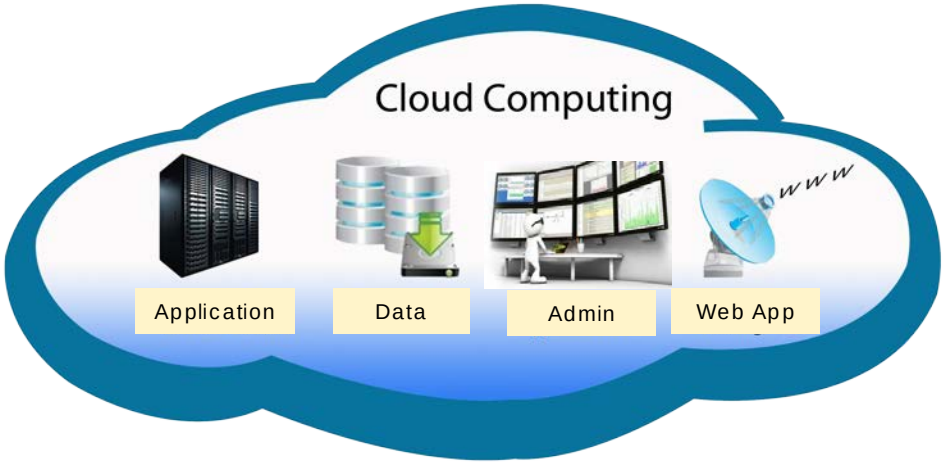


Only send up quality data (security and cost)

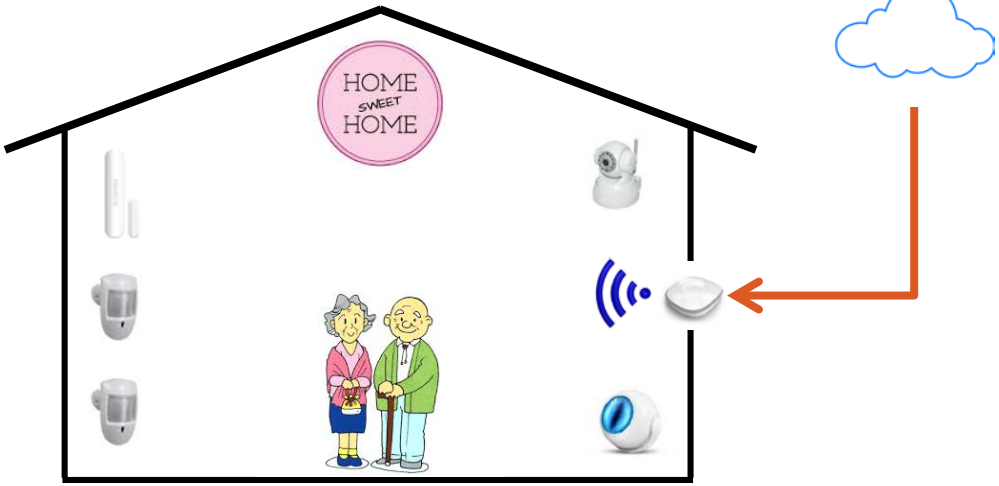


Activity recognition (part of tracker)

Activity tracker

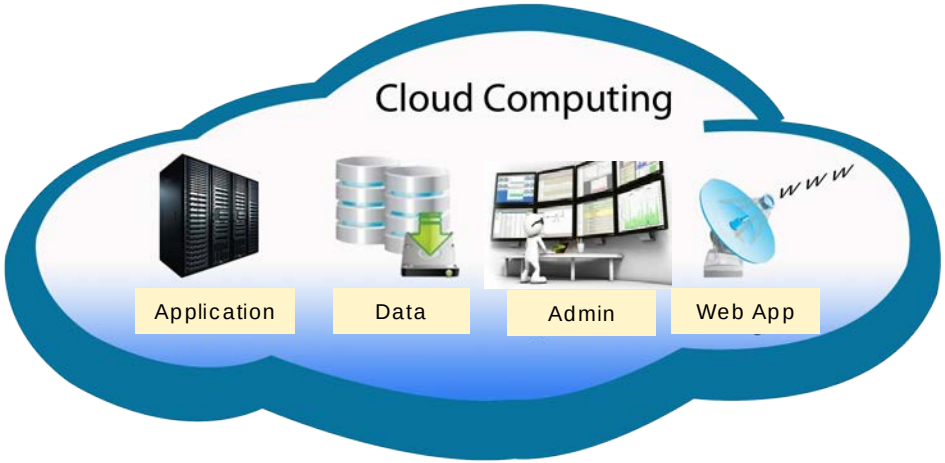


Automate configuration (Internet/4G/...)

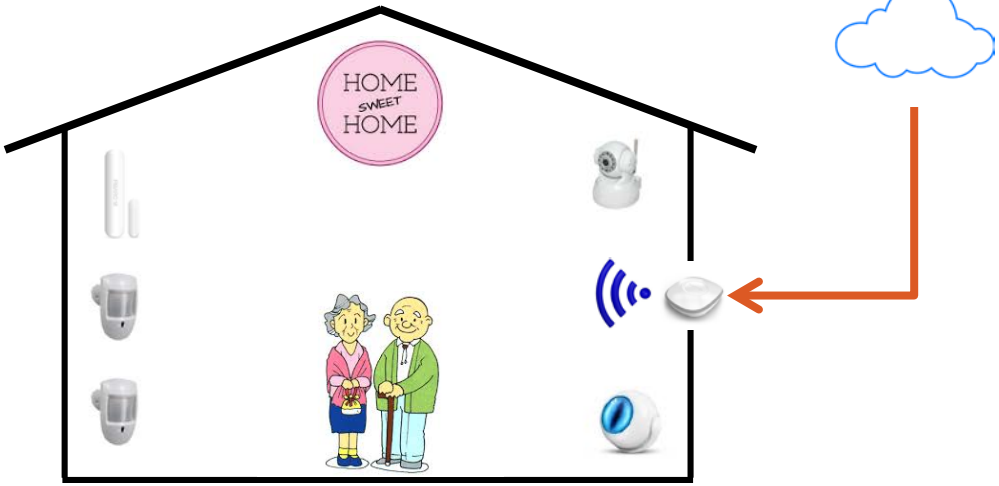


Activity recognition (part of tracker)

Activity tracker



Cache data in gateways, cloud, browsers (availability)



Distribution – architecture example

In the mobile industry, edge servers are more and more used to (partly) run applications backend near the mobiles. It improves

efficiency

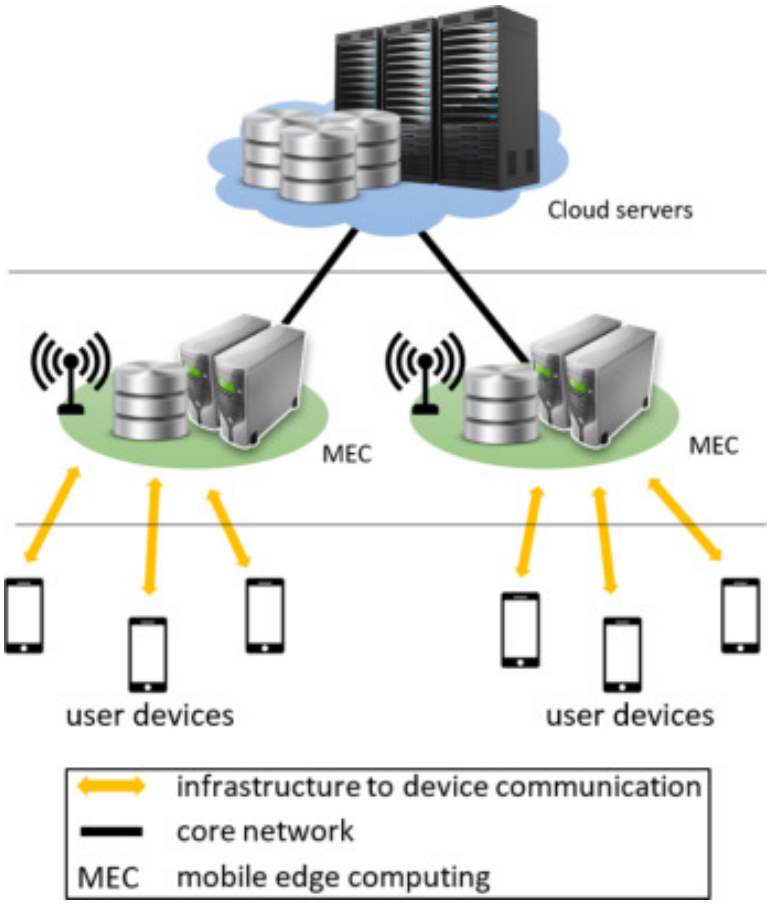
bandwidth

security

homogeneity

But application management is more complicated

Same kind of architecture in pervasive are coming



Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

More non functional properties

Conclusion

Heterogeneity

Pervasive architectures are by nature very heterogeneous.

Competing device and software providers are numerous. This implies different:

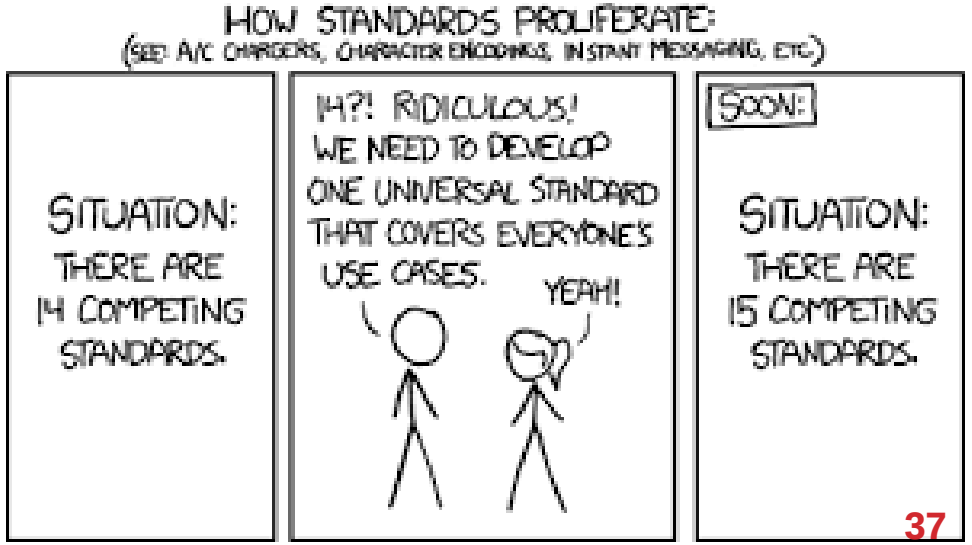
communication protocols

data formats

data semantic

supporting technologies

Many standards are proposed without much success (in software)



Heterogeneity – issues

Heterogeneous software is complex to build and to manage. It needs to be frequently updated (every time new data/devices come in).

Main issues

Protocol heterogeneity (even with the same specification)

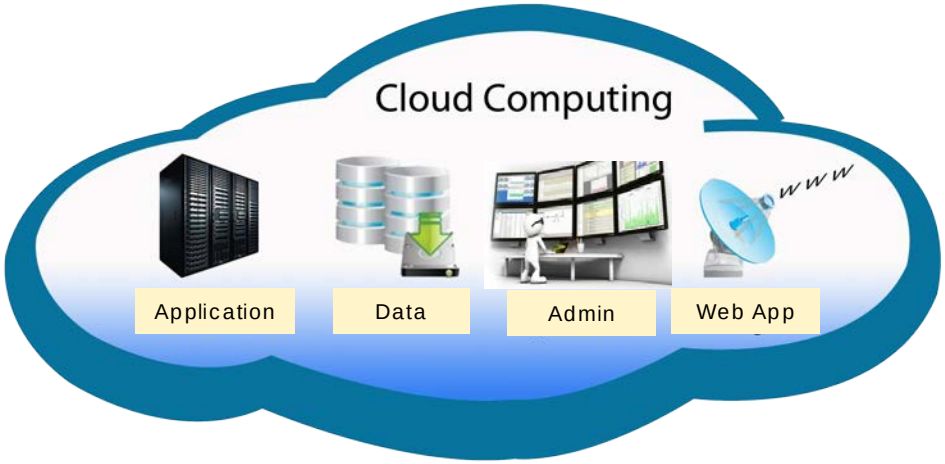
Syntactic and semantic differences

QoS merging

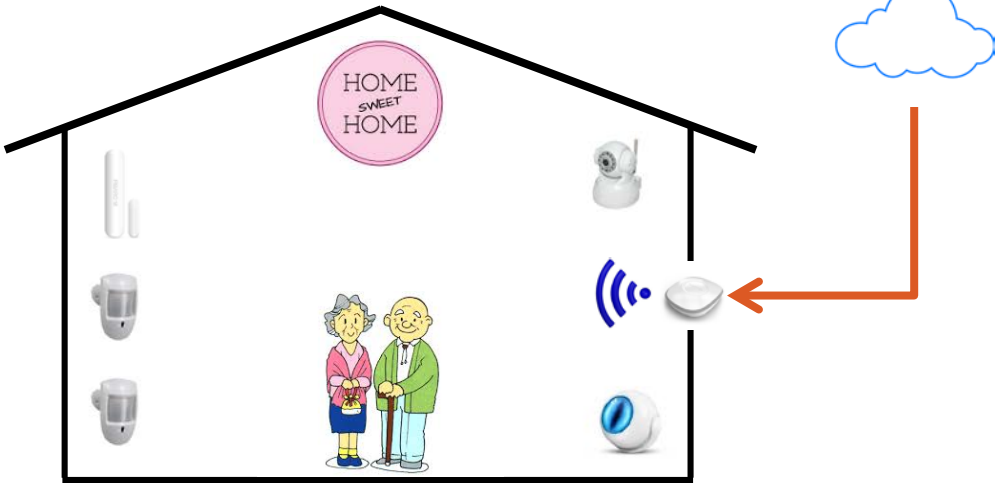
Loss of information

Activity tracking

Activity tracker

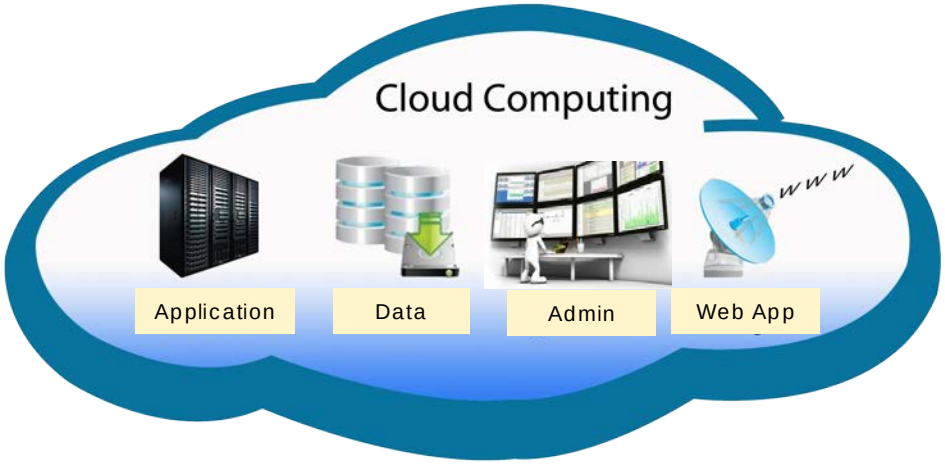


Use of devices of various brands

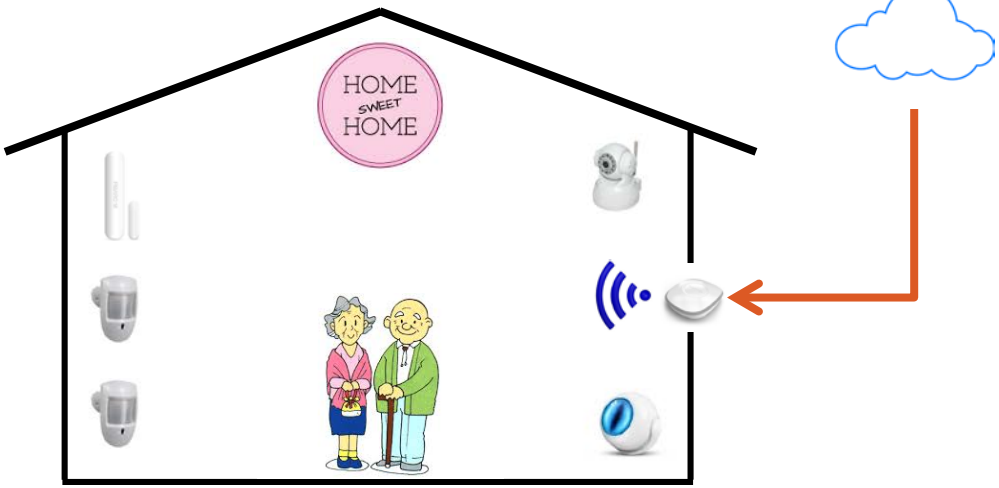


Activity tracking

Activity tracker



Use of devices of different browsers

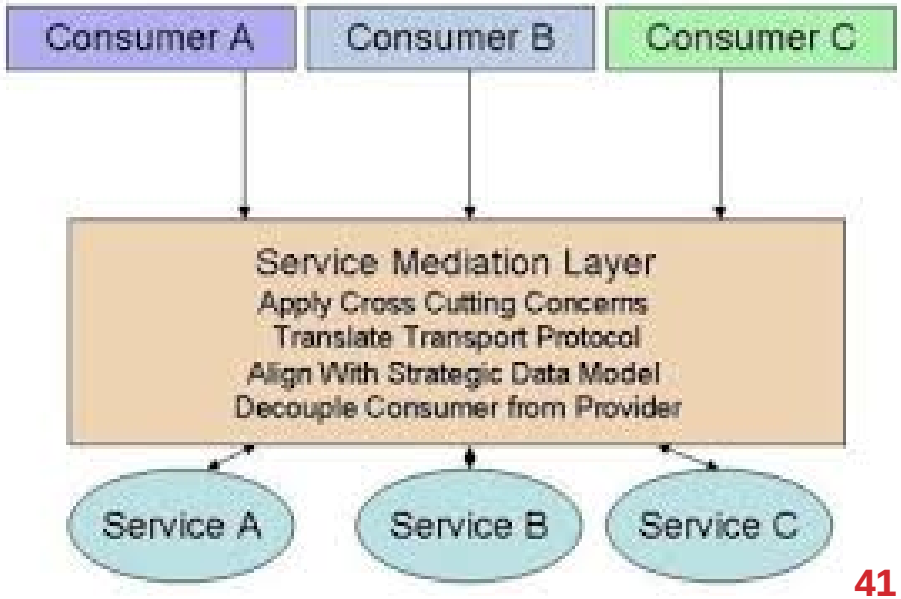


Mediation code

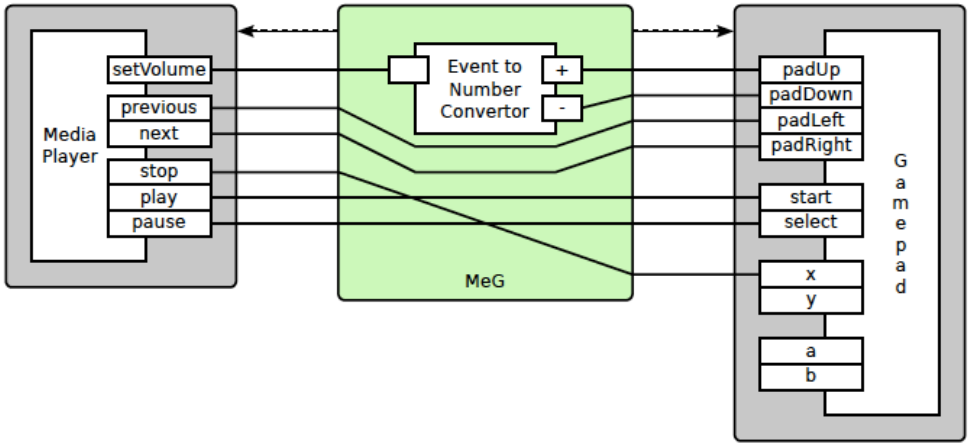
Mediation software have been proposed to integrate disparate information sources

- Communication alignment
- Syntactic alignment
- Semantic alignment
- Non-functional property alignment
- Persistency
- Monitoring

Generally, component based



Mediation for multi-modality



Interaction entre MediaPlayer et Gamepad

Issues with mediation

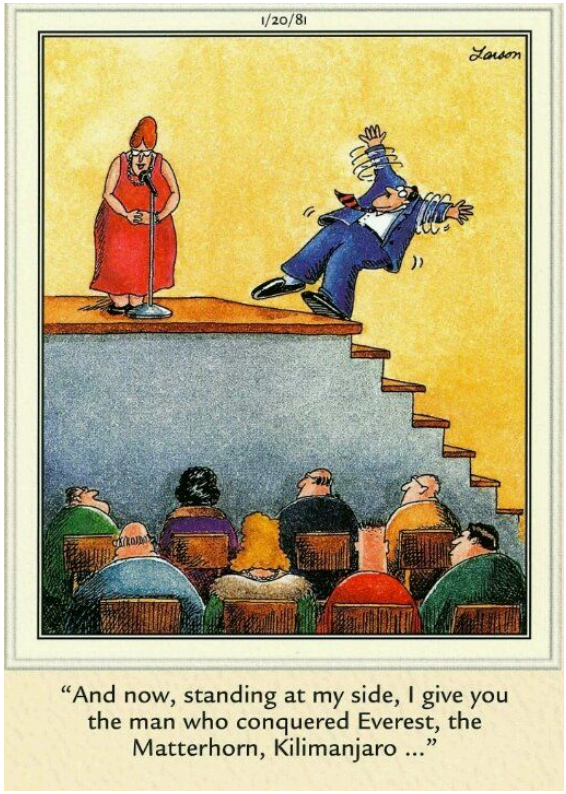
Several technologies must be used

highly skilled engineers

difficult to find

not in the same company

New mediation chains must be provided very frequently



Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

More non functional properties

Conclusion

Dynamicity

Every device, every system that contribute to the computing environment evolves and changes through time. It may also disappear/reappear anytime

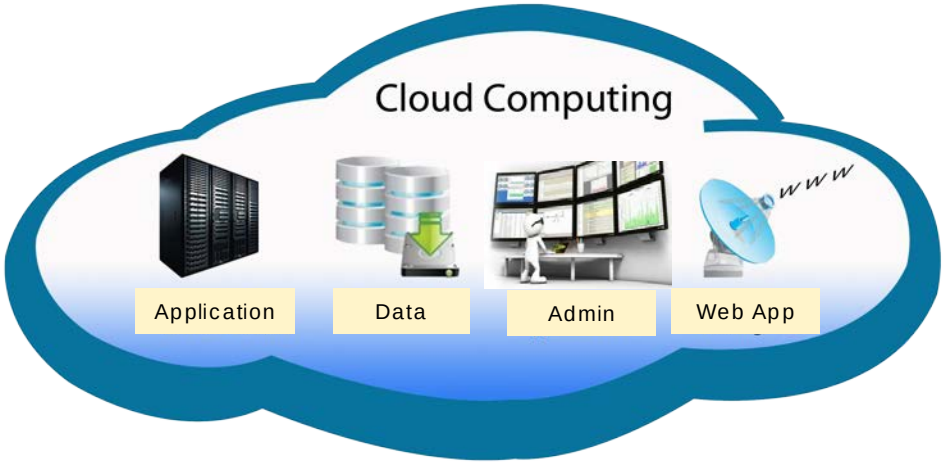
- Hardware or software failures
- Energy shortage
- Maintenance operations
- Functional evolutions

Applications must recover, generally in a dynamic fashion

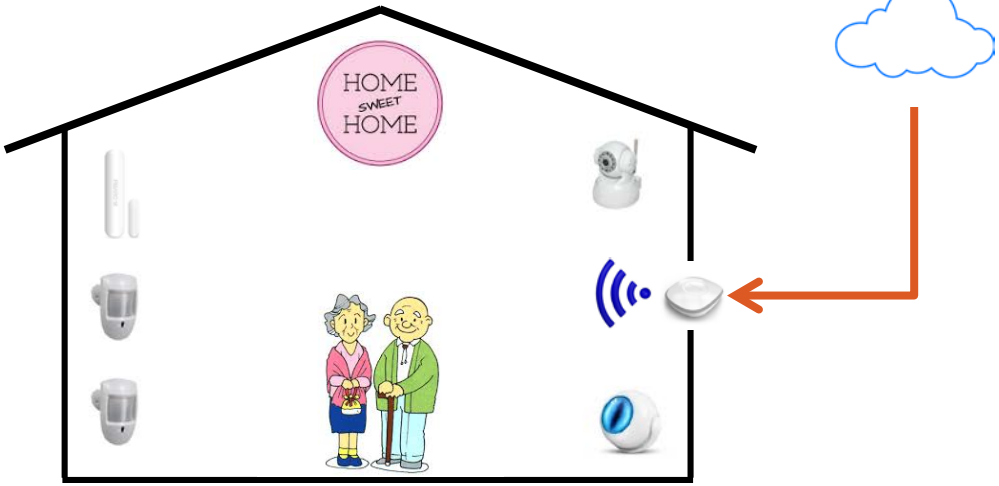
- they cannot be restarted often

Activity tracking

Activity tracker

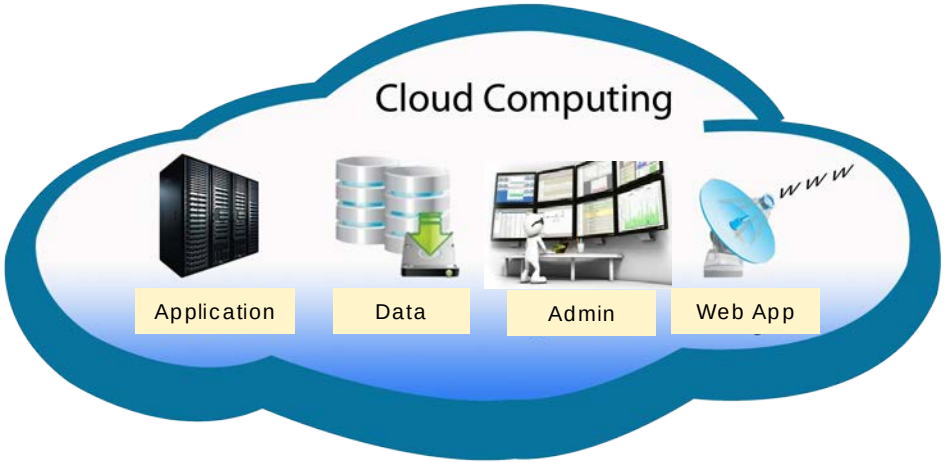


Device appear/disappear

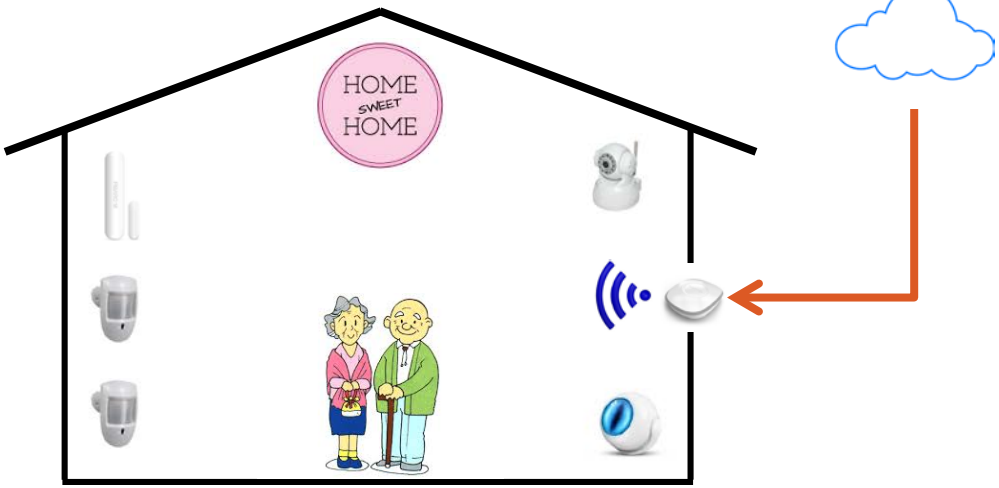


Activity tracking

Activity tracker



Computing resources appear/disappear



Dynamicity - issues

A pervasive applications must be able to opportunistically use the available resources.

Dynamic programming is however very complex

maintain control flow

don't loose data

avoid crashes (extremely complex synchronization code)

Variability in programs

Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

More non functional properties

Conclusion

Energy management- devices

Some sensors are connected to the mains power source

Others rely on battery (energy is then a limited resource)

- wake up on events (motion sensor)
- wake up on periods (thermometer)
- configurable (periods, motion, etc.)
- their use is not straightforward

Energy management – data storage

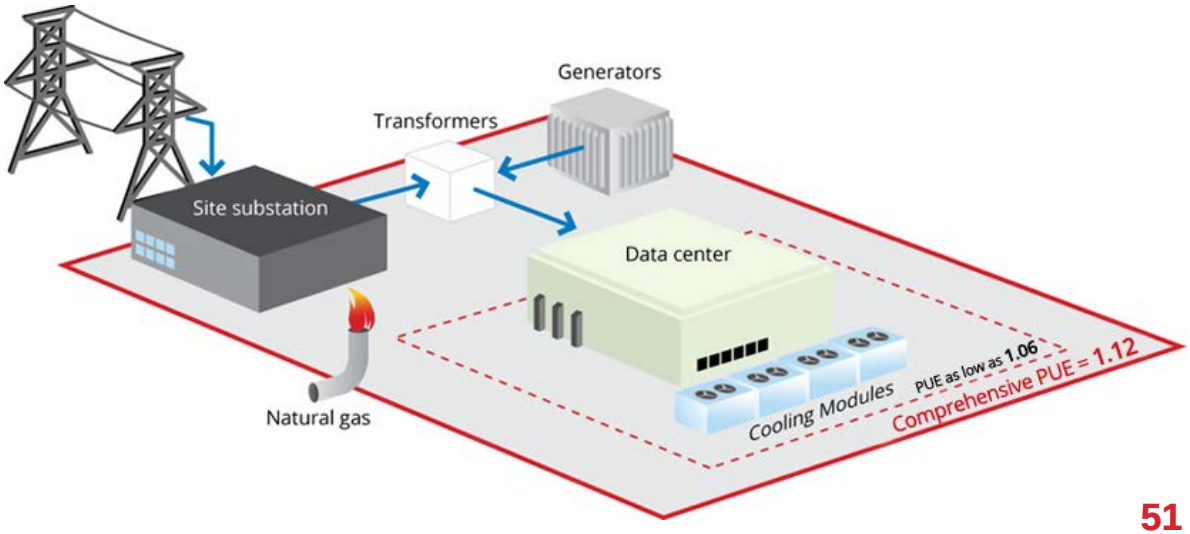
Data center are very demanding

they use as much energy as a little town

In a near future, consumption will have to be controlled

select data to be stored

decide on the duration



Security

Security is certainly the main brake to the development of pervasive computing

- protect privacy
- protect data
- protect against denial of service
- protect against malicious actions on devices

The issue is made worst by distribution/heterogeneity/dynamicity

Scalability

Success is an option!

Software must be able to rapidly scale

at the gateway level (more apps, more gateways)

at the transport level (more data)

at the cloud level (more resources needed, more analytics, etc.)

Structure of this lecture

Introduction

Context-awareness and adaptation

Distribution

Heterogeneity

Dynamicity

More non functional properties

Conclusion

Software challenges

Impacts on software are tremendous.

What is needed

- new architectures
- new development paradigms
- new platforms and middleware
- new technologies
- new interaction means, ...

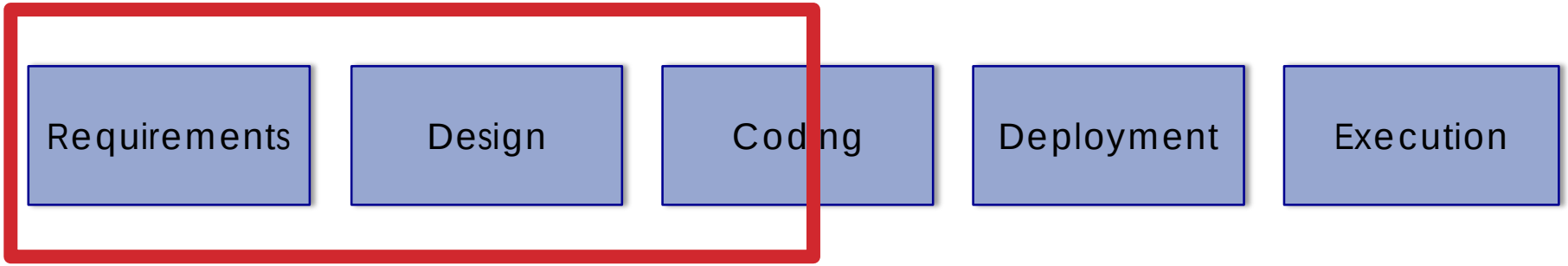
Software engineering is not ready

- recent focus was on design activities

Software engineering

Software engineering defines repeatable processes and technologies for supporting software development and maintenance activities.

Unfortunately, it has been mainly thought for deterministic environments and focuses on early phases.



Defining Determinism, Take 1

Definition 1:

A program is deterministic if and only if its outputs are 100% defined by its inputs

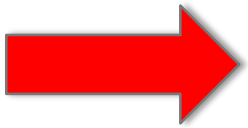
Research agenda

Most activities have to be rethought and moved to runtime

- requirements must be explicitly considered at runtime
- application goals must be known at runtime
- applications have to be opportunist and adaptable
- testing must be partially made at runtime
- deployment and integration must be continuous

Research agenda

Pervasive platform is a highly studied domain



More to come !