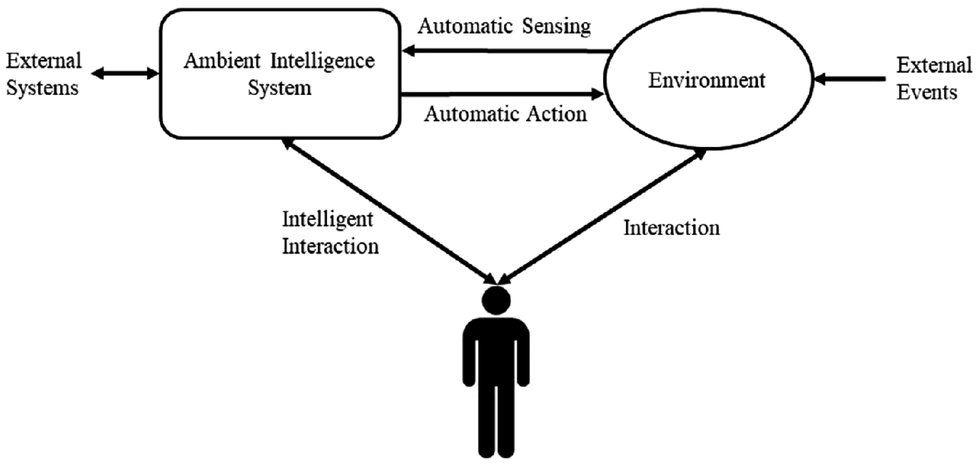
Ambient **In**telligence

Course by A. Sgorbissa

Notes by D. Lanza, A. Grillo & A. Ghiotto

EMARO+ M2

2019-2020



Outline

Ambient Intelligence presupposes the presence of a certain number of devices (sensors and/or actuators) that are integrated into the environment and capable of communicating with each other, in order to support people in carrying out their everyday activities. The course analyzes how to design Ambient Intelligence applications, presenting methodological and technological solutions.

The goal of the course is to enable students to understand the Ambient Intelligence computing paradigm, which envisions a world where people (and possibly robots) are surrounded by intelligent sensors/actuators and interfaces embedded in the everyday objects around them.

At the end of the course, the student will be able to:

- Understand the characteristics and problems of Ambient Intelligence applications, and their relationship with other areas including IoT, IA, and Robotics;
- Understand the methodologies and technological tools for the design of Ambient Intelligence applications;
- Extend the acquired knowledge to understand how to use new methodologies and tools that have not been treated in the course;
- Apply the methodologies and tools for solving problems, in particular for the design of Ambient Intelligence applications

Contact Antonio Sgorbissa (antonio.sgorbissa@unige.it) – www.laboratorium.dist.unige.it/ sgorbiss

Contents The teaching program will cover the following topics:

- Ambient Intelligence
    - – Base principles;
- Localization of people and devices
    - – Sensors for localization;
    - – Geometric approaches;
    - – Topological approaches;
    - – Probabilistic location: Particle Filter;
- Knowledge representation
    - - Descriptive logic;
    - - Ontologies: OWL and Protégé;
    - - SWRL rules;
    - - Bayesian Networks and Hidden Markov Models
- Context and Context Awareness
    - - The Context Toolkit;
    - - Context Awareness with ontologies;
    - - Context Awareness with Bayesian Networks
- Middleware for Ambient Intelligence
- Execution of plans: AgentSpeak and Jason

References (...)

Exam The exam provides that the student is able to deal with the design of an Ambient Intelligence application with given characteristics using the theoretical bases and programming tools learned during the lessons and exercises. The final grade results from the composition of "continuous assessment" vote (30%) and exam grade (70%) .

Contents

Chapter 1

Ontologies & Description Logics

1.1 Introduction

Similarly to robotics, people were waiting for Ambient Intelligence before technology was currently available, but the same does not happen for all technology (were people expecting mobile phones? GPS navigators?)

What is the main goal of Ambient Intelligent technologies and in what they differs from robotics? It is summarized by this quote by *The Computer for the 21st Century*'s writer Mark Weiser:

> "The most profound technologies are those that disappear."

An example could be **writing**: it is ubiquitous, we don't realize that we are using it and it's pervasive and useful (not only, it's necessary). In fact not only do books, magazines and newspapers convey written information, but so do street signs, billboards, shop signs and even graffiti. The constant background presence of these products of "**literacy technology**" does not require active attention, but the information to be conveyed is ready for use at a glance.

Silicon-based information technology, in contrast, is far from having become part of the environment. Computers are approachable only through complex specialized skills that has nothing to do with the tasks for which people actually use computers (the state of the art is perhaps **analogous** to the period when **scribes** had to know as much about making ink or baking clay as they did about writing).

> " ... we are trying to conceive a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background."

1.2 Prolegomena to ontologies

Ontologies are used to **represent knowledge and to reason** upon such knowledge. An ontology is a formal conceptualisation of the world which specifies a set of constraints and declares what should necessarily hold in any possible world.

Given an ontology, a **legal world description** is a possible world satisfying the constraints.

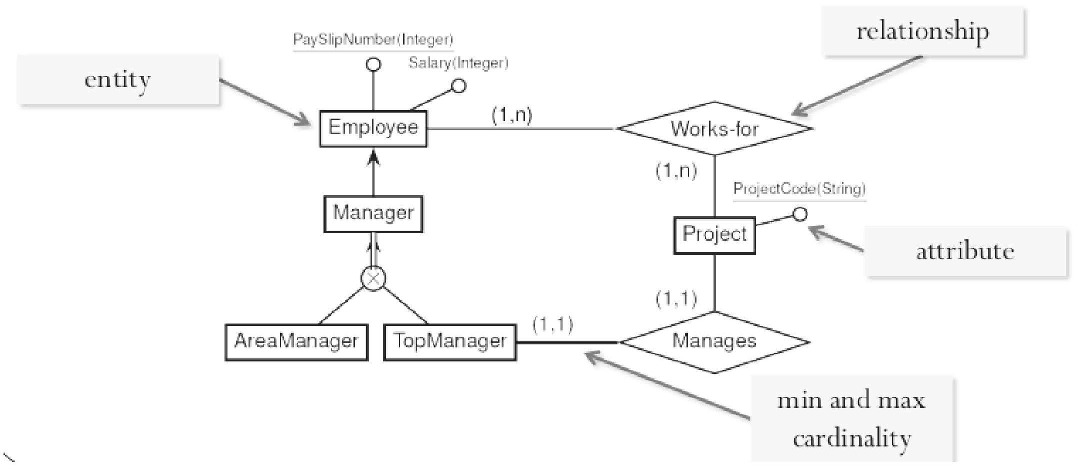
An **ontology language** usually introduces concepts (aka classes, entities),properties of concepts (aka slots, attributes, roles), associations (aka relationships) and additional constraints. Ontology languages may be simple (e.g., having only concepts), frame-based(having only concepts and properties), or logic-based, and can be expressed by means of **diagrams**.[1]

1.2.1 ER Diagram

The Entity-relationship diagram is used for modeling databases:

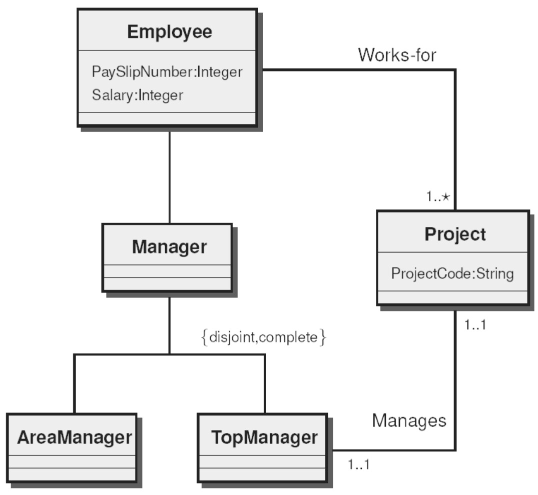
[1] The Entity-Relationship conceptual data model and UML Class Diagrams can be considered as ontology languages, even if the term "ontology" in Computer Science has been introduced more recently.

- Concepts are represented as rectangles;
- Relationships between concepts are represented as diamonds, possibly specifying the cardinality of the involved concepts (e.g., each Top Manager manages exactly one Project);
- Generalizations are represented with arrows (e.g., a Manager is also an Employee);
- Concepts and relationships can have attributes (basic data types), which are inherited from parents to children (e.g., each Employee has a Salary; a Top Manager is an Employee, and therefore it has a Salary).



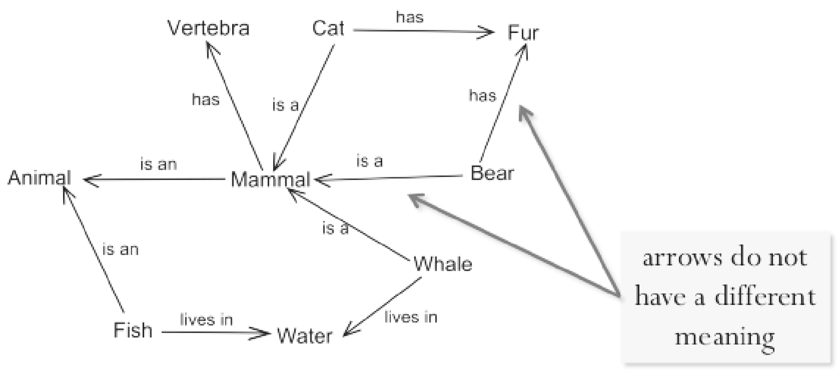
1.2.2 UML diagram

UML class diagram, used to model classes in object oriented programming, can be used in a similar way (see the correspondences of this figure with the previous one)



1.2.3 Older approaches

Semantic networks A semantic network is a network which represents semantic relations among concepts. Invented for computers by Richard H. Richens of the Cambridge Language Research Unit in 1956 as an "interlingua" for machine translation of natural languages. The problem is that arcs do not make any difference between generalizations and relationships (or roles): we have a label that says that a "Cat" is a "Mammal", but this generalization is not expressed through a proper linguistic construct, that is, it is not possible to infer that a "Cat" is an "Animal" from the fact that a "Cat" is a "Mammal" and a "Mammal" is an "Animal".

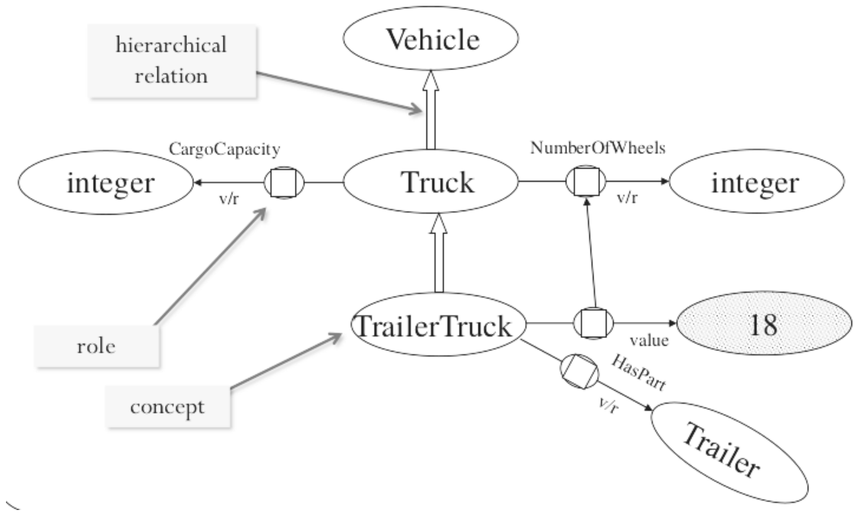


KL-ONE KL-ONE is a well known knowledge representation system in the tradition of semantic networks. The system is an attempt to overcome semantic indistinctness in semantic network representations and to explicitly represent conceptual information as a structured **inheritance network**. Basic elements in KL-ONE are called **concepts**. Concepts form **hierarchies** using subsume-relations; in the KL-ONE terminology a super class is said to subsume its subclasses. Multiple inheritance is allowed. All concepts, except the top concept “Thing”, must have at least one super class. The slot-concept is called **roles** and the values of the roles are **role-fillers**. There are several different types of roles to be used in different situations.

KL-ONE allows for making complex descriptions by composing concepts: i.e., “A vehicle with cargo capacity 10 tons, a trailer and 18 wheels”.

KL-ONE provides a graphical representation for these structured descriptions based on a large set of knowledge-structuring primitives, and provides tools for reasoning, for instance to check if a concept/description subsumes or not another concept/description or to find all concepts which subsumes another concept/description.

Example: consider a vehicle with cargo capacity 10tons, a trailer and 18 wheels. What is it?



1.3 Description Logics

KL-one is an implementation of Description Logics, a formalism to introduce logic- based semantics into knowledge-based representation systems. Description Logics can be interpreted as a **subset of first order predicate logics** which guarantees efficient (**polynomial time**) **reasoning**. More recent languages (OWL-DL and different OWL 2 profiles) still relies on DL, hence it is fundamental to study how it works.

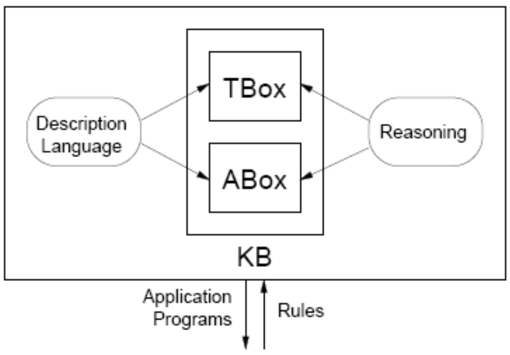
Description Logics is a logic (formal) language for describing knowledge (OWL and OWL2 are an implementation of DL). Description Logics is based on the concept of **concepts** (aka classes, entities) which describe sets, and **individuals** (aka instances) which describe elements of the set.

DL supports **inference patterns** that occur in many applications of intelligent information processing systems, which are also used by humans to structure and understand the world. We are mostly interested in the classification of concepts and instance checking of individuals:

- **Classification of concepts** determines subconcept/superconcept relationships (called subsumption relationships in DL) between the concepts of a given terminology
- **Instance checking of individuals** determines whether a given individual is always an instance of a certain concept (i.e., whether this instance relationship is implied by the description of the individual and the definition of the concept)

!!! The output of reasoning may trigger the application of rules that insert additional facts into the knowledge base. !!!

1.3.1 Knowledge Base



A **knowledge base** (KB) comprises two components, the TBox and the Abox:

- The **TBox** introduces the **terminology**, i.e., the vocabulary of an application domain (intensional knowledge). The vocabulary consists of concepts, which denote sets of individuals, and roles, which denote binary relationships between individuals belonging to the involved concepts.
- The **ABox** contains **assertions** about named individuals in terms of this vocabulary (extensional knowledge).

In addition to atomic concepts and roles (concept and role names), all DL systems allow their users to build **complex descriptions** of concepts and roles. The TBox can be used to assign names to complex descriptions.

As anticipated, a DL system not only stores terminologies and assertions, but also offers services that **reason** about them.

Typical **reasoning tasks for a TBox** are:

- Determine whether a description is **satisfiable** (i.e., non-contradictory)
- Determine if one description is more general than another one, that is, whether the first **subsumes** the second (when iterated over all concepts, it leads to “classification”)

Important **problems for an ABox** are, on the other hand:

- **instance checking**: find out whether the assertions in the ABox entail that a particular individual is an instance of a given concept description
- **knowledge base consistency**: which amounts to verifying whether every concept in the knowledge base admits at least one individual
- **realization**: which finds the most specific concept an individual object is an instance of
- **retrieval**: which finds the individuals in the knowledge base that are instances of a given concept

1.3.2 Concepts, roles and descriptions

An **atomic concept** in the Tbox can be, for example,

$$Person$$

which presumably indicates the "set of all persons" (the semantic of a concept will be formally defined in the following). An **atomic role** could be

$$hasChild$$

which can be used to introduce the concept of "somebody that has a child"

> ⇒ by using the concept *Person* and the role *hasChild* above, it is possible to describe "somebody who is a person and has a child who is a person"

How is it possible to build complex descriptions starting from pre-existing concepts and roles? **Elementary descriptions** are atomic concepts and atomic roles, while **complex descriptions** can be built from them inductively with concept constructors using the production rules below:

> In the following notation, we use the letters A and B for atomic concepts, the letter R for atomic roles, and the letters C and D for descriptions (which are concepts as well).
>
> Description languages are distinguished by the linguistic constructors they provide. Concept descriptions in $\mathcal{AL}$ language are formed according to the following syntax rule:

$C, D \rightarrow$	$A\vert$	(atomic concept)
(context-free	$\top\vert$	(universal concept, which subsumes every other concept)
grammar)	$\bot\ \vert$	(bottom concept, which is subsumed by every other concept)
	$\neg A\vert$	(atomic negation)
	$C \cap D$	(intersection)
	$\forall R.C\vert$	(value restriction)
	$\exists R.\top$	(limited existential quantification)

The rules above say, among the others, that a **complex** description can be **iteratively built** as the intersection of two other descriptions, or by properly expressing relationships with other concepts using roles.

The production rules in the previous slide are part of a context-free grammar. In formal language theory, a **context-free grammar** (CFG) is a formal grammar in which every production rule is of the form

$$V \rightarrow w$$

where V is a **single** nonterminal symbol, and w is a string of terminals and/or nonterminals (w can be empty). A formal grammar is considered "context free" when its **production rules** can be applied **regardless of the context** of a nonterminal. It does not matter which symbols the nonterminal V is surrounded by, the single nonterminal V on the left hand side **can always be replaced** by the right hand side.

What we just saw allows to iteratively build, for example, the description of

"Somebody who is a person and has a child"

starting from the atomic concept A corresponding to *Person* and the atomic concept role R corresponding to *hasChild*:

$$C \rightarrow \exists R.\top = \exists hasChild.\top$$

$$D \rightarrow A \cap C = Person \cap \exists hasChild.\top$$

Notice that $\mathcal{AL}$ does not allows to specify that the child itself must be a *Person*, since in $\mathcal{AL}$ the **existential quantifier admits only** to use the **superconcept** $\top$ as a role descriptor (**!!!**). In order to define the concept of "somebody who is a person and has a child that is a person", one needs to apply additional rules for **value restriction**:[2]

$$E \to \forall R.A = \forall hasChild.Person$$

$$F \to D \cap E = Person \cap \exists hasChild.\top \cap \forall hasChild.Person$$

1.3.3 DL languages

There are many varieties of Description Logic and there is an informal naming convention, roughly describing the operators allowed. The expressivity is encoded in the label for a logic starting with one of the following basic logics:

- $\mathcal{AL}$ ($\mathcal{A}$ttributive $\mathcal{L}$anguage) is the base language which allows:
  - Atomic negation (negation of concept names that don't appear on the left hand side of axioms)
  - Concept intersection
  - Universal restrictions
  - Limited existential quantification
- $\mathcal{FL}$ ($\mathcal{F}$rame based description $\mathcal{L}$anguage) allows:
  - Concept intersection
  - Universal restrictions
  - Limited existential quantification
  - Role restriction
- $\mathcal{EL}$ allows:
  - Concept intersection
  - Existential restrictions (of full existential quantification)

Basic logics can be followed by any of the following extensions:

$\mathcal{F}$ – Functional properties

$\mathcal{E}$ – Full existential qualification (existential restrictions that have fillers other than `owl:Thing`)

$\mathcal{U}$ – Concept union

$\mathcal{C}$ – Complex concept negation

$\mathcal{H}$ – Role hierarchy (subproperties - `rdfs:subPropertyOf`)

$\mathcal{R}$ – Limited complex role inclusion axioms, reflexivity and irreflexivity, role disjointness

$\mathcal{O}$ – Nominals (enumerated classes of object value restrictions - `owl:oneOf, owl:hasValue`)

$\mathcal{I}$ – Inverse properties

$\mathcal{N}$ – Cardinality restrictions (`owl:cardinality, owl:maxCardinality`)

$\mathcal{Q}$ – Qualified cardinality restrictions (available in OWL 2, cardinality restrictions that have fillers other than `owl:Thing`)

$^{(\mathcal{D})}$ – Use of datatype properties, data values or data types

As an example, $\mathcal{ALC}$ is a centrally important description logic from which comparisons with other varieties can be made. $\mathcal{ALC}$ is simply $\mathcal{AL}$ with complement of any concept allowed, not just atomic concepts.

The Protégé ontology editor supports $\mathcal{SHOIN}^{(\mathcal{D})}$ (where $\mathcal{S}$ is an abbreviation for $\mathcal{ALC}$).

Three major biomedical informatics terminology bases, $\mathcal{SNOMEDCT}$, $\mathcal{GALEN}$, and $\mathcal{GO}$, are expressible in $\mathcal{EL}$ (with additional role properties)

[2] Notice that other (more expressive) allow for a full existential quantifier. In such case (not supported by $\mathcal{AL}$) it is possible to express:

$$G \to Person \cap \exists hasChild.Person$$

Notice that the resulting concept G is different from the previous F. In fact, the concept G does not exclude a hypothetical person that has a child that is a person, and an additional child that is a dog. In fact, there is not a universal value restriction on the filler of the role *hasChild*. On the opposite, the concept F requires that all the fillers of the role *hasChild* are subconcepts of *Person*.

OWL 2 provides the expressiveness of $\mathcal{SHROIQ}^{(\mathcal{D})}$, OWL-DL is based on $\mathcal{SHOIN}^{(\mathcal{D})}$ and for OWL-Lite it is $\mathcal{SHIF}^{(\mathcal{D})}$.

1.3.4 Language expressivity vs. complexity

Investigating the **computational complexity** of a given DL with decidable inference problems is an important issue. **Decidability and complexity** of the inference problems depend on the **expressive power** of the DL at hand.

- On the one hand, very expressive DLs are likely to have inference problems of high complexity, or they may even be undecidable.
- On the other hand, very weak DLs (with efficient reasoning procedures) may not be sufficiently expressive to represent the important concepts of a given application.

The **Barber paradox** supposes a barber who shaves all men if and only if they do not shave themselves (Russel Paradox). Does the barber shave himself? In First order logics:

$$(\exists x)\Big(man(x) \wedge (\forall y)\Big(man(y) \rightarrow \Big(shaves(x,y) \leftrightarrow \neg shaves(y,y)\Big)\Big)\Big)$$

This sentence is **unsatisfiable** (a contradiction) because of the universal quantifier: by assigning the value x to y, this yields the contradiction:

$$\Big(shaves(x,x) \leftrightarrow \neg shaves(x,x)\Big)$$

This kind of contradiction emerges from the expressivenes of the language, e.g., the concept of "the set of all normal sets".[3] It can be a good idea to **limit expressiveness** to avoid these problems.

1.3.5 Some DL examples

Let us see other examples, by adding the atomic concept

$$Female$$

The persons that are female can be described as

$$Person \sqcap Female$$

The people that are not female can be described as

$$Person \sqcap \neg Female$$

The people whose children (if any) are all female (it does not mean that the person has children...) can be described as

$$Person \sqcap \forall hasChild.Female$$

The people that have children, and whose children are all female can be described as

$$Person \sqcap \forall hasChild.Female \sqcap \exists hasChild.\top$$

[3] If you are interested to know about Russel's paradox in its original formulation:
Let start by calling a set "**abnormal**" if it is a member of itself, and "**normal**" otherwise. For example, take the set of all squares. That set is not itself a square, and therefore is not a member of the set of all squares. So it is "normal". On the other hand, if we take the complementary set that contains all non-squares, that set is itself not a square and so should be one of its own members. It is abnormal.
Now we consider **the set of all normal sets**, R. Attempting to determine whether R is normal or abnormal is impossible: if R were a normal set, it would be contained in the set of normal sets (itself), and therefore be abnormal; if R were an abnormal set, it would not be contained in the set of normal sets (itself), and therefore be normal. This leads to the conclusion that R is **both normal and abnormal**.

The people that do not have children

$$Person \cap \forall hasChild. \perp$$

As anticipated, additional constructors can be available (not in $\mathcal{AL}$).[4]

1.3.6 Formal semantics & interpretations

With DL it is possible to define a formal semantics. To this purpose, we consider the **interpretation** I, which consists of a non-empty set Δ^I (the **domain** of the interpretation) and an interpretation function which assigns

- to every atomic concept A a set $A^I \in \Delta^I$
- to every atomic role R a binary relation $R^I \in \Delta^I \times \Delta^I$

In the examples in the previous section we had the atomic concepts *Female* and *Person*, and the atomic role *hasChild*. A possible interpretation could be the following:

$$\begin{aligned}
\Delta^I &= \{Paul, Tom, Julia, Laura, Mariah, Andrea\} \\
Person^I &= \{Paul, Tom, Julia, Laura, Mariah, Andrea\} \\
Female^I &= \{Julia, Laura, Mariah\} \\
hasChild^I &= \{(Tom, Julia), (Julia, Laura), (Mariah, Andrea), (Mariah, Tom), (Andrea, Paul)\}
\end{aligned}$$

Notice that the interpretation has the purpose of "anchoring" symbols (concepts and then individuals) to their meaning in the real world: the **domain** of the interpretation is meant to **refer to entities in the real world**. For this reason, the following interpretation could be equally valid, by assuming an unambiguous interpretation to the sentences among braces:

$$\begin{aligned}
\Delta^I &= \{\text{All Italian citizens}\} \\
Person^I &= \{\text{All Italian citizens}\} \\
Female^I &= \{\text{All Italian citizens whose gender is female}\} \\
hasChild^I &= \{\text{All couples of citizens such that the } 1^{st} \text{ element of the couple is parent of the } 2^{nd} \text{ one}\}
\end{aligned}$$

A formal semantics is necessary because the names of concepts and roles are mere symbols: even if, up to now, we have used names whose meaning (i.e., its connection to entities in the real world) is "intuitive", this is not required. In fact, we could define

$$\begin{aligned}
\Delta^I &= \{\text{All Italian citizens}\} \\
X11^I &= \{\text{All Italian citizens}\} \\
Qurtz^I &= \{\text{All Italian citizens whose gender is female}\} \\
Cupurufucuski^I &= \{\text{All couples of cit. s.t. the } 1^{st} \text{ element of the couple is parent of the } 2^{nd} \text{ one}\}
\end{aligned}$$

In this case a formal semantics is obviously required to give an interpretation to symbols.

Starting from the interpretation of atomic concepts and roles, we obviously want that the interpretation of the two descriptions between parentheses is identical:

$$(Person \cap \forall hasChild.Female)^I = (X11 \cap \forall Cupurufucuski.Qurtz)^I$$

[4] For example, cardinality restrictions, to describe "a person that has more than 2 female children" as

$$Person \cap \geq 3 hasChild.Female$$

Negation of non-atomic concepts, to describe "something that is not a person with children" as

$$\neg(Person \cap \exists hasChild.Person)$$

Union of concepts (a person that has a child or a dog)

$$Person \cap (\exists hasChild.Person \cup \exists hasPet.Dog)$$

> Remember that we are manipulating symbols, but – ultimately – we want to make statements about entities in the real world!

After defining the interpretation of atomic concepts, it is necessary to define the **interpretation of the concept constructors** that are used to build complex descriptions. For example, in the case of $\mathcal{AL}$:

$$
\begin{array}{rcl}
\top^I & = & \Delta^I \\
\bot^I & = & \emptyset \\
(\neg A)^I & = & \Delta^I \setminus A^I \\
(C \cap D)^I & = & C^I \cap D^I \\
(\forall R.C)^I & = & \{a \in \Delta^I \mid \forall b \quad (a,b) \in R^I \rightarrow b \in C^I\} \\
(\exists R.\top)^I & = & \{a \in \Delta^I \mid \exists b \quad (a,b) \in R^I\}
\end{array}
$$

- Δ^I={Paul, Tom, Julia, Laura, Mariah, Andrea}
- Person^I={Paul, Tom, Julia, Laura, Mariah, Andrea}
- Female^I ={Julia, Laura, Mariah}
- hasChild^I ={(Tom, Julia), (Julia, Laura), (Mariah, Andrea), (Mariah, Tom), (Andrea, Paul)}

For example:

(Person∩Female)^I = (Person)^I∩ (Female)^I ={Julia, Laura, Mariah}
(Person∩hasChild)^I ={Tom, Julia, Mariah, Andrea}
(Person∩∃hasChild.⊤∩∀hasChild.Female)^I ={Tom, Julia}
(Person∩∀hasChild.Female)^I ={Paul,Tom, Julia, Laura} ⟵ This set includes those that do not have children!

We say that two concepts are **equivalent** when:

$$C \equiv D \iff C^I = D^I \quad \forall \text{ interpretations } I$$

For example, let us introduce a new concept

$$Student$$

with the following interpretation:

$$Student^I = \{Tom, Laura\}$$

It can be verified that the following concepts are equivalent:

$$Person \cap \exists hasChild.Female \cap \exists hasChild.Student = Person \cap \exists hasChild(Female \cap Student)$$

Note: it is obviously possible that two concepts have the same interpretation

$$C^I = D^I$$

without being equivalent! In the case

$$Woman^I = Teacher^I$$

but for a different interpretation it may not hold:

$$
\begin{aligned}
\Delta^I &= \{\text{All Italian citizens}\} \\
Woman^I &= \{Paola, Enrica\} \\
Teacher^I &= \{Paola, Enrica\}
\end{aligned}
$$

1.3.7 Terminological axioms

It is now possible to introduce **terminological axioms**, which make statements about how concepts or roles are related to each other. There are two kind of axioms:

- **Inclusions:** $C \subseteq D \qquad (R \subseteq S)$
- **Equalities:** $C \equiv D \qquad (R \equiv S)$

(C, D are concepts and R, S are roles)

Definitions are specific axioms (either inclusions or equalities) by which we can introduce new atomic concepts as abbreviations or names for complex descriptions:

- Equality definition: $NewName \equiv Complex\ Description$

  This is useful when we want to give a strict, detailed definition, for example:

$$Parent \equiv (Person \cap \exists hasChild.T)$$

- Inclusion definition: $NewName \subseteq Complex\ Description$

  This can be used, for example, when it is not possible or convenient to give a detailed definition, for example:

$$Father \subseteq (Person \cap \exists hasChild.T)$$

Considering the last example, we can see how it would be possible to give a strict definition by assuming the atomic concept $Male$:

$$Father \equiv (Person \cap Male \cap \exists hasChild.T)$$

It is important to note as well that definitions can be iteratively used for creating new definitions:

$$Father \equiv (Male \cap Parent)$$

1.3.8 Interpretation satisfiability

Let's introduce now the concept of interpretation **satisfiability**:

- We say that an interpretation I satisfies an inclusion $C \subseteq D$ if $C^I \subseteq D^I$.
- We say that an interpretation I satisfies an equality $C \equiv D$ if $C^I = D^I$.

For example, consider this additional statement which integrate the previously seen interpretation:

$$Parent^I = \{Tom, Julia, Mariah, Andrea\}$$

The resulting interpretation satisfies the equality

$$Parent \equiv (Person \cap \exists hasChild.\top)$$

because it can be verified that it holds

$$Parent^I = (Person \cap \exists hasChild.\top)^I$$

If $\mathcal{T}$ is a **set of axioms**, then I satisfies $\mathcal{T}$ if and only if I satisfies each element of $\mathcal{T}$ (we say that I is a **model** for this set of axioms). Hence, to summarize:

$$\boxed{\begin{array}{c} I \text{ satisfies/is a model for } \mathcal{T} \Leftrightarrow I \text{ satisfies } a, \ \forall \text{ axiom } \ a \in \mathcal{T} \\[1em] I \text{ satisfies } C \subseteq D \Leftrightarrow C^I \subseteq D^I \qquad I \text{ satisfies } C \equiv D \Leftrightarrow C^I = D^I \end{array}}$$

Two axioms or two sets of axioms are equivalent if they have the same models.

The satisfiability is a very important concept, because it **checks if your set of axioms $\mathcal{T}$ is consistent** or if for some axioms there is not an available interpretation!

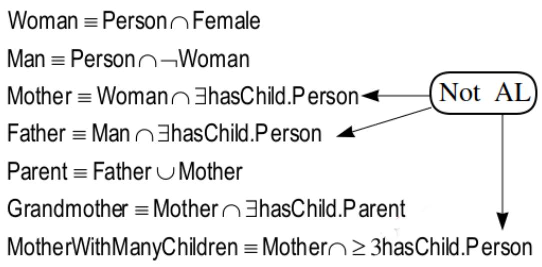
1.3.9 TBox (terminology)

A set of definitions should be unequivocal: we call a finite set of definitions $\mathcal{T}$ a terminology or **TBox** if **no symbolic name is defined more than once**, that is:

For every atomic concept A there is at most one axiom in T whose left-hand side is A

Also, **definitions must be acyclic** in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them.

Example of a TBox:



Woman $\equiv$ Person $\cap$ Female

Man $\equiv$ Person $\cap \neg$Woman

Mother $\equiv$ Woman $\cap \exists$hasChild.Person

Father $\equiv$ Man $\cap \exists$hasChild.Person

Parent $\equiv$ Father $\cup$ Mother

Grandmother $\equiv$ Mother $\cap \exists$hasChild.Parent

MotherWithManyChildren $\equiv$ Mother $\cap \geq 3$hasChild.Person

Named symbols are often called **defined concepts** and base symbols **primitive concepts**: we expect that the terminology ultimately defines the named symbols in terms of the base symbols.

1.3.10 ABox (world description)

The second component of a knowledge base, in addition to the terminology or TBox, is the world description or **ABox**. In the ABox, one describes a **specific state** of affairs of an application domain in terms of **concepts and roles**.

!! Some of the concept and role atoms in the ABox may be defined names of the Tbox**!!**

In the ABox, one introduces **individuals**, by giving them names, and one asserts properties of these individuals. In particular, after introducing the named individuals a and b, we can **give a semantics to the Abox** by extending interpretations to individual names:

From now on an interpretation I:
– not only maps atomic concepts and roles to sets and relations
– but in addition maps each individual name a to an element $a^I \in \Delta^I$

This is done by making the **unique name assumption** (UNA), that is:

$$a, b \text{ distinct names } \Rightarrow a^I \neq b^I$$

Using concepts C and roles R, and by referring to the named individuals a, b, c one can make assertions of the following two kinds in an Abox:

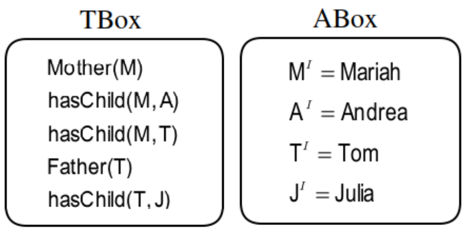
- **Concept assertions** $C(a)$

  One states that the interpretation of a belongs to the interpretation of $C(a^I \in C^I)$

- **Role assertions** $R(b; c)$

  One states that c is a filler of the role R for b, i.e., that $(b^I, c^I) \in R^I$

The following example shows concepts and role assertions using concepts and roles in the Tbox (left) and a possible interpretation of the corresponding individuals (right). Notice that we used letters A, B, C, D, E to name individuals, instead of referring to the names $Paul, Tom, Julia, Laura, Mariah, Andrea$ to avoid confusion between individuals in the ABox (that, once again, are mere symbols) and their interpretation (that refer to entities in the real worlds):



Given an interpretation I we have:

$$[I \text{ satisfies } C(a)] \leftrightarrow [a^I \in C^I]$$
$$[I \text{ satisfies } R(a,b)] \leftrightarrow [(a^I, b^I) \in R^I]$$
$$[I \text{ is a model for/satisfies the ABox } \mathcal{A}] \leftrightarrow [I \text{ satisfies each assertion in } \mathcal{A}]$$
$$[I \text{ is a model for } \mathcal{A} \text{ w.r.t. the TBox } \mathcal{T}] \leftrightarrow [I \text{ is a model for } \mathcal{A} \wedge I \text{ is a model for } \mathcal{T}]$$

1.3.11 Reasoning with DL

A knowledge representation system based on DLs is able to perform specific kinds of reasoning. As said before, the purpose of a knowledge representation system goes beyond storing concept definitions and assertions.

A **KB** (TBox and ABox) **has a semantics** that makes it **equivalent** to a set of axioms in **first-order** predicate **logic**. Thus, like any other set of axioms, it contains implicit knowledge that can be made explicit through inferences. For example, from the previous TBox and ABox one can conclude that the individual M is an instance of the concept *Grandmother*, although this knowledge is not explicitly stated as an assertion.

◦ **Reasoning in the TBox** :

- **Satisfiability**: A concept C is satisfiable with respect to $\mathcal{T}$ if there exists a model I of $\mathcal{T}$ such that C^I is nonempty. In this case we also say that

$$I \text{ (model of } \mathcal{T}) \text{ is a model of } C$$

- **Subsumption**: A concept C is subsumed by a concept D with respect to $\mathcal{T}$ if $C^I \subseteq D^I$ for every model I of $\mathcal{T}$. In this case we write

$$\mathcal{T} \models C \subseteq D$$

- **Equivalence**: Two concepts C and D are equivalent with respect to $\mathcal{T}$ if $C^I = D^I$ for every model I of $\mathcal{T}$. In this case we write

$$\mathcal{T} \models C \equiv D$$

Satisfiability is important, for example, to check if the Tbox is consistent. For example, assume that the TBox includes the following definitions:

$$\text{Woman} \equiv \text{Person} \cap \text{Female}$$
$$\text{Man} \equiv \text{Person} \cap \neg\text{Woman}$$
$$\text{Mother} \equiv \text{Woman} \cap \exists\text{hasChild.Person}$$
$$\text{Father} \equiv \text{Man} \cap \exists\text{hasChild.Person}$$
$$\boxed{\text{BetelgeuseAlien} \equiv (\text{Father} \cup \text{Mother}) \cap \neg\exists\text{hasChild.Person}}$$

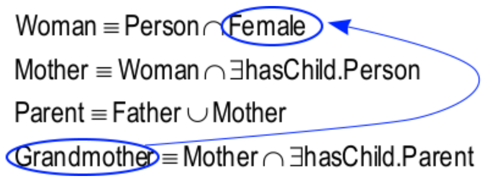
it is never possible
to satisfy this TBox!!!

Traditionally, the **basic reasoning** mechanism provided by DL systems **checks the subsumption** of concepts. It can be shown that this, in fact, is **sufficient** to implement also the **other inferences**.

For example, can we say the following?

$$\textit{Female} \text{ subsumes } \textit{Grandmother} \quad \text{or} \quad T \models \textit{Grandmother} \subseteq \textit{Female}$$

It is easy to check this for every model I of $\mathcal{T}$:



Woman ≡ Person ∩ Female
Mother ≡ Woman ∩ ∃hasChild.Person
Parent ≡ Father ∪ Mother
Grandmother ≡ Mother ∩ ∃hasChild.Parent

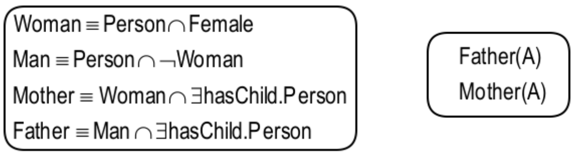
1. To satisfy the definition
$$Grandmother \equiv Woman \cap \exists hasChild.Parent$$
the concept on the left and the concept on the right must have the same interpretation.
2. The interpretation of
$$Woman \cap \exists hasChild.Parent$$
refers to a subset of the interpretation of $Woman$.
3. To satisfy the definition
$$Woman \equiv Person \cap Female$$
the concept on the left and the concept on the right must have the same interpretation.
4. The interpretation of
$$Person \cap Female$$
refers to a subset of the interpretation of $Female$.

⇒ Hence, $Female$ subsumes $Grandmother$ or $T \models Grandmother \subseteq Female$

◦ **Reasoning in the ABox:**

- **Consistency checking**: An ABox $\mathcal{A}$ is consistent with respect to a TBox $\mathcal{T}$, if there is an interpretation that is a model of both $\mathcal{A}$ and $\mathcal{T}$. We simply say that
$$\mathcal{A} \text{ is consistent } \leftrightarrow \mathcal{A} \text{ is consistent w.r.t. the empty TBox}$$
- **Instance checking**: We say that an assertion a is **entailed** by $\mathcal{A}$ with respect to a TBox $\mathcal{T}$ if every model of $\mathcal{A}$ and $\mathcal{T}$ also satisfies a:
$$\mathcal{A} \models a \text{ w.r.t. } \mathcal{T} \leftrightarrow \text{ every model of } \mathcal{A} \text{ and } \mathcal{T} \text{ also satisfies } a$$

Consistency checking is very important. In the following example the ABox on the right is not consistent with respect to the TBox on the left (however, it could be consistent with respect to a different Tbox... remember that $Father$ and $Mother$ are only symbols):

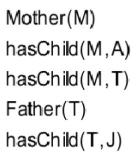


Woman ≡ Person ∩ Female
Man ≡ Person ∩ ¬Woman
Mother ≡ Woman ∩ ∃hasChild.Person
Father ≡ Man ∩ ∃hasChild.Person

Father(A)
Mother(A)

For example, can we say the following?

The individual $M \in \mathcal{A}$ (below) is an instance of $Grandmother$ w.r.t. $\mathcal{T}$ or $A \models Grandmother(M)$



Mother(M)
hasChild(M,A)
hasChild(M,T)
Father(T)
hasChild(T,J)

It is easy to check that, for every model of $\mathcal{A}$ and $\mathcal{T}$:

1. It must necessarily hold
$$M^I \in Mother^I \qquad (M^I, T^I) \in hasChild^I \qquad T^I \in Father^I$$
2. Since the concept *Father* is subsumed by the concept *Parent* in the Tbox, it must necessarily hold
$$T^I \in Parent^I \qquad \text{and then } M^I \in (hasChild.Parent)^I$$
3. Since it must necessarily hold
$$Grandmother^I \equiv (Mother \sqcap \exists hasChild.Parent)^I \qquad M^I \in Mother^I \qquad M^I \in (hasChild.Parent)^I$$

$\Rightarrow$ This finally requires that
$$M^I \in Grandmother^I$$

1.4 Semantic Web Technologies

Nowadays, ontologies play a fundamental role in the Semantic Web. **Semantic Web** is a group of methods and technologies allowing machines to understand the meaning - or "semantics" - of information on the World Wide Web. The term was coined by World Wide Web Consortium (W3C).

According to the original vision, the availability of machine-readable metadata would enable automated agents and other software to access the Web more intelligently. Related technologies include:

- the **Resource Description Framework** (RDF)
- a variety of **data interchange formats** (e.g. RDF/XML, N3, Turtle, N-Triples)
- and **notations** such as RDF Schema (RDFS) and the **Web Ontology Language** (**OWL**), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

Many of the technologies proposed by the W3C already exist and are used in various projects. The Semantic Web as a global vision, however, has remained largely unrealized and its critics have questioned the feasibility of the approach.

> "*I have a dream for the Web* [in which computers] *become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize.*"
>
> – Tim Berners-Lee, 1999

Web pages are written in HTML, which is very limited. It is based on the concept of "tag" and "content":

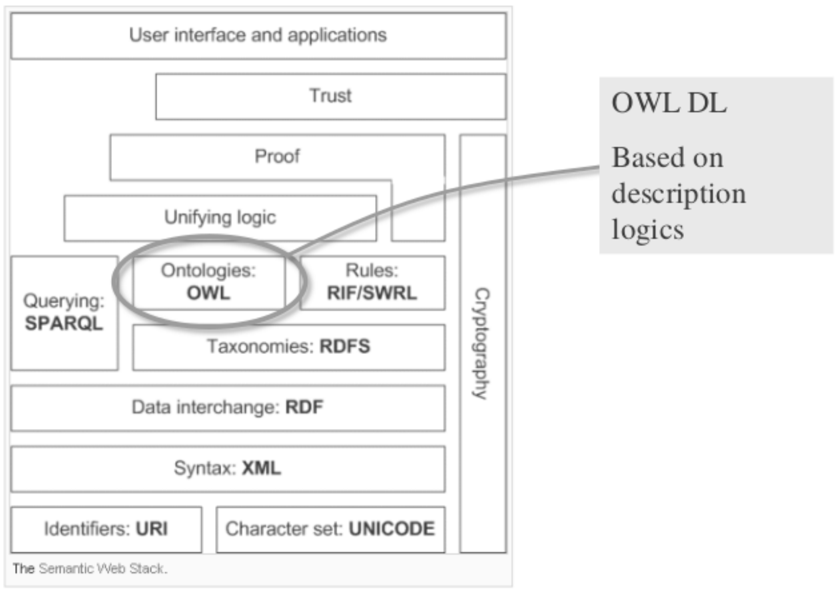
```
<html>
<head>
    <title> favorites / bookmark title goes here </title>
    <meta name="keywords" content="shoes,␣trousers,␣shop">
    <meta name="description" content="My␣shop">
</head>
<body bgcolor="yellow" text="blue">
    <b> My shop </b> <!-- ``bold'' tag -->
    <br> <!-- line break tag -->
    Buy shoes for 56$
    <br>
    Buy trousers for 120$
</body>
</html>
```

Meta tags can be used to add information. However, there is not an easy way to describe the fact that shoes are something that can be bought, and their price is 56$, in a way that is understandable by a computer.

Understanding the semantics of text is a prerequisite for designing autonomous software agents!

1.4.1 The Semantic web stack

Since HTML itself is not sufficient to associate a “semantic” to text, a different approach is proposed, which is based on a **hierarchy of different languages**:



→ Each language is based on the linguistic constructs made available by languages at the underlying level:

- **XML** provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within.

  XML Schema is a language for providing and restricting the structure and content of elements contained within XML documents

  It is possible to define a schema which describes HTML starting from XML.
- **RDF** is a simple language for expressing data models, which refer to objects ("resources") and their relationships.

  An RDF-based model can be represented in XML syntax

  RDFS (RDF Schema) extends RDF and is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized-hierarchies of such properties and classes.
- **OWL** adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

  SPARQL is a protocol and query language for semantic web data sources.

1.4.2 XML

XML gives more freedom that HTML, in that it **separates the structure from the presentation style**. In this sense, it can be used to “structure knowledge”. However, it does not allow to provide a “meaning” to content.

XML Key terminology:

- By definition, an XML document is a string of characters. Almost every legal Unicode character may appear in an XML document.
- In order to structure knowlegde, the characters which make up an XML document are divided into markup and content. Markup and content may be distinguished by the application of simple syntactic rules.

All strings which constitute **markup** either begin with the character "<" and end with a ">", or begin with the character "&" and end with a ";".

Strings of characters which are not markup are **content**.

- **Structured knowledge** can be **processed** and provide the input for an application: the processor (**XML Parser**) analyzes the markup and passes structured information (or part of it) to the application.

More specifically, an XML document can be structured in the following components:

- **Tag**: A markup construct that begins with < and ends with >. There are three types of tags: start-tags, end-tags, empty-element tags. For example:

  start-tag: `<section>`
  end-tag: `</section>`
  empty-element tag: `<line-break/>`

- **Element**: A logical document component which either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag. The element may contain markups, including other elements, which are called child elements. For example, the following element has two child elements:

```
<section>
    <title>This is the title </title>
    <body>This is the body </body>
</section>
```

- **Attribute**: A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag. In the example (below) the element section has two attributes, number and page:

```
<section number="IV" page="234">
```

1.4.3 DTD and XML Schema

In addition to being well-formed (i.e., meaning that it satisfies a list of syntax rules provided in the specification), an XML document may be **valid**. This means that it contains a **reference to a Document Type Definition** (DTD) and that its elements and attributes are **declared in** that DTD and **follow the grammatical rules** for them that the DTD specifies.

A newer schema language than DTD is **XML Schema**, which typically constrain:

- the set of elements that may be used in a document
- which attributes may be applied to them
- the order in which they may appear
- the allowable parent/child relationships

This is an example of XML Schema definition:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/
    ↪ XMLSchema">
    <!-- This is the element that we want to define-->
    <xs:element name="Address">
        <xs:complexType>
            <xs:sequence>
                <!-- These are the child elements it must include -->
                <xs:element name="Recipient" type="xs:string" />
                <xs:element name="House" type="xs:string" />
                <xs:element name="Street" type="xs:string" />
                <xs:element name="Town" type="xs:string" />
                <xs:element name="County" type="xs:string" minOccurs="0" />
                <xs:element name="PostCode" type="xs:string" />
                <xs:element name="Country" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
```

```
                            <!-- These is a restriction on the possible valuest
                                ↪ for Country -->
                            <xs:enumeration value="IN" />
                            <xs:enumeration value="DE" />
                            <xs:enumeration value="ES" />
                            <xs:enumeration value="UK" />
                            <xs:enumeration value="US" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

And this is an example XML document conforming to the previous schema:

```
<?xml version="1.0" encoding="utf-8"?>
<Address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ↪ xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
    <Recipient>Mr. Walter C. Brown</Recipient>
    <House>49</House>
    <Street>Featherstone Street</Street> <!-- This is a valid address -->
    <Town>LONDON</Town>
    <PostCode>EC1Y 8SY</PostCode>
    <Country>UK</Country>
</Address>
```

1.4.4 RDF

Can XML be used to define ontologies? **XML does not associate a shared semantics** to the tags. Suppose that another document uses a similar schema, but maybe with some missing tags, or tags that are in a different languages, or simply synonyms, or are mere sequences of symbols with no evident meaning:

```
<?xml version="1.0" encoding="
    ↪ ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>"Don't␣forget␣me␣this␣
    ↪ weekend!"</body>
</note>
```

```
<?xml version="1.0" encoding="
    ↪ ISO-8859-1"?>
<message>
<subject1>Tove</subject1>
<subject2>Jani</subject2>
<text>"Don't␣forget␣me␣this␣
    ↪ weekend!"</text>
</message>
```

How can an autonomous agent use the available knowledge, i.e., for taking decisions and act? Notice that tags on the right are not a mere translation of tags on the left: how can we know that `subject1` is the sender and `subject2` is the receiver (if they are...)? How can we know that a `message` and a `note` refer to the same concept?

- For example, in the left case, the XML Schema schema can say that

  `<note>` must end with `</note>`, and must have four children
  `<to>` must end with `</to>`
  `<from>` must end with `</from>`
  `<heading>` must end with `</heading>`
  `<body>` must end with `</body>`

- In the right case, the XML Schema schema can say that

  `<message>` must end with `</message>`, and must have four children
  `<subject1>` must end with `</subject1>`
  `<subject2>` must end with `</subject2>`
  `<text>` must end with `</text>`

There is no information to say that the two documents refer to a similar concept! $\longrightarrow$ The **Resource Description Framework** (**RDF**) is a **first step** towards **building ontologies**, i.e., associating a semantics to documents.

What is RDF? RDF is a framework for describing resources on the web, designed to be read and understood by computers but **not designed for being displayed to people**.

RDF is **(usually) written in XML** and it's a part of the W3C's Semantic Web Activity and is a W3C Recommendation (see also: http://www.w3schools.com/rdf/)

RDF – examples of use:

- Describing properties for shopping items, such as price and availability
- Describing time schedules for web events
- Describing information about web pages (content, author, created and modified date)
- Describing content and rating for web pictures
- Describing content for search engines
- Describing electronic libraries

As we said, RDF documents are often written in XML. The XML language used by RDF is called **RDF/XML**. By using XML, RDF information can easily be exchanged between different types of computers using different types of operating systems and application languages.

RDF identifies things using **Web identifiers** (Uniform Resource Identifiers - **URI**s), and describes resources with properties and property values:

- A **Resource** is anything that can have a URI, such as `http://www.w3schools.com/rdf`
- A **Property** is a Resource that has a name, such as `author` or `homepage`
- A **Property Value** is the value of a Property, such as `Jan Egil Refsnes` or `http://www.w3schools.com` (note that a Property Value can be another resource)

Example The following RDF document could describe the resource `http://www.w3schools.com/rdf`. This example comprises two triplets (Resource, Property, Property values) but it is simplified, since it does not contain namespaces (see later):

```
<?xml version="1.0"?>

<rdf:RDF>
    <Description about="http://www.w3schools.com/rdf">
        <author>Jan Egil Refsnes</author>
        <homepage>http://www.w3schools.com</homepage>
    </Description>
</rdf:RDF>
```

- The first one says that `http://www.w3schools.com/rdf` has the `author` that is `Jan Egil Refsnes`
- The second one says that `http://www.w3schools.com/rdf` has the `homepage` that is `http://www.w3schools.com`

Do not confuse RDF (which is conceptually a set of triplets) with its XML implementation!

The combination of a Resource, a Property, and a Property value forms a **Statement** (known as the subject, predicate and object of a Statement). In the previous example:

- the Statement `http://www.w3schools.com/rdf` has the `author` that is `Jan Egil Refsnes`

  (the subject of the statement above is `http://www.w3schools.com/rdf`, the predicate is `author` and the object is `Jan Egil Refsnes`)
- the Statement `http://www.w3schools.com/rdf` has the `homepage` that is `http://www.w3schools.com`

  (the subject of the statement above is `http://www.w3schools.com/rdf`, the predicate is `homepage` and object is `http://www.w3schools.com`)

The RDF data model is **similar** to classic conceptual modeling approaches such as **ER relationship or class diagrams**. RDF it is based upon the idea of making statements about resources (in particular web resources) in the form of **subject-predicate-object expressions**.

These expressions are known as **triples** in RDF terminology:

- For example, one way to represent the notion "The sky has the color blue" in RDF is using the triple composed of:

  a subject denoting "the sky"
  a predicate denoting "has the color"
  and an object denoting "blue"

- A collection of RDF statements intrinsically represents a labeled, directed multi-graph, where

  the subject is the predecessor node
  the object is the successor node
  the predicate is the arc.

Two common **serialization formats** of the **RDF graph** are in use:

1. The first is an **XML format**. This format is often called simply RDF because it was introduced among the other W3C specifications defining RDF. However, it is important to distinguish the XML format from the abstract RDF model itself.
2. In addition to serializing RDF as XML, the W3C introduced **Notation 3** (or N3) as a non-XML serialization of RDF models designed to be easier to write by hand, and in some cases easier to follow.

Skipped slides "Description Logics" from 66 to 73

1.4.5 RDFS

RDF Schema (RDFS) is an extension to RDF. If RDF describes resources with classes, properties, and values (*to* triplets which define a graph that is similar to Richard H. Richens semantic networks of 1956), it still lacks of expressivity, **missing** a way to **define application-specific classes and properties**, as well as **taxonomies**.

RDF Schema is an extension of RDF, and it provides the framework to **describe application-specific classes and properties**. **Classes** in RDF Schema are much like classes in object oriented programming languages, and this allows **resources** to be defined as **instances of classes**, and subclasses of classes.

In the following example the resource "horse" is a subclass of the class "animal":

```xml
<?xml version="1.0"?>

<!-- This resource (the name can be used by other RDF file to refer to the
    ↪ definition introduced here)-->
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.animals.fake/animals#">

<rdf:Description rdf:ID="animal">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="horse">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <!-- Horse is defined as a subclass of animal -->
    <rdfs:subClassOf rdf:resource="#animal"/>
    </rdf:Description>
</rdf:RDF>
```

RDF is metadata (data about data), used to describe information resources. The **Dublin Core Metadata Initiative** (DCMI) has created some predefined properties for describing documents. The Dublin Core is a set of predefined properties for describing documents. The first Dublin Core properties were defined at the Metadata Workshop in Dublin, Ohio in 1995 and is currently maintained by the

Dublin Core Metadata Initiative. This and other initiatives allow for a common vocabulary shared among different agents operating in the network!

1.5 OWL

The **Web Ontology Language** (**OWL**)[5] is built on top of RDF. It was designed for processing information on the web and to be interpreted by computers (not for being read by people). It is written in XML and has three sublanguages: OWL lite, OWL DL and OWL full.

For the web, ontology is about the exact description of web information and relationships between web information. OWL is a part of the "**Semantic Web Vision**" - a future where:
– Web information has exact meaning
– Web information can be processed by computers
– Computers can integrate information from the web

OWL was designed for **Processing Information**, in order to provide a common way to process the content of web information (instead of displaying it).

OWL is **different from RDF**. Even though they are much of the same thing, OWL is a stronger language with greater machine interpretability than RDF, and it comes with a larger vocabulary and stronger syntax than RDF.

OWL is **written in XML**, and by using XML, OWL information can easily be exchanged between different types of computers using different types of operating system and application languages.

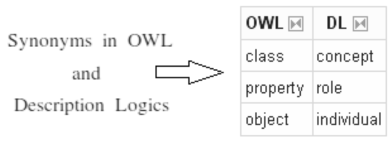
OWL is a **Web Standard** and became a W3C (World Wide Web Consortium) recommendation in February 2004. A W3C Recommendation is understood by the industry and the web community as a web standard, and it is a stable specification developed by a W3C Working Group and reviewed by the W3C Membership.

1.5.1 OWL versions

There are three kind of OWL languages, with different expressiveness:

- **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints. Since expressivity is low, OWL Lite also has a lower formal complexity than OWL DL;
- **OWL DL** (good compromise between complexity and expressiveness) supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time).

  OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.


Synonyms in OWL
and
Description Logics

OWL	DL
class	concept
property	role
object	individual

- **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

Every legal	OWL Lite ontology	is a legal	OWL DL ontology
Every legal	OWL DL ontology	is a legal	OWL Full ontology
Every valid	OWL Lite conclusion	is a valid	OWL DL conclusion
Every valid	OWL DL conclusion	is a valid	OWL Full conclusion.

There exist many interpreter, translators and reasoner available. The more recent OWL 2 proposes three different profiles, with increasing expressivity (and hence increasing computational complexity in reasoning): OWL2 EL, OWL2 QL and OWL2 RL.

[5]OWL is a W3C standard.

The OWL Web Ontology Language is intended to provide a language that can be used to **describe the classes and relations** between them that are inherent in Web documents and applications.

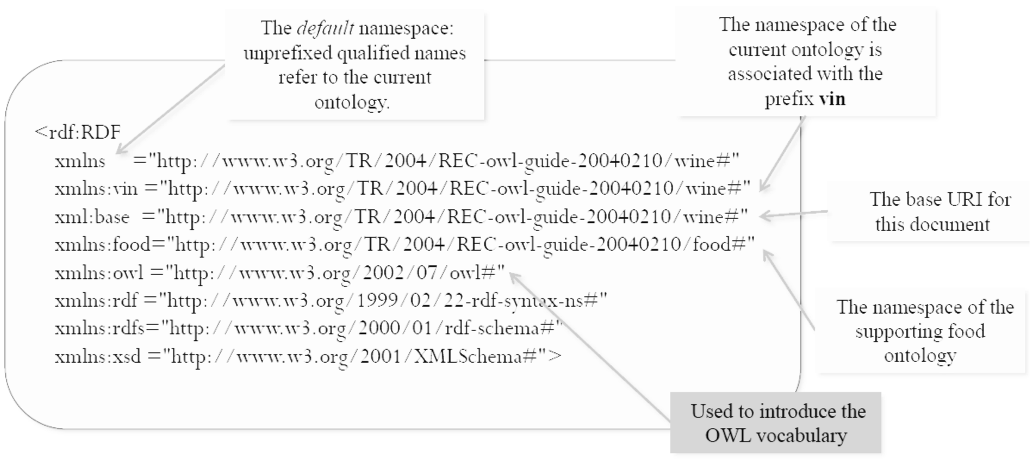
We demonstrate the use of the OWL language to:

- **formalize a domain** by defining classes and properties of those classes
- **define individuals** and **assert properties** about them.

This constitute the basis to reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language (see also: http://www.w3schools.com/rdf/rdf_owl.asp)

For a tutorial on how to design an ontology using the Protégŵ ontology editor check the webpage http://protege.stanford.edu/doc/owl/getting-started.html

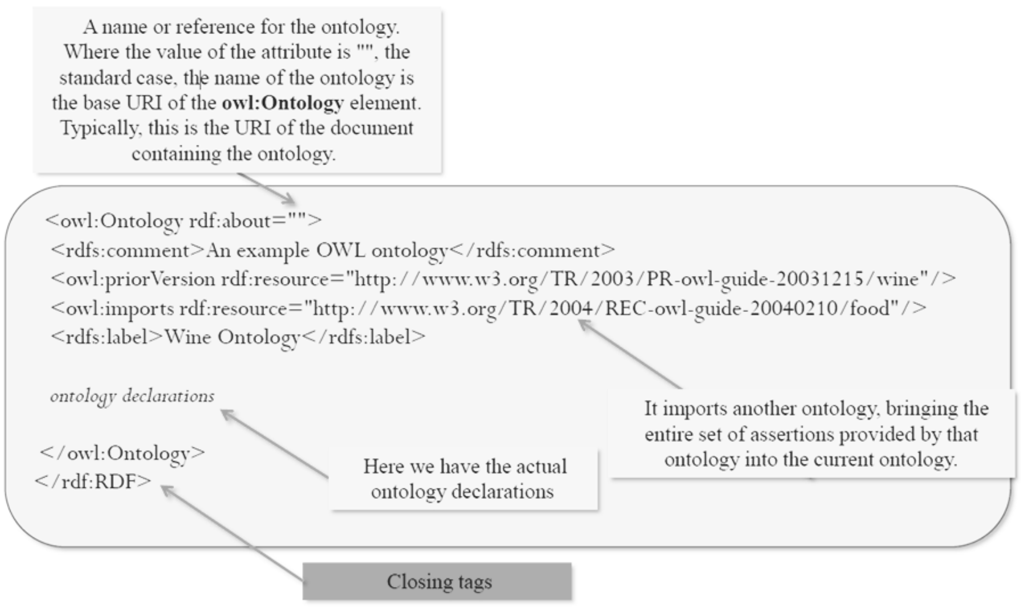
Before we can use a set of terms, we need a precise indication of what specific vocabularies are being used. A standard initial component of an ontology includes a set of XML namespace declarations enclosed in an opening `rdf:RDF` tag. These provide a means to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable. A typical OWL ontology begins with a namespace declarations similar to the following (of course, the URIs of the defined ontologies will not usually be w3.org references)



```
<rdf:RDF
  xmlns    ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xml:base  ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
  xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

The *default* namespace: unprefixed qualified names refer to the current ontology.
The namespace of the current ontology is associated with the prefix **vin**
The base URI for this document
The namespace of the supporting food ontology
Used to introduce the OWL vocabulary

Once namespaces are established we normally include a collection of assertions about the ontology grouped under an `owl:Ontology` tag.



```
<owl:Ontology rdf:about="">
 <rdfs:comment>An example OWL ontology</rdfs:comment>
 <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031215/wine"/>
 <owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food"/>
 <rdfs:label>Wine Ontology</rdfs:label>

 ontology declarations

</owl:Ontology>
</rdf:RDF>
```

A name or reference for the ontology. Where the value of the attribute is "", the standard case, the name of the ontology is the base URI of the **owl:Ontology** element. Typically, this is the URI of the document containing the ontology.
It imports another ontology, bringing the entire set of assertions provided by that ontology into the current ontology.
Here we have the actual ontology declarations
Closing tags

Importing another ontology brings the entire set of assertions provided by that ontology into the current ontology. In order to make best use of this imported ontology it would normally be coordinated with a namespace declaration. Notice the distinction between these two mechanisms:

- The namespace declarations provide a convenient means to **reference** names defined in other OWL ontologies;

- The actual instruction `owl:imports` is provided to indicate your intention to **include** the assertions of the target ontology.

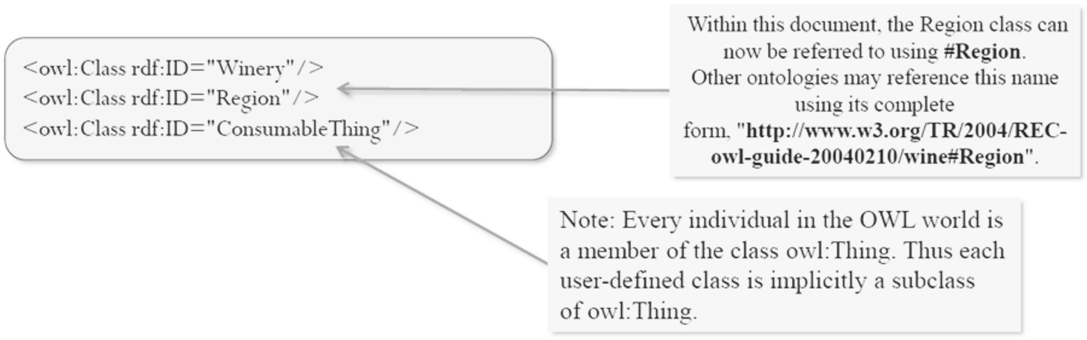
Importing another ontology,O2, will also import all of the ontologies thatO2imports.

Note that `owl:imports` may not always succeed. As you would expect when dealing with the Semantic Web, access to resources distributed across the Web may not always be possible. Tools will respond to this situation in an implementation defined manner.

1.5.2 Classes and Individuals

Many uses of an ontology will depend on the ability to reason about individuals. In order to do this in a useful fashion we need to have a mechanism to describe the classes that individuals belong to and the properties that they inherit by virtue of class membership. We can always assert specific properties about individuals, but much of the power of ontologies comes from class-based reasoning. Sometimes we want to emphasize the distinction between a class as an object and a class as a set containing elements. We call the set of individuals that are members of a class the **extension** of the class.

The most basic concepts in a domain should correspond to classes that are the roots of various taxonomic trees. OWL also defines the **empty class**, `owl:Nothing`.



```
<owl:Class rdf:ID="Winery"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="ConsumableThing"/>
```

Within this document, the Region class can now be referred to using **#Region**. Other ontologies may reference this name using its complete form, "**http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Region**".

Note: Every individual in the OWL world is a member of the class owl:Thing. Thus each user-defined class is implicitly a subclass of owl:Thing.

Formally, we know almost nothing about these classes other than their existence, despite the use of familiar English terms as labels. For all we know at the moment, these classes might as well have been called Thing1, Thing2, and Thing3.

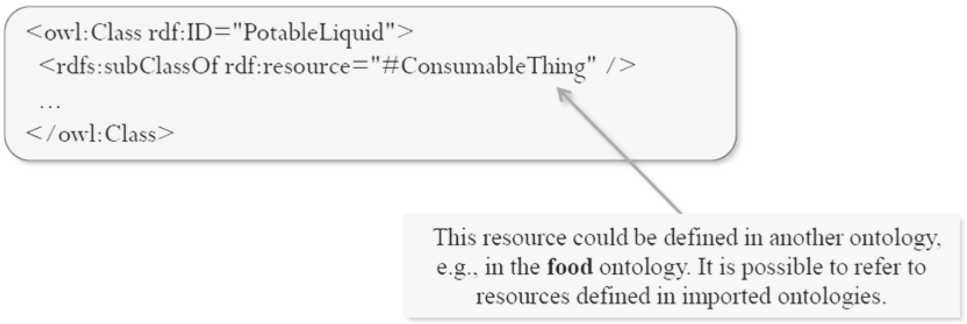
In the Description Logics terminology we refer to classes as to "concepts".

1.5.3 Class constructor

The fundamental taxonomic constructor for classes is `rdfs:subClassOf`. It relates a more specific class to a more general class: if X is a subclass of Y, then every instance of X is also an instance of Y.

The `rdfs:subClassOfrelation` is transitive. If X is a subclass of Y and Y a subclass of Z then X is a subclass of Z.

We define `PotableLiquid` (liquids suitable for drinking) to be a subclass of `ConsumableThing`.



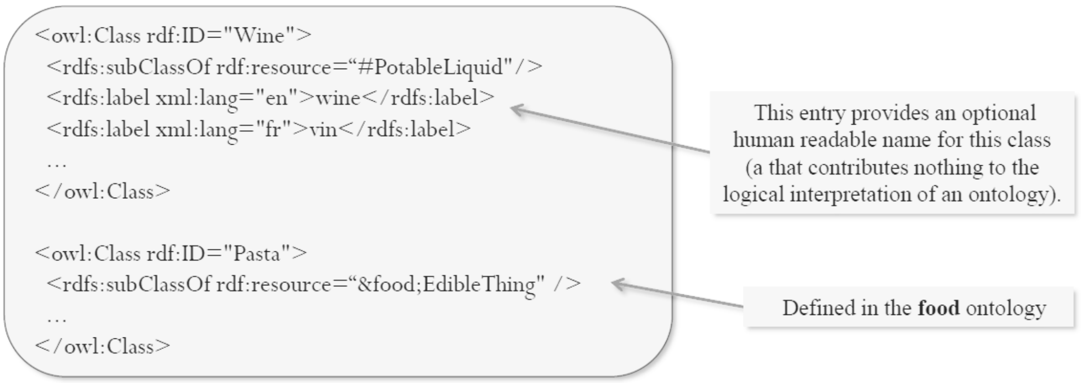
```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
  …
</owl:Class>
```

This resource could be defined in another ontology, e.g., in the **food** ontology. It is possible to refer to resources defined in imported ontologies.

A **class definition** has two parts:

- A name introduction or reference.
- A list of restrictions.

Each of the immediate contained expressions in the class definition further restricts the instances of the defined class. Instances of the class belong to the intersection of the restrictions. So far we have only seen examples that include a single restriction, forcing the new class to be a subclass of some other named class.



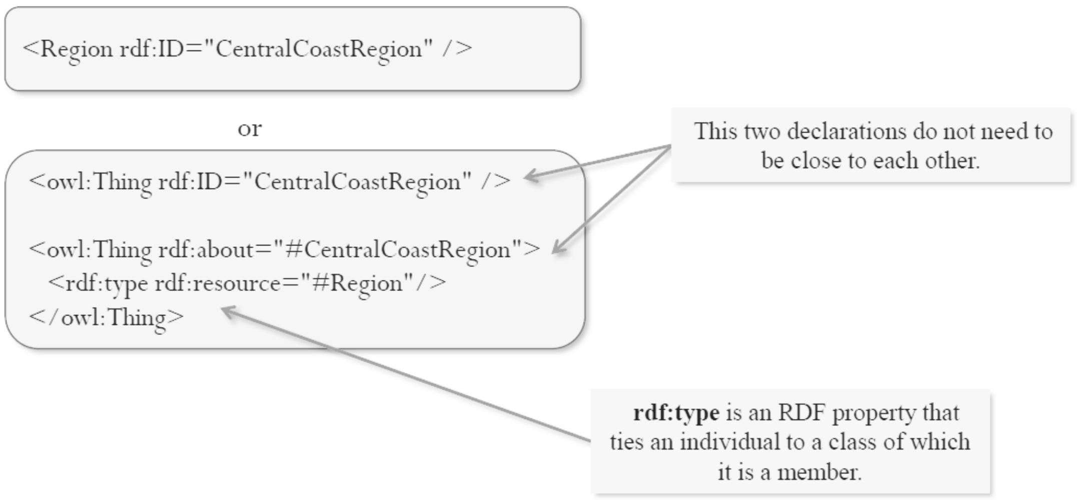
```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="#PotableLiquid"/>
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
  …
</owl:Class>

<owl:Class rdf:ID="Pasta">
  <rdfs:subClassOf rdf:resource="&food;EdibleThing" />
  …
</owl:Class>
```

This entry provides an optional human readable name for this class (a that contributes nothing to the logical interpretation of an ontology).

Defined in the **food** ontology

In addition to classes, we want to be able to describe their members. We normally think of these as individuals in our universe of things.

An individual can be introduced in two ways:



```
<Region rdf:ID="CentralCoastRegion" />
```

or

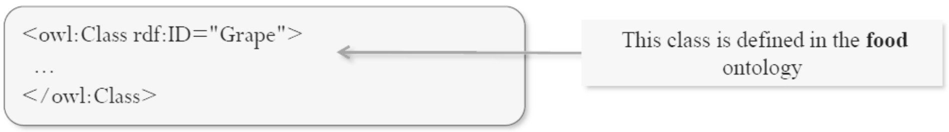
```
<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">
  <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

This two declarations do not need to be close to each other.

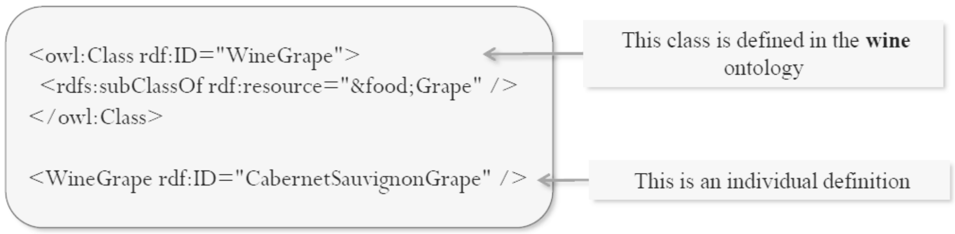
rdf:type is an RDF property that ties an individual to a class of which it is a member.

In the Description Logics we use the same terminology used in OWL to refer to individuals.

We add a few more classes to be used in the following examples.



```
<owl:Class rdf:ID="Grape">
  ...
</owl:Class>
```



```
<owl:Class rdf:ID="WineGrape">
  <rdfs:subClassOf rdf:resource="&food;Grape" />
</owl:Class>

<WineGrape rdf:ID="CabernetSauvignonGrape" />
```

This class is defined in the **wine** ontology
This is an individual definition

1.5.4 Classes and individual

There are important issues regarding the distinction between a class and an individual in OWL. A **class** is simply a name and collection of properties that describe a set of individuals. **Individuals** are the members of those sets. Thus classes should correspond to naturally occurring sets of things in a domain of discourse, and individuals should correspond to actual entities that can be grouped into these classes.

It is very easy to confuse the instance-of relationship with the subclass relationship. For example, it may seem arbitrary to choose to make `CabernetSauvignonGrape` an individual that is an instance of Grape,as opposed to a subclass of `Grape`. This is not an arbitrary decision. The `Grape` class denotes the set of all grape varietals, and therefore any subclass of `Grape` should denote a subset of these varietals. Thus, `CabernetSauvignonGrape` should be considered an **instance** of "Grape", and not a subclass. It does not describe a subset of `Grape` varietals, it is a grape varietal.

In different contexts, it could be different: if `Grape` represents the class of all bunches of grapes I have on my table, `CabernetSauvignonGrape` could represent the subclass of all bunches of grapes of that specific varietal.

1.5.5 Properties

A **property** is a binary relation that lets us assert **general facts** about the members of classes and **specific facts** about individuals.

Two types of properties are distinguished:

- **Datatype properties**: relations between instances of classes and RDF literals and XML Schema datatypes.

- **Object properties**: relations between instances of two classes.

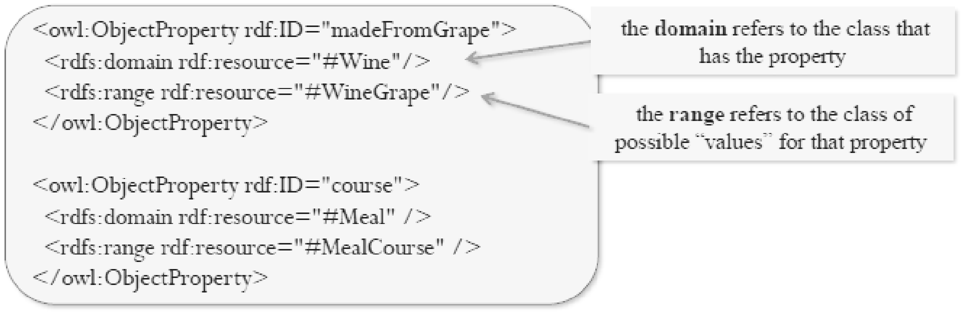
When we define a property there are a number of ways to restrict the relation, the domain and range can be specified or the property can be defined to be a specialization (subproperty) of an existing property. More elaborate restrictions are possible.

Remember that in **OWL**, a sequence of elements without an explicit operator represents an implicit conjunction / intersection. The property **madeFromGrape** has

- A domain `ofWine`

- A range `ofWineGrape`

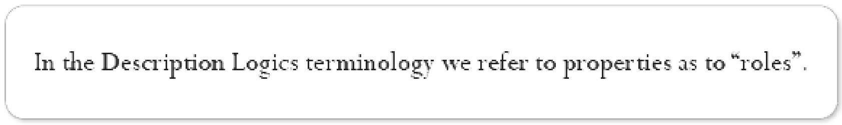
That is, it relates instances of the class `Wine` to instances of the class `WineGrape`. Multiple domains mean that the domain of the property is the intersection of the identified classes (and similarly for range).



```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="course">
  <rdfs:domain rdf:resource="#Meal" />
  <rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>
```

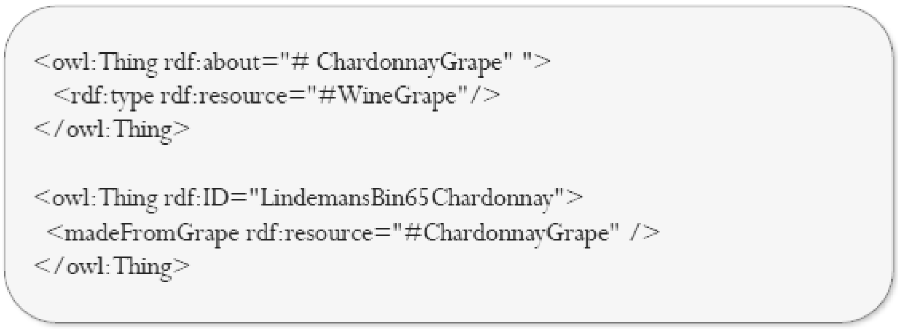
the **domain** refers to the class that has the property

the **range** refers to the class of possible "values" for that property



In the Description Logics terminology we refer to properties as to "roles".

Note that the use of range and domain information in OWL is different from type information in a programming language. Among other things, types are used to check consistency in a programming language. In OWL, a range may be used to infer a type. For example, given:



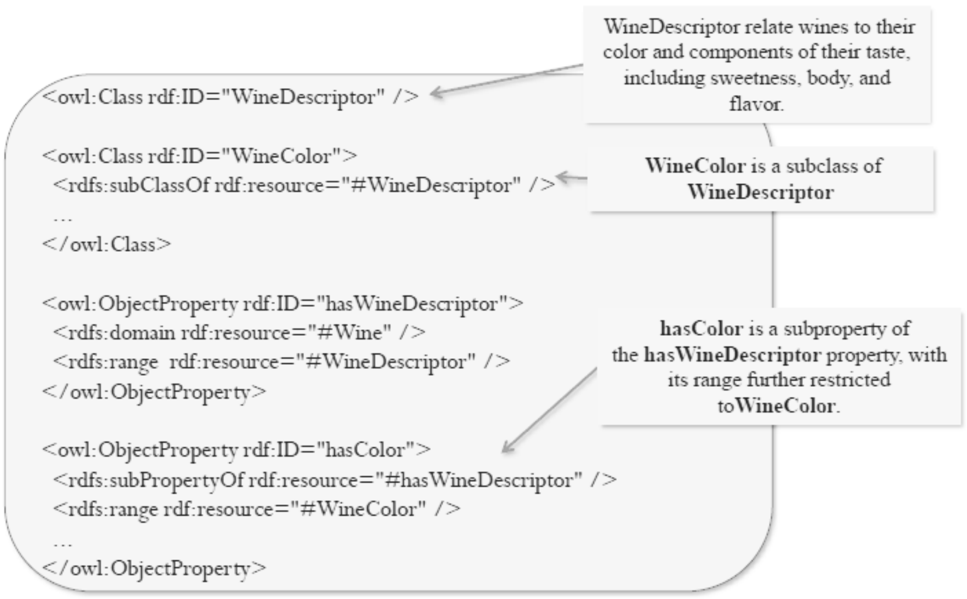
```
<owl:Thing rdf:about="# ChardonnayGrape" ">
  <rdf:type rdf:resource="#WineGrape"/>
</owl:Thing>

<owl:Thing rdf:ID="LindemansBin65Chardonnay">
  <madeFromGrape rdf:resource="#ChardonnayGrape" />
</owl:Thing>
```

we can **infer** that `LindemansBin65Chardonnay` is a wine because the domain of `madeFromGrape` is `Wine`.

This reasoning procedure is also referred to as **instance checking**, i.e., checking the class an individual belongs to.

1.5.6 Hierarchy

Properties, like classes, can be arranged in a hierarchy:



```
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range  rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
  ...
</owl:ObjectProperty>
```

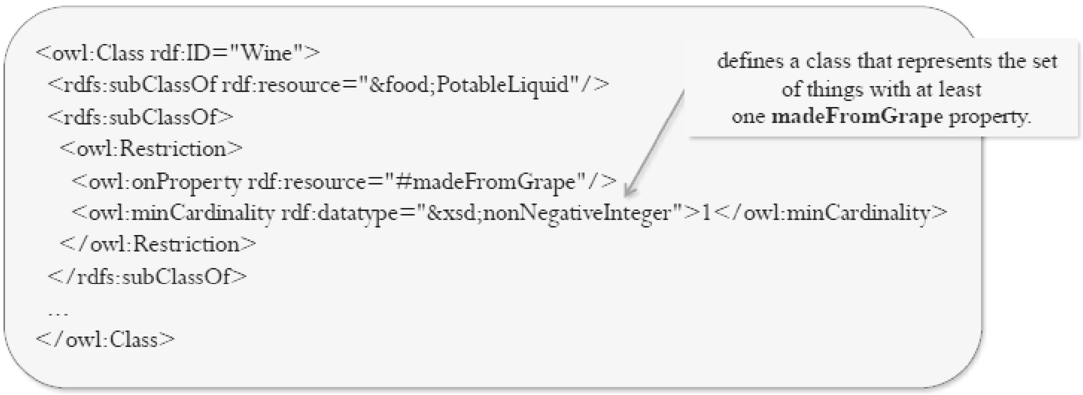
WineDescriptor relate wines to their color and components of their taste, including sweetness, body, and flavor.

WineColor is a subclass of **WineDescriptor**

hasColor is a subproperty of the **hasWineDescriptor** property, with its range further restricted to**WineColor**.

The `rdfs:subPropertyOf` relation in this case means that anything with a `hasColor` property with value X also has a `hasWineDescriptor` property with value X.

It is now possible to expand the definition `ofWine` to include the notion that a wine is made from at least one `WineGrape`. As with property definitions, class definitions have multiple subparts that are implicitly conjoined.



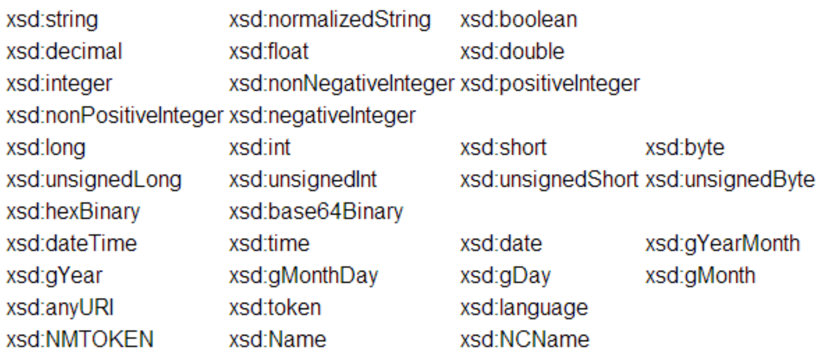
```
<owl:Class rdf:ID="Wine">
 <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#madeFromGrape"/>
   <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 ...
</owl:Class>
```

defines a class that represents the set of things with at least one **madeFromGrape** property.

We **distinguish** properties according to whether they relate

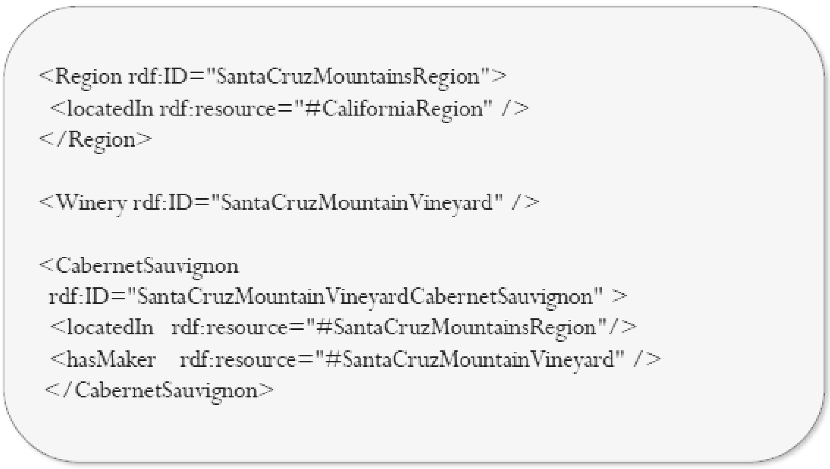
- Individuals to individuals (object properties)
- Individuals to datatypes (datatype properties)

Datatype properties may range over RDF literals or simple types defined in accordance with XML Schema datatypes. An example list of dataset is the following:



xsd:string	xsd:normalizedString	xsd:boolean	
xsd:decimal	xsd:float	xsd:double	
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger	
xsd:nonPositiveInteger	xsd:negativeInteger		
xsd:long	xsd:int	xsd:short	xsd:byte
xsd:unsignedLong	xsd:unsignedInt	xsd:unsignedShort	xsd:unsignedByte
xsd:hexBinary	xsd:base64Binary		
xsd:dateTime	xsd:time	xsd:date	xsd:gYearMonth
xsd:gYear	xsd:gMonthDay	xsd:gDay	xsd:gMonth
xsd:anyURI	xsd:token	xsd:language	
xsd:NMTOKEN	xsd:Name	xsd:NCName	

First we describe `Region` and `Winery` individuals, and then we define a `Cabernet Sauvignon`.



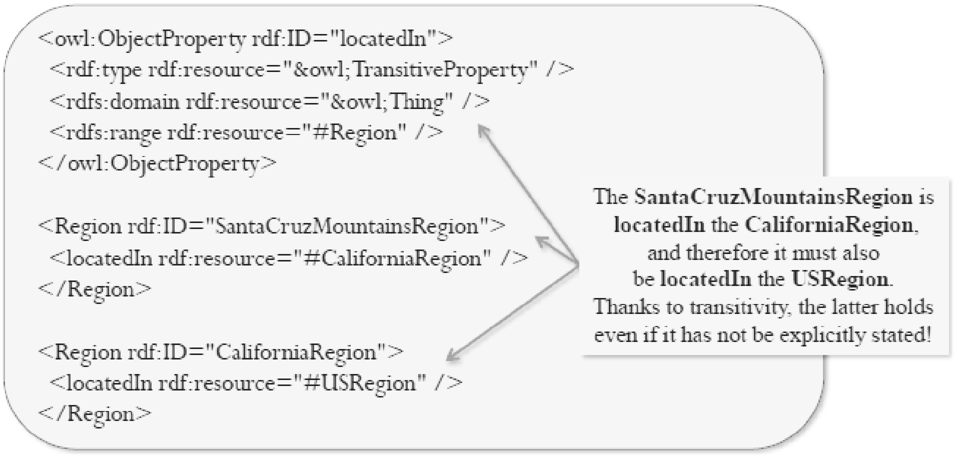
```
<Region rdf:ID="SantaCruzMountainsRegion">
 <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Winery rdf:ID="SantaCruzMountainVineyard" />

<CabernetSauvignon
 rdf:ID="SantaCruzMountainVineyardCabernetSauvignon" >
 <locatedIn  rdf:resource="#SantaCruzMountainsRegion"/>
 <hasMaker   rdf:resource="#SantaCruzMountainVineyard" />
 </CabernetSauvignon>
```

1.5.7 List of properties

Transitive property: if a property, P, is specified as transitive then for any x, y, and z:

```
P(x,y)  and  P(y,z)  implies  P(x,z)
```

The property `locatedIn` is transitive.



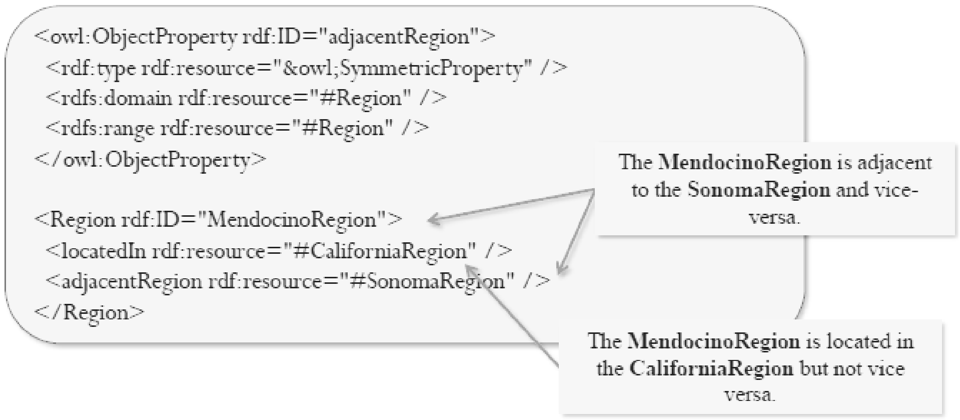
```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

The **SantaCruzMountainsRegion** is **locatedIn** the **CaliforniaRegion**, and therefore it must also be **locatedIn** the **USRegion**. Thanks to transitivity, the latter holds even if it has not be explicitly stated!

Symmetric property: if a property, P, is tagged as symmetric then for any x and y:

```
P(x,y)  iff  P(y,x)
```

The property `adjacentRegion` is symmetric, while `locatedInis` not.



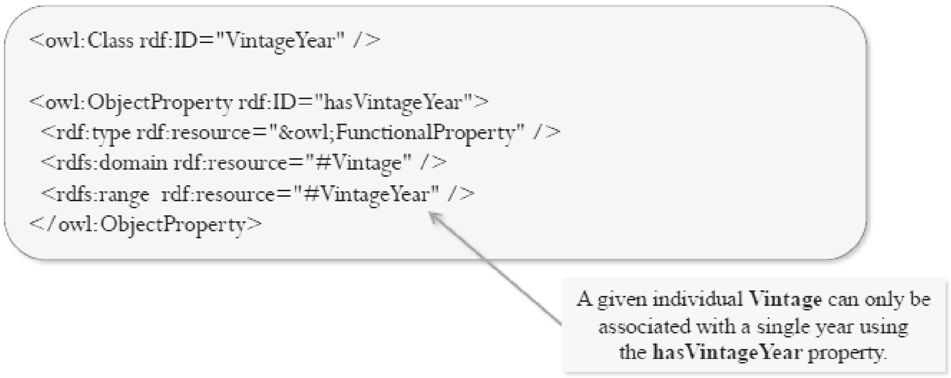
```
<owl:ObjectProperty rdf:ID="adjacentRegion">
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#Region" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="MendocinoRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
  <adjacentRegion rdf:resource="#SonomaRegion" />
</Region>
```

The **MendocinoRegion** is adjacent to the **SonomaRegion** and vice-versa.

The **MendocinoRegion** is located in the **CaliforniaRegion** but not vice versa.

Functional property If a property, P, is tagged as functional then for all x, y, and z:

```
P(x,y)  and  P(x,z)  implies  y = z
```

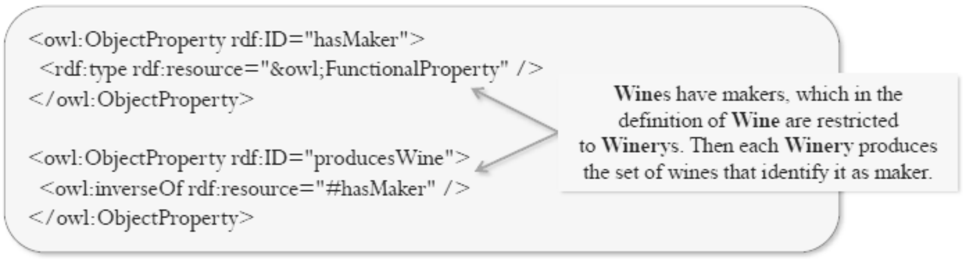


```
<owl:Class rdf:ID="VintageYear" />

<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range  rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```

A given individual **Vintage** can only be associated with a single year using the **hasVintageYear** property.

Inverse property if a property, P1, is tagged as the `owl:inverseOf` P2, then for all x and y:

```
P1(x,y)  iff  P2(y,x)
```

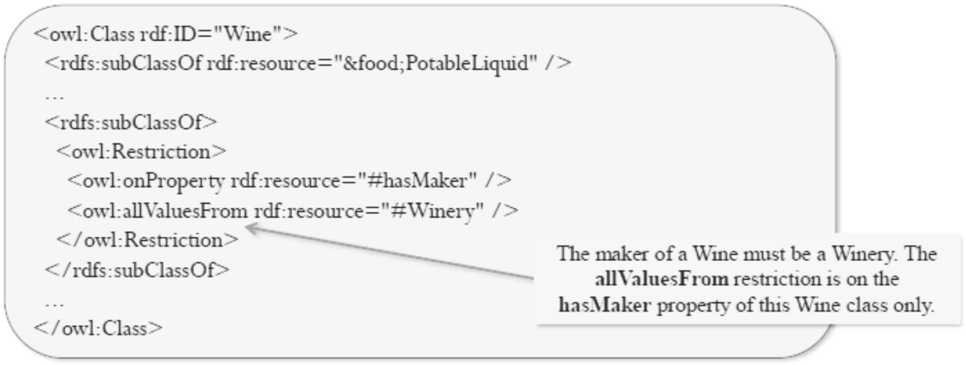


```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="producesWine">
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

Wines have makers, which in the definition of **Wine** are restricted to **Winerys**. Then each **Winery** produces the set of wines that identify it as maker.

1.5.8 Property Restrictions

In addition to designating property characteristics, it is possible to further constrain the range of a property in specific contexts in a variety of ways. We do this with **property restrictions**. The various forms described below can only be used within the context of an `owl:Restriction`. The `owl:onProperty` element indicates the restricted property.



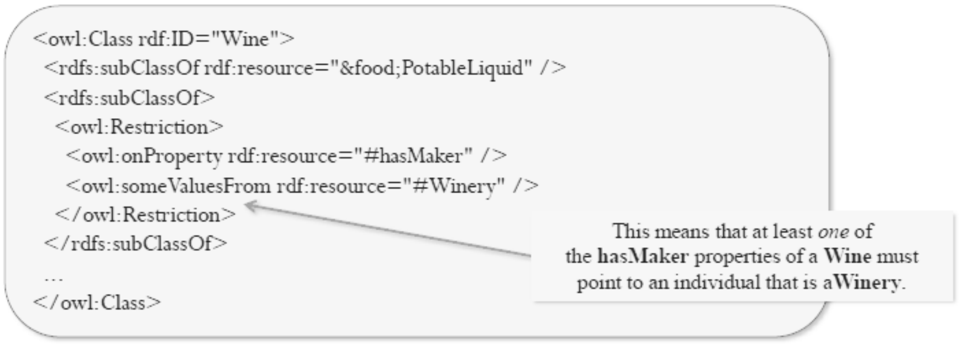
```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

The maker of a Wine must be a Winery. The **allValuesFrom** restriction is on the **hasMaker** property of this Wine class only.

The property restriction `owl:someValuesFrom` is similar. The difference between the two formulations is the difference between a universal and existential quantification.

`allValuesFrom`: For all wines, if they have makers, all the makers are wineries. This does not require a wine to have a maker. If it does have one or more, they must all be wineries.

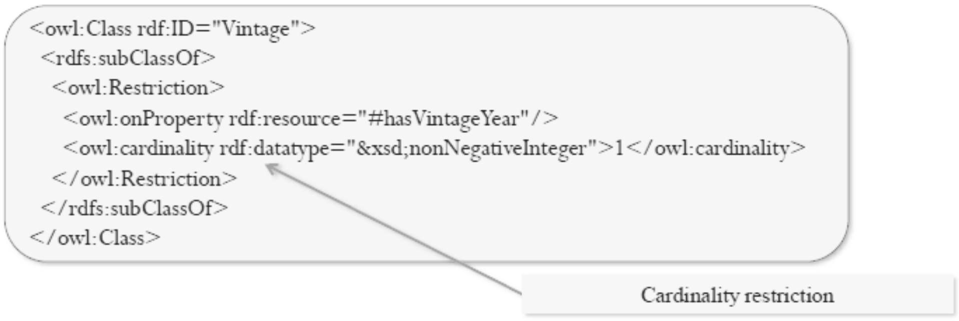
`someValuesFrom`: For all wines, they have at least one maker that is a winery. This requires that there be at least one maker that is a winery, but there may be makers that are not wineries.



```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

This means that at least *one* of the **hasMaker** properties of a **Wine** must point to an individual that is a **Winery**.

Cardinality We have already seen examples of cardinality constraints. To date, they have been assertions about minimum cardinality. Even more straight-forward is `owl:cardinality`, which permits the specification of **exactly** the number of elements in a relation. For example, we specify `Vintage` to be a class with exactly one `VintageYear`.



```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasVintageYear"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Once the ontology has been built, it is possible to reason upon it.

Some example are

- Determine whether or not the ontology is consistent
- Identify subsumption relationships between classes
- Checking if an individual is an instance of a class or not

Some popular reasoners are:

- Hermit
- RACER
- FaCT
- FaCT++
- Pellet

1.6 SWRL

More advanced reasoning can be made using SWRL rules. SWRL **extends** the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be **combined** with an OWL knowledge base. The proposed rules are of the form of an **implication** between an antecedent (body) and consequent (head).

The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Both the antecedent (body) and consequent (head) consist of zero or more atoms.

- An empty antecedent is treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation;

- An empty consequent is treated as trivially false (i.e., not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction. Note that rules with conjunctive consequents could easily be transformed into multiple rules each with an atomic consequent.

Atoms in these rules can be of the form `C(x)`, `P(x,y)`, `sameAs(x,y)` or `differentFrom(x,y)`, where

- `C` is an OWL description
- `P` is an OWL property
- `x`,`y` are either variables, OWL individuals or OWL data values.

It is easy to see that OWL DL becomes undecidable when extended in this way.

We will give an XML syntax for these rules based on RuleML and the OWL XML presentation syntax.

We start with an **abstract** syntax, which does not correspond to any implementation language. The abstract syntax is specified here by means of a version of Extended BNF.

- Terminals are quoted
- Non-terminals are bold and not quoted
- Alternatives are either separated by vertical bars (|) or are given in different productions.
- Components that can occur at most once are enclosed in square brackets ([...])
- Components that can occur any number of times (including zero) are enclosed in braces (...).
- Whitespace is ignored in the productions here.

1.6.1 Rules and axioms

An **OWL ontology** in the abstract syntax contains a sequence of axioms and facts.

Axioms may be of various kinds, e.g., `subClass` axioms and `equivalentClass` axioms. It is proposed to extend this with rule axioms.

$$axiom ::= rule$$

A **rule axiom** consists of an antecedent (body) and a consequent (head), each of which consists of a (possibly empty) set of atoms. A rule axiom can also be assigned a URI reference, which could serve to identify the rule.

$$\begin{aligned}
&rule ::=' Implies(' [URIreference] annotation antecedent consequent ')'\\
&antecedent ::=' Antecedent(' atom ')'\\
&consequent ::=' Consequent(' atom ')'
\end{aligned}$$

Informally, **a rule may be read as** meaning that if the antecedent holds (is "true"), then the consequent must also hold.

- An empty antecedent is treated as trivially holding (true). Rules with an empty antecedent can thus be used to provide unconditional facts; however such unconditional facts are better stated in OWL itself, i.e., without the use of the rule construct.
- An empty consequent is treated as trivially not holding (false).

Non-empty antecedents and consequents hold iff all of their constituent atoms hold, i.e., they are treated as conjunctions of their atoms.

```
atom ::= description '(' i-object ')'
       | dataRange '(' d-object ')'
       | individualvaluedPropertyID '(' i-object i-object ')'
       | datavaluedPropertyID '(' i-object d-object ')'
       | sameAs '(' i-object i-object ')'
       | differentFrom '(' i-object i-object ')'
       | builtIn '(' builtinID { d-object } ')'
builtinID ::= URIreference
```

Atoms can be of the form C(x), P(x,y), sameAs(x,y) differentFrom(x,y), or builtIn(r,x,...) where:

- C is an OWL description or data range
- P is an OWL property
- r is a built-in relation
- x and y are either variables, OWL individuals or OWL data values, as appropriate.

Informally

- An atom C(x) holds if x is an instance of the class description or data range C
- An atom P(x,y) holds if x is related to y by property P
- An atom sameAs(x,y) holds if x is interpreted as the same object as y
- An atom differentFrom(x,y) holds if x and y are interpreted as different objects.
- And builtIn(r,x,...) holds if the built-in relation r holds on the interpretations of the arguments.

Note that the "sameAs" and "differentFrom" two forms can be seen as "syntactic sugar": they are convenient, but do not increase the expressive power of the language (i.e., such (in)equalities can already be expressed using the combined power of OWL and rules without explicit (in)equality atoms).

```
i-object ::= i-variable
           | individualID
d-object ::= d-variable
           | dataLiteral
```

Atoms may refer to individuals, data literals, individual variables or data variables. Variables are treated as universally quantified, with their scope limited to a given rule.

As usual, only variables that occur in the antecedent of a rule may occur in the consequent (a condition usually referred to as "safety").

```
i-variable ::= 'I-variable(' URIreference ')'
d-variable ::= 'D-variable(' URIreference ')'
```

1.6.2 Human Readable Syntax

While the abstract EBNF syntax is consistent with the OWL specification, and is useful for defining XML and RDF serialisations, it is rather verbose and not particularly easy to read.

In the following we will, therefore, often use a relatively informal "human readable" form:

$$antecedent \Rightarrow consequent$$

where both **antecedent** and **consequent** are conjunctions of atoms written $a_1 \wedge ... \wedge a_n$.

Variables are indicated using the standard convention of prefixing them with a question mark (e.g., ?x).

Using this syntax, a rule asserting that the composition of parent and brother properties implies the uncle property would be written:

$$parent(?x, ?y) \ \wedge \ brother(?y, ?z) \ \Rightarrow \ uncle(?x, ?z)$$

In this syntax, built-in relations that are functional can be written in functional notation, i.e., "op:numeric-add(?x,3,?z)" can be written instead as

$$?x = op : numeric - add(3, ?z)$$

A simple use of these rules would be to assert that the combination of the **hasParent** and **hasBrother** properties implies the "hasUncleproperty".

Informally, this rule could be written as:

$$hasParent(?x1, ?x2) \ \wedge \ hasBrother(?x2, ?x3) \ \Rightarrow \ hasUncle(?x1, ?x3)$$

In the abstract syntax the rule would be written like:

$$\begin{aligned}&Implies(Antecedent(hasParent(I - variable(x1)I - variable(x2))\\&hasBrother(I - variable(x2)I - variable(x3)))\\&Consequent(hasUncle(I - variable(x1)I - variable(x3))))\end{aligned}$$

From this rule, if John has Mary as a parent and Mary has Bill as a brother then John has Bill as an uncle.

An even simpler rule would be to assert that "Students" are "Persons", as in

$$Student(?x1) \ \Rightarrow \ Person(?x1)$$

Or, in the abstract syntax:

$$Implies(Antecedent(Student(I - variable(x1)))\, Consequent(Person(I - variable(x1))))$$

However, this kind of use for rules in OWL just duplicates the OWL subclass facility. It is logically equivalent to write instead that the class "Student" is a subclass of "Person", which would make the information directly available to an OWL reasoner.

A very common use for rules is to move property values from one individual to a related individual, as in the following example that expresses the fact that the style of an art object is the same as the style of the creator.

$$Artist(?x) \ \& \ artistStyle(?x, ?y) \ \& \ Style(?y) \ \& \ creator(?z, ?x) \ \Rightarrow \ style/period(?z, ?y)$$

Or, in the abstract syntax

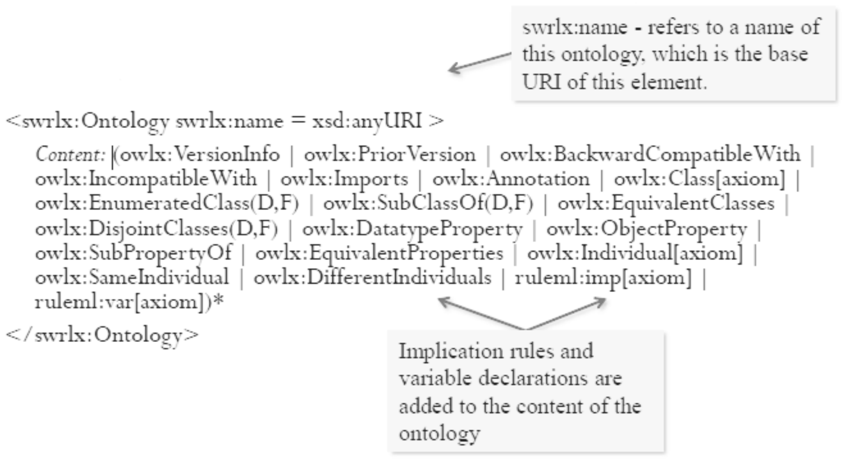
$$\begin{aligned}Implies(Antecedent(Artist(I - variable(x))artistStyle(I - variable(x)\\I - variable(y))Style(I - variable(y))creator(I - variable(z)I - variable(x)))\\Consequent(style/period(I - variable(z)I - variable(y))))\end{aligned}$$

1.6.3 The XML Concrete Syntax

The XML Concrete Syntax is a combination of the OWL Web Ontology Language XML Presentation Syntax with the RuleML XML syntax.

The **Ontology root element** of the OWL XML Presentation Syntax is extended to include "imp" (implication rule) and "var" (variable declaration) axioms as found under the rulebase root of RuleML.

Element Ontology


swrlx:name - refers to a name of this ontology, which is the base URI of this element.
Implication rules and variable declarations are added to the content of the ontology

```
<swrlx:Ontology swrlx:name = xsd:anyURI >
   Content: (owlx:VersionInfo | owlx:PriorVersion | owlx:BackwardCompatibleWith |
   owlx:IncompatibleWith | owlx:Imports | owlx:Annotation | owlx:Class[axiom] |
   owlx:EnumeratedClass(D,F) | owlx:SubClassOf(D,F) | owlx:EquivalentClasses |
   owlx:DisjointClasses(D,F) | owlx:DatatypeProperty | owlx:ObjectProperty |
   owlx:SubPropertyOf | owlx:EquivalentProperties | owlx:Individual[axiom] |
   owlx:SameIndividual | owlx:DifferentIndividuals | ruleml:imp[axiom] |
   ruleml:var[axiom])*
</swrlx:Ontology>
```

Let us consider the "elementruleml:var" axiom. Variable (var) axioms are statements about variables, indicating that the given string is to be used as a variable.

$$< ruleml : var > xsd : string < /ruleml : var >$$

A var axiom simply defines the existence of a variable. This is taken from the RuleML namespace. For example:

$$Parents : swrlx : Ontology$$

Let us consider the "elementruleml:imp" axiom. Rule axioms (imp elements) are taken from the RuleML namespace. A rule axiom can be read as a logical implication between the antecedent (_body) and consequent (_head).

$$< ruleml : imp > Content : (\_rlab?, owlx : Annotation*, \_body,_h ead) < /ruleml : imp >$$

Note:This element allows one to say that every binding that satisfies the _body of the rule must also satisfy the _head of the rule. All the child elements will now be described in details.

A **rule axiom** may optionally be named using a URI. elemen truleml:_rlab

$$< ruleml : \_rlabruleml : href = xsd : anyURI(required) > Content : () < /ruleml : \_rlab >$$

Both _body and _head are lists of atoms and are read as the conjunction of the component atoms. elementruleml:_body

$$< ruleml : \_body > Content : (swrlx : atom*) < /ruleml : \_body >$$

Parents:ruleml:imp elementruleml:_head

$$< ruleml : \_head > Content : (swrlx : atom*) < /ruleml : \_head >$$

Atoms can be formed from **unary predicates** (classes), **binary predicates** (properties), **equalities** or **inequalities**.

- Model group "swrlx:atom"

```
Content:(swrlx:classAtom | swrlx:datarangeAtom | swrlx:
    ↪ individualPropertyAtom
| swrlx:datavaluedPropertyAtom | swrlx:sameIndividualAtom | swrlx:
    ↪ differentIndividualsAtom | swrlx:builtinAtom)
```

  Class atoms consist of a description and either an individual name or a variable name.

  – Element `swrlx:classAtom`

```
<swrlx:classAtom> Content:( owlx:description, swrlx:iObject) </
    ↪ swrlx:classAtom>
```

Parents:swrlx:atom

For example

```
<swrlx:classAtom>
        <owlx:Class owlx:name="Person" />
        <ruleml:var>x1</ruleml:var>
</swrlx:classAtom>
```

Datarange atoms consist of a data range and either a literal or a variable name.

– Element `swrlx:datarangeAtom`

```
<swrlx:datarangeAtom> Content:( owlx:datarange, swrlx:dObject)
    ↪ </swrlx:datarangeAtom>
```

Parents: `swrlx:atom`

The description in a datarange atom may be a datatype ID, or may be a set of literals. For example:

```
<swrlx:datarangeAtom>
<owlx:Datatype owlx:name="\&xsd;int" />
<ruleml:var>x1</ruleml:var>
</swrlx:datarangeAtom>
```

– Element `swrlx:individualPropertyAtom`

```
<swrlx:individualPropertyAtom swrlx:property = xsd:anyURI {
    ↪ required} >
Content:( swrlx:iObject, swrlx:iObject)
</swrlx:individualPropertyAtom>
```

Attribute: `swrlx:property` -a reference to an individual property `nameParents:swrlx:atom`

For example

```
<swrlx:individualPropertyAtom swrlx:property="hasParent">
<ruleml:var>x1</ruleml:var>
<owlx:Individual owlx:name="John" />
</swrlx:individualPropertyAtom>
```

– Element `swrlx:datavaluedPropertyAtom`

```
<swrlx:datavaluedPropertyAtom swrlx:property = xsd:anyURI {
    ↪ required} >
Content:( swrlx:iObject, swrlx:dObject)
</swrlx:datavaluedPropertyAtom>
```

Attribute: `swrlx:property` -a reference to an datavalued property `nameParents:swrlx:atom`

For example

```
<swrlx:datavaluedPropertyAtom swrlx:property="grade">
<ruleml:var>x1</ruleml:var>
<owlx:DataValue owlx:datatype="&xsd;int">4</owlx:DataValue>
</swrlx:datavaluedPropertyAtom>
```

- Model group `swrlx:iObject`

```
Content:( owlx:Individual[ID] | ruleml:var[ID] )
```

Parents: `swrlx:classAtom`, `swrlx:individualPropertyAtom`, `swrlx:sameIndividualAtom`, `swrlx:different`

- Model group `swrlx:dObject`

```
Content:( owlx:DataValue | ruleml:var[ID] )
```

Parents: `swrlx:datavaluedPropertyAtom`

– Element `ruleml:var ID`

```
<ruleml:var>xsd:string</ruleml:var>
```

Parents: `swrlx:iObject, swrlx:dObject`

Note: this element is used for solely referring to a variable ID, and does not actually define any variable, unlike avar axiom.

1.6.4 Rule assertion

We can use SWRL to assert that the combination of the "hasParent" and "hasBrother" properties implies the "hasUncle" property:

```
<ruleml:imp>
<ruleml:_rlab ruleml:href="#example1"/>
<ruleml:_body>
<swrlx:individualPropertyAtom swrlx:property="hasParent">
<ruleml:var>x1</ruleml:var>
<ruleml:var>x2</ruleml:var>
</swrlx:individualPropertyAtom>
<swrlx:individualPropertyAtom swrlx:property="hasBrother">
<ruleml:var>x2</ruleml:var>
<ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom>
</ruleml:_body>
<ruleml:_head>
<swrlx:individualPropertyAtom swrlx:property="hasUncle">
<ruleml:var>x1</ruleml:var>
<ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom>
</ruleml:_head>
</ruleml:imp>
```

This rule asserts that if "x1" "hasParent x2", "x2" "hasSibling x3", and "x3" "hasSex male", then "x1" "hasUncle x3"

```
<ruleml:imp>
<ruleml:_rlab ruleml:href="#example2"/>
<ruleml:_body>
<swrlx:individualPropertyAtom swrlx:property="hasParent">
<ruleml:var>x1</ruleml:var>
<ruleml:var>x2</ruleml:var>
</swrlx:individualPropertyAtom>
<swrlx:individualPropertyAtom swrlx:property="hasSibling">
<ruleml:var>x2</ruleml:var>
<ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom>
<swrlx:individualPropertyAtom swrlx:property="hasSex">
<ruleml:var>x3</ruleml:var>
<owlx:Individual owlx:name="#male" />
</swrlx:individualPropertyAtom>
</ruleml:_body>
<ruleml:_head>
<swrlx:individualPropertyAtom swrlx:property="hasUncle">
<ruleml:var>x1</ruleml:var>
<ruleml:var>x3</ruleml:var>
</swrlx:individualPropertyAtom>
</ruleml:_head>
</ruleml:imp>
```

1.6.5 Ontology

The top layers of the Semantic Web stack contain technologies that are **not** yet **standardized** and are required to realize the Semantic Web.

OWL plus SWRL rules plus SPARQL query language are **integrated** together in a common Unifying Logics framework.

Cryptographyis required to verify that information is coming from a trusted source.

Trust to derived statements will be supported by (a) verifying that the premises come from trusted source and by (b) relying on formal logic during deriving new information.

User interface is the final layer enabling users to interact with semantic web applications.

What can be used ontologies in AmI for? As usual, to represent, exchange, reason upon knowledge.

For example, agents can exchange information with other agents, or collect information on the internet. In addition, they can be used to perform "**context assessment**" and "**situation assessment**"

- Context assessment means understanding the status of "things" that are related to a given entity (where am I located? Is the oven on or off)
- Situation assessment means how the context evolves in time.

The **epistemic problem:** how is it possible to acquire new knowledge and store it in the Knowledge base?

Chapter 2

Sensors

2.1 Introduction

Sensing devices provide the first step towards context awareness. **The W5+ questions:**

- **Who**: Tracking and identifying persons and pets, i.e. the actors of the Ambient Intelligence (AmI) environment;
- **Where** and **When**: Providing a time frame for location and object associations to determine context;
- **What**: Recognizing activities, interactions, spatio temporal relations, but also linguistic and non linguistic messages, signals, and signs;
- **Why**: Association of actions with action semantics, scripts and plans, identification of tasks and behaviour patterns;
- **How**: Tracing the information flow through multiple modalities, recognizing expressions, movements, gestures.

2.2 Cameras

Traditionally, research has focused mainly on audio visual observations as these modalities provide almost all signals that humans make use of in interpersonal communication. The visual domain represents all the following categories:

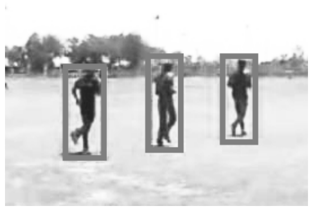
- Face detection and identification.
- Person and object tracking;
- Facial expression recognition;
- Body posture recognition;
- Attention direction sensing;
- Hand tracking and hand gesture recognition;
- More recently, object and place recognition.

2.2.1 Face detection / Facial expression recognition

- What happens when the face is seen under a different perspective?
- What about occlusion and changing lighting conditions?

2.2.2 Person and object tracking / Body posture recognition

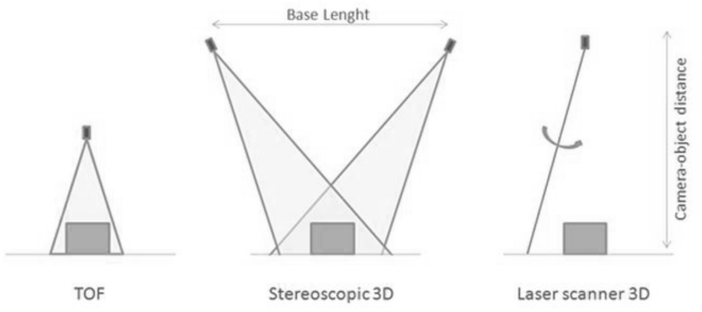
- What about occlusions? Current research focuses on people tracking in the crowd!



Cameras can be used for objects and places recognition, for recognizing emotions and so on. All these computation can be performed with the help of some kind of cloud computing platform such as clarifai, Microsoft Azure, Google Coolab and so on.

2.3 RGB-D Cameras

An RGB-D camera is a time of flight camera (TOF camera) which is able to create distance data with help of the time of flight (TOF) principle. The principle is similar to that of laser scanners with the advantage that whole scene is captured at the same time. Infrared light from the camera's internal lighting source is reflected by objects in the scene and travels back to the camera, where its precise time of arrival is measured independently by each of tens of thousands of sensor pixels. Resolution is low (about 320x240 pixel in 2011) and it requires an illumination unit, i.e., it is not a passive sensor.



The output data of and RGB-D camera can look as follows:

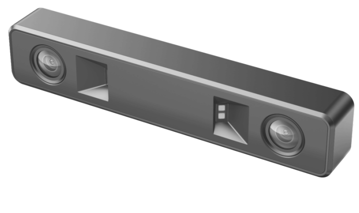
2.3.1 Microsoft Kinect

An example of RGB-D camera available in the market is the Microsoft Kinect, a device that features an RGB camera, depth sensor and multi array microphone. The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions. The Kinect sensor has a ranging limit of 0.7 6 m (2.3 20 ft) and has an angular field of view of 57 degrees horizontally and 43 degrees vertically. At the minimum distance resolution is 1.3 mm (0.051 in) per pixel.

In the kinect, the depth map is constructed by analyzing a speckle pattern of infrared laser light. It used the structured light general principle: it projects a known pattern onto the scene and infer depth from the deformation of that pattern.

2.4 Stereo Cameras

Stereo cameras provide the same information about the three dimensional structure of the environment



2.5 Microphones

The auditory domain is primarily about speech recognition, which is the process of detecting speech and identify and localize speakers determining properties of the source, e.g. detecting head orientation of speaking persons in the room (relies on microphone arrays). Recently more attention has been paid to general auditory signal analysis in order to identify broader classes of sounds (ShotSpotter system)

2.6 Event Cameras

In event cameras, only local pixel level changes are transmitted at the time they occur. The result is a stream of events at microsecond time resolution, equivalent to conventional vision sensors running at thousands of frames per second but with far less data. Power, data storage and computational requirements are also drastically reduced, and sensor dynamic range is increased by orders of magnitude due to local processing.

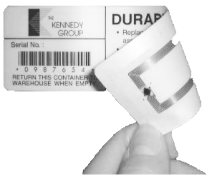
2.7 Presence sensors

Passive infrared sensors (PIR sensor) register the infrared emissions from objects (notably humans, animals and vehicles). They provide complementary features for motion and object classification. With a network of PIR devices deployed in the AmI environment, it becomes possible to detect fire and smoke, but also to learn movement patterns of the inhabitants.

PIR sensors measure infrared (IR) light radiating from objects in its field of view. They are often used in the construction of PIR based motion detectors. Apparent motion is detected when an infrared source with one temperature, such as a human, passes in front of an infrared source with another temperature, such as a wall. The term passive means that the PIR device does not emit an infrared beam but merely passively accepts incoming infrared radiation. The window may have multiple Fresnel lenses moulded into it, i.e., a lens with large aperture and short focal length without the mass and volume of material that would be required by a lens of conventional design.

2.8 RFID Antennas

Radio frequency identification (RFID) technology is making inroads in the AmI setting as it allows both sensing of object proximity and identification. Passive RFID tags are small, flexible and do not require endogenous power to operate. They can therefore be placed on every day objects. Strategically positioned RFID antennas will be able to identify the objects or persons that pass in their proximity. Research is looking for smaller RFID, with a higher communication range and bandwidth.



Today, RFID tags are used for a lot of different applications:

- Identity recognition
- Transportation payments
- Product tracking
- Clothes

Collecting the data from different readers allows the system to infer activities of the main actors in an AmI scenario, such as assuming that people wear a bracelet which registers the ID tags on all the objects the user touches and to perform localization.

By sticking a large number of RFID tags on a variety of objects and appliances (e.g. kettle, coffee mugs, etc.) in the household or office, it becomes possible to infer what activity a person is involved in (e.g. making a cup of coffee). Tagging the users of an AmI system with RFID is also very useful for collecting ground truth data for face, gesture, body posture and speech recognition applications, which usually incorporate statistical models that require large amounts of data for robust operation.

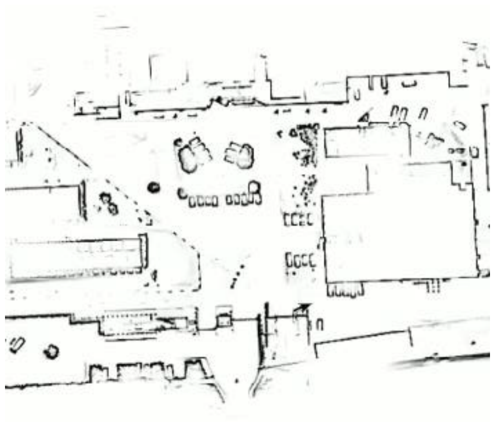
2.9 Laser Scanner

Range sensing sensors are used in robotics for mapping and obstacle detection, as well as for localization. In AmI, they can be used to implement light curtains, that are opto electronic devices used to safeguard personnel in the vicinity of moving machinery with the potential to cause harm such as (but not limited to) presses, winders and palletizers. They are also used in Museums, e.g., to preserve paintings from vandalism.

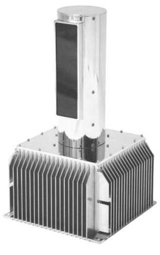
How can they be used in Ambient Intelligence? Laser scanners return range measurements with a given angular resolution (e.g., 0.5 degrees with a Field of view of 270 degree). Range measurement can be used for:

- Build a 2D geometrical map of the environment, which describes the profile of walls, furniture, people, etc. Perform Cartesian localization.
- Define safety regions in which nobody is allowed to enter.

The output of a laser scanning of an area will look as follows:



2.10 3D laser scanner



A 3D Laser scanner is ideally composed of a laser scanner and a Pan unit. The output of a 3D laser scanning of an area will look as follows:

2.11 Accelerometers and gyroscopes

Inertial Measurement Units measure linear acceleration and orientation, they use a combination of accelerometers and gyroscopes. They are typically used to manoeuvre aircraft, including UAVs, but also mobile robots. They can be used to measure human motion and posture (gait, falls, etc.) and for games.

They can measure acceleration and orientation along 6 axes and they can be integrated to provide Cartesian position.

2.12 GPS and environmental sensors

GPS sensors and environmental sensors for light, temperature, pressure or vibration are often already equipped with wireless communication capabilities.

2.13 Switches

Switches are the most simple but most useful sensors for AmI.

They have a lot of very good properties:

- They are very unobtrusive
- They are very reliable
- They can be easily embedded in many objects that plays a fundamental role in human activity (door and window handles, light switches, taps, ovens, fridges).

2.14 Artefacts

Sensing devices can be used to build artefact. Turning an object into an artefact is a process that aims at enhancing its characteristics and properties and abilities so that the new affordances will emerge.

Affordance is a term introduced by psychologist J. Gibson.

> “"action possibilities" latent in the environment, objectively measurable and independent of the individual’s ability to recognize them, but always in relation to the actor and therefore dependent on their capabilities.”

For instance, a set of steps which rises four feet high does not afford the act of climbing if the actor is a crawling infant. The term is used in the field of design and Human Computer Interaction.

In practical terms, building an artefact is about providing the object with the necessary hardware and software modules, by embedding the hardware modules into the object and installing the software modules that will determine its functionality,

Generally, this phase involves embedding into the artefact

- A power source
- An array of sensors and actuators
- A processor board
- A wireless module
- A fes buttons and a screen

Nevertheless, some of these modules may not appear in all artefacts. For example:

- Artefacts, especially mobile ones, may have batteries but large artefacts can be directly connected to an electricity socket;
- The number of sensors may range from tens to none. An artefact may simply receive sensor data from other artefacts;
- The processor board may be embedded in the artefact, but it may also be the case that the artefact "rents" processing time and storage space at a nearby (using network, not physical terms) server;
- The wireless module is probably the most distinguishing module; nevertheless, an artefact may only carry a passive tag, which, when tracked, triggers an action at another artefact. Also, it can be connected to the network;
- Screens and buttons are generally avoided, especially in small size artefacts, as they occupy a lot of space and consume a lot of power.

Most usually, the hardware modules will be embedded in the artefact and unique ID (could be a serial number) will be assigned to it at the time of its manufacture.

Example: Smart Fridge

- It senses object inside (through RFID antenna)
- It detects the expiration date of food
- It warns the user if some food has expired or has finished
- If the user plans to prepare a recipe, it checks if all ingredients are available
- If some ingredients are not available, it orders them online

2.15 Sensing Devices Classification

We skip details about the sensor hardware, instead, we propose a classification of sensing devices which is significant for AmI applications (we also speak about "sensing services"). In particular, sensing devices are classified according different axes:

- The "dimensions" and the "domain" of data they return, i.e., boolean , scalar, named values, 1 dimensional vector, 2 dimensional vector, etc;
- How much "simple" they are;
- If they are localized/able to localize themselves and other objects.

We do not try to give a rigorous definition: instead, we refer to the intuitive meaning of the previous notions.

2.15.1 Dimensions and domain of data

Cameras return 2D data. However, motion detection camera returns boolean data (i.e., TRUE or FALSE, depending on the fact that there is something moving or not). One could object that, in the second case, we are not considering raw data, but the information obtained by a motion detection algorithm.

What about a boolean camera which returns TRUE if and only if it recognizes me? What about an "Antonio Detection Camera"? (or a "Daniel Dennet Detection Camera"?).

What should we consider as being part of the sensing device, the hardware? The firmware? Both of them plus processing software?

The key point is that motion detection with a camera is reliable , (and in fact it is often implemented in the firmware), i.e., it has a low error rate (<1%). The "AD Camera" is not reliable, it often fails when in presence of occlusions, when perspective changes, etc

Definition 1 We call "sensing device" the hardware that come out of the box, including firmware, plus additional software which process data by still providing results with a very low error rate

2.15.2 Dimensional space and data processing

The higher is the "dimension" of the data returned, the more complex is data processing.

Sensing devices can be ordered in increasing order with respect to both dimensional space and processing requirements.

1. **Single Boolean values**: cameras for motion detection, Passive InfraRed Sensors, safety laser scanners, switches.
2. **Single Named values**: RFID antenna and tags.
3. **Single scalar value**: temperature, pressure, light, humidity, microphones.
4. **Ordered set of scalar values**: GPS, inertial measurement units, laser scanner.
5. **Grid of scalar values**: B&W camera, 3D Laser scanner
6. **Grid of vectors**: RGB camera, 3D Stereo Camera, TOF camera.

2.15.3 A simple device

The definition of "simple" is even "fuzzier":

Definition 2 A sensing device is "simple" if it say something about entities and facts of the word at the level of abstraction commonly used by humans to describe the world.

Examples

- A switch says that "somebody has turned the light on".
- A PIR sensor says that "somebody is present in the room".
- A camera for motion detection says the same thing (at a higher cost).

Both information can be easily expressed in natural language, and have an "intuitive" meaning.

A camera (not used for motion detection) returns a grid of pixel. The image need additional processing to say something about entities and facts of the world which are "interesting" for humans.

Unfortunately, the information "somebody is here" returned by the motion detection camera is reliable, object recognition is not. Therefore, according to Definition 1, a "camera that detects armchairs" does not exist as a "reliable" sensing device.

The only sensing device that exists is the camera, which unfortunately is not simple sensor since it does not provide information at a high level of abstraction.

An alternative definition could be the following:

Definition 2bis A sensing device is "simple" if it say something that can be "reliably" expressed in symbolic form by using only first order predicates available in the "domain".

At time t the motion detection camera in the kitchen says:

∃x, moving(x, kitchen, t)

Similarly one could state that:

At time t the armchair detecting camera says

∃x, is_in (x, hallway, t) ∧ armchair(x)

But that information is not very reliable, this is the well known problem of symbol grounding.

If the perceptual anchoring of symbols is wrong, all subsequent inferences are wrong as well.

Ideally, one should compute the error probability in detecting objects and to propagate such probability in all subsequent inferences.

However the standard video camera V1 says

pixel(V1, 1, 1, 203,t)
pixel(V1, 1, 2, 202,t)
...
pixel(V1, 640, 480, 157,t)

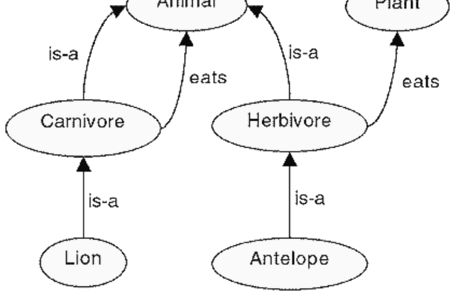
By assuming here that V1 has resolution 640x480 and returns images in gray tones:

- The second and third variables correspond to row and column.
- The fourth variable corresponds to the gray tone.

This appears a little strange but it still makes sense.

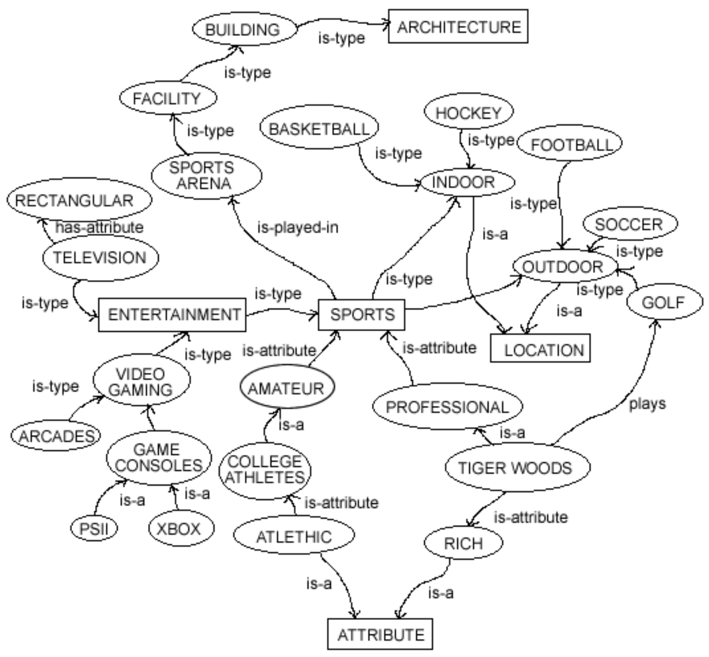
Does it mean that a standard video camera is a "simple" sensing device? It depends on the "domain" considered.

Ontology is the philosophical study of the nature of being, existence or reality in general, as well as of the basic categories of being and their relations. Ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences, as it can be seen from the following image:



The informatics meaning of ontology is more specific: here we mean "what exists" in the domain considered.

A simple ontology can reach very different domains:



If we are interested, for example, in perceiving objects, humans, as well as fact and relationships between objects and humans, we can assume a "human environment ontology". It is unlikely that "pixel" will be part of the "human environment ontology", as well as "electrons" are not part of the "System biology ontology". "Electrons" are presumably described in the "Particle physics ontology".

The process of describing entities belonging to an ontology through its component entities belonging to a lower level ontology is a type of "reductionism".

2.15.4 From simple to complex devices

We call a sensing device**"simple"** in a given domain if is able to say something reliable (i.e., with a low error rate) about entities in the ontology which describe that domain (e.g., humans, objects, etc.).

We call a sensing device **"complex"** in a given domain if it is able to say something reliable only about entities belonging to an ontology at a lower level of abstraction (e.g., pixels, range measurements), and it requires further processing (which introduce errors) to describe higher level entities on the basis of lower level ones.

In this sense, all efforts in **statistical data fusion** (Kalman Filters, Particle Filters, Information Filters, Bayesian extimators , etc.) can be interpreted as an attempt to build "simple" sensing devices for the "human environment ontology", i.e., sensing devices able to reliably ground symbols by anchoring them to perceptual data.

This is very important: simple sensors produce results that can be merged at a symbolic level which is very intuitive for humans , in order to describe more complex situations starting from basic elements.

$$\neg\exists x, \text{moving}(x, \text{kitchen}, t) \wedge \text{opened}(\text{fridge}) \Rightarrow \text{alarm}(\text{kitchen})$$

Ontologies can be ideally written in first order predicate logic, however there are language which include only a subset of first order predicate logic to guarantee computational efficiency and completeness (description logics, Resource Description Framework RDF, Ontology Web Language OWL, etc.)

Most ontologies are equipped with "reasoners".

When going from simple to complex devices there is an obvious correlation with the dimensions of data returned.

2.15.5 Other classification methods

Sensing devices can be also classified according to

- Their sensing range / field of view.
- Their capability to sense the distance of objects.

These characteristics can play a fundamental role when coupled with the ability to localize themselves.

Sensing range / Field of view A switch has a sensing range which is 0, however, if it is part of some artifact (e.g., TV set, fridge, shower tap, etc.), it is localized, i.e., it is associated with a given location in the environment.

As long as somebody operates the switch, it is possible to say something about the location of the person with a great accuracy and almost null error probability! (it can be difficult to disambiguate between different persons).

∃x, is_in (x, shower, t)

Or even, by knowing the XY coordinates of the shower,

∃x, is_at (x, X-shower, Y-shower, t)

In most case, knowing XY coordinates is not relevant in the human environment domain. RFID tags can be used in the same way.

The problem with sensors that have a low sensing range is that a very dense distribution is required (e.g RFID carpet). As the sensing range increases (as in the ordered list below), the accuracy in localizing object decreases and a denser distribution of sensors is required.

- Switches, IMUs
- RFID antenna and tags.
- Temperature, pressure, light, humidity, microphones.
- Passive InfraRed Sensors, TOF camera.
- Laser scanners, motion detection camera.
- Video camera.
- GPS

Capability to sense distance objects Cameras, laserscanner , etc., have a higher sensing range, and hence positioning accuracy is lower

∃x, is_in (x, bathroom, t)

However, their ability to return an estimated of the distance of an object that is segmented from the background, allow to infer the object position (as long as the sensors themselves are localized).

Wireless sensor network (WSN) motes (i.e nodes) usually have an even higher sensing range, however they are able to measure the distance from another mote depending on the power of the received signal.

In general, if sensors provide metric information about the sensed object, it can be used to infer the position of objects (trilateration, triangulation, inverse perspective, Kalman Filtering, etc.)

- Trilateration requires to measure distances, and it is at the basis of GPS, microphone arrays, and WSN motes localization.
- Triangulation requires to measure angles, and can be performed with cameras.
- Inverse perspective can be performed with cameras.

Sensing devices can also be classified according to the resolution in returning the distance or the relative position of the sensed object:

- BOOLEAN switches, IMUs, RFID antenna and tags.
- PIR Sensors, Motion detection camera where alarm frequency depend on distance.
- Microphones, WSN motes.
- Laser scanners, TOF cameras, standard cameras, GPS.

2.15.6 Ability to localize themselves

It is very important, since it constitutes a prerequisite for the sensing range / field of view of devices and their capability to sense the distance of objects. The position of all sensors which do not move in the environment can be recorded (this can be expensive, especially in large environments).

The position of moving sensors require the ability to localize themselves:

- GPS provides this information outdoor, and can be coupled to another sensor to provide positioning data.
- WSN motes are able to localize themselves with respect to each other and a base station on the basis of the wireless signal strength.
- Video cameras, laserscanner , inertial measurement units, etc., are capable of performing self localization but require complex localization algorithms (basically the same algorithms that are required to localize objects).
- When the same sensor is used to localize objects and to localize the sensor itself on the basis of detected objects, we are performing SLAM (Simultaneous Localization and Mapping).

Chapter 3

Self-localization

3.1 Introduction

In Ambient Intelligence, it is fundamental that **sensors** know their own location (**self-localization**) and that they are able to compute the of significant **actors'/objects' locations** (the problem is the same as in autonomous robotics)

The main difference with mobile robots relies on the types of sensors that, in Ambient Intelligence, are available (either **distributed** in the environment and/or **carried** around by actors) $\rightarrow$ the case of an actor carrying sensors is just a particular case of sensor self-localization.

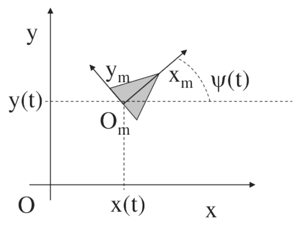
Sensors must be able to infer their own position in the environment. Hence, sensors which are not able to self-localize can be coupled with sensors which have this ability (e.g., GPS, accelerometers, other).With reference to Wireless Sensor networks technology, we call also "**unknown nodes**" the sensors whose position must be computed. Similarly, we call "**beacons**" or "seed nodes" the sensors whose position is known *a priori*, which can be used as a reference for "unknown nodes".

In this chapter we will consider **geometric 2D localization** (returning x, y, ϕ position and orientation with respect to a fixed reference frame) and **topological localization** (returning $vertex_x$, the vertex of a topological representation of the environment).

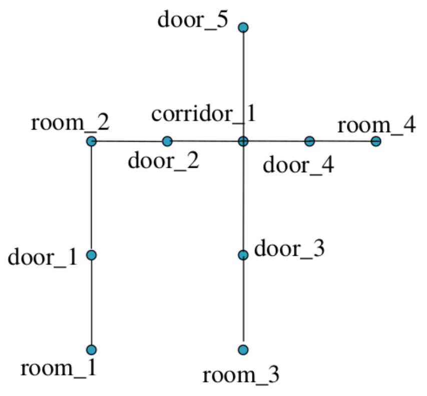
3.2 Geometric and topological localization

Geometric 2D localization $O(x, y)$ is a **fixed reference frame** and $O_m(x_m, y_m)$ is a **moving** reference frame. At time t, the goal is to express the configuration $\big(x(t), y(t), \psi(t)\big)$ of O_m with respect to O.

In the case of sensor self-localization, it is often sufficient to consider only $x(t), y(t)$ and to ignore $\psi(t)$. Differently from mobile robots, localization is not the basis for autonomous navigation. Instead, it is important for **understanding the current context** and consequently adapt the system.



Topological localization G is a **topological representation** of the environment (**graph**) with m vertices and n edges. The **vertices** represent significant locations (room, corridor, door), and the **edges** represent adjacency relationships between vertices. The **goal** is to recognize the vertex which best "represents" the sensor current position.



⇒ Topological localization is interesting since the sensor position can be expressed through **first order predicates** asserting something in the "human activity ontology":

$$is\_in(actor, room1)$$
$$linked(room1, door1)$$

⇒ **Geometrical** localization instead requires **further computation**. Ideally:

$$\begin{aligned}
&at(actor, X, Y)\\
&area(room1, Xminr, Xmaxr, Yminr, Ymaxr)\\
&at(S, X, Y) \land area(P, Xmin, Xmax, Ymin, Ymax) \land X \geq Xmino\\
&\qquad \land X \leq Xmaxo \land Y \geq Ymino \land X \leq Ymaxo \land is\_in(S, P)\\
&linked(room1, door1)
\end{aligned}$$

However, for this very reason, it **allows merging** information coming from different sources, for example using a statistical approach (e.g., Kalman Filter, Particle Filter, etc.)

3.3 Relative localization

3.3.1 Relative vs. absolute localization

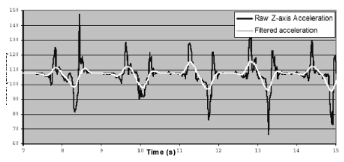
Approaches are usually classified as **relative localization** (incremental methods that measure the new position of the sensor with respect to the old position) and **absolute localization** (methods that compute the absolute position in the space without requiring a previous estimate). The two approaches can be merged to provide better results.

3.3.2 Human Odometry (1)

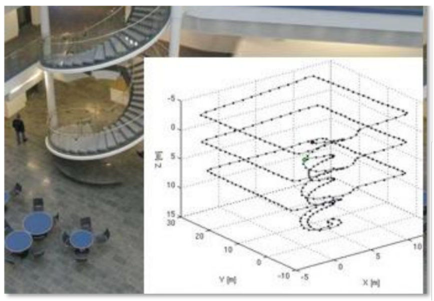
One of the methods for relative localization is the inertial navigation. Uses **gyroscopes and accelerometers** to measure rotation velocity and acceleration. These measures are **integrated** (twice for acceleration) in order to obtain orientation and position. If available, **compasses** can provide additional information to be merged.

It does not require external reference points but the **positioning error grows** with time.

In mobile robotics, **odometry** is often used for this purpose. A different approach, more similar to odometry is **counting the footsteps** of a human actor wearing an accelerometer mounted on her boot:



Acceleration experienced by a pedestrian's foot in the vertical plane whilst gently strolling. Raw and filtered data.

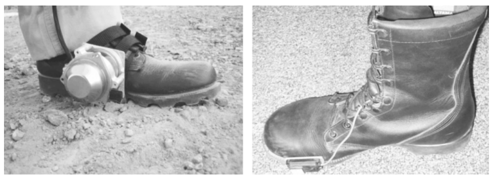


One commercially available personal navigation system based on this principle is the **Dead Reckoning Module**(DRM) (in 2007). The DRM uses accelerometers to identify steps, and linear displacement is computed assuming that the step length is constant.

In DRM heading is measured using a digital compass, which is combined with the traveled distance (step counts) to estimate 2-D position. Under this condition, **Pointresearch/Honeywell** claims an accuracy of up to 5% of the traveled distance. However, the requirement for a constant step length is **impractical**, since step size changes as a function of operational needs, fatigue, and weight carried by the user.

General Pedestrian Tracking, also called "**Personal Dead-reckoning**" (PDR) system, tracks and records or transmits the location of a walking person relative to a known starting position. The PDR system works by using an **Inertial Measuring Unit** (IMU) mounted to the user's boot. The algorithm proposed correct the drift of the accelerometers in the IMU with every step, thereby **preventing the accumulation of errors**.

When equipped with a high-grade IMU, the PDR system produces position **errors of under 2%** of distance traveled for walks of **up to 15 minutes**. Accuracy degrades gracefully in longer walks.



The main problem is that **errors increase with time**. How to solve it? With error correction through external information source?

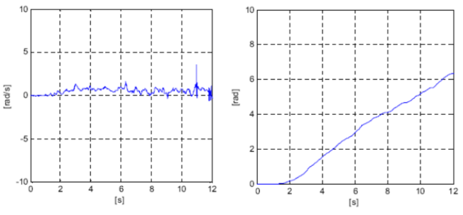


Fig. 4. The measurements taken by one gyroscope: speed (on the left) and the relative angle (on the right).

Fig. 1. IMU and laser scanner are copuled on a helmet.

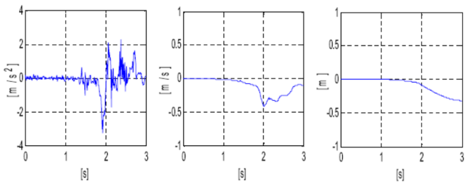


Fig. 5. The measurements taken by one accelerometer: acceleration (on the left), speed (in the middle), and position (on the right).

An idea to solve this it's the **Zero Velocity Update**: there must be a time interval $\Delta_T = [T_1, T_2]$ such that, unless the sole is slipping on the ground, A is not moving relative to the ground and the velocity vector of A is $V_A = 0$.

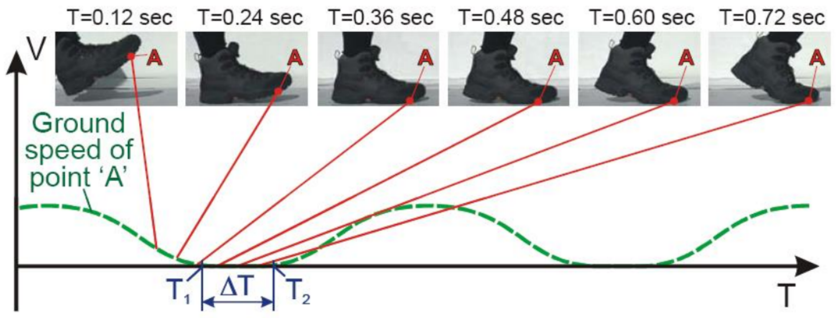
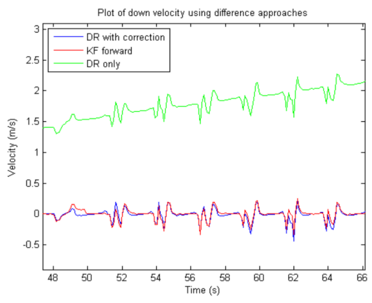


Figure 4: Key phases in a stride. During *ΔT*, all velocities and accelerations of point A in the sole of the boot are zero.

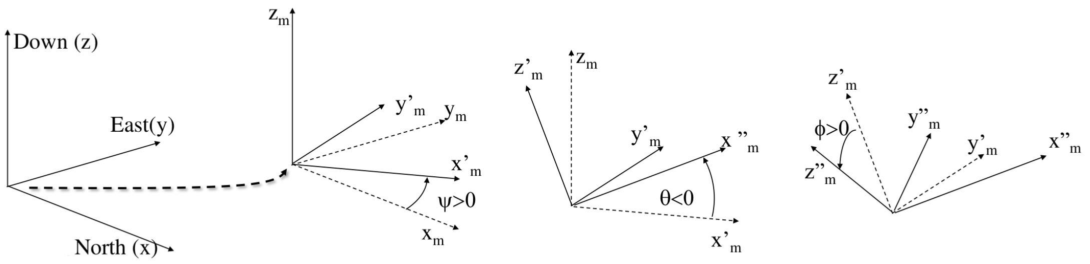
Since the condition $V_A = 0$ is maintained for the significant period of time Δ_T and not just for an instance, we reason that at least sometime during Δ_T the velocity vector of Point A is also zero. We expect the three velocities to show readings of zero during this time. If the reading is not zero, then we assume that the difference between zero and the momentary reading is the result of accumulated errors during the step interval. Sometimes, $[T_1, T_2]$ is found through additional sensors, e.g., pressure sensors.



Plot of down velocity using DR with correction, DR only and KF forward

3.3.3 Euler angles

Euler angles to express the orientation of the body frame, with respect to the fixed frame North-East-Down:



- a_x, a_y, a_z are the **linear accelerations** along the axes of the body frame (obtained by integrating accelerometers)
- v_x, v_y, v_z are the **linear velocities** of the body frame with respect to the fixed frame
- $\omega_x, \omega_y, \omega_z$ are the **rotational velocities** around the axes of the body frame (obtained through gyroscopes)
- (x, y, z) is the **position of the origin** of the body frame with respect to the the fixed frame
- ψ, θ, φ are the **Euler angles** describing the orientation of the body frame with respect to the fixed frame

The "hat" in the following pages means that we are dealing with measured values, not actual ones.

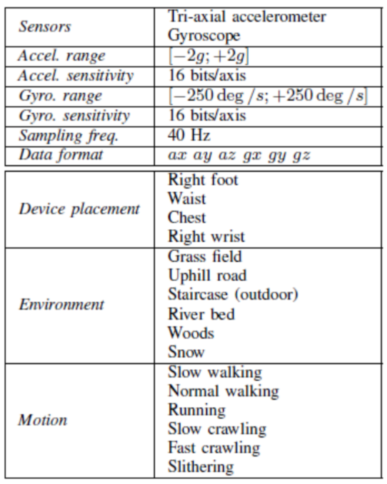
$$
\begin{aligned}
\dot{v}_x &= \hat{a}_x \cos\theta\cos\psi + \hat{a}_y(\sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi) + \\
&\quad + \hat{a}_z(\sin\varphi\sin\psi + \cos\varphi\sin\theta\sin\psi) \\
\dot{v}_y &= \hat{a}_x \cos\theta\sin\psi + \hat{a}_y(\cos\varphi\cos\psi + \sin\varphi\sin\theta\sin\psi) + \\
&\quad + \hat{a}_z(\cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi) \\
\dot{v}_z &= -\hat{a}_x\sin\theta + \hat{a}_y\sin\varphi\cos\theta + \hat{a}_z\cos\varphi\cos\theta \\
\dot{x} &= v_x \\
\dot{y} &= v_y \\
\dot{z} &= v_z \\
\dot{\varphi} &= \hat{\omega}_x + \hat{\omega}_y\sin\varphi\tan\theta + \hat{\omega}_z\cos\varphi\tan\theta \\
\dot{\theta} &= \hat{\omega}_y\cos\varphi - \hat{\omega}_z\sin\varphi \\
\dot{\psi} &= \hat{\omega}_y\frac{\sin\varphi}{\cos\theta} + \hat{\omega}_z\frac{\cos\varphi}{\cos\theta}.
\end{aligned}
$$

This system is not linear, i.e., it cannot be written as $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$ because kinematics equations **must be numerically integrated** $\rightarrow$ Euler method (other methods yield a smaller error) $\rightarrow$ h is the integration step (properly chosen)

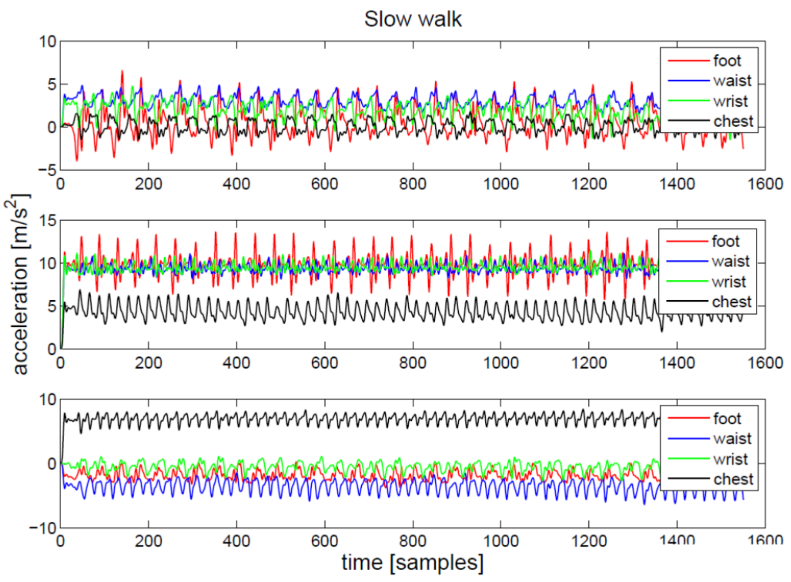
3.3.4 Human Odometry (2)

In general, walking is not the only way in which humans move in the environment, and environment itself can be different. Finally, the sensor placement can be different. Hence, how Human Odometry through step counting changes depending on **diverse outdoor** environments, **different motion types and sensor placements**?

We can perform a **performance analyses** over a **dataset** that considers 6 real environments (staircases, flat grass field, uphill road, flat rough terrain - river bed, uphill rough terrain - woods, snow), 6 motion types (slow walking, normal walking, running, slow crawling, fast crawling, slithering) and 4 sensor placements (foot, waist, wrist, chest). The dataset has been made publicly available and it's the HOOD (**Human Odometry Outdoor Dataset**). HOOD is a public dataset for the evaluation of HO systems based on accelerometer and gyroscope data. The dataset is composed of 168 trials, referring to the combinations listed previously. The same experiment was repeated twice: a person was walking along a straight line and then walking along a zig-zag path, i.e., alternatively taking left and right turns. Each trial records the 6DoF acceleration and angular rate values registered during one execution of one combination and is annotated with the number of steps effectively taken.



Sensors	Tri-axial accelerometer Gyroscope
Accel. range	$[-2g; +2g]$
Accel. sensitivity	16 bits/axis
Gyro. range	$[-250\,\deg/s; +250\,\deg/s]$
Gyro. sensitivity	16 bits/axis
Sampling freq.	40 Hz
Data format	$ax\ ay\ az\ gx\ gy\ gz$
Device placement	Right foot Waist Chest Right wrist
Environment	Grass field Uphill road Staircase (outdoor) River bed Woods Snow
Motion	Slow walking Normal walking Running Slow crawling Fast crawling Slithering



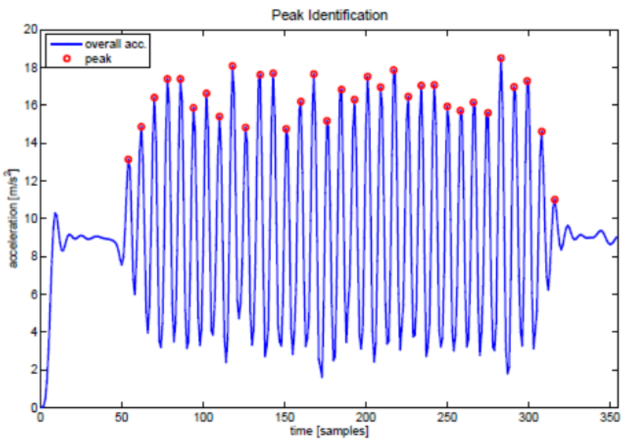
Step detection procedures analyse the accelerations generated by the motion to identify each step taken by the person, implicitly assuming that there is a periodic pattern in the signal.

Tests confirm that all motions generate a **periodic acceleration pattern** along at least one axis, for at least one sensor placement. For some placements, vertical-stance motions (walking, running) produce **clean periodic patterns** along more than one axis (as an example, consider the chest placement for run). In such cases, **merging** the information coming from different axes may **increase the step recognition accuracy**.

A well-known, simple technique is applied for step detection. The raw acceleration signals a_x, a_y, a_z and the raw angular velocity signals are first filtered with a low-pass filter to remove higher frequencies. A standard signal analysis algorithm is used to identify the peaks. Given the considerations about merging axes to increase peak identification accuracy, the overall acceleration and the overall angular velocity signals are computed as:

$$ov_{a,i} = \sqrt{a_{x,i}^2 + a_{y,i}^2 + a_{z,i}^2},$$

$$ov_{\omega,i} = \sqrt{\omega_{x,i}^2 + \omega_{y,i}^2 + \omega_{z,i}^2},$$



3.3.5 Visual Odometry

What if we have additional sensors? Other methods for relative localization are the visual odometry ones. Visual odometry uses **cameras** (single cameras, stereo cameras, or omnidirectional cameras). It is used especially in outdoor applications, e.g., planetary exploration (cameras can be mounted on the robot body, however there are no examples of such application in Ambient Intelligence).

Significant features are extracted from the picture (e.g., SIFT). When the sensor moves, angular and linear displacements returns an **incremental estimate** of the sensor position and orientation.

Scale-invariant feature transform (or **SIFT**) is an algorithm in computer vision to detect and describe local features in images (David Lowe in 1999). It is **local** and based on the appearance of the object at particular **interest points**. It is **invariant** to image scale and rotation, and **robust** to changes in illumination, noise, and minor changes in viewpoint. The object is described by set of **SIFT features** is also **robust to partial occlusion** (as few as 3 SIFT features from an object are enough to compute its location and pose).

Most existing approaches to visual odometry are based on the following stages:

1. Acquire input images: using either single cameras, stereo cameras, or omnidirectional cameras
2. Image correction: apply image processing techniques for lens distortion removal, etc
3. Feature detection: define interest operators, and match features across frames and construct optical flow field (no long term feature tracking)

– Feature extraction and correlation (Lucas–Kanade method).

– Construct optical flow field

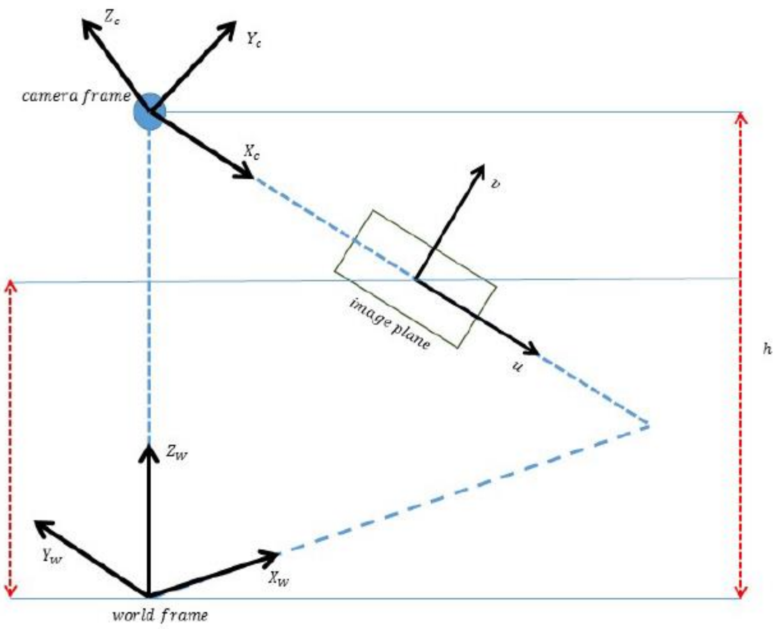
4. Check flow field vectors for potential tracking errors and remove outliers
5. Estimation of the camera motion of the camera from the optical flow

   – Choice 1: Kalman filter for state estimate distribution maintenance

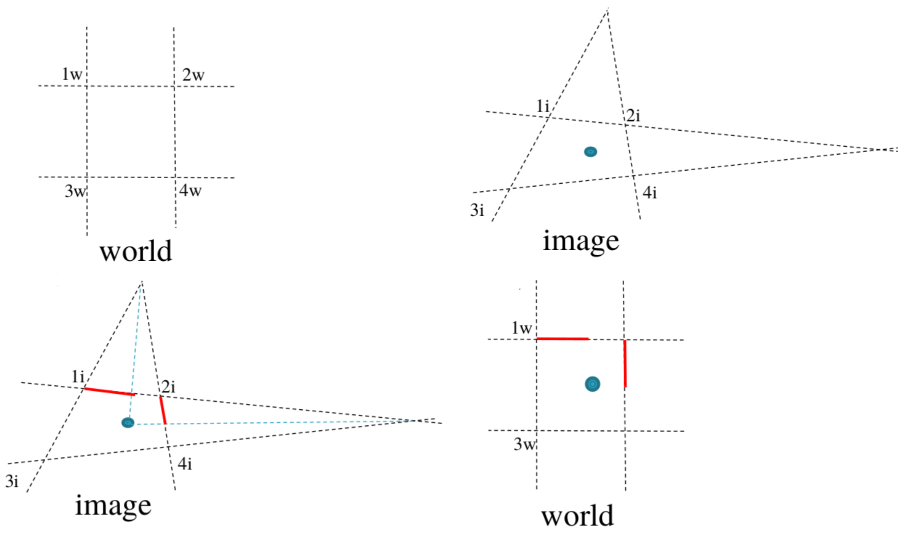
   – Choice 2: find the geometric and 3D properties of the features that minimize a cost function based on the re-projection error between two adjacent images. This can be done by mathematical minimization or random sampling.
6. Periodic repopulation of trackpoints to maintain coverage across the image.

3.4 Absolute localization

Stereo vision returns 3D spatial information. If some assumptions can be made (e.g. objects and actors are on a flat terrain) we can **segment object** from the background with a standard camera and **apply** inverse perspective mapping (**IPM**). This is very efficient, since it is **based on a lookup table** that defines correspondences between pixels and real world coordinates.



IPM localization with a fixed camera:

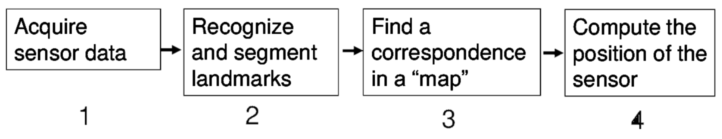


3.4.1 Landmarks

Landmarks are significant features of the environment in known position (doors, lamps, etc.). The sensor **knows the position** of all landmarks in the environment and computes its own position on the basis of the detected features.

They can be natural landmark (environmental features, SIFT) or artificial landmarks (beacons). In general, landmarks are chosen/designed in such a way as to be recognized easily (e.g., when using cameras, they have a high contrast with the background).

The general problem can be described as described in the picture:


Acquire sensor data
Recognize and segment landmarks
Find a correspondence in a "map"
Compute the position of the sensor
1
2
3
4

Step 2 is usually complex with natural landmarks: this is not the case with artificial landmarks.

Landmarks can be used in different way:

- **Topological localization**: a single landmark is sufficient

  sensors: RFID tags, RF antenna (through signal detection), cameras, etc...

- Geometrical localization with **trilateration**: three landmarks are necessary in the 2D plane

  sensors: RF antenna (through signal strength measurement), cameras, microphones, WSN etc...

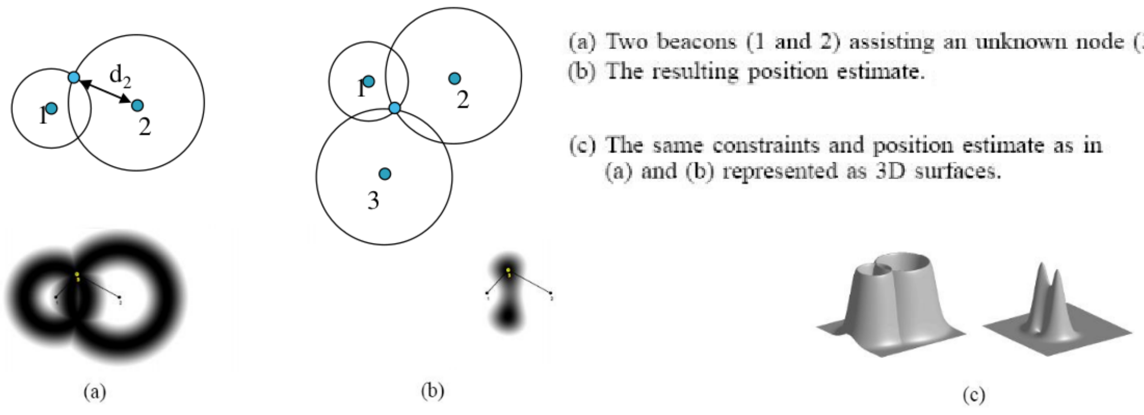
- Geometrical localization with **triangulation**: three landmarks are necessary in the 2D plane

  sensors: directional RF antenna, cameras, etc...

3.4.2 Trilateration

The general idea is to measure the distance from landmarks/beacons-seed nodes. Unknown nodes require to compute the distance from at least 3 seed nodes (2 seed nodes if an approximate estimate is available).

Each landmark defines a circumference; by intersecting them a single point is obtained:



(a) Two beacons (1 and 2) assisting an unknown node (3).
(b) The resulting position estimate.

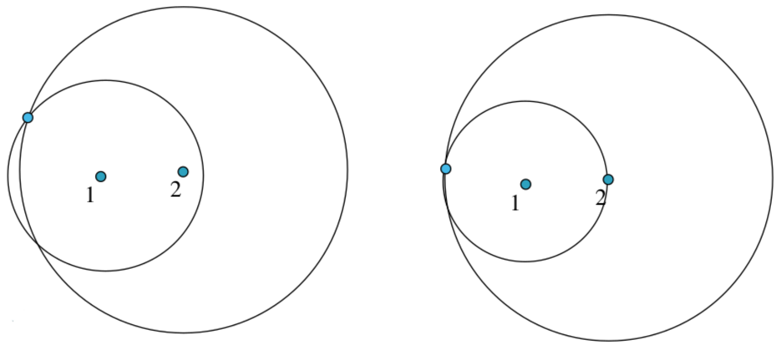
(c) The same constraints and position estimate as in (a) and (b) represented as 3D surfaces.

The position of landmarks must be known a priori (e.g., in a map). It is possible to take into account measurement errors, by considering the mean value and the standard deviation.

Formally, it is necessary to solve the following system:

$$\begin{cases}(x-x_1)^2+(y-y_1)^2=d_1^2\\(x-x_2)^2+(y-y_2)^2=d_2^2\end{cases}$$

Whichever is the sensor adopted, the approach has the well known problem of error sensitivity due to its non-linearity: in some configurations, a small error in distance measurements can produce big errors in position.



If more beacons are available, it is possible to **use the additional information**, for example, by using **Least Squares Methods**:

$$\begin{cases}(x-x_1)^2+(y-y_1)^2=d_1^2\\ \vdots\\(x-x_i)^2+(y-y_i)^2=d_i^2\\ \vdots\\(x-x_n)^2+(y-y_n)^2=d_n^2\end{cases}$$

$$\min_{x,y}\left\{\left[(x-x_1)^2+(y-y_1)^2-d_1^2\right]^2+\ldots+\left[(x-x_i)^2+(y-y_i)^2-d_i^2\right]^2+\ldots+\left[(x-x_n)^2+(y-y_n)^2-d_n^2\right]^2\right\}$$

A **camera mounted on the moving sensor** or on the human body can ideally be used to measure distances with natural and artificial landmark. With artificial landmark, if the shape of each landmark is known, it is easy to compute the distance. Otherwise SIFT features can be used.

Radio frequency signal strength can be used as a measure of the distance from beacons (ZigBee, Bluetooth, Wi-Fi, active RFID tags, etc...).[1] Beacons broadcast information about their own position in a fixed reference frame. Unknown nodes work only as relay.

The transmission power at the transmitting device (**PTX**) directly affects the receiving power at the receiving device (**PRX**). The detected signal strength decreases quadratically with the xdistance to the sender. Next the Received Signal Strength indicator is computed ($P_{Ref} = 1mW$):

$$P_{RX} = P_{TX} \cdot G_{TX} \cdot G_{RX} \left(\frac{\lambda}{4\pi d} \right)^2 \qquad RSSI = 10 \cdot \log \frac{P_{RX}}{P_{Ref}}$$

P_{TX}	= Transmission power of sender
P_{RX}	= Remaining power of wave at receiver
G_{TX}	= Gain of transmitter
G_{RX}	= Gain of receiver
λ	= Wave length
d	= Distance between sender and receiver

In practical scenarios, the **ideal distribution of PRX is not applicable**, because the propagation of the radio signal is interfered with a lot of influencing effects (e.g. reflections on metallic objects, superposition of electro-magnetic fields, diffraction at edges, refraction by media with different propagation velocity, polarization of electro-magnetic fields, unadapted MAC protocols, inapplicable receiving circuits...).

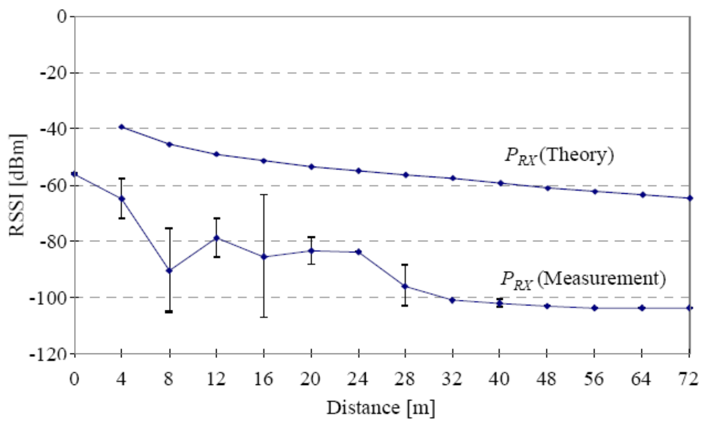


Figure 1: a) RSSI as quality identifier of the received signal power P_{RX} b) Received Signal Strength of a Chipcon CC1010 sensor node

Another possibility, if available, is to measure **link quality indicator** (**LQI**) of the transmission. According to IEEE 802.15.4, LQI is a characterization of the strength and/or the quality of a received packet. It must be proportional to signal level (**RSSI**, a signal-to-noise estimation or a combination of these methods and shall be a value between 0 and 255) and, again, experiments show that LQI is indirect proportional to distance, but outliers are present.

[1]The Figures in the following refer to ZigBee

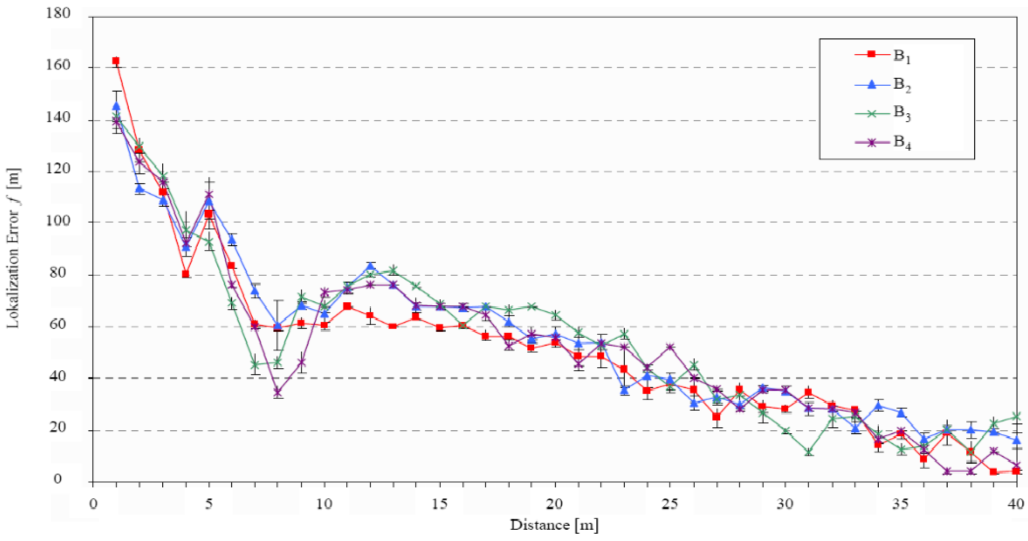


Figure 2: LQI vs. distance d between Zigbee-based sensor nodes B_1..B_4 and a coordinator node (CC2420DB) in 20 loops, ideal outdoor environment

In this case, trilateration can be problematic. Embedded algorithms such as CL use **centroid determination** to calculate their own position:

- In the first phase, all beacons send their position $B_j(x, y)$ to all sensor nodes within their transmission range.
- In the second phase, all sensor nodes calculate their own position $P_i(x, y)$ by a centroid determination from all n positions of the beacons in range.
- The localization error $f_i(x, y)$ is defined as distance between the exact position $P_i(x, y)$ and the approximated position $P_i(x, y)$ of a sensor node.
- The idea is to integrate more information sources, by ignoring the information about distance at all.

$$P_i^{'}(x,y) = \frac{1}{n}\sum_{j=1}^{n} B_j(x,y) \qquad f_i(x,y) = \sqrt{(x'-x)^2 + (y'-y)^2}$$

We saw the **centroid algorithm**, now we can introduce as well the **weight centroid algorithm**:

- The weight w_{ij} is a function depending on the distance and the characteristics of the sensor node's receivers.
- In WCL, shorter distances are more weighted than higher distances. Thus, w ij and d ij are inversely proportional
- As an approximation, the correlation is equivalent to the function $1/d$
- To weight longer distances marginally lower, the distance is raised to a higher power of g

$$P_i^{''}(x,y) = \frac{\sum_{j=1}^{n}\left(w_{ij} \cdot B_j(x,y)\right)}{\sum_{j=1}^{n} w_{ij}}$$

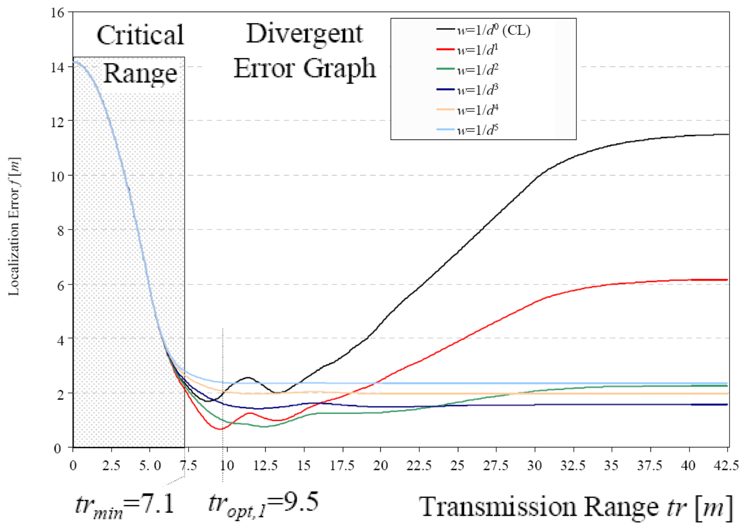


Figure 3: Simulation of the localization error versus transmission range tr with different weight functions in a full equipped sensor network enclosed by 4x4 beacons (dimension: 30mx30m, fq=10m).

Due to using distances only as additional weights, the **Weighted Centroid Localization** (WCL) is featuring a very high robustness against scaling errors. In comparison with the least squares method (LS) the WCL algorithm results in smaller localization errors. As the following figure visualizes, only correct scaling leads to a smaller localization error in comparison to the weighted centroid localization (in all other cases, WCL yields smaller errors):

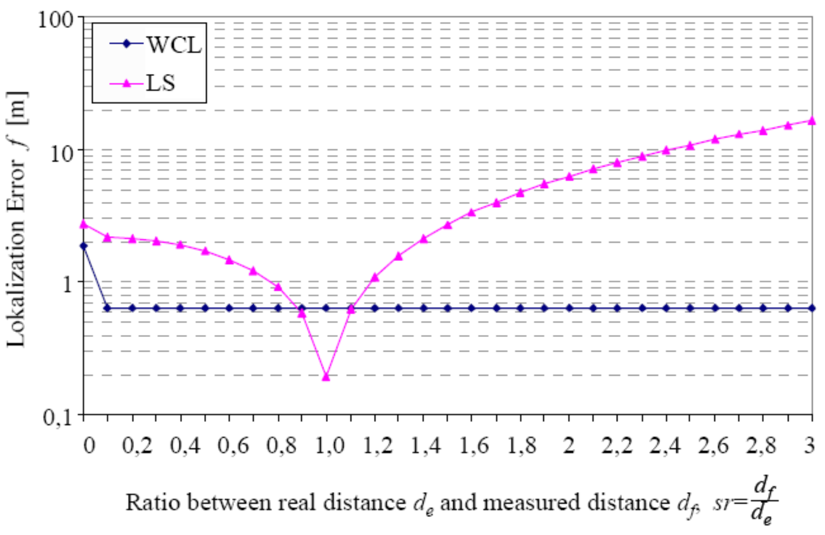


Figure 4: Localization error vs. scaling ratio of WCL and Least Squares.

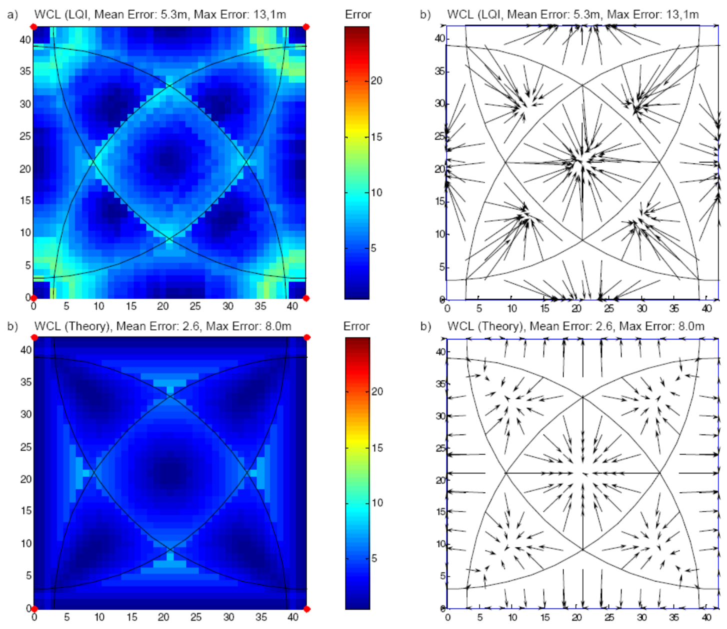
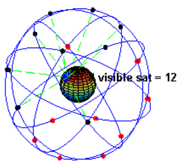


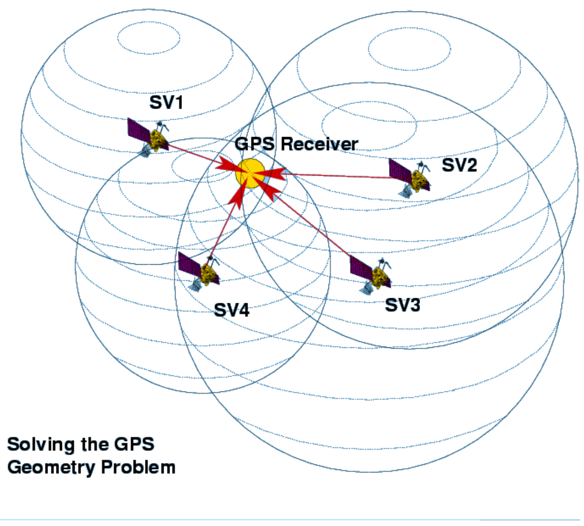
Figure 6: Top view and error vectors of localization errors
using Weighted Centroid Localization (WCL) based on
a,b) Link Quality Indicator (LQI) in Zigbee- based devices (CC2420)
c,d) theoretical model with exact distances in a sensor network of 43mx43m

3.4.3 Global Positioning System (Trilateration)

GPS is constituted of a constellation of 24 satellites with an orbit around the world of 12 hours. GPS can provide users with **longitude, latitude, altitude, and time information** (as well as velocity), given that **at least 4 satellites** are visible from the receiver.

GPS is made up of three parts: satellites, tracking stations and the GPS receivers owned by users. The localization algorithms is basically trilateration based on **"pseudo-range" measurements**: the Master Control station measures signals from the satellites to incorporate into precise orbital mathematical models, which are then used to compute **corrections** for the clocks on each satellite. These corrections, and orbital (ephemeris) data are then uploaded to the satellites, which then transmit them to GPS user's receivers.

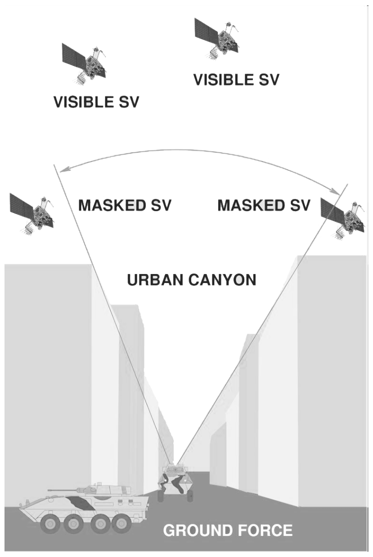




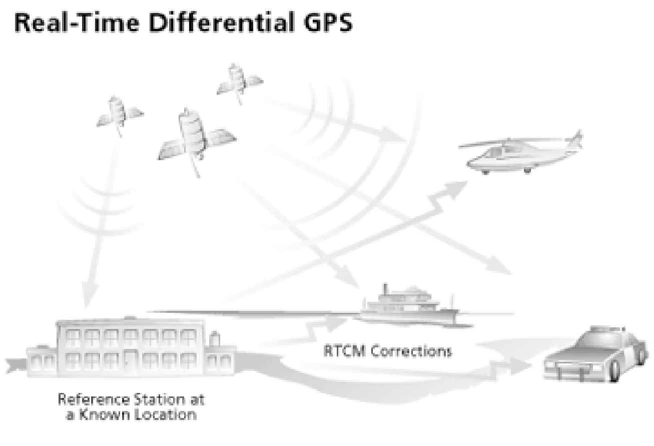
Common (non differential) GPS has an accuracy within 15 meters, which is usually considered suffcient for car navigation systems. More specifically, GPS measurements are corrupted by different error sources, which degrade the positioning accuracy:

- ionospheric effects, i.e., changes in the speed of the GPS signals as they pass through the ionosphere, as a consequence of changing atmospheric conditions
- inaccurate computation of the satellites position due to ephemeris errors
- inaccurate synchronization between the satellite and the receiver clock
- tropospheric effects, i.e., changes in the speed of the GPS signals depending on humidity
- multipath distortion, due to reflecting surfaces in the immediate surroundings of the receiver
- signal blockage, i.e., when the amount of visible satellites changes due to occlusion (e.g., because of tall buildings), and possibly becomes insufficient for positioning
- numerical errors

A well-known problem with GPS: "Urban Canyons"

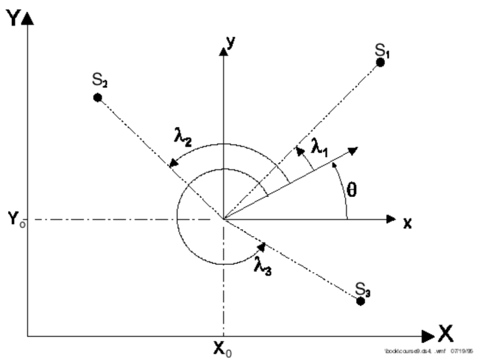


D-GPS (**Differential GPS**) is essentially a system to provide positional corrections to GPS signals. DGPS uses a fixed, known position to adjust real time GPS signals to eliminate pseudorange errors


Real-Time Differential GPS
RTCM Corrections
Reference Station at a Known Location

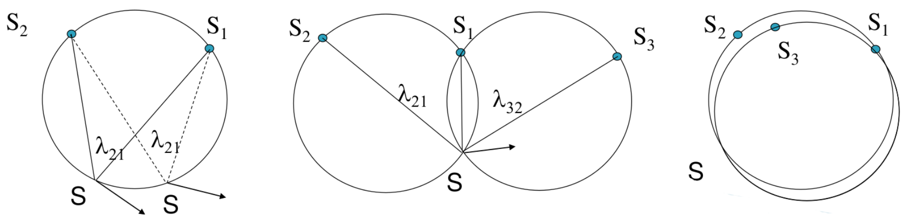
3.5 Triangulation

Triangulation is similar to trilateration, but it requires **angular information**. This information can be acquired with camera, but for RF localization a directional antenna is required. Again, 3 or more beacons are required $\rightarrow$ the three azimuth angles $\lambda_1, \lambda_2, \lambda_3$ are recorded:



To compute the position through triangulation it is possible to use

1. Pothenot method (or Snellius method), used in topography
2. Circle intersection (considered here)



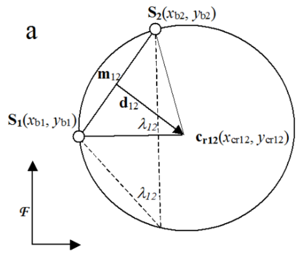
$\lambda_{21} = \lambda_2 - \lambda_1$ defines a circumference
$\lambda_{32} = \lambda_3 - \lambda_2$ defines another circumference

- In fact, given a circumference, all angles at the circumference $\lambda_{21}(\lambda_{32})$ standing on the same arc $S_2S_1(S_3S_2)$ are identical
- Inversely, the angle $\lambda_{21}(\lambda_{32})$ and the arc $S_2S_1(S_3S_2)$ define univocally a circumference with the previous property
- By intersecting the two circumferences it is possible to infer positions (one position is invalid since it corresponds to a beacon).
- Next, orientation can be computed by considering one of the three angles $\lambda_1, \lambda_2, \lambda_3$
- In practice, there are efficient ways for doing this analytically [2]

[2] **Circle intersection algorithm (mathematics details)**: let us imagine, for sake of simplicity, to choose angles λ_{12} and λ_{23}, and to consequently consider the circles C_{r12} and C_{r23}. First, we compute the centre of C_{r12}: by elementary geometric considerations, we know that the angle between $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$, when measured from the center $\boldsymbol{c}_{r12}$, is doubled with respect to λ_{12}. Thus, the distance between $\boldsymbol{c}_{r12}$ and S_1S_2 can be computed as follows:

$$d_{12} = 0.5\sqrt{(x_{b2} - x_{b1})^2 + (y_{b2} - y_{b1})^2}\,\tan(\lambda_{12})$$

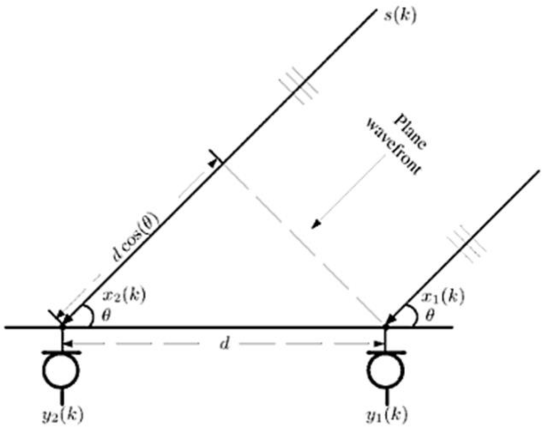
a



After some computations, this yields the following expression for $\boldsymbol{c}_{r12}$ coordinates (the same can be done for $\boldsymbol{c}_{r23}$)

3.5.1 Microphone triangulation

Microphone arrays can be used for triangulation.
Case 1: Sound source is in the **far field**. We detect only the direction, but not the distance:



Case 2: Sound source is in the **close field**. We detect the direction, and the distance:

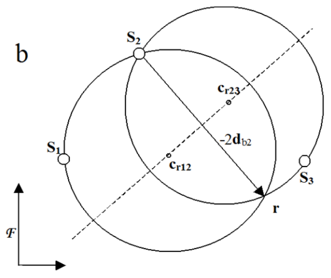
$$x_{cr12} = 0.5\left(x_{b1} + x_{b2} + \frac{y_{b2} - y_{b1}}{\tan \lambda_{12}} \right) \qquad y_{cr12} = 0.5\left(y_{b1} + y_{b2} - \frac{x_{b2} - x_{b1}}{\tan \lambda_{12}} \right)$$

There are some singularities in the equation: $\lambda_{12} = 0, \pi$ corresponds to a circle with infinite radius (i.e., collapsing to a straight line), a singularity which can be handled by considering a different couple of landmarks.
Next we compute the equation of the straight line connecting the two centres

$$-\left(x_{cr23} - x_{cr12}\right) y + \left(y_{cr23} - y_{cr12}\right) x - y_{cr23}x_{cr12} + y_{cr12}x_{cr23} = 0$$

The distance d_{b2} between C_{r12} C_{r23} and S_2 (i.e., the first intersection of $\boldsymbol{c}_{r12}$ and $\boldsymbol{c}_{r23}$)

$$d_{b2} = \frac{-\left(x_{cr23} - x_{cr12}\right) y_{b2} + \left(y_{cr23} - y_{cr12}\right) x_{b2} - y_{cr23}x_{cr12} + y_{cr12}x_{cr23}}{\sqrt{\left(y_{cr23} - y_{cr12}\right)^2 + \left(x_{cr23} - x_{cr12}\right)^2}}$$

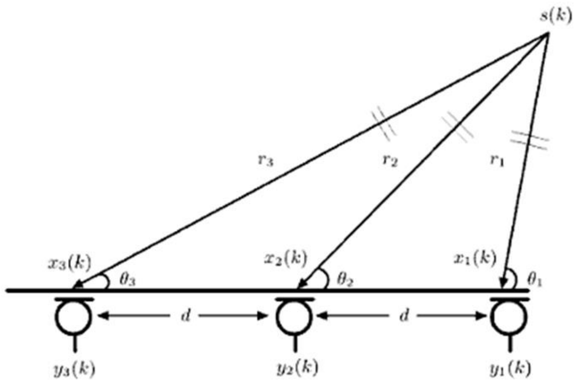


Since the second intersection (i.e., corresponding to x_r , y_r) is symmetrical to S_2 with respect to $\boldsymbol{c}_{r12}\boldsymbol{c}_{r23}$, we compute it by adding to S_2 a vector $\boldsymbol{d}_{b2}$ which is $\perp$ to $\boldsymbol{c}_{r12}$ $\boldsymbol{c}_{r12}$ and is doubled than $\boldsymbol{d}_{b2}$ (but has an opposite sign)

$$x_r = x_{b2} - 2d_{b2}\left(y_{cr23} - y_{cr12}\right) \Big/ \sqrt{\left(y_{cr23} - y_{cr12}\right)^2 + \left(x_{cr23} - x_{cr12}\right)^2}$$
$$y_r = y_{b2} + 2d_{b2}\left(x_{cr23} - x_{cr12}\right) \Big/ \sqrt{\left(y_{cr23} - y_{cr12}\right)^2 + \left(x_{cr23} - x_{cr12}\right)^2}$$

To compute the orientation ψ

$$\psi = \arctan\left(\frac{y_{b2} - y_r}{x_{b2} - x_r} \right) - \lambda_2 = \arctan\left(-\frac{x_{cr23} - x_{cr12}}{y_{cr23} - y_{cr12}} \right) - \lambda_2$$



How can we compute distances r_1, r_2, r_3 and angles q_1, q_2, q_3. Let c be a constant corresponding to the velocity of the sound.

$$
\begin{aligned}
&t_{12} = \frac{(r_2 r_1)}{c} \qquad t_{23} = \frac{(r_3 r_2)}{c} \\
&r_2^2 = r_1^2 + d^2 + 2dr_1 \cos\theta_1 \quad \text{(cosine rule)} \\
&r_3^2 = r_1^2 + 4d_2 + 4dr_1 \cos\theta_1 \quad \text{(cosine rule)}
\end{aligned}
$$

θ_2 and θ_3 computed with the sine rule.

$\Rightarrow$ we can "zoom" with the microphone! Every possible sound source in the environment produces a sound. If you know that and you want to focus on that area, you shift the signals w.r.t the computed delays in that area!

3.6 Statistical approaches

In presence of measurement errors, many approaches to localization rely on the idea of merging data coming from different sensors in a statistical framework.

In particular, incremental approach are merged with absolute approaches; an estimate is first produced incrementally, and it is subsequently refined through absolute measurement. In this way:

- The errors which are inherent in incremental approaches are periodically reduced through absolute measurements
- Absolute approaches can take benefit of having an approximate estimate of the sensor position, which can be corrected as soon as a single absolute measurement is available (instead of waiting for more measurement to be available, e.g., trilateration or triangulation)

Different approaches are available but we consider here two of them: the **Kalman Filter** and the **Particle Filter**

3.6.1 Introduction

The general idea (in the 2D plane) is to **compute** a probability density/distribution that describes, for each point in the space (x, y, ψ), the probability that the sensor is in the corresponding configuration. Such density/distribution is updated whenever new sensorial data are received:

- The Kalman Filter takes into account the first two moments of the probability density (mean and covariance)

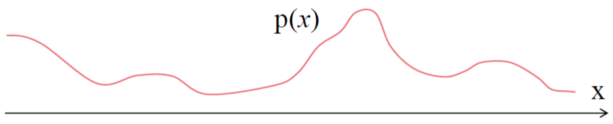
  KF is adequate only for **position tracking**, since it requires an estimate a priori of the position.
- Particle Filters takes into account the whole probability distribution.

  PF can be used for **Global Localization**, I.e., when an initial estimate is absent.

Hence, while in Kalman Filter you describe a probability distribution only with a mean and covariance (describing only a certain probability distribution $\rightarrow$ good only for position tracking), the particle filter can be used instead for general localization, when we don't give an initial estimate or multiple hypothesis distributions.

3.6.2 Kalman filter

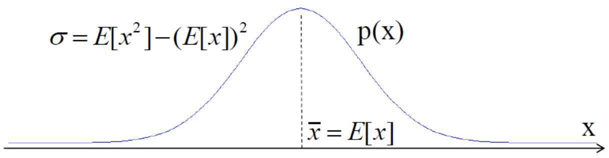
One-dimensional case: the state to be estimated comprises only the x component.



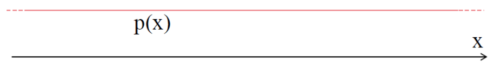
In this case, we need methods to describe the probability density of x taking into account uncomplete knowledge/errors in measurement. We will use the **probability density** to compute an estimate of x to be fed to other tasks and algorithms (the most probable state?) The Kalman Filter **assumes** that the probability density can be described as a **Gaussian**

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\dfrac{(x-\bar{x})^2}{2\sigma^2}}$$

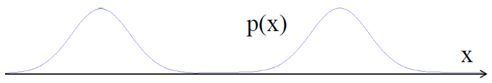
The expected value of x is used as an estimated of the state to be fed to other tasks and algorithms:



However, the assumption about the shape of the probability density does not always hold... Suppose, for example, that we do not have any idea about the initial value of the state (**kidnapped robot problem**[3])...



...or there exist two symmetrical configuration and we want to take into account both hypotheses...



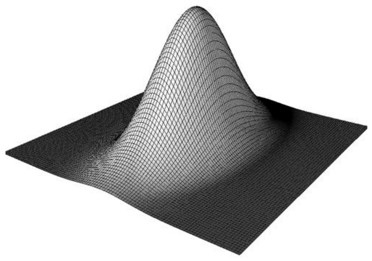
Two-dimensional case: $x = (x, y)$ is a vector with two components. The probability density function can be expressed as

$$K^{-\frac{1}{2}(\boldsymbol{x}-\hat{\boldsymbol{x}})^T \Sigma^{-1} (\boldsymbol{x}-\hat{\boldsymbol{x}})}$$

where K is a normalizing factor and

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix} = \begin{bmatrix} E[(x-\bar{x})^2] & E[(x-\bar{x})(y-\bar{y})] \\ E[(x-\bar{x})(y-\bar{y})] & E[(y-\bar{y})^2 \end{bmatrix}$$

[3]Kidnap the robot problem: you ideally steal put the robot, put it in a bag with no localization possibility and then you release it $\rightarrow$ General Localization problem



The eigenvector are the two axis of the ellipse $\rightarrow$ that's why normally we represent it as an ellipse of uncertainty rather that 3D bell shaped.

The Kalman Filter solves the general problem of estimating the state $x \in \mathbb{R}^n$ of a controlled time discrete process through available measure. The process is described by the following linear[4] finite state equation (w, v are noises)

$$x_{k+1} = Ax_k + Bu_k + w_k$$

and the measure $o \in \mathbb{R}^m$ is given by

$$o_{k+1} = Hx_{k+1} + v_{k+1}$$

The random variables w_k and v_k represent, respectively, **additional noise** on the **process and** on the **measure**. It is assumed that they are **independent AWG** noises

$$p(w) = N(0, Q) \qquad p(v) = N(0, R)$$

In theory, **covariance matrices** Q (covariance of the process noise) and R (covariance of the measurement noise) could be different at every time instant, but they are **assumed to be constant**.

Matrix A $(n \times n)$ in the state equations describes the state at instant k depending on the state at the previous time instant $k-1$ in **absence of control inputs and noise**. It is **assumed** here that A does **not change in time**.

Matrix B $(n \times l)$ puts in relation the **control input** $u \in \mathbb{R}^l$ **with the state** x. It is assumed here that B does **not change in time**.

Matrix $H(m \times n)$ in the measurement model puts in relation **the state with measure** o_k. It is assumed here that H does **not change in time**.

Example: Localization of a moving person with beacons. It is obviously possible to use the Kalman Filter with every kind of sensor: it is sufficient to be able to model the measurement. $x \in \mathbb{R}^3$ is the state to be estimated (coordinates x, y, ψ in the 2D plane). Suppose that the person is wearing a IMU. By oversimplifying, assume that at time instant k, the IMU measures a control input u_k. In general, the state can be updated as

$$x_{k+1} = f(x_k, u_k, w)$$

Assume that the system is linear: a time instant $k+1$, the state can be updated as

$$x_{k+1} = Ax_k + Bu_k + w_k$$

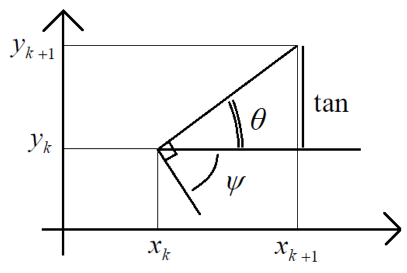
(noise w_k takes into account the fact that Bu_k does not correspond
to the real displacement in the world)

$o_{k+1} = o_i \in \mathbb{R}^l$ is the measurement, i.e., the azimut angle l of the i^{th} beacon.

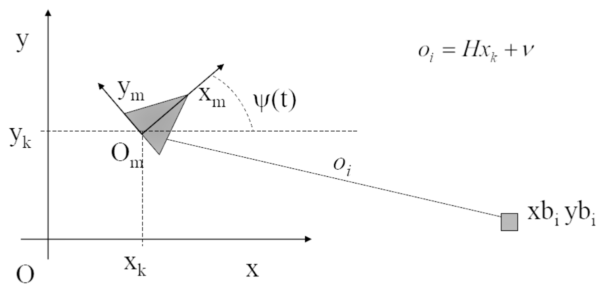
The relationship between o_i and x_k (x_k, y_k, ψ_k) is obviously

[4]The Kalman Filter relies on a **linear assumption**! We have to use the **Extended Kalman Filter** to do a sort of "linearization"

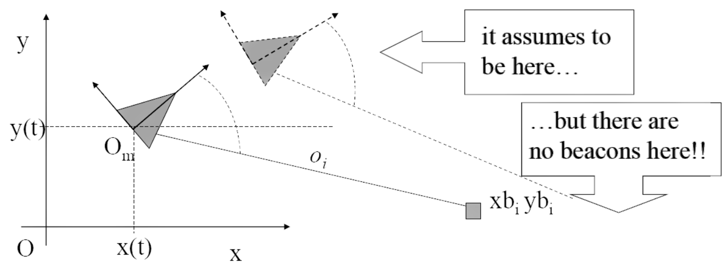
$$o_i = h(x_k, y_k, \psi_k, i) = \tan^{-1}_{(-\pi,\pi)}\left(\frac{yb_i - y_k}{xb_i - x_k}\right) - \psi_k + \nu$$



Assume however that, for every beacon i, such relation is linear, i.e. in the following figure



How it corrects its errors?



We define $\hat{\boldsymbol{x}}_k^- \in \mathbb{R}^n$ (notice the superscript "minus") as the a priori estimate of the state a time k. This estimate follows from knowing the process in instants which precede k and from the control input u_k a time k. We define $\hat{\boldsymbol{x}}_k \in \mathbb{R}^n$ (without the superscript "minus") the a posteriori estimate at time k. This estimate follows from the a priori estimate and the measurement o_k at time k.

We define the error of the **a priori estimate and the error of the a posteriori estimate** as:

$$\Delta\boldsymbol{x}_k^- = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-$$
$$\Delta\boldsymbol{x}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k$$

The covariance of the a priori error is defined as:

$$P_k^- = E[\Delta\boldsymbol{x}_k^- \Delta\boldsymbol{x}_k^{-T}]$$

whereas the covariance of the a posteriori error is defined as

$$P_k = E[\Delta\boldsymbol{x}_k \Delta\boldsymbol{x}_k^T]$$

At time k+1, the a posteriori estimate $\hat{\boldsymbol{x}}_{k+1}$ is computed as a lin. combination of the a priori estimate $\hat{\boldsymbol{x}}_{k+1}^-$ and a weighted difference between the current measurement o_{k+1} and a prediction of the measurement $H\hat{\boldsymbol{x}}_{k+1}$

$$\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_{k+1}^- + K(o_{k+1} - H\hat{\boldsymbol{x}}_{k+1}^-)$$

In our case (beacons):

- $\hat{\boldsymbol{x}}_{k+1}^{-}$ (a priori estimate) is provided by the IMU
- o_{k+1} (measurement) is the azimuth direction of the i th beacon
- $\hat{\boldsymbol{x}}_{k+1}$ is the new a posteriori estimate (which will be used to update dead reckoning)

The difference between measurement and prediction is called **innovation**:

$$o_{k+1} = H\hat{\boldsymbol{x}}_{k+1}^{-}$$

Innovation reflects the difference between the predicted measurement and the measurement that has been really done. When innovation is null, it means that there is a perfect agreement between prediction and actual measurement that is, the estimate of the position corresponds to the real position.

It is required to compute K, which determines the relative weight of the a priori estimate and the innovation in computing the a posteriori estimate. The matrix K is chosen in such a way as to minimize the covariance of the a posteriori error P_k. A possibility is:

$$K_{k+1} = P_{k+1}^{-}H^T(HP_{k+1}^{-}H^T + R)^{-1}$$

In this way, the a posteriori estimate minimizes the a posteriori error (the Kalman Filter is optimal). Here it is not reported why this choice for K leads to optimality.

Some observations:

- When the measurement error covariance R tends to zero, K weights innovation more:

  $$\lim_{R_k \to 0} K_{k+1} = H^{-1}$$

  $\rightarrow$ Intuitively, the system relies **more on the measurement**
- Viceversa, when the covariance of the a priori (state) error P_k^{-} tends to zero, K weights innovation less:

  $$\lim_{P_k \to 0} K_{k+1} = 0$$

  $\rightarrow$ Intuitively, the system relies **more on the prediction**

Kalman Filter Algorithm Let us describe the algorithm to update the estimate of the state through the Kalman Filter:

- The Filter keeps in memory the first two moments of the probability distribution x_k:

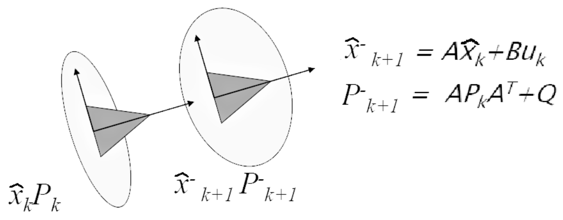
  $$E[\boldsymbol{x}_k] = \hat{\boldsymbol{x}}_k \qquad \text{(a posteriori estimate of the state)}$$
  $$E[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^T] = P_k \qquad \text{(a posteriori estimate of the error covariance)}$$

- Kalman Filter equations can be divided into two groups:

  Equations for **updating the estimate in time**, which are responsible of projecting forward in time the current estimate of the state and of the error covariance

  Equations for **updating the estimate as a consequence of measurements**, which are responsible of incorporating a new measurement in the a priori estimate, with the purpose of obtaining a new a posteriori estimate.

1. **Time update phase**:


$\hat{x}^{-}_{k+1} = A\hat{x}_k + Bu_k$
$P^{-}_{k+1} = AP_kA^T + Q$
$\hat{x}_k P_k$
$\hat{x}^{-}_{k+1} P^{-}_{k+1}$

Filter inizialization: (x_0, P_0): x_0P_0 chosen by the user or computed through **triangulation**

2. **Measurement update phase**:

$$
\begin{aligned}
K_{k+1} &= P^-_{k+1} H^T (H P^-_{k+1} H^T + R)^{-1} \\
\hat{x}_{k+1} &= \hat{x}^-_{k+1} + K_{k+1}(o_{k+1} - H \hat{x}^-_{k+1}) \\
P_{k+1} &= (I - K_{k+1} H) P^-_{k+1}
\end{aligned}
$$

   (a) Compute the Kalman gain K_k
   (b) Acquire measurement o_{k+1}
   (c) Produce the a posteriori estimate of the state
   (d) Produce the a posteriori estimate of the error covariance

Problems with the Kalman Filter:

- It is not always obvious how to estimate P and R ; moreover, the process and measurement noise do not necessarily have a Gaussian profile.
- It assumes that the process and the measurement are linear (not true in this case). It is necessary to use the Extended Kalman Filter, which linearize the process and /or the measurement in the current estimate. EKF is no more optimal.
- The approach is adequate when the whole pdf has a Gaussian profile, I.e., for position tracking, not for global localization.

An alternative is the Particle Filter.

3.6.3 Particle Filter

The main objective of particle filtering is to **"track" a variable of interest** as it evolves over time. It works with non Gaussian and potentially multi-modal pdf.

The idea is to **represent the posterior probabilities by a set of randomly chosen[5] weighted samples**. It constructs a sample-based representation of the entire pdf. Such representation is modified as the state evolves and/or when measurements are available.

Increasing number of samples implies (almost sure) convergence to true pdf.

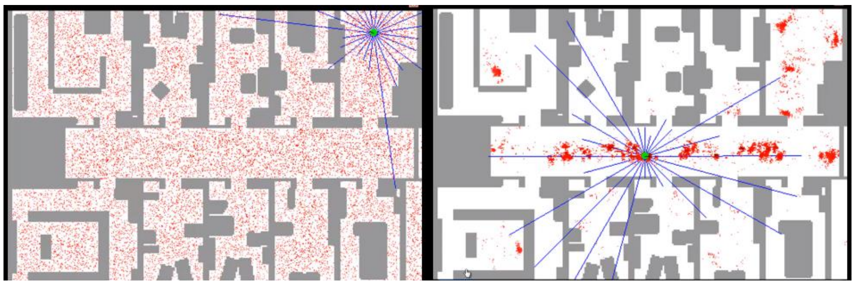
In particular, to do this, multiple copies of the variable of interest are used, each associated with a weight that signifies its quality. An estimate of the variable of interest is obtained by the weighted sum of all particles. The algorithms is recursive and operates in two phases: prediction and update.

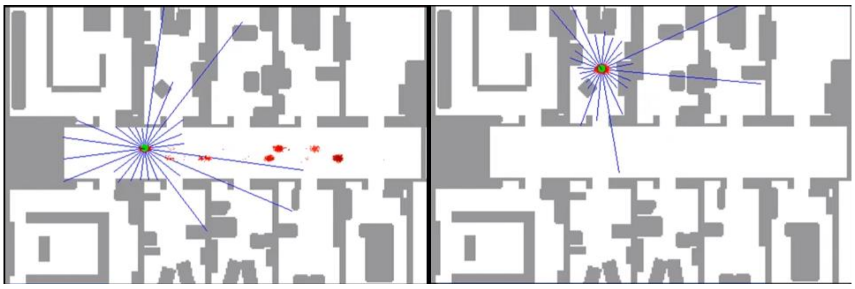
- Prediction: after each action, each particle is modified according to the existing model by adding noise
- Update: each particle's weight is re-evaluated based on the sensory information, by eliminating particles with small weights (re-sampling)

More formally:

- The variable of interest (the pose of the person at time k)
$$\boldsymbol{x}^k = [x^k, y^k, \psi^k]^T$$
is represented as a set of M samples (particles) S^k
- $S^k = [x^k_j, w^k_j], \quad j = 1...M$ where each particle is a copy of the variable of interest and w_j defines the contribution of this particle to the overall estimate of the variable.
- If at time $t = k + 1$ we know the pdf of the system at the previous instant $t = k$ it is possible to model the effect of the action to obtain an a priori estimate at time $t = k + 1$ (**prediction phase**)
- Next, the information obtained for sensing is used to update the particle weights in order to accurately describe the pdf (**update phase**)
- Finally, a method of evaluation is used to obtain an estimate of the pose starting from all particles and their corresponding weights (e.g., weighted mean, best particle, weighted mean in a small window around the best particle)

[5]"randomly chosen": "Monte Carlo" method (we are playing roulette / throwing the dice)





```
Require: A set of Particles at time 0: $S^0 = [\mathbf{x}_j, w_j : j = 1 \dots M]$.
  $W = w_j : j = 1 \dots M$
  while (Exploring) do
    $k = k + 1$;
    if ($\underline{\text{ESS}}(W) < \beta * M$) then {Particle Population Depleted (Equation 5)}
      Index=Resample($W$);
      $S^k = S^k(Index)$;
    end if
    for ($j = 1$ to $M$) do {Prediction after action $\alpha$}
      $\mathbf{x}_j^{k+1} = \hat{f}(\mathbf{x}_j^k, \mathbf{u})$
    end for
    s=$\underline{\text{Sense}}$()
    for ($j = 1$ to $M$) do {Update the weights}
      $w_j^{k+1} = w_j^k * \mathcal{W}(s, \mathbf{x}_j^{k+1})$
    end for
    for ($j = 1$ to M) do {Normalize the weights}
      $w_j^{k+1} = \frac{w_j^{k+1}}{\sum_{j=1}^{M} w_j^{k+1}}$
    end for
  end while
  {ESS is the Effective Sample Size, see Equation 5}
```

the hat is because it is not necessary linear! that's why we don't need to invert anything!!
works in forward mode, computes the next move for each particle!

i just use the errors to simulate what happens to each particle!

I then compute the weights w.r.t. how much coherent the next particle location for each one

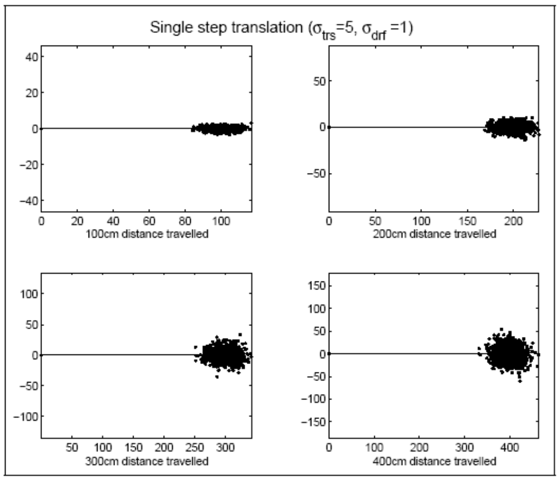
The **prediction** does **not** make **any assumption** on the fact that the process is linear or the noise is Additive White Gaussian Noise. For each particle $x_j^k \in S_k$ we have:

$$x_j^{k+1} = f(x_j^k, u^k, w^k)$$

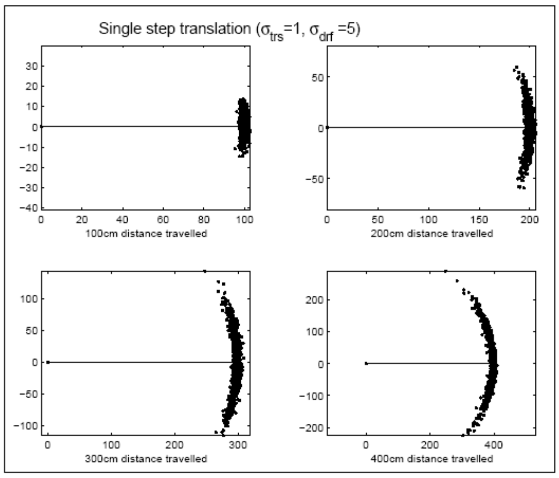
Nevertheless, there is a **problem**: after a few steps, particles tend to disperse depending on the noise:

- σ_{trs} is the error due to translation
- σ_{drft} is a lateral drift due to orientation errors

The effect of $\sigma_{trs}, \sigma_{drft}$ for the forward translation:



(a) $\sigma_{trs} = 5cm/m, \sigma_{drft} = 1°/m$



(b) $\sigma_{trs} = 1cm/m, \sigma_{drft} = 5°/m$.

The solution is the **resampling**: when some conditions hold, the set $S^k = [x_j^k, w_j^k], \quad j = 1...M$ is substituted with a new set $S'^k = S^k(\text{index})$. Depending on their weight, some particles in S^k are deleted and some other particles are replicated. All weights in S'^k are identical and normalized, i.e., $w'_j = 1/M$. The conditions for resampling are computed as follow:

Coefficient of variation
increase if the difference between high and low weights increase

$$cv_t^2 = \frac{var(w_t(i))}{E^2(w_t(i))} = \frac{1}{M}\sum_{i=1}^{M}(Mw(i) - 1)^2$$

Effective sample size

$$ESS_t = \frac{M}{1 + cv_t^2}$$

$\rightarrow$ when the effective sample size falls below a given threshold, particles are resampled.

Different resampling schemes have been proposed.

- Select with replacement: for each particle, the number of replica that are propagated statistically depend on the particle weight.
- Technical details: for each particle j, compute the cumulative sums of weights from particle $l = 0$ to particle $l = j$. Store all values in Q.
- Compute M random numbers, each with uniform distribution between 0 and 1. Store all values in T. Sort T in increasing order.

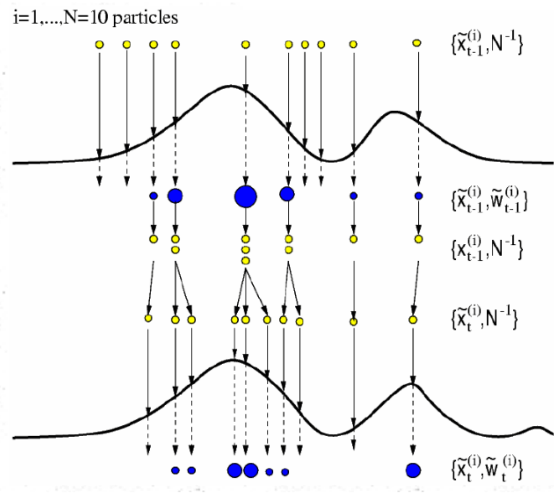
```
  Input: double W[$M$]
Require: $\sum_{i=1}^{N} W_i = 1$
  Q = cumsum(W); { calculate the running totals $Q_j = \sum_{l=0}^{j} W_l$}
  t = rand($M$+1); {t is an array of $M$+1 random numbers.}
  T = sort(t); {Sort them (O(m log m) time)}
  T($M$+1) = 1; i=1; j=1; {Arrays start at 1}
  while ($i \leq M$) do
    if $T[i] < Q[j]$ then
      Index[i]=j;
      i=i+1;
    else
      j=j+1;
    end if
  end while
  Return(Index)
```

Algorithm 3: Select with Replacement Resampling Algorithm; functions are noted as underlined text. Comments are inside curly brackets "{}".

The resampling principle:

we ignore the low weight particle and we redistribute them w.r.t. the ones with high weight

we don't need to keep track of all the particles with like "0.000001" weight!!!



i=1,...,N=10 particles
$\{\tilde{x}_{t-1}^{(i)}, N^{-1}\}$
$\{\tilde{x}_{t-1}^{(i)}, \tilde{w}_{t-1}^{(i)}\}$
$\{x_{t-1}^{(i)}, N^{-1}\}$
$\{\tilde{x}_{t}^{(i)}, N^{-1}\}$
$\{\tilde{x}_{t}^{(i)}, \tilde{w}_{t}^{(i)}\}$

(graphics taken from Van der Merwe et al.)

After an action, the observation done $o_k i$ is used to update the weights:

- Measurement model is

$$o_i^k = h(x^k, y^k, \psi^k, i) = \tan^{-1}_{(-\pi,\pi)}\left(\frac{yb_i - y^k}{xb_i - x^k}\right) - \psi^k + \nu$$

- Therefore, for every particle x_j^k, the estimated measurement o_{ij}^k is

$$o_{ij}^k = h(x_j^k, y_j^k, \psi_j^k, i) = \tan^{-1}_{(-\pi,\pi)}\left(\frac{yb_i - y_j^k}{xb_i - x_j^k}\right) - \psi_j^k + \nu$$

- In particular, at time $k+1$, the weight is set as proportional to the probability of x_j^{k+1} given $o_i k + 1$
- This is simpler to be computed by assuming the measurement noise ν as Gaussian (however, it is not necessary)

$$P\left(x_j^{k+1} \mid o_i^{k+1}\right) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{\left(o_i^{k+1} - o_{ij}^{k+1}\right)^2}{2\sigma_\nu^2}}$$

- Weights are then normalized such that their sum equals to 1.

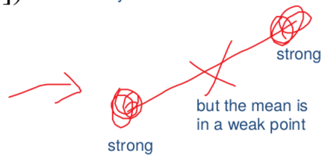
How do we chose the best pose in which we are after having computed all the probabilities? There are different techniques in order to choose the best pose:

- The best particle, I.e., the one with maximum weight

$$\bar{x} = x^{\max} = x_j,\ j = \arg\max(W[M])$$ not very robust

- Weighted mean:

$$\bar{x} = \sum_{j=1}^{M} x_j w_j$$ not very good



- Robust mean:

$$\bar{x} = \frac{\sum_j x_j w_j}{\sum_j w_j} : \left|x_j - x^{max}\right| \le \varepsilon$$ better choice

3.6.4 Summary

Calculation of belief for robot localization:

$$\underbrace{\overset{\text{Pdf of current state}}{p(x_t \mid d_{o\ldots t})}}_{\substack{\text{Resample:}\\ p(x)\text{ – the posterior probability, our belief about the state variables}}} = \underbrace{\underset{\text{Normalization constant}}{\eta}\ \overset{\text{Perception model}}{p(z_t \mid x_t)}}_{\substack{\text{Update:}\\ w(x)\text{ – the importance weights}}} \underbrace{\int \underset{\text{Motion model}}{p(x_t \mid u_{t-1}, x_{t-1})}\ \overset{\text{Pdf from last time step}}{p(x_{t-1} \mid d_{o\ldots t-1})}\, dx_{t-1}}_{\substack{\text{Prediction:}\\ q(x)\text{ – the prior probability}}}$$

The equation on this slide shows the formalization of the steps taken in the particle filter algorithm. It is derived from applying Bayes rule to the posterior, and then using the Markov assumption. While executing the particle filter algorithm, we are calculating this equation from right to left.

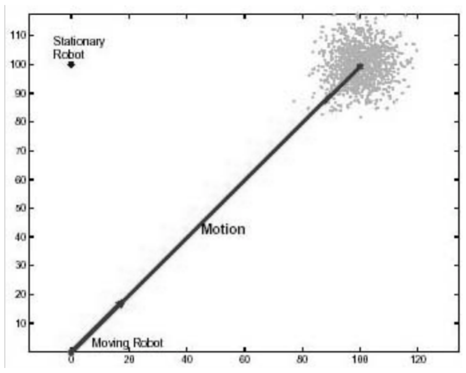
→ **First** we start with the pdf from the last time step, and then we multiply it by the motion model in the "prediction" step to get $q(x)$**, the prior probability**. The integral is there only to say that we can end up in the same state in time (t) from more than one state in time $(t-1)$, and thus we have to integrate over the states from time $(t-1)$. But we do not have to worry about this detail in the particle filter algorithm, since the particle representation takes care of the integral.

→ **Next**, we **find** the **importance weights** $w(x)$ using the perception model and normalize them so that they sum to 1.

→ $\boxed{q(x) \times w(x) = p(x)}$ the **posterior probability**, which we use resampling based on the importance weights to achieve.

More in detail:

1. **Prediction**: for each particle, sample and add random, noisy values from action model:

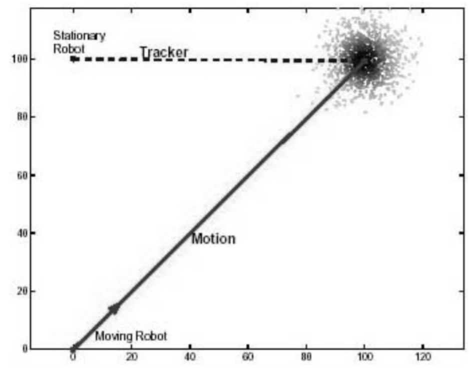


Resulting proposal distribution $(q(x))$ approximates:

$$\int p(x_t \mid x_{t-1}, u_{t-1})\, p(x_{t-1} \mid d_{o\ldots t-1})\ dx_{t-1}$$

→ in the prediction step, we take each particle and **add a random sample from the motion model**. In the figure, the robot starts from the lower left, and moves to the upper right. The resulting position, from the motion model, will be somewhere in that cloud of particles. The resulting distribution of particles approximates the prior distribution.

2. **Update**: each particle's weight is the likelihood of getting the sensor readings from that particle's hypothesis:



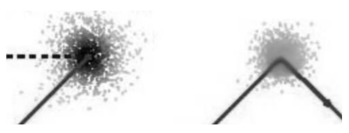
The weight associated with each particle is:

$$w(x) = \frac{p(x)}{q(x)} = p(z_t|x_t)$$

normalized so that all the weights sum to 1.

→ During the update step, we take the sensor measurements and assign each particle a **weight** that is equal to the **probability of observing the sensor measurements from that particle's state**. Those weights are then normalized so that they sum to 1. In the figure, the robot has observed the 'stationary robot' landmark at the top left, and based on that measurement, it has assigned weights to each particle. Darker particles have higher weights.

3. **Resample**: new set of particles are chosen such that each particle survives in proportion to its weight



Resulting distribution $p(x)$ is:

$$\underbrace{p(x_t \mid d_{0\ldots t})}_{\text{p(x) - the posterior probability}} = \eta\, \underbrace{p(z_t \mid x_t)}_{\text{w(x) - the importance weights}} \int \underbrace{p(x_t \mid u_{t-1}, x_{t-1})\, p(x_{t-1} \mid d_{0\ldots t-1})\, dx_{t-1}}_{\text{q(x) - the prior probability}}$$

→ Finally, in the resample step, a new set of particles is chosen so that each particle **survives in proportion to its weight**. As you can see in the picture, the weighted cloud of particles turns into the somewhat more condensed and smoother cloud of unweighted particles on the right. Highly unlikely particles at the fringe are not chosen, and the highly likely particles near the center of the cloud are replicated so that the high-probability region has a high density, correctly representing $p(x)$, our posterior distribution.

Chapter 4

Context-awareness

4.1 What is the context?

Humans are quite successful at **conveying ideas** to each other and **reacting appropriately**. This is due to many factors:

- the richness of the language they share;
- the common understanding of how the world works;
- and an implicit understanding of everyday situations.

When humans talk with humans, they are able to use **implicit situational information**, or context, to increase the conversational bandwidth.

> "*The context of the phonebook can be changed and another meaning can be derived from it: ask a hard life related question of it and start reading. Inspiration can then be found in the names numbers and addresses. Try it, it is called the phonebook oracle.*"
> – Lourens Coetzer

Devices are often used in changing environments, yet they do not adapt to those changes very well: although moving away from the desktop brings up a new variety of situations in which an application may be used, computing devices are often left **unaware of** their surrounding **environment**.

One hypothesis that a number of ubiquitous computing researchers share is that enabling devices and applications to **automatically adapt to changes in their surrounding** physical and electronic environments will lead to an enhancement of the user experience. **Current** examples of context awareness are based on localization:

- Map-based information;
- Items available in a shop

But *contex awareness* and *situational awareness* are usually given a different interpretation.

Understanding **what is context** will enable application designers to choose what context to use in their application. Understanding **how context can be used** will help application designers determine what context-aware behaviors to support in their applications. Finally, **architectural support** will enable designers to build their applications more easily.

In the work that first introduces the term "context-aware", **Schilit and Theimer** refer to context as *location, identities of nearby people and objects, and changes to those objects* (i.e., where you are, who you are with, and what resources are nearby): These types of **definitions** that define context by example are **difficult to apply**. When we want to determine whether a type of information not listed in the definition is context or not, it is not clear how we can use the definition to solve the dilemma.

Other definitions have **simply** provided **synonyms** for context; for example, referring to context as the *environment* or *situation*: definitions that simply use synonyms for context are extremely difficult to apply in practice.

Pascoe defines context to be the subset of physical and conceptual states of interest to a **particular entity**.

These definitions are **too specific**. Context is all about the whole situation relevant to an application and its set of users.

Anind Dey proposes the following definition:

> "*Context is* ***any information*** *that can be used to* ***characterize the situation*** *of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*"
> – Dey A., "Understanding and Using Context",
> in *Personal and Ubiquitous Computing* 5 : 1, Feb 2001, Pages 4-7

If a piece of information can be used to **characterize the situation** of a participant in an interaction, then that information is **context**.

Take the canonical context-aware application, an indoor mobile tour guide, as an example:

- Entities: the user, the application and the tour sites.

→ The weather is part of the context? **No**, the weather does not affect the application because it is being used indoors. Therefore, it is not context.

→ The presence of other people is part of the context? **Yes**, the presence of other people can be used to characterize the user's situation. If a user is traveling with other people, then the sites they visit may be of particular interest to her. Therefore, the presence of other people is context because it can be used to characterize the user's situation.

4.1.1 Categories of Context

Given the diversity of context information, it is useful to attempt to categorize it.

The **entities** we identified as most relevant are places, people and things:

- **Places** are regions of geographical space such as rooms, offices, buildings, or streets.
- **People** can be either individuals or groups, co-located or distributed.
- **Things** are either physical objects or software components and artifacts, for instance an application or a file.

We introduce four essential **categories**, or characteristics, of context information:

- **Identity** refers to the ability to assign a unique identifier to an entity.
- **Location** is more than just position information in 2-D space. It is expanded to include orientation and elevation, as well as all information that can be used to deduce spatial relationships between entities, such as co-location, proximity, or containment.
- **Status** (or **activity**) identifies intrinsic characteristics of the entity that can be sensed. For a place, it can be the current temperature, or the ambient light
- Finally, **time** is context information as it helps characterize a situation. In some cases, just knowing the relative ordering of events or causality is sufficient.

Simple inference, or **derivation**, of context information happens when related context information is deduced from a single known piece of context information.

More **complex inference** is required when context is deduced from multiple sources of information

4.1.2 Context-aware computing

Context-aware computing was first discussed by **Schilit and Theimer** in 1994. They described context-awareness in systems the ability to:

> "*adapt according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time.* "

Dey proposes the following definition:

> "*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.* "

For Dey there are three categories of **features** that a context-aware application can support:

- presentation of information and services to a user;
- automatic execution of a service for a user;
- tagging of context to information to support later retrieval.

The first category, **presenting information and services**, refers to applications that either present context information to the user, or use context to propose appropriate selections of actions to the user (e.g., showing the user's or her vehicle's location on a map, indicating nearby sites of interest, presenting a choice of printers close to the user etc...).

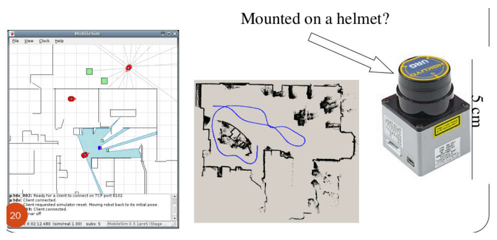
The second category, **automatically executing a service**, describes applications that trigger a command, or reconfigure the system on behalf of the user according to context changes (e.g., car navigation systems that recompute driving directions when the user misses a turn; a recording whiteboard that senses when an informal and unscheduled encounter of individuals occurs and automatically starts recording the ensuing meeting, a "teleport" system in which a user's desktop environment follows her as she moves from workstation to workstation etc...).

In the third category, **attaching context information for later retrieval**, applications tag captured data with relevant context information (e.g, a zoology application tags notes taken by the user with the location and the time of the observation; the informal meeting capture system mentioned above provides an interface to access informal meeting notes based on who was there, when the meeting occurred and where the meeting was located etc...).

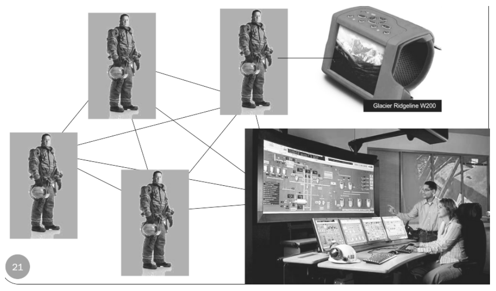
A case study: ORPHEUS One of the main issues to be addressed by first responders engaged in crisis scenario is resource localization and tracking. In outdoor scenarios the problem can be dealt with through the use of Global Positioning Systems technology, but a different approach must be pursued in all situations when the GPS signal is be absent or heavily corrupted by noise (e.g., inside buildings, tunnels and subways, under heavy tree canopy, etc.).

A precise positioning of all available resources is of the uttermost importance, especially when resources are limited or particularly costly, like for survey missions, search & rescue, or rapid intervention in case of terrorist or organized crime attack. In all these scenarios, resource localization plays a key role for exploring the environment, planning evacuation routes (possibly considering alternatives whenever a previously planned route becomes unavailable) and defining coordinated intervention strategies and optimal resource allocation.

The solution is to implement **wearable computing**: embedded processors, lightweight sensors and actuators (inertial sensors, environmental sensors, biofeedback) all using wireless communication (short range/long range). As **absolute localization systems** they used laser rangefinders ($50 \times 50 \times 70$mm) that allow to update the map of the environment and allow to reset the position in the environment $\rightarrow$ SLAM (simultaneous localization and mapping; statistical approaches: Kalman/Particle filters, etc)



The information dynamically acquired by the sensor network and by first responders will constitute the conditioning information set for a family of control schemes, enabling **decision support** (e.g., suggesting optimal evacuation routes/plans, optimal resource allocation etc...).



Which kind of context-aware behaviour could be interesting?

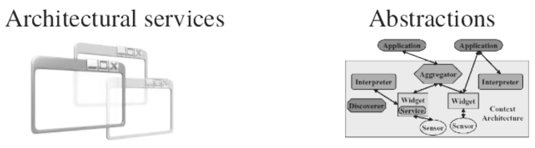
- *Context awareness* (of something/somebody): being aware of all information related to something/-somebody (position, environmental state, health state,...)
- *Situational awareness*: how the context of multiple objects or subjects evolve in time, and which information is relevant at a give instant.

4.1.3 Support for Building Applications

Application builders may need help moving from the design to an actual implementation.This help can come in two forms:

- The first is a combination of **architectural services** or features that designers can use to build their applications from;
- The second form is **abstractions** that allow designers to think about their applications from a higher level.

Dey proposes an architecture, the **Context Toolkit**, that contains a combination of features and abstractions to support context-aware application builders.



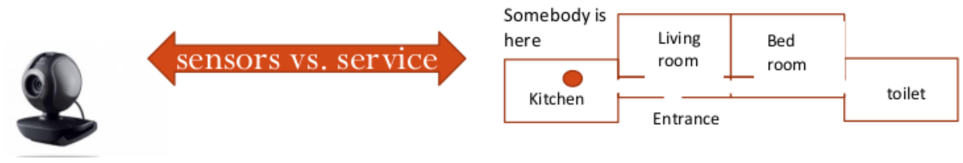
The requirements for dealing with Context are:

1. Separation of concerns;
2. Context interpretation;
3. Transparent, distributed communications;
4. Constant availability of context acquisition;
5. Context storage;
6. Resource discovery.

1. Separation of concerns There are two common ways in which context has been handled:

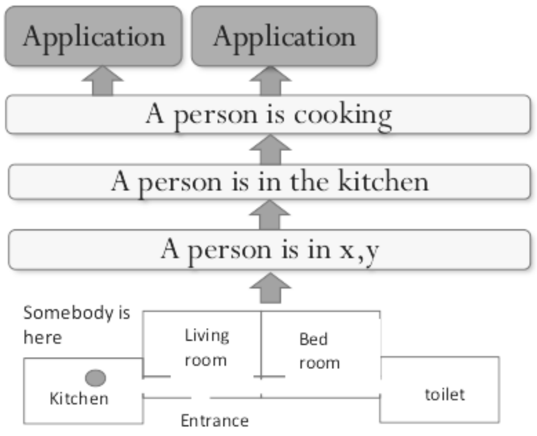
- connecting sensor drivers directly into applications
- using servers to hide sensor details

By separating how context is acquired from how it is used, applications can now use contextual information without worrying about the details of a sensor and how to acquire context from it. These details are not completely hidden and can be obtained if needed (see universAAL section)



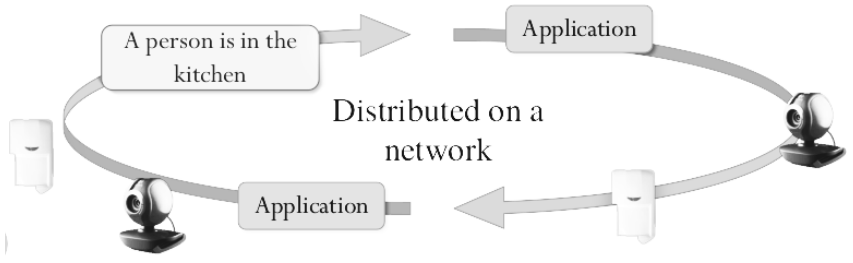
2. Context Interpretation There is a need to extend the notification and querying mechanisms to allow applications to **retrieve context from distributed computers**.

There may be **multiple layers** that context data go through before reaching an application: from an application designer's perspective, the use of these multiple layers should be **transparent**. In order to support this transparency, context must often be **interpreted before** it can be **used** by an application. In order for the **interpretation** to be easily reusable by multiple applications, it needs to be **provided by the architecture**.



3. Transparent, Distributed Communications When dealing with context, the devices used to sense context most likely are not attached to the same computer running an application that will react to that context.

- For example, an indoor infrared positioning system may consist of many infrared emitters and detectors;
- In addition, multiple applications may require use of that location information and these applications may run on multiple computing devices;
- The fact that communication is distributed should be transparent to both sensors and applications;
- A related requirement is the need for a global timeclock mechanism.



4. Constant Availability of Context Acquisition Differently from GUI applications, context-aware applications should **not instantiate individual components** that provide **sensor data**, but must be able to **access existing ones**, when they require it.

- Multiple applications may need to access the same piece of context.
- This leads to a requirement that the components that acquire context must be executing independently from the applications that use them.
- This eases the programming burden on the application designer.
- Because these components run independently of applications, there is a need for them to be persistent, available all the time.

Sensor data are periodically acquired and the context is periodically updated, even if no application requires it!

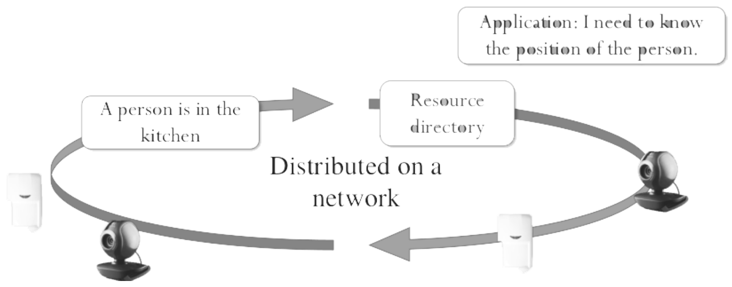
5. Context Storage and History A requirement linked to the need for constant availability is the desire to maintain historical information.

- A component that acquires context information, should maintain a history of all the context it obtains.
- Context history can be used to establish trends and predict future context values.
- A component may collect context when no applications are interested in that particular context information. Therefore, there may be no applications available to store that context. However, there may be an application in the future that requires the history of that context.

The history of past contexts is stored

6. Resource Discovery In order for an application to communicate with a sensor (or rather its software interface), it must know **what** kind of information the sensor can provide, **where** it is located **and how** to communicate with it (protocol, language and mechanisms to use).

- To be able to effectively hide details from the application, the architecture needs to support a form of resource discovery.
- With a resource discovery mechanism, when an application is started, it could specify the type of context information required.
- The mechanism would be responsible for finding any applicable components and for providing the application with ways to access them.

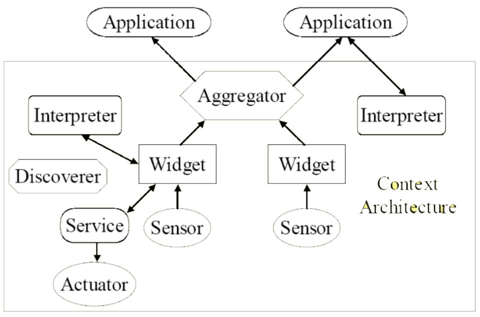

Application: I need to know the position of the person.
A person is in the kitchen
Resource directory
Distributed on a network

4.1.4 Context toolkit

The context toolkit was probably the first attempt to face the problem of context-awareness at an architectural level, but now the project has **not** been **updated since 2003**.

Software can be downloaded here http://contexttoolkit.sourceforge.net/

The applications though were not very exciting... some demonstrations are available in the "Georgia Tech Aware Home".



The Context Toolkit architecture is built on the concept of enabling applications to **obtain** the context they require without them having to worry about **how** the context was sensed: it makes the distribution of the context architecture transparent to context-aware applications, mediating all communications between applications and components.

◦ **Context Toolkit Widgets:** A context widget is responsible for **acquiring a certain type of context information** and making that information available to applications in a **generic manner**, regardless of how it is actually sensed. Applications can access context from widgets using **traditional poll and subscribe methods**, commonly available with graphical user interface (GUI) widgets.
While in most **GUI applications widgets** are instantiated, controlled and used by only a single application, **context-aware widget based applications** do not instantiate individual context widgets, but must be able to access existing ones, when they require.
To meet this requirement, context widgets operate **independently** from the applications that use them: this eases the programming burden by not requiring to maintain the context widgets, while allowing an easy communication with them.
Because context widgets run independently of applications, there is a need for them to be textbfpersistent, available all the time.
Moreover, context widgets should automatically store the **history** of the context they sense and make this history available to any interested applications so that they are able to **predict** the future actions or intentions of users.
This prediction or interpretations functionality is encapsulated in the context **interpreter abstraction**: interpreters accept one or more types of context and produce a single piece of context (i.e. converting from a name to an e-mail address or interpreting context from all the widgets in a conference room to determine that a meeting is occurring).
Widgets also provide a **separation of concerns** by hiding the complexity of the actual sensors used from the application. Whether the location of a user is sensed using different sensors or a combination of these, it should not impact the application.

To **access** context information applications can:

- be notified to changes in the widget context;
- query a widget;
- poll a widget.

They abstract context information to **suit** the expected needs of applications: a widget that tracks the location of a user within a building or a city notifies the application only when the user moves from one room to another, or from one street corner to another, and does not report less significant moves to the application.

Widgets provide reusable and customizable **building blocks of context sensing**: a widget that tracks the location of a user can be used by a variety of applications, from tour guides to car navigation to office awareness systems.

◦ **Context Toolkit Interpreters** Interpreters transform context information by **raising its level of abstraction**: typically takes information from one or more context sources and produces a new piece of context information.
Simple inference or derivation transforms geographical coordinates (low level of abstraction) into street names (higher level of abstraction) using for example a geographic information database.
Complex inference using **multiple pieces of context** also provides **higher-level information**.
For example, if a room contains several occupants and the sound level in the room is high, one can guess that a meeting is going on by combining these two pieces of context.

◦ **Context Toolkit Aggregators** Aggregation refers to collecting multiple pieces of context information that are **logically related** into a common repository.
The need for aggregation comes in part from the distributed nature of context information which must often be retrieved from distributed sensors, via widgets.
Rather than have an application query each distributed widget in turn, aggregators gather **logically related information** relevant for applications and make it available within a single software component.
Accessing information provided by aggregators is, therefore, a **simplified operation** for the application, and several applications can access the same information from a single aggregator.
For example, an application may have a context-aware behavior to execute when the following conditions are met: an individual is happy, located in his kitchen, and is making dinner. With no support for aggregation, an application (via the specification mechanism) has to use a combination of subscriptions and queries on different widgets to determine when these conditions are met.
In conclusion:

- an aggregator is responsible for collecting all the context about a given entity;
- an aggregator has similar capabilities as a widget;
- applications can be notified to changes in the aggregator's context, can query/poll for updates, and access stored context about the entity, represented by the aggregator.

◦ **Context Toolkit Services** Services are components in the framework that execute actions on behalf of applications. A context service is an analog to the context widget.
Whereas the context widget is responsible for retrieving state information about the environment from a sensor (i.e.input), the context service is responsible for **controlling or changing state information in the environment** using an actuator (i.e. output). As with widgets, applications do not need to understand the details of how the service is being performed in order to use them.
Context services can be synchronous or asynchronous. An example of a synchronous context service is to send an e-mail to a user. An example of an asynchronous context service is to send a message to a user on a two-way pager containing a number of possible message responses.

◦ **Context Toolkit Discoverers** Discoverers are the final component in the conceptual framework. They are responsible for maintaining a **registry** of what capabilities exist in the framework, including the knowledge about widgets, interpreters, aggregators and services currently available.
When any of these components are started, it **notifies a discoverer** of its presence which comprises how to contact that component (e.g. language, protocol, machine hostname) and its capabilities:

- widgets indicate what kind(s) of context they can provide;
- interpreters indicate what interpretations they can perform;
- aggregators indicate what entity they represent and the type(s) of context they can provide about that entity;
- Services indicate what context-aware service they can provide and the type(s) of context and information required to execute that service.

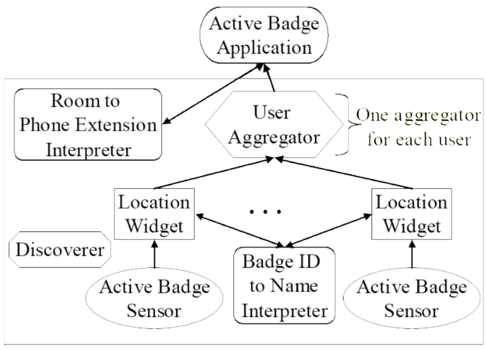
4.1.5 Context Toolkit Situation Abstraction

A situation is an abstraction at a **level above** widgets, interpreters and aggregators. Currently, application designers need to explicitly poll and subscribe to widgets and aggregators for context information and call on interpreters to determine when relevant entities are in a particular state so they can take action. This **collection of states** can be described as a **situation**.
The situation abstraction is exactly that: a **description of the states of relevant entities**. The Context Toolkit is responsible for the translation of the description to the "wiring" of the context components and also for determining when the individual elements of the situation have been collectively satisfied.

4.1.6 Context Toolkit implementation

The ContextToolkit provides designers with the abstractions above: widgets, interpreters, aggregators, services and discoverers as well as a distributed infrastructure. Developed in Java, although programming language-independent mechanisms were used, allowing the creation and interoperability of widgets, interpreters, aggregators and applications in any language (applications can be been written in C++, Frontier, Visual Basic and Python.). Each widget, aggregator, interpreter and discoverer component is implemented as a single process. Typically, different components are distributed on different processors. Currently, it utilizes a simple object communication mechanism based on HTTP (HyperText Transfer Protocol) and XML (eXtensible Markup Language) encoding of messages.

Example: Active Badge Call-Forwarding



The Active Badge application helps a receptionist to forward a phone calls to the extension nearest the recipient's most recent location in a building. A dynamically updating table displays the phone extension and location for each person in the system. In addition, a user can:

- request a list of the locations an individual has been in the past five minutes or past hour;
- locate an individual and find the names of others in the same location;
- request a list of the people at a particular location;
- request to be notified when an individual is next sensed by the system.

The Active Badge location system consisted of a number of infrared sensors distributed throughout a building that detected the presence of user-worn Active Badges.
A Location Widget represents each sensor: when a sensor detects a badge, its corresponding Location Widget determines the wearer's identity (using the Badge ID to Name Interpreter), stores this information and passes it to User Aggregator.
Each User Aggregator (one for each user in the system) collects and stores all the location information from each of the Location Widgets for their corresponding user and makes it available to the application.
All of these components (the interpreters, aggregators and widgets) register with the Discoverer to allow them to locate each other as needed (widgets need to locate the Badge ID to Name Interpreter and the aggregator needs to locate the widgets).
The application then uses the Discoverer to locate the User Aggregators and the Room to Phone Extension Interpreter.

4.2 Context assessment with Bayesian Networks

The purpose of this section is to build a model to infer higher level contexts starting from basic blocks and, ultimately, from sensor readings. In the context toolkit terminology, we want to build the "**interpreters**". There are several approaches to do this:

- Bayesian networks
- Dynamic Bayesian Networks
- Hidden Markov Models
- Ontologies / Description Logics

4.2.1 Bayes and joint probability tables

Let A be a discrete random variable with possible states $dom(A) = \{a_1, ..., a_n\}$ (also known as the the **domain** of the variable):

- Then $P(A)$ denotes a **probability distribution over these states** (i.e., a function which assigns a probability value in the range [0 1] to each value in the domain)

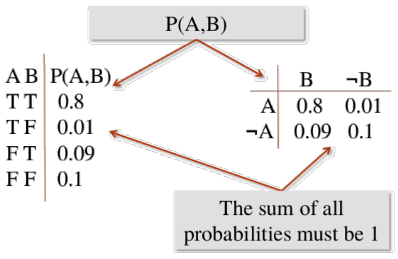
$$P(A) = (x_1, ..., x_n) \qquad x_i \geq 0 \qquad \sum_{i=1}^{n} x_i = 1$$

  where x_i is the probability of A being in state a_i.
- The probability of A being in state a_i is denoted $P(A = a_i)$ or simply $P(a_i)$ when obvious from the context.
- If the variable B has $dom(B) = b_1, ..., b_m$, then $P(A, B)$ represents the **joint probability distribution** over the Cartesian product $dom(A) \times dom(B)$.
- $P(A, B)$ assigns a probability value in the range [0 1] to each couple of states (a_i, b_j).
- $P(A = a_i, B = b_j)$ or simply $P(a_i, b_j)$ denotes the probability of A being in state a_i and B being in state b_j.
- $P(A, B)$ can be represented with an $n \times m$ table containing the numbers $P(a_i, b_j)$.

Let's make an examples with boolean variables:

- Variable A: $outside\_raining$
- Variable B: $slippery\_floor$
- $dom(A) = dom(B) = \{T, F\}$

→ There are two different ways of representing the joint probability $P(A, B)$:



A B	P(A,B)
T T	0.8
T F	0.01
F T	0.09
F F	0.1

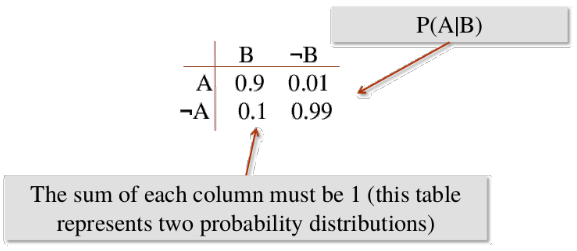
	B	¬B
A	0.8	0.01
¬A	0.09	0.1

- Similarly, $P(A|B)$ is a function which assign a probability value in the range [0 1] to each couple of states (a_i, b_j)
- $P(A = a_i|B = b_j)$ or simply $P(a_i|b_j)$ represents the conditioned probability of A being in state a_i given that B is in state b_j
- $P(A|B)$ should be considered as a **set of probability distributions, one for each value of the conditioning variable**
- $P(A|B)$ can be represented with an $n \times m$ table containing the numbers $P(a_i|b_j)$.

Let's make an examples with boolean variables:

- Variable A: *outside_raining*
- Variable B: *slippery_floor*
- $dom(A) = dom(B) = \{T, F\}$

→ $P(A|B)$ is a couple of probability distribution, one for each value of the conditioning variable:



	B	¬B
A	0.9	0.01
¬A	0.1	0.99

P(A|B)

The sum of each column must be 1 (this table represents two probability distributions)

- For all states a_i of A and b_j of B we have

$$P(a_i|b_j)P(b_j) = P(a_i, b_j)$$

- This procedure can be applied to a whole table, i.e., to all the $n \times m$ configurations (a_i, b_j)

$$P(A|B)P(B) = P(A, B)$$

- Since $P(A, B)$ is obviously equal to $P(B, A)$, this leads to the **Bayes rule**:

$$P(A|B)P(B) = P(B|A)P(A) = P(A, B)$$

- **From a table** $P(A, B)$, the **probability distribution** P(A) can be **calculated**. For a state a_i

$$P(a_i) = \sum_{j=1}^{m} P(a_i, b_j) = \sum_{j=1}^{m} P(a_i|b_j)P(b_j)$$

  This calculation is called **marginalization**, and we say that the variable B is marginalized out of $P(A, B)$

$$P(A) = \sum_{B} P(A, B)$$

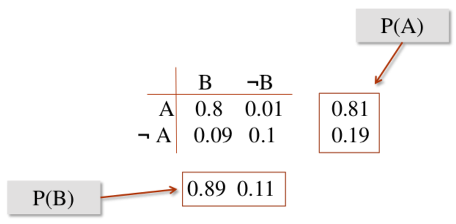
Let's make an examples with boolean variables:

- Variable A: *outside_raining*
- Variable B: *slippery_floor*
- $dom(A) = dom(B) = \{T, F\}$

→ For computing $P(A)$ it is sufficient to sum the probabilities of all combinations (or "possible world") such that A is T. For example, in this case $P(A) = 0.81$:

A B	P(A,B)
T T	0.8
T F	0.01
F T	0.09
F F	0.1

The term marginalization can be better understood by considering the alternative representation of $P(A, B)$:

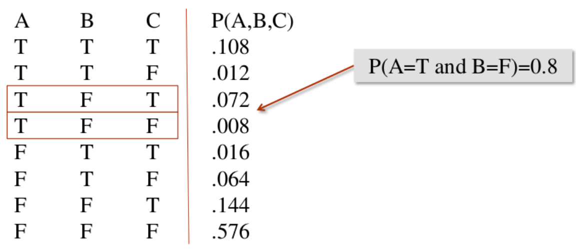


	B	¬B	
A	0.8	0.01	0.81
¬ A	0.09	0.1	0.19
	0.89	0.11	

Let's make another example with three boolean variables:

- Variable A: cavity (I have a cavity in my tooth)
- Variable B: tootache (I have a tootache)
- Variable C: catch (the proble catches the cavity)

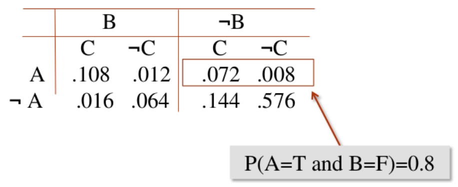
→ Which is $P(A = T \text{ and } B = F)$? (i.e., I have a cavity but not a tootache?)



A	B	C	P(A,B,C)
T	T	T	.108
T	T	F	.012
T	F	T	.072
T	F	F	.008
F	T	T	.016
F	T	F	.064
F	F	T	.144
F	F	F	.576

P(A=T and B=F)=0.8

Using an alternative representation:



	B		¬B	
	C	¬C	C	¬C
A	.108	.012	.072	.008
¬ A	.016	.064	.144	.576

P(A=T and B=F)=0.8

!!! Given a joint distribution, e.g. $P(X, Y, Z)$ we can compute any distributions over any smaller sets of variables. **!!!**

4.2.2 Belief updating, marginal and conditional independence

A basic task in artificial intelligence is **belief updating**: **(1)** you are considering a part of the world, and you have a certain belief in the state of a particular variable A. Next **(2)** you receive the information that the state of the variable B is b, and you wish to use this information to update your belief in the state of A.
→ formally, you have a prior distribution $P(A)$ and, by knowing the joint probability distribution, you wish to compute the posterior $P(A|b)$.

For example, consider the previous case "a person goes to the dentist":

- The prior probability of having a cavity $P(cavity = T)$ should be revised if you know that there is toothache

$$P(cavity = T|toothache = T)$$

- It should be revised again if you were informed that the probe did not catch anything

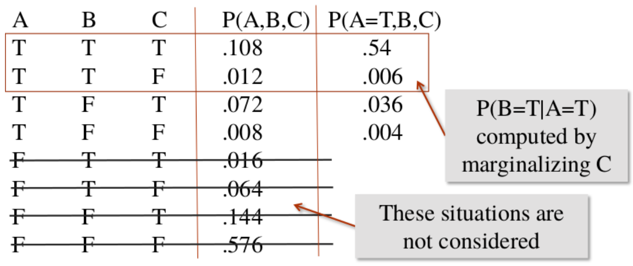
$$P(cavity = T|toothache = T, catch = F)$$

- What about the probability to have cavity given that it is a sunny day? They are not correlated (see later):

$$P(cavity = T|sunny = T) = P(cavity = T)P(sunny = T)$$

- What happens in term of "possible worlds" if we know the value of a random var (or a set of random vars)? **Some worlds are ruled out, others change their probability**.

  → Suppose that I have a cavity, i.e., $A = T$: how can I compute the probability of having a tootache? Recalling that $P(B|A) = P(B, A)/P(A)$ we have:



A	B	C	P(A,B,C)	P(A=T,B,C)
T	T	T	.108	.54
T	T	F	.012	.006
T	F	T	.072	.036
T	F	F	.008	.004
~~F~~	~~T~~	~~T~~	~~.016~~	
~~F~~	~~T~~	~~F~~	~~.064~~	
~~F~~	~~F~~	~~T~~	~~.144~~	
~~F~~	~~F~~	~~F~~	~~.576~~	

P(B=T|A=T) computed by marginalizing C

These situations are not considered

In general, to **express the joint probability** in terms of conditioned probabilities it is possible to use the **chain rule**:

$$\begin{aligned}
&P(X_1, ...X_{n-1}, X_n) = \\
&P(X_1, ...X_{n-1})P(X_n|X_1, ...X_{n-1}) = \\
&P(X_1, ...X_{n-2})P(X_{n-1}|X_1, ...X_{n-2})P(X_n|X_1, ...X_{n-1}) = \\
&(...) \\
&P(X_1)P(X_2|X_1)...P(X_{n-1}|X_1, ...X_{n-2})P(X_n|X_1, ...X_{n-1})
\end{aligned}$$

For our previous example:

$$P(A, B, C) = P(A)P(B|A)P(C|A, B)$$

Do we always **need to revise** beliefs when there is a new evidence? ? It depends **if variables are dependent** or not.

$\rightarrow$ We say that variables A and C are **marginally independent** if

$$P(a_i|c_k) = P(a_i) \quad \forall i, k \qquad \text{can be written as} \qquad P(A|C) = P(A)$$

Then, in they are not dependent, the useful consequence is the following one:

$$P(A = a_i, B = b_j) = P(A = a_i|B = b_j)P(B = b_j) = P(A = a_i)P(B = b_j)$$

If A is not dependent on B, new evidence on B does not affect current belief in A. As anticipated before:

- Variable A: *cavity* (I have a cavity in my tooth)
- Variable B: *toothache* (I have a tootache)
- Variable C: *catch* (the proble catches the cavity)
- Variable D: *sunny* (today is a sunny day)

$$P(A, B, C, D) = P(A, B, C)P(D)$$

The joint probability can be stored using two smaller tables (with 8 and 2 entries) instead of a big one (with 16 entries)! $\rightarrow$ **independence leads to few, smaller tables**

Marginal independence is rare. However there is another (weaker but very important) type of independence:

- The variables A and C are **conditionally independent** given the variable B if

$$P(a_i|b_j) = P(a_i|b_j, c_k) \quad \forall i, j, k$$

this means that **if the state of B is known, than no knowledge of C will alter the probability of A.**

- This can also be written as

$$P(A|B) = P(A|B, C)$$

$\rightarrow$ That's very important! That means that if you know B, A and C will become independent.

Let's consider the previous example:

- Variable A: cavity (I have a cavity in my tooth)
- Variable B: tootache (I have a tootache)
- Variable C: catch (the proble catches the cavity)

→ Are tootache and catch independent?

$$P(B|C) = P(B)?$$

A	B	C	P(A,B,C)
T	T	T	.108
T	T	F	.012
T	F	T	.072
T	F	F	.008
F	T	T	.016
F	T	F	.064
F	F	T	.144
F	F	F	.576

If you try to compute them, it turns out they are not independent

BUT if I have a cavity, does the probability that the probe catches it depend on whether I have a toothache?

$$P(C|B, A = T) = P(C|A = T)$$

What if I haven't got a cavity? If you try to compute them, it turns out they are **conditionally independent**

$$P(C|B, A = F) = P(C|A = F)$$

→ Both toothache and catch are directly caused by the cavity, but neither has a direct effect on the other.

- In general,C $Catch$ is conditionally independent of B (Toothache) given A (Cavity):

$$P(C|B, A) = P(C|A)$$

Equivalent statements are:

$$P(B|A, C) = P(B|A) \qquad P(B, C|A) = P(B|A)P(C|A)$$

- Sometimes, two variables might not be marginally independent. However, they **may become independent after we observe some third variable**.

- This is very important when computing the joint probability distribution over C (Catch) is conditionally independent of B (Toothache) given A (Cavity):

$2^3 - 1 = 7$ values (probabilities sum up to 1)

$$P(A,C,B) = P(B \mid A,C)P(C \mid A)P(A)$$
$$= P(B \mid A)P(C \mid A)P(A)$$

2+2+1=5 values

- The use of **conditional** independence often **reduces the size** of the representation of the **joint distribution** from exponential in n **to linear** in n (with n number of variable)
- Conditional independence is our most basic and robust form of knowledge about uncertain environments.

Summary:
Assume that the world consist of three finite variables A, B, C and the model of the world is the joint probability distribution $P(A, B, C)$ (i.e., an $n \times m \times p$ table)

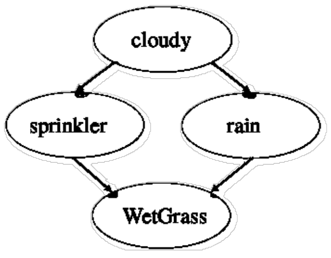
$$P(A) = \sum_{B,C} P(A, B, C) \qquad P(A, b) = \sum_{C} P(A, b, C) \qquad P(A|b) = \frac{P(A, b)}{\sum_A P(A, b)}$$

The problem arises when the number of variables increases:
- A very big table is required
- A lot of computations to handle it
- **Bayesian networks** provide a **compact representation** by **identifying conditional independence** between variables

4.2.3 Bayesian Networks

Consider a very typical example:

- Every variable can assume boolean values $True$ or $False$
- The probability of *rain* and *sprinkler* depends on *cloudy*
- The probability of $WetGrass$ depends on *sprinkler* and *rain*
- What we said in the previous summary here is evident: Bayesian networks provide a compact representation by **identifying conditional independence** between variables. From the network we can clearly see that, given a well-defined value (true/false) for *rain* and $sprinkler$, the probability of $WetGrass$ does not depend on *cloudy*:



- This information (which determines the structure of the following graph) comes from knowledge about the domain, i.e., causal relationships.

!!! It's **forbidden** to have **cycles** in a Bayesan Networks!

!!! On the **top** just the **a priori** probability, from which the first layer depends. The more is deep, the more are the dependencies

Let CN be a **causal network** over the universe $\boldsymbol{\mathcal{U}} = \{A_1, ..., A_n\}$, where A_i is a discrete random variable (boolean in all the following examples). Let $pa(A)$ denote the **causal parents** of A and $desc(A)$ the **causal descendents** of A. Let $\boldsymbol{\mathcal{R}}$ denote the remaining nodes:

$$\text{remaining nodes } \boldsymbol{\mathcal{R}} = \boldsymbol{\mathcal{U}} \backslash [\{A\} \cup pa(A) \cup desc(A)]$$

then:

The structure of the network tells that A is conditionally independent of $\boldsymbol{\mathcal{R}}$ given $pa(A)$:

$$P(\boldsymbol{\mathcal{U}}) = P(A_1, ..., A_n) = \prod_{i=1}^{n} P[A_i|pa(A_i)]$$

Pay **attention!** In general, it is not true that A is conditionally independent of $\boldsymbol{\mathcal{R}}$: this holds **only** if the **value of** A**'s parents is given!**

In the example of Cloudy (C), Sprinkler (S), Rain (R), WetGrass (W) it turns out that, from chain rule:

$$P(C,S,R,W) = P(W|C,S,R)P(C,S,R) =$$
$$= P(W|C,S,R)P(R|C,S)P(C,S) = P(W|C,S,R)P(R|C,S)P(S|C)P(C)$$

This is a four-dimensional matrix

But **using the information about causal relationships** that are **encoded in** the structure of **the network**, the computation is simplified as follows:

$$P(C,S,R,W) = P(W|S,R)P(R|C)P(S|C)P(C)$$

This is a three-dimensional matrix

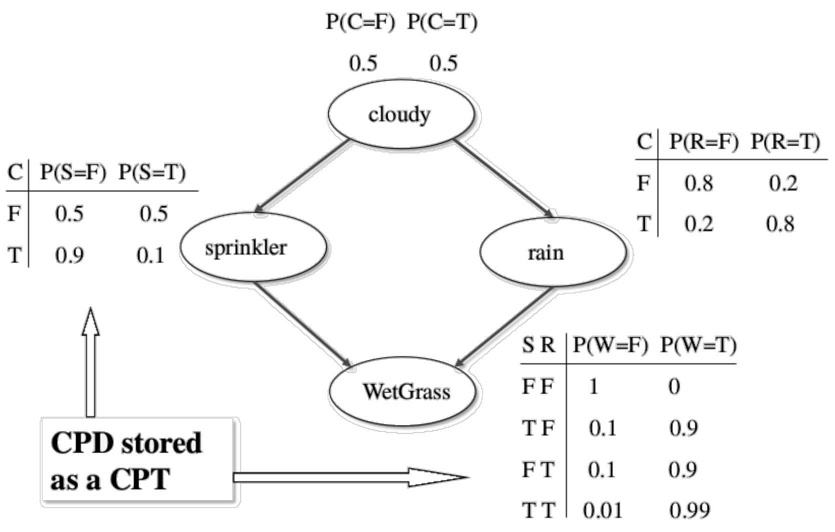
→ the dimensions of the tables required to represent information can be considerably smaller!

A **Bayesian network** (or **belief network**) is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (**DAG**). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are **directed acyclic graphs**:

- **Nodes** represent **random variables** they may be observable quantities, or hidden (not known) variables
- **Edges** represent **conditional dependencies**; nodes which are not connected represent variables which are conditionally independent of each other
- Each node is associated with a probability function (**Conditional Probability Distribution** – CPD) that takes as **input** a particular set of values for the node's **parent variables** and gives the probability of the variable represented by the node
- If the parents are m Boolean variables then the probability function could be represented by a table of 2^m entries (**Conditional Probability Table** – CPT), one entry for each of the 2^m possible combinations of its parents being true or false
  - → remember that, for each of the 2^m combinations of the parents, we need to **store only the probability of the child node to true** (or false), since the complementary probability of the child node to be false (respectively, true) can be computed as a consequence.

For the previous example:



P(C=F)	P(C=T)
0.5	0.5

C	P(S=F)	P(S=T)
F	0.5	0.5
T	0.9	0.1

C	P(R=F)	P(R=T)
F	0.8	0.2
T	0.2	0.8

S R	P(W=F)	P(W=T)
F F	1	0
T F	0.1	0.9
F T	0.1	0.9
T T	0.01	0.99

Let's analyze this basic net:

- We see that the event "grass is wet" (W=true) has two possible causes: either the water sprinker is on ($S = true$) or it is raining ($R = true$). The strength of this relationship is shown in the table:

$$P(W = true|S = true, R = false) = 0.9 \quad \leftarrow \text{(second row)}$$

and hence

$$P(W = false|S = true, R = false) = 0.1$$

since each row must sum up to one.

- Since the C node has no parents, its CPT specifies the prior probability that it is cloudy (in this case, $P(C = T) = 0.5$).
- The conditional independence relationships allow us to represent the joint probability distribution more compactly:
  - $\rightarrow$ in general, if we had n binary nodes, the full joint would require $O(2^n)$ space to represent
  - $\rightarrow$ the factored form would require $O(n \cdot 2^k)$ space to represent (k = max fan-in of a node)
  - $\rightarrow$ in addition, fewer parameters makes learning easier

4.2.4 Statistical inference with BN

Suppose **we observe the fact $W = T$** (the grass is wet). There are **two possible causes**: either it is raining ($R = T$), or the sprinkler is on ($S = T$). Which is more likely? We can use Bayes' rule to compute the posterior probability of each explanation:

$$P(S=T \mid W=T) = \frac{P(S=T, W=T)}{P(W=T)} = \frac{\sum_{C,R} P(C, S=T, R, W=T)}{P(W=T)} =$$

Explanation 1

$$= \frac{\sum_{C,R} P(W=T \mid R, S=T, C) P(S=T \mid R, C) P(R \mid C) P(C)}{P(W=T)} =$$

$$= \frac{\sum_{C,R} P(W=T \mid R, S=T) P(S=T \mid C) P(R \mid C) P(C)}{P(W=T)} = ?$$

Explanation 2

$$P(R=T \mid W=T) = \frac{P(R=T, W=T)}{P(W=T)} = \frac{\sum_{C,S} P(C, S, R=T, W=T)}{P(W=T)} = ?$$

Let us compute the former quantity:

$$\sum_{C,R} P(W=T \mid R, S=T) P(S=T \mid C) P(R \mid C) P(C) = 0.2781$$

It is possible to start by computing all terms of the product

C	R	$P(C)$	$P(R \mid C)$	$P(S=T \mid C)$	$P(W=T \mid R, S=T)$
F	F	0.5	0.8	0.5	0.9
F	T	0.5	0.2	0.5	0.99
T	F	0.5	0.2	0.1	0.9
T	T	0.5	0.8	0.1	0.99

Analogously, it is possible to compute all other quantities

$$P(W=T) = \sum_{C,R,S} P(C, S, R, W=T) = 0.6471$$
$$P(S=T \mid W=T) = 0.2781/0.6471 = 0.4298$$
$$P(R=T \mid W=T) = 0.4581/0.6471 = 0.7079$$

By comparing the two competing explanation, it turns out that it is **more likely that $W = T$ because $R = T$** (it is more likely that "the grass is wet beacause it has rained").

4.2.5 Explaining away BN

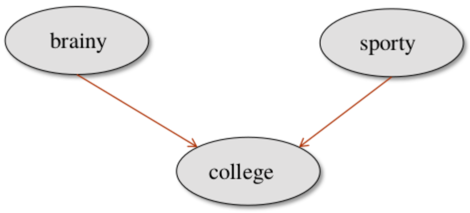
In the above example, notice that the **two causes "compete"** to "explain" the observed data, hence S and R are **conditionally dependent** given that their common child, W, is observed, even though they are marginally independent.

For example, suppose the grass is wet, but that we also know that it is raining. Then the **a-posteriori** probability that the sprinkler is on **($S=T$) goes down**:

$$\downarrow P(S = T|W = T, R = T) = 0.1945$$

This is called "**explaining away**". In statistics, this is known as Berkson's paradox, or "**selection bias**".

For a dramatic example of this effect, consider a college which admits students who are either brainy or sporty (or both). Let C denote the event that someone is admitted to college, which is made true if they are either brainy (B) or sporty (S). Suppose in the general population, B and S are independent. We can model our conditional independence assumptions using a graph which is a V-structure:



Now look at a population of college students (those for which C is observed to be true). It will be found that being **brainy makes you less likely to be sporty and vice versa (!)**, because either property alone is sufficient to explain the evidence on C.

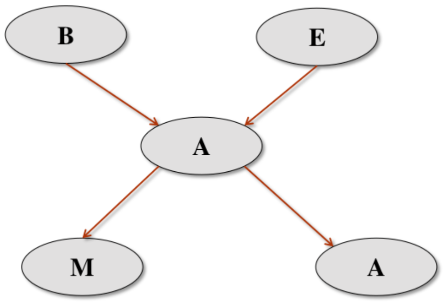
4.2.6 Top-down and bottom-up reasoning in BN

In the water sprinkler example, we had evidence of an effect (wet grass), and inferred the most likely cause. This is called **diagnostic**, or **"bottom up"**, reasoning, since it goes from effects to causes. It is a common task in expert systems.

Bayes nets can also be used for **causal**, or **"top down"**, reasoning. For example, we can compute the probability that the grass will be wet given that it is cloudy. Hence Bayes nets are often called **"generative" models**, because they specify how causes generate effects.

Consider another example: there might be a **burglar** in my house (variable B) and the anti-burglar alarm in my house may go off (variable A). I have an agreement with two of my neighbors, John and Mary, that they call me if they hear the alarm go off when I am at work (variable J and M). Minor earthquakes may occur and sometimes they set off the alarm (variable E):

– Variables: B, A, J, M, E
– The joint probability distribution has $2^5 - 1$ entries
- Typically we order variables to reflect causal knowledge (i.e., causes before effects):
– A burglar (B) can set the alarm (A) off
– An earthquake (E) can set the alarm (A) off
– The alarm can cause Mary to call (M)
– The alarm can cause John to call (J)



- The remaining dependencies are expressed as a network
– Each var is a node
– For each var, the conditioning vars are its parents
– Associate to each node corresponding conditional probabilities
- Some possible queries:

(Q1) I'm at work,

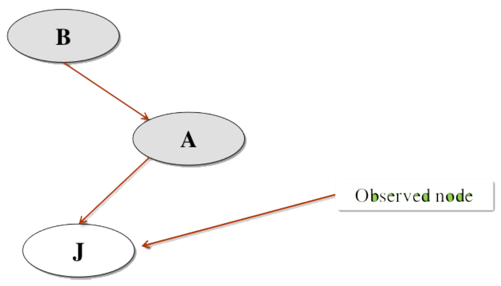
- neighbor John calls to say my alarm is ringing,
- neighbor Mary doesn't call.
- No news of any earthquakes.
- Is there a burglar?

(Q2) I'm at work,

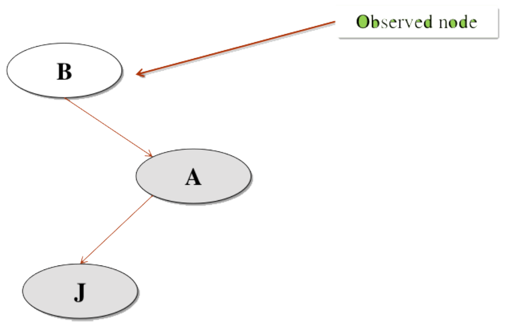
- Receive message that neighbor John called ,
- News of minor earthquakes.
- Is there a burglar?

Let's see now some inference types:

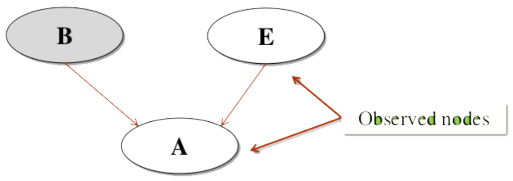
- **Diagnostic**: which is the probability that an event is the cause of a detected event?
- I know the apriori probability of a burglar: $P(B)$
- I receive a phone call from John: $P(J = T) = 1$
- I can compute the a posteriori probability that a burglar entered my house, given that John has called me: $P(B|J = T)$



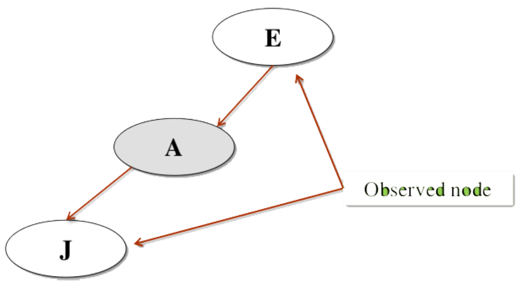
- **Predictive**: which is the probability that an event is the consequence of another event?
- I know the apriori probability that John will call me: P(J);
- I see a burglar entering my house: $P(B = T) = 1$
- I can compute the a posteriori prob. that John calls me, given that a burglar entered: $P(J|B = T)$



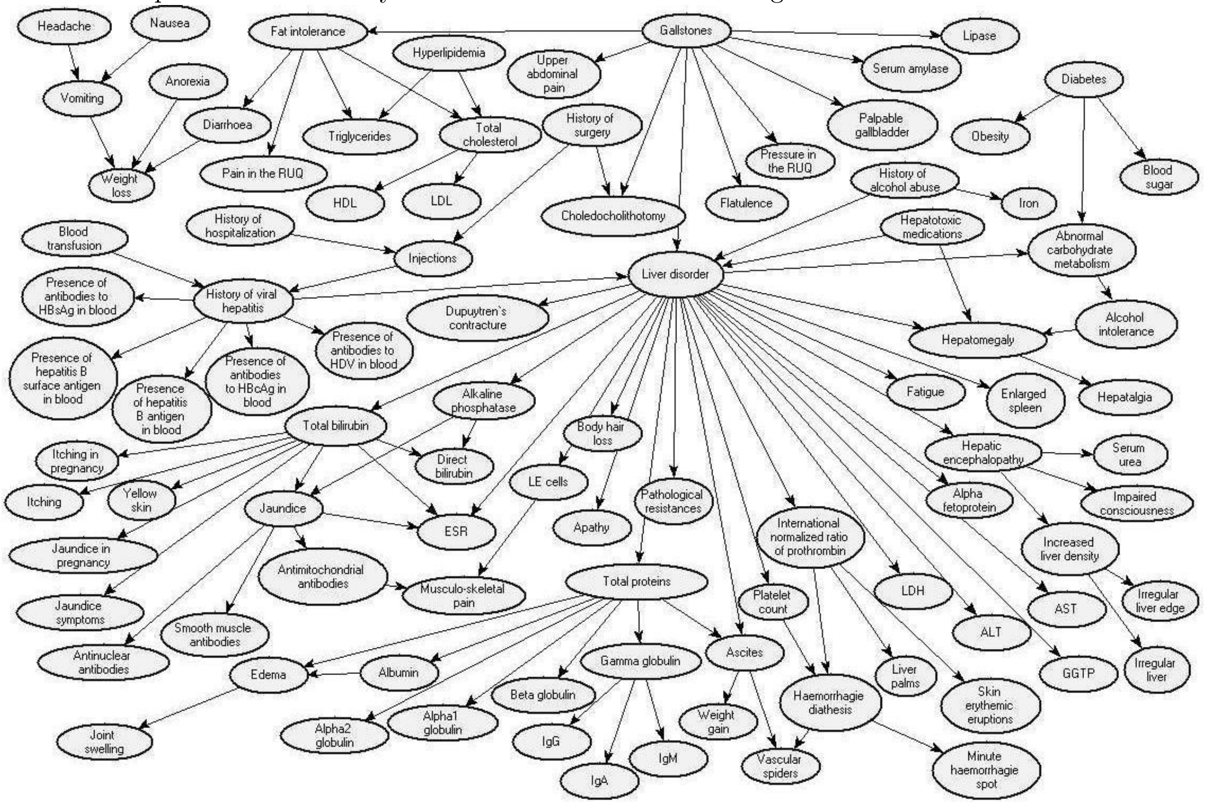
- **Intercausal**: given two possible causes of an event, given that the event has been decteded and one of the causes has been observed, which is the probability of the other cause?
- I know the apriori probability of a burglar: $P(B)$
- The alarm is set off and an earthquake is detected: $P(A = T) = P(E = T) = 1$
- I can compute the a posteriori probability that of a burglar, given the observed nodes: $P(B|E = T, A = T)$



- **Mixed**: given that a cause and the effect of an event have been observed (or not observed), which is the probability of the event?
- I know the apriori probability of an alarm: P(A);
- The earthquake is not detected and John calls me: $P(E = F) = P(J = T) = 1$
- I can compute the a posteriori probability that the alarm is set off, given the observed nodes: $P(A = T|E = F, J = T)$



Here an example of a realistic Bayesian Network used for Liver diagnosis:



Some variable values are binary, others are continuous but discretized to one of {low, normal, high, very high}

Conditional probabilities were obtained by counting from a data set of 600 test cases

Missing measurements in the test data can be significant -- the doctor felt that such a test was not worth taking

The disorder variable has 16 possible outcomes (16 possible liver disorders could be computed)

Model took about 40 hours to construct in consultation with medical experts

In trial runs, they generated the top 4 diagnostic conclusions with an accuracy of 34% for the most probable conclusion, and 67% that the actual diagnosis was found among the top 4

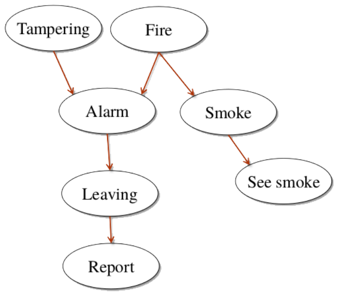
4.2.7 Independencies in BN

A BN encodes more independencies than the ones specified by construction. Thanks to **compactness** of BNs, we reduce the number of probabilities from to $O(2^n)$ to $O(n \cdot 2^k)$, with n number of variables, maximum fan out of a variable. There are two **problems** though: **(1)** there are still too many variables and **(2)** there are no data/experts for accurate assessment. **Solution**: Make stronger (approximate) independence assumptions.

Implied Conditional Independence relations in a Bnet Consider this case

- $P(Report|Alarm, Fire) = P(Report|Alarm)$?
- $P(Leaving|Seesmoke, Fire) = P(Leaving|Fire)$?

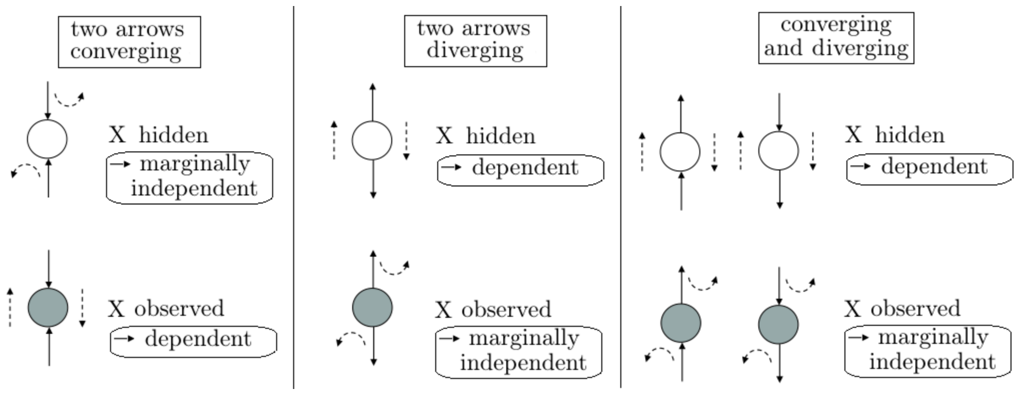
→ Knowing these independence relationships is very important!



Independence relationships allow us not to update the whole probabilities in the Bayesian Network even when new evidence occur.

Conditional independence in Bayes Nets In general, the conditional independence relationships encoded by a Bayes Net are best be explained by means of the "**Bayes Ball**" algorithm (due to Ross Shachter), which is as follows:

- Two (sets of) nodes A and B are conditionally independent (d-separated) given a set C if and only if there is no way for a ball to get from A to B in the graph, where the allowable movements of the ball are shown below:



- Hidden nodes are nodes whose values are not known, and are depicted as unshaded; observed nodes (the ones we condition on) are shaded. The dotted arcs indicate direction of flow of the ball.

The most interesting case is the **first column**, when we have two arrows converging on a node X (so X is a "leaf" with two parents):

- if X is hidden, its parents are marginally independent, and hence the ball does not pass through (the ball being "turned around" is indicated by the curved arrows);
- if X is observed, the parents become dependent, and the ball does pass through, because of the explaining away phenomenon.

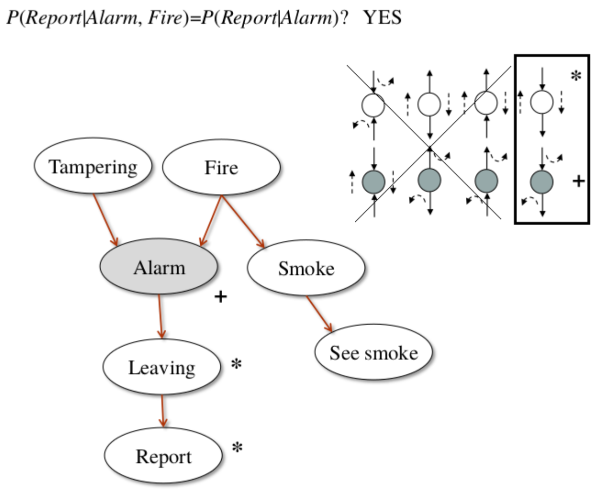
Now consider the **second column** in which we have two diverging arrows from X (so X is a "root"):

- if X is hidden, the children are dependent, because they have a hidden common cause, so the ball passes through;
- if X is observed, its children are rendered conditionally independent, so the ball does not pass through.

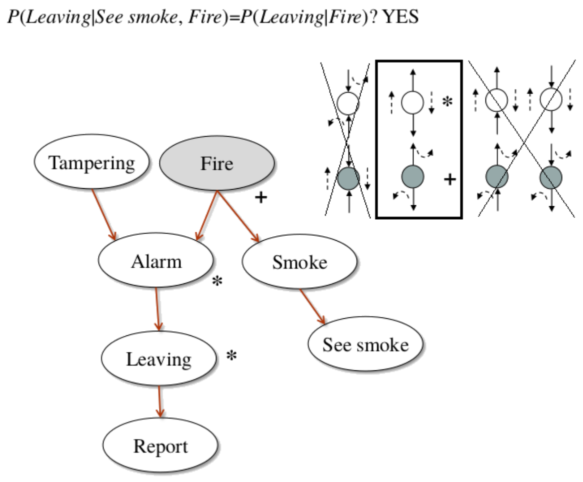
Finally, consider the **the third and the fourth column** in which we have an incoming and an outgoing arrow to X.

- It is intuitive that the nodes upstream and downstream of X are dependent iff X is hidden, because conditioning on a node breaks the graph at that point.

Now let's see some examples:



When an *Alarm* is observed, *Fire* and *Leaving* become conditionally independent, according to the Bayes Ball Algorithm (fourth column).
Therefore, the probability of a *Report*, given *Alarm* and *Fire*, is equal to the probability of *Report*, given *Alarm*.



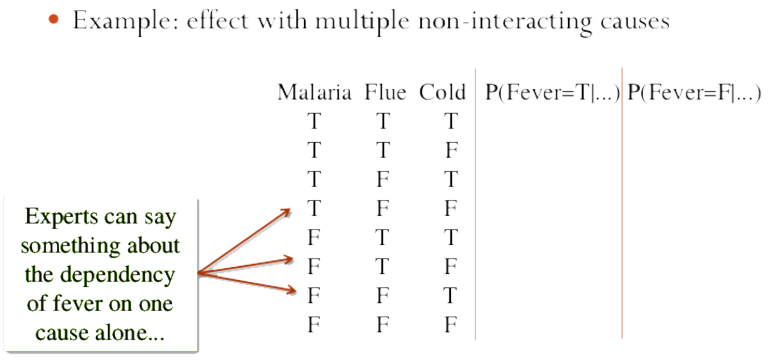
When a *Fire* is observed, *Leaving* and *See smoke* become conditionally independent, according to the Bayes Ball Algorithm (second column).
Therefore, the probability of *Leaving*, given *Fire* and *See smoke*, is equal to the probability of *Leaving*, given *Fire*.

4.2.8 Representation of Compact Conditional Distributions

Once we have established the topology of a Bnet, we still need to specify the conditional probabilities: how?

- From Data
- From Experts

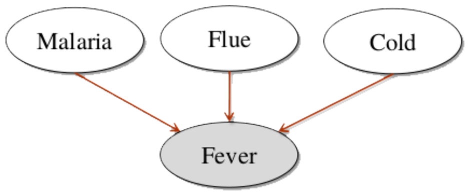
To facilitate acquisition, we aim for **compact representations** for which data/experts can provide accurate assessments.
Moreover, even if Bayesian Networks significantly reduce the dimension of the Joint Probability Distribution, CPT grows exponentially with number of parent: in realistic model of internal medicine with 448 nodes and 906 links 133,931,430 values are required. This means that often there are no data/experts for accurate assessment!.
We can try to simplify this problem, for example, considering an event like having the fever as generated by **non-interacting** causes such as a cold, a flue or contracting malaria.



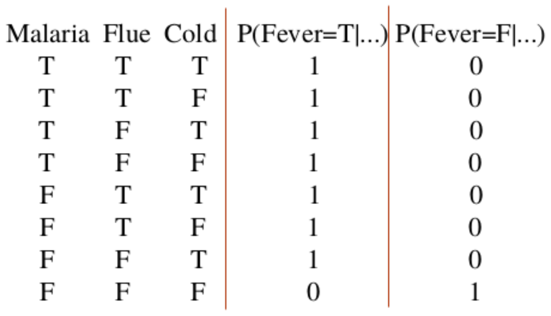
- Example: effect with multiple non-interacting causes

Malaria	Flue	Cold	P(Fever=T|...)	P(Fever=F|...)
T	T	T		
T	T	F		
T	F	T		
T	F	F		
F	T	T		
F	T	F		
F	F	T		
F	F	F		

Experts can say something about the dependency of fever on one cause alone...



We make such assumption in order to gather data because it is easier to check the dependency of fever on one cause alone. Still, in order to compute the truth table we can not use a logic OR because the output would be the following:



Malaria	Flue	Cold	P(Fever=T|...)	P(Fever=F|...)
T	T	T	1	0
T	T	F	1	0
T	F	T	1	0
T	F	F	1	0
F	T	T	1	0
F	T	F	1	0
F	F	T	1	0
F	F	F	0	1

In order to have more meaningful results, a possible solution is the Noisy-OR Distribution. The Noisy-OR model allows for **uncertainty** in the ability of each cause to generate the effect (e.g.. one may have a cold without a fever). Two assumptions are considered:

- All possible causes a listed;
- For each of the causes, whatever inhibits it to generate the target effect is independent from the inhibitors of the other causes.

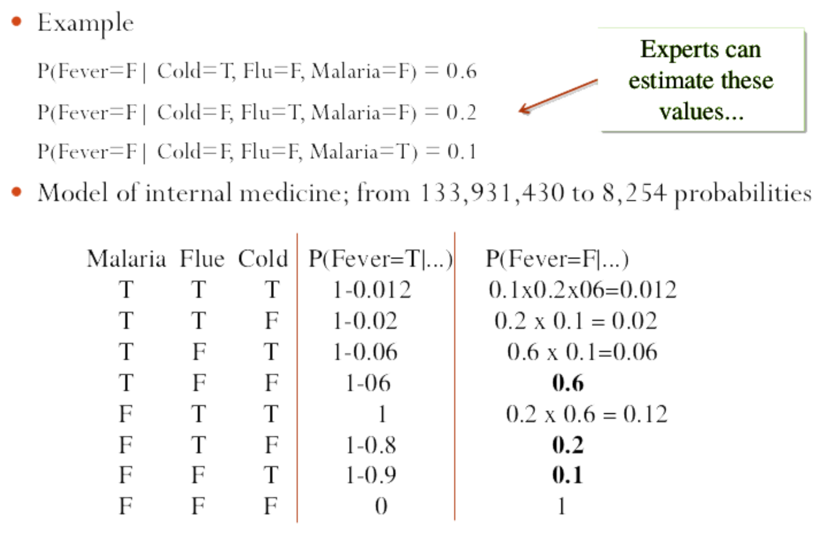
The **independent probability of failure** q_i for each cause alone is given by:

$$P(Effect = F \mid C_i = T, \text{and no other causes}) = q_i$$

$$P(Effect = F \mid C_1 = T, \ldots, C_j = T, C_{j+1} = F, \ldots, C_k = F) = \prod_{i=1}^{j} q_i$$

$$P(Effect = T \mid C_1 = T, \ldots, C_j = T, C_{j+1} = F, \ldots, C_k = F) = 1 - \prod_{i=1}^{j} q_i$$

Noisy-OR - Fever example:



- Example

P(Fever=F | Cold=T, Flu=F, Malaria=F) = 0.6

P(Fever=F | Cold=F, Flu=T, Malaria=F) = 0.2

P(Fever=F | Cold=F, Flu=F, Malaria=T) = 0.1

Experts can estimate these values...

- Model of internal medicine; from 133,931,430 to 8,254 probabilities

Malaria	Flue	Cold	P(Fever=T|...)	P(Fever=F|...)
T	T	T	1-0.012	0.1x0.2x06=0.012
T	T	F	1-0.02	0.2 x 0.1 = 0.02
T	F	T	1-0.06	0.6 x 0.1=0.06
T	F	F	1-06	**0.6**
F	T	T	1	0.2 x 0.6 = 0.12
F	T	F	1-0.8	**0.2**
F	F	T	1-0.9	**0.1**
F	F	F	0	1

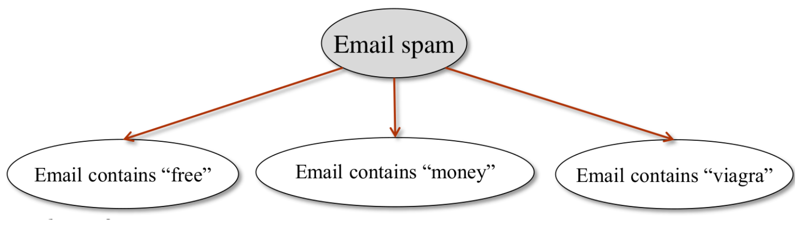
4.2.9 Naive Bayesian Networks

Naive Bayesian Classifier is a very simple and successful Bnet that allows to **classify** entities in a set of classes C, given a set of attributes.

Email Spam example - Determine whether an email is spam (only two classes spam=T and spam=F) After having established what are the useful attributes of an email, we can make two assumptions:

- The value of each attribute depends on the classification;
- (Naive) The attributes are **independent** of each other given the classification: P("bank"| "account" , spam=T) = P("bank" | spam=T).

Naive Bayesian Network Structure

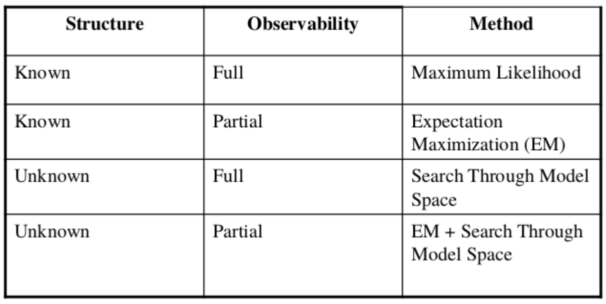


Data is easy to acquire: if you have a large collection of emails for which you know if they are spam or not, it is easy to compute CPT.
Now suppose Email contains "free"=T, Email contains "money"=T, Email contains "viagra"=T.
How can I say that P(spam=T|all successors) > P(spam=F|all successors)?

4.2.10 Learning in BN

One needs to specify two things to describe a BN: the graph topology (**structure**) and the **parameters** of each CPD. It is possible to learn both of these from data. However, learning structure is much harder than learning parameters.

Also, learning when some of the nodes are hidden, or we have missing data, is much harder than when everything is observed.This gives rise to 4 cases:



Structure	Observability	Method
Known	Full	Maximum Likelihood
Known	Partial	Expectation Maximization (EM)
Unknown	Full	Search Through Model Space
Unknown	Partial	EM + Search Through Model Space

Case 1 Known structure, full observability:

We assume that the goal of learning in this case is to find the values of the parameters of each CPD which maximizes the likelihood of the training data, which contains s cases (assumed to be independent).

- Define θ as the parameters to be learned, i.e., the conditional probabilities written in the CPT.
- $D = \{Y(1), ...Y(n)\}$ are the observation made in n subsequent steps (assumed to be independent), where $Y(i)$ is a vector that describes the observation for each node $Y_j^{(i)}$ at step i.
- We want to find θ which maximizes the likelihood of observing D, i.e.:

$$P(D|\theta) = \prod_{i=1}^{n} P(Y(i)|\theta)$$

The parameters are obtained by maximizing the likelihood or, equivalently, the **log likelihood**:

$$L(\theta) = \sum_{i=1}^{n} \log P(Y^{(i)}|\theta)$$

If the observation vector includes all variables in the Bayesian network, then each term factors as:

$$\log P(Y^{(i)}|\theta) = \log \prod_j P(Y_j^{(i)}|Y_{pa(j)}^{(i)}, \theta) = \sum_j \log P(Y_j^{(i)}|Y_{pa(j)}^{(i)}, \theta)$$

and then:

$$\sum_{i=1}^{n} \sum_j \log P(Y_j^{(i)}|Y_{pa(j)}^{(i)}, \theta)$$

!!! very important: the maximization of the likelihood **can be computed locally**, by considering each node and its parents separately $\longrightarrow$ consider estimating the Conditional Probability Table for the W node. If we have a set of training data, we can just count the number of times the grass is wet when it is raining and the sprinkler is on,

$$N(W = T, S = T, R = T)$$

the number of times the grass is wet when it is raining and the sprinkler is off,

$$N(W = T, S = F, R = T)$$

and so on...

Given these counts (which are the sufficient statistics), we can find the Maximum Likelihood Estimate of the CPT as follows:

$$P(W = w|S = s, R = r) \approx N(W = w, S = s, R = r)/N(S = s, R = r)$$

where the denominator is

$$N(S = s, R = r) = N(W = 0, S = s, R = r) + N(W = 1, S = s, R = r)$$

Thus "learning" **just** amounts to **counting** (in the case of multinomial distributions), **but** for other kinds of distributions, more complex procedures are necessary.

Case 2 Known structure, partial observability:

When some of the nodes are hidden, we can use the EM (Expectation Maximization) algorithm to find a (locally) optimal Maximum Likelihood Estimate of the parameters.
The basic idea behind EM is that, if we knew the values of all the nodes, learning (the M step) would be easy, as we saw above.
In the E step, we compute the expected values of all the nodes using an inference algorithm.
In the M step we treat these expected values as though they were observed (distributions).

E step:

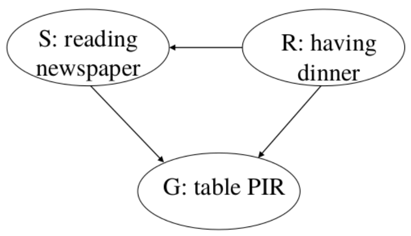
For hidden nodes we replace the observed counts of the events with the number of times we expect to see each event.
For example, in the case of the W node:

$$P(W = w|S = s, R = r) \approx EN(W = w, S = s, R = r)/EN(S = s, R = r)$$

where EN(.) returns the expected counts.
Given the expected counts, we maximize the parameters, and then recompute the expected counts, etc. This **iterative** procedure is guaranteed to converge to a local maximum of the likelihood surface.
When nodes are hidden, inference becomes a subroutine which is called by the learning procedure; hence fast inference algorithms are crucial.

4.2.11 Example of context assessment

Suppose that there are two events which could cause the PIR to detect a person: either the user is having dinner or he is reading the newspaper. Also, suppose that having dinner has a direct effect on reading the newspaper: when the user is having dinner, usually he does not read news. Then the situation can be modeled with a Bayesian network:



All three variables have two possible values T (for true) and F (for false). We assume to know the structure, which could be learned, and to learn only parameters. Given the Training Set:

R	**S**	**G**		**R**	**S**	**G**
T	T	T		F	T	T
T	F	T		T	T	T
F	F	F		F	T	T
F	T	F		T	F	T

We can make the following computations:

$$P(R=T) \approx N(R=T)/8 = 0.5$$
$$P(R=F) \approx N(R=F)/8 = 0.5$$

$$N(R=T) \;=\; 4$$
$$P(S=T \mid R=T) \approx N(S=T, R=T)/N(R=T) = 0.5$$
$$P(S=F \mid R=T) \approx N(S=F, R=T)/N(R=T) = 0.5$$

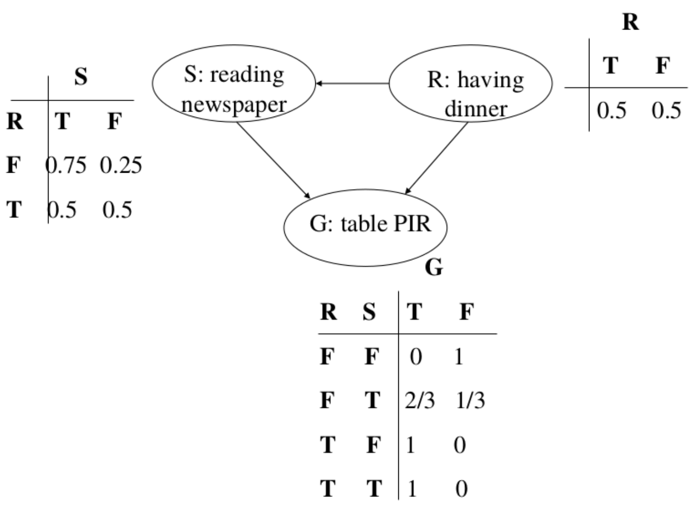
$$N(R=F) \;= 4$$
$$P(S=T \mid R=F) \approx N(S=T, R=F)/N(R=F) = 3/4$$
$$P(S=F \mid R=F) \approx N(S=F, R=F)/N(R=F) = 1/4$$

$$N(S=T,R=T) \;=\; 2$$
$$P(G=T \mid S=T, R=T) \approx N(G=T, S=T, R=T)/N(S=T, R=T) = 1$$
$$P(G=F \mid S=T, R=T) \approx N(G=F, S=T, R=T)/N(S=T, R=T) = 0$$

$$N(S=F,R=T) \;= 3$$
$$P(G=T \mid S=T, R=F) \approx N(G=T, S=T, R=F)/N(S=T, R=F) = 2/3$$
$$P(G=F \mid S=T, R=F) \approx N(G=F, S=T, R=F)/N(S=T, R=F) = 1/3$$

$$N(S=T,R=F) \;= 2$$
$$P(G=T \mid S=F, R=T) \approx N(G=T, S=F, R=T)/N(S=F, R=T) = 1$$
$$P(G=F \mid S=F, R=T) \approx N(G=F, S=F, R=T)/N(S=F, R=T) = 0$$

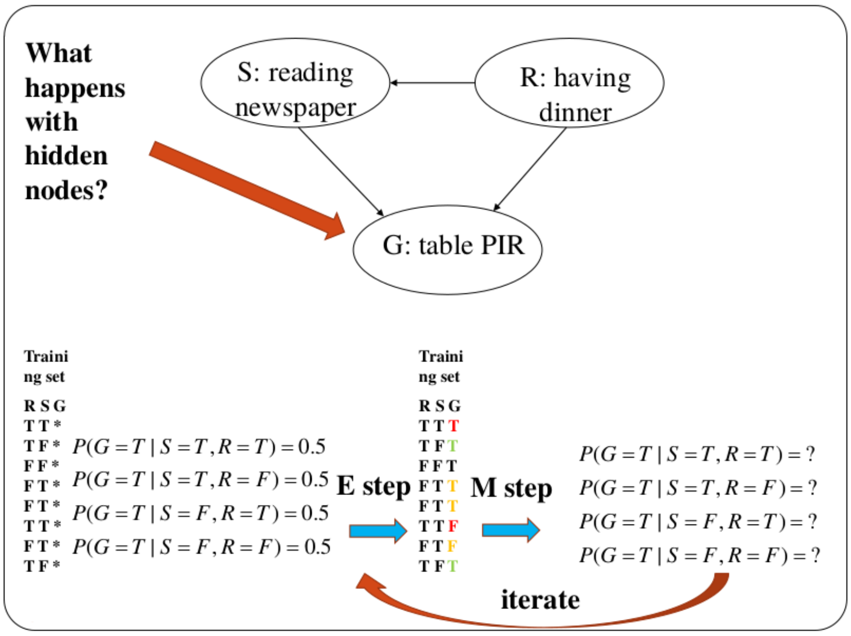
$$N(S=F,R=F) \;= 1$$
$$P(G=T \mid S=F, R=F) \approx N(G=T, S=F, R=F)/N(S=F, R=F) = 0$$
$$P(G=F \mid S=F, R=F) \approx N(G=F, S=F, R=F)/N(S=F, R=F) = 1$$



	S	
R	T	F
F	0.75	0.25
T	0.5	0.5

R	
T	F
0.5	0.5

		G	
R	S	T	F
F	F	0	1
F	T	2/3	1/3
T	F	1	0
T	T	1	0



Training set

```
R S G
T T *
T F *
F F *
F T *
F T *
T T *
F T *
T F *
```

$P(G=T \mid S=T, R=T) = 0.5$
$P(G=T \mid S=T, R=F) = 0.5$
$P(G=T \mid S=F, R=T) = 0.5$
$P(G=T \mid S=F, R=F) = 0.5$

Training set

```
R S G
T T T
T F T
F F T
F T T
F T T
T T F
F T F
T F T
```

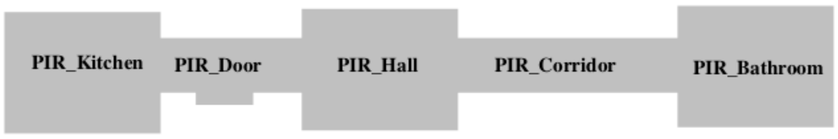
$P(G=T \mid S=T, R=T) = ?$
$P(G=T \mid S=T, R=F) = ?$
$P(G=T \mid S=F, R=T) = ?$
$P(G=T \mid S=F, R=F) = ?$

4.2.12 Dynamic Bayesian Network

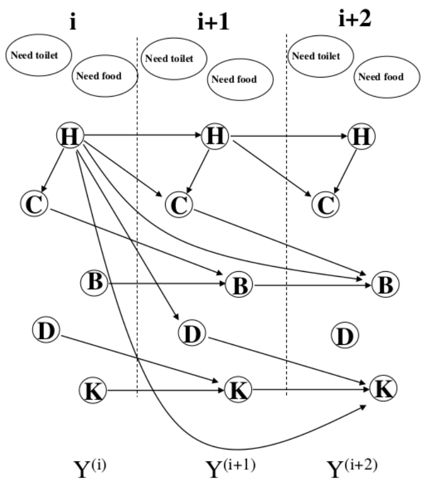
So far we have used Bayesian networks to perform inference in **static environments**, for instance, the system keeps collecting evidence to diagnose the cause of a fault in a system (e.g., a car). The environment (values of the evidence, the true cause) does not change as I gather new evidence

What does **change**? The system?s **beliefs** over possible causes. What are we missing? **Time**.

When I go to the bathroom, at time i I am detected by PIR_hall, at time $i+1$ I am detected by $PIR\_corridor$... When I go to the kitchen, at time i I am detected by PIR_hall, at time $i+1$ I am detected by PIR_door... Then, observing PIR_hall is not sufficient to explain the causes:



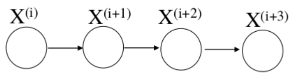
It is possible to build a Dynamic Bayesian Network in which every **slice** correspond to a time interval, the **links** between different slices determines how a variable at time i influences variables at time $i+n$ (the model is very complex):



Information at a higher level of abstraction can be incorporated (arrows not shown in figure). This allows **symptomatic reasoning** like “which is the most probable cause of my going to the kitchen?”

4.2.13 Markov Model

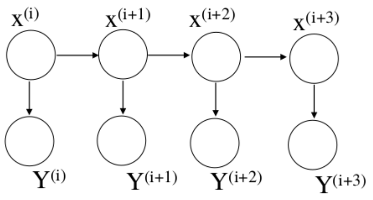
First Order Markov Model Given a variable $X(i)$ (corresponding to a set of observable nodes at time i), the network is defined as a **first order Markov Model** if the value of each node at time (i) is influenced only by its value at time $(i-1)$.



This can also be said as follows:

$$P(X^{(T)}, X^{(T-1)}, X^{(T-2)}, ...) \equiv P(X^{(T)}|X^{(T-1)})P(X^{(T-1)}|X^{(T-2)})...P(X^{(1)}|X^{(0)})$$

We make a **stationary process assumption**: the mechanism that regulates how state variables change overtime is stationary, that is it can be described by a single transition model.



A very common way to see Markov Model is to assume that **observations** are **dependent on** a discrete **hidden** variable called the **state** which can assume N values, and that the sequence of states is a Markov Process (I.e., it has the Markov property). In this this case we will speak about a *Hidden Markov Models*.

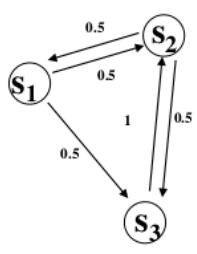
The transition probabilities from states s can be expressed in transition tables:

	s_1	s_2	s_3
s_1	.5	.5	0
s_2	.3	.3	.4
s_3	.2	0	.8

Markov processes are interesting in that they allow a very compact representation $P(x^{(i)}) = \pi^{(i)} = [\pi_1^{(i)}, ...\pi_N^{(i)}]^T$ is the probability distribution at time i, which describes the probability of being in each of the N states

$P(x^{(0)}) = \pi^{(0)}$ is the initial probability distribution over all possible states.

The conditional probabilities $P(x_k^{(i+1)}|x_j^{(i)})$ are stored in a NxN transition matrix P, where the element $p_{jk} = P(x_k^{(i+1)}|x_j^{(i)})$ specifies the probability of going from state i to state j in two subsequent time steps.



$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\pi^{(i+1)} = P^T \pi^{(i)}$$

$$\pi^{(0)} = [0 \quad 0 \quad 1]^T \qquad \pi^{(0)} = [0 \quad 1 \quad 0]^T$$

$$\pi^{(1)} = [0 \quad 1 \quad 0]^T \qquad \pi^{(1)} = [0.5 \quad 0 \quad 0.5]^T$$

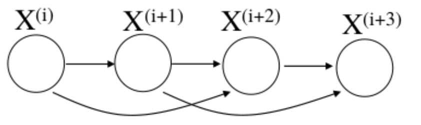
$$\pi^{(i+2)} = P^T P^T \pi^{(i)}$$

$$\pi^{(2)} = [0 \quad 0.75 \quad 0.25]^T$$

Starting from this representation it is possible to infer interesting properties:

- ff the matrix P is irreducible (I.e., the corresponding graph is strongly connected), P has one eigenvector with eigenvalue 1 (Perron-Frobenius theorem) the distribution $\pi^{(i)}$ tends to a steady-state distribution π as i tends to infinity;
- strongly connected: it is possible to find a path from every node to every other node;
- the steady state distribution can be computed as $\pi = P^T\pi$;
- it gives information about the probability of each state after a transient behaviour.

Second order Markov Model Given a variable $X(i)$ (corresponding to a set of observable nodes at time i), the network is defined as a **second order Markov Model** if the value of each node at time (i) is influenced only by its values at time $(i-1)$ and $(i-2)$.



What can Markov Models be used for? Consider the sentence "Book me a room near UBC".
We could be interested in:

- **Assigning a probability** to the sentence (e.g. Part-of-speech tagging, word-sense disambiguation, probabilistic)
- **Parsing** a word or **predicting** the next word (e.g. Speech recognition, hand-writing recognition, augmentative communication for the disabled)

But how can we compute the Joint Probability of a sequence of n words?

$$P(w_1, .., w_n) \text{ is impossible to estimate!}$$

Assuming 10^5 words and considering that an average sentence contains $n = 10$ words, this yields 10^{50} probabilities!
For example, Google language repository (22 Sept. 2006) contained "only": 95,119,665,584 sentences (10^{11})
Most sentences will not appear or appear only once: we can assume that a sentence is generated by a first order Markov Chain:

$$P(w_1,\dots,w_n) = P(w_1 \mid < S >)P(w_2 \mid w_1)P(w_3 \mid w_2)\dots P(w_k \mid w_{k-1})$$

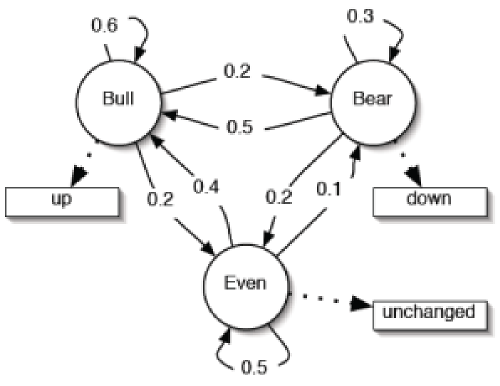
$$P(\text{The big red dog barks}) =$$
$$P(\text{The} \mid < S >)P(\text{big} \mid \text{The})P(\text{red} \mid \text{big})P(\text{dog} \mid \text{red})P(\text{barks} \mid \text{dog})$$

These can be computed! It is sufficient to check how many times a given pattern of two words can be found!

Hidden Markov Model A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with **unobserved state**.
An HMM can be considered as the simplest dynamic Bayesian network. In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters.
In a hidden Markov model, the **state** is not directly visible, but **output**, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of **tokens** generated by an HMM gives some information about the sequence of states.
Hidden Markov models are especially **known** for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

4.2.14 Markov Processes

Imagine to have a Markov process, and that every observation is **deterministically** associated to a state. The model has three states, Bull, Bear and Even, and three index observations up, down, unchanged. Given a sequence of observations we can easily verify which state sequence produced those observations.



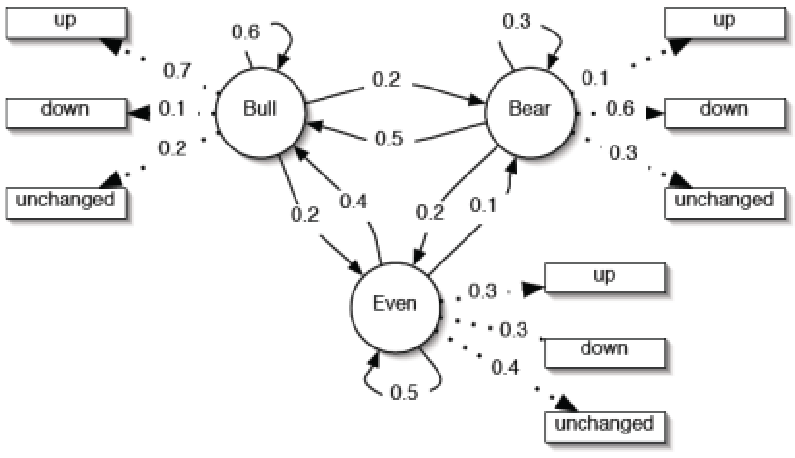
E.g.: up-down-down corresponds to Bull-Bear-Bear, and the probability of the sequence is $0.2 * 0.3 * 0.3$.

The key difference with a Markov Model is that now if we have the observation sequence *up-down-down* then we **cannot say exactly** what state sequence produced these observations and thus the state sequence is 'hidden'.
We can **however** calculate:

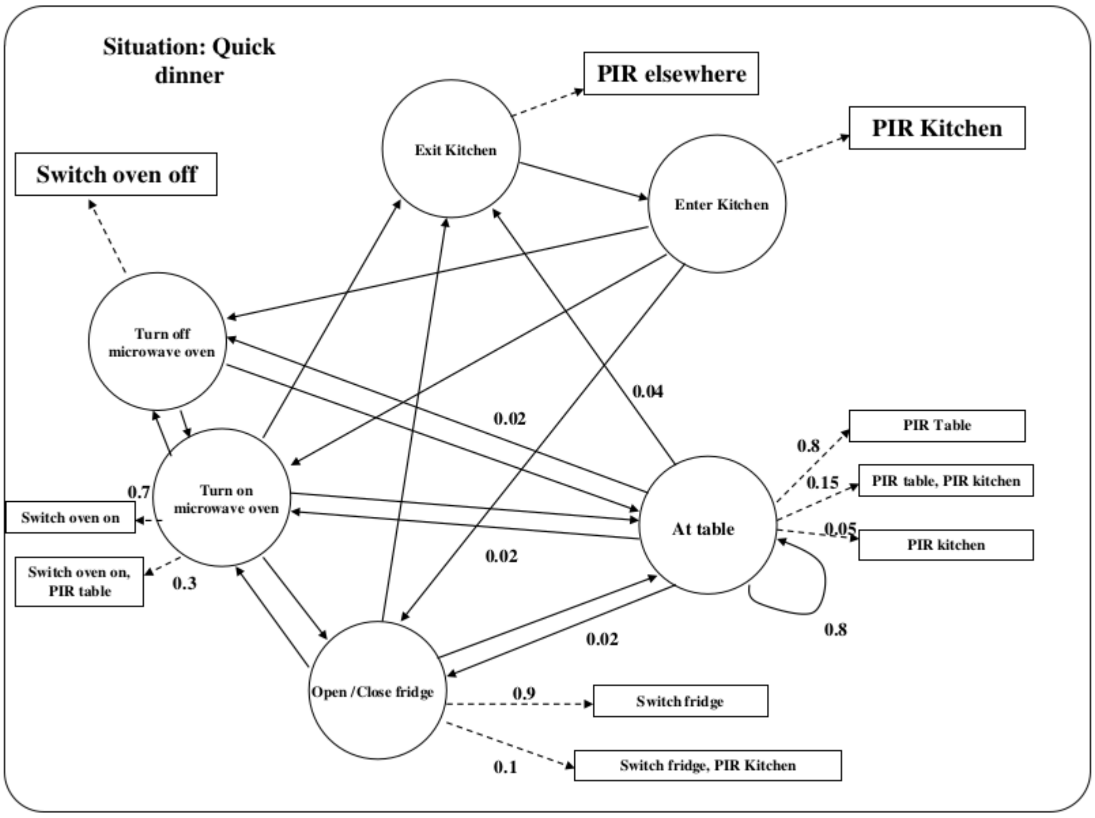
1. which state sequence was most likely to have produced the observations;

2. the probability that the model produced the sequence.

Point 1 is not different from what we already saw, point 2 is something different... we imagine to have different models, and to find the ones that fits the best.



Another example can be a hidden Markov Model of a quick dinner situation.



But what is the main problem of such a model? It can not consider states before the previous one.
For example, if you *turn on the microwave* and after that you *open the fridge*, you could reasonably expect that a *turn off the microwave* will probably happen in the future, but a first order Markov model (like this one) can't take that into account and that "information" will be lost.

Formal definition of HMM The **formal definition of a HMM** is as follows:

$$\lambda = (A, B, \pi)$$

S is our **state alphabet set**, and V is the**observation alphabet** set:

$$S = (s_1, s_2, ..., s_N)$$
$$V = (v_1, v_2, ..., v_M)$$

We define Q to be a **fixed state sequence** of length T, and corresponding **observations** O:

$$Q = (q_1, q_2, ..., q_T)$$
$$O = (o_1, o_2, ..., o_T)$$

A is a transition array, storing the probability of state j following state i . Note the state transition probabilities are independent of time:

$$A = [a_{ij}] \ , \ a_{ij} = P(q_t = s_j j | q_{t-1} = s_i)$$

B is the observation array, storing the probability of observation k being produced from the state j, independent of t:

$$B = [b_i(k)] \ , \ b_i(k) = P(x_t = v_k | q_t = s_i)$$

π is the initial probability array:

$$\pi = [\pi_i] \ , \ \pi_i = P(q_1 = s_i)$$

Assumptions Two assumptions are made by the model:

- Markov assumption, states that the current state is dependent only on the previous state, this represents the memory of the model:

$$P(q_t|q_1^{t-1}) = P(q_t|q_{t-1})$$

- The independence assumption states that the output observation at time t is dependent only on the current state, it is independent of previous observations and states:

$$P(o_t|o_1^{t-1}, q_1^t) = P(o_t|q_t)$$

Evaluation of a HMM Given a HMM, and a sequence of observations, we would like to be able to compute the probability of the observation sequence given a model. This problem could be viewed as one of **evaluating** how well a model **predicts** a given observation sequence, and thus allow us to choose the most appropriate model from a set.

The probability of the observations O for a specific state sequence Q is:

$$P(O|Q,\lambda) = \prod_{t=1}^{T} P(o_t|q_t,\lambda) = b_{q_1}(o_1) * b_{q_2}(o_2) * ... * b_{q_T}(o_T)$$

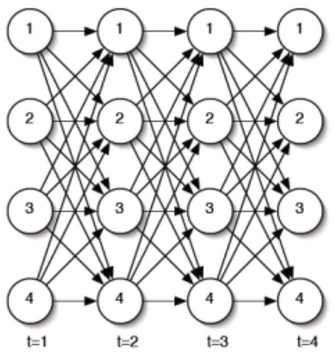
and the probability of the state sequence is:

$$P(Q|\lambda) = \pi_{q_1} * a_{q_1 q_2} * a_{q_2 q_3} * ... * a_{q_{T-1} q_T}$$

so we can calculate the probability of the observations given the model as:

$$P(O|\lambda) = \sum_Q P(O|Q,\lambda)P(Q|\lambda) = \sum_{(q_1 ... q_T)} \pi_{q_1} * b_{q_1}(o_1) * a_{q_1 q_2} * b_{q_2}(o_2) * ... * a_{q_{T-1} q_T * b_{q_T}(o_T)}$$

This result allows the **evaluation** of the probability of O, but to evaluate it directly would be exponential in T.
A better approach is to recognise that many redundant calculations would be made by directly evaluating the equation, and therefore caching calculations can lead to **reduced complexity**.



We implement the cache as a trellis of states at each time step, calculating the cached valued (called $P(O|\lambda)$) for each state as a sum over all states at the previous time step.
We define the **forward probability** variable as $\alpha_t(i) = P(o_1 o_2 ... o_t, q_t = s_i|\lambda)$.

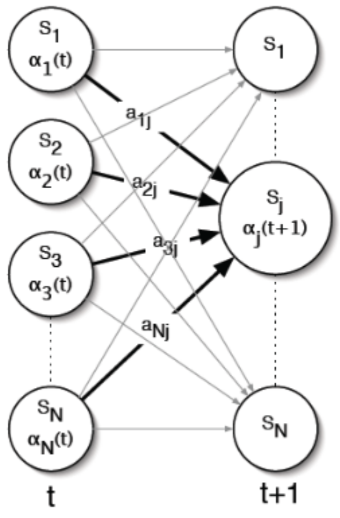
The algorithm for this process is called **the forward algorithm** and is as follows:

- Initialisation:

$$\alpha_1(i) = \pi_i b_i(o_1) \quad , \quad 1 \le i \le N$$

- Induction

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}) \quad , \quad 1 \le t \le T-1 \quad , \quad 1 \le j \le N$$



- Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

It is apparent that by caching values the forward algorithm reduces the complexity of calculations involved to N^2T rather than $2TN^T$.
In particular, with hidden variables $H = X^{(1)}...X^{(n)}$, the log likelihood cannot be composed in local terms to be maximized separately.

$$L(\theta) = \sum_{i=1}^{N} logP(Y^{(i)}|\theta) = \sum_{i=1}^{N} log \sum_{X^{(i)}} P(Y^{(i)}, X^{(i)}|\theta)$$

By considering a single observation, it holds:

$$\begin{aligned}\log \sum_X P(Y,X\mid\theta) &= \log \sum_X Q(X)\frac{P(Y,X\mid\theta)}{Q(X)}\\ &\geq \sum_X Q(X)\log\frac{P(Y,X\mid\theta)}{Q(X)}\\ &= \sum_X Q(X)\log P(Y,X\mid\theta) - \sum_X Q(X)\log Q(X)\\ &= F(Q,\theta)\end{aligned}$$

Jensen inequality

Entropy of Q

The **Expectation Maximization** algorithm alternates between maximizing F with respect to Q and θ, respectively, holding the other fixed.
Starting from some initial parameters θ_0, we iteratively compute:
E step $Q_{k+1} \leftarrow \arg\max_{Q} F(Q, \theta)$
M step $\theta_{k+1} \leftarrow \arg\max_{\theta} F(Q_{k+1}, \theta)$
It is easy to show that the maximum in the E step results when

$$Q_{k+1}(X) = P(X|Y, \theta_k)$$

In fact:

$$L(\theta_k) = \log P(Y \mid \theta_k) = \log \sum_X P(Y, X \mid \theta_k)$$

$$\geq \sum_X P(X \mid Y, \theta_k) \log \frac{P(Y, X \mid \theta_k)}{P(X \mid Y, \theta_k)} = \sum_X P(X \mid Y, \theta_k) \log \frac{P(X \mid Y, \theta_k) P(Y \mid \theta_k)}{P(X \mid Y, \theta_k)} =$$

$$= \log P(Y \mid \theta_k) \sum_X P(X \mid Y, \theta_k) = \log P(Y \mid \theta_k)$$

The M step becomes:

$$\theta_{k+1} \leftarrow \arg\max_\theta \textstyle\sum_X P(X|Y,\theta_k) logP(X,Y|\theta)$$

Since $logP(X,Y|\theta)$ contains both hidden and observed variables, it can be factored as before as the sum of log probabilities of each node given its parents.
By referring as Z as a generic (hidden or observable) node:

$$\theta_{k+1} \leftarrow \arg\max_\theta \textstyle\sum_X P(X|Y,\theta_k) \sum_Z logP(Z|pa(Z),\theta)$$

For observable nodes compute:

$$\textstyle\sum_X P(X|Y,\theta_k) \sum_Y logP(Y|pa(Y),\theta) = \sum_Y logP(Y|pa(Y),\theta)$$

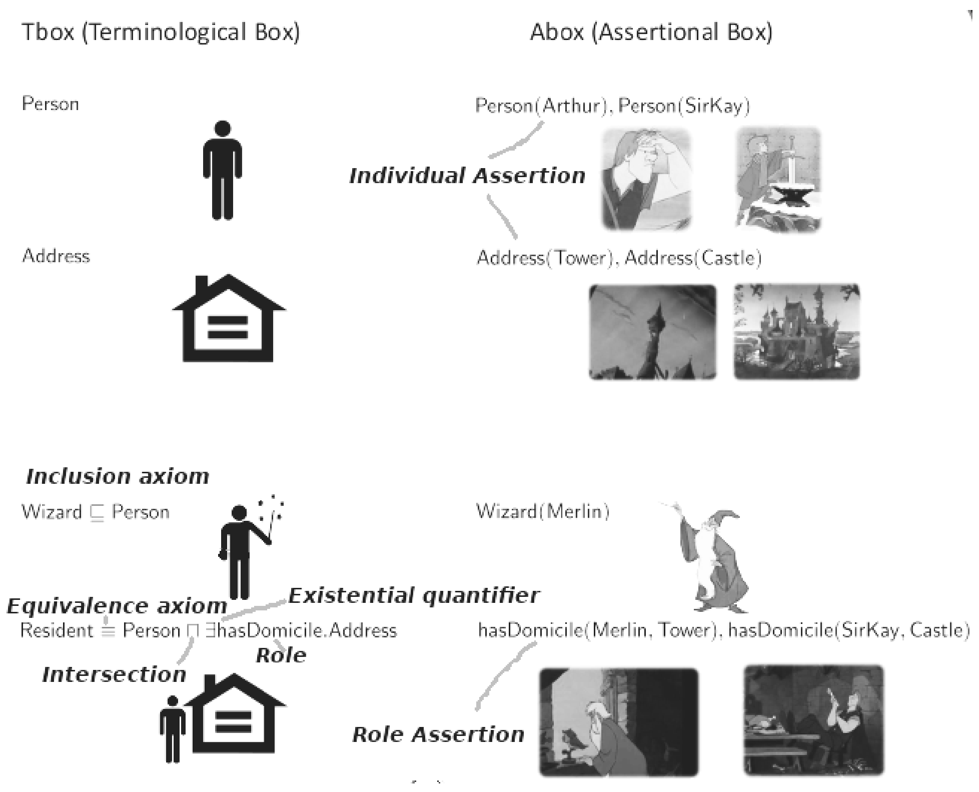
4.3 Context awareness

4.3.1 State of the art

Temporal patterns are used in order to develop a more relevant Context Awareness. For example, if the stove sensors detect somebody **cooking** at time t and the table PIR sensor detects somebody **sitting** at the table at time $t+\Delta t$, then somebody is **having a meal**. While temporal patterns are supported in **Allen's algebra** *(B. Gottfried, H. Guesgen and S. Hubner, 2006 ; M. Cirillo, F. Lanzellotto, F. Pecora and A. Saffiotti, 2009)* and in **temporal data mining and machine learning** *(P. Rashidi and D. Cook, 2009)*, they are not supported in conventional **Ontologies**.
Temporal constructs, like Temporal RDF, and Temporal Description Logics were developed in order to add **temporal functionalities** into ontologies, but they were not compatible with existing standard languages, tools for editing and reasoning.
Another attempt was developing a **rule-based reasoning**, but it increased the complexity in design and reasoning.
In order to recognize temporal patterns of events, a system requires an ontology implemented in OWL DL with basic reasoning mechanisms, without any additional rule-based mechanisms.
Such system has been implemented in $\mathcal{EL}^{++}$ (a lightweight Desciption Logic) which is implemented in the OWL 2 EL profile and allows for reasoning in polynomial time *(F. Baader, S. Brand and C. Lutz, 2005)*

4.3.2 Description Logic Basic Principles

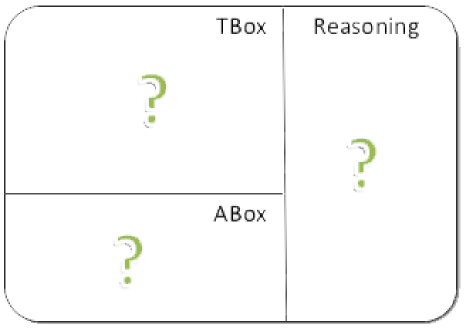
The basic principles of Description Logic are described in the following picture:


Tbox (Terminological Box)
Abox (Assertional Box)
Person
Person(Arthur), Person(SirKay)
Individual Assertion
Address
Address(Tower), Address(Castle)
Inclusion axiom
Wizard ⊑ Person
Wizard(Merlin)
Equivalence axiom
Existential quantifier
Resident ≡ Person ⊓ ∃hasDomicile.Address
hasDomicile(Merlin, Tower), hasDomicile(SirKay, Castle)
Role
Intersection
Role Assertion

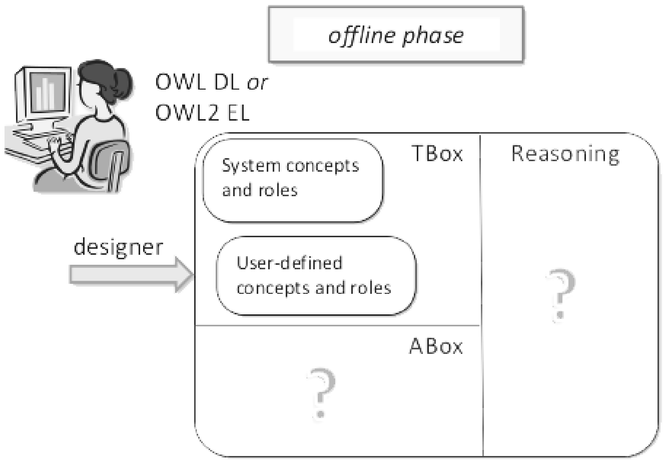
The constructors available to build complex descriptions determine the **expressiveness** of a specific sub-language.
$\mathcal{AL}$: negation of atomic concepts, concept intersection, universal value restrictions and limited existential quantification.
$\mathcal{SHOIN}^{(\mathcal{D})}$: negation of non atomic concepts, union of concepts, full existential quantification, role hierarchies, usage of nominals in concept definitions, inverse on roles, role cardinality restrictions, and usage of basic data types.
$\mathcal{EL}$: Since what we are interested in is the instance checking for context assessment, it has been proven that if the ontology is written in EL description logic the checking time will be polynomial! That's very good since we have an upper bound.

4.3.3 System Architecture

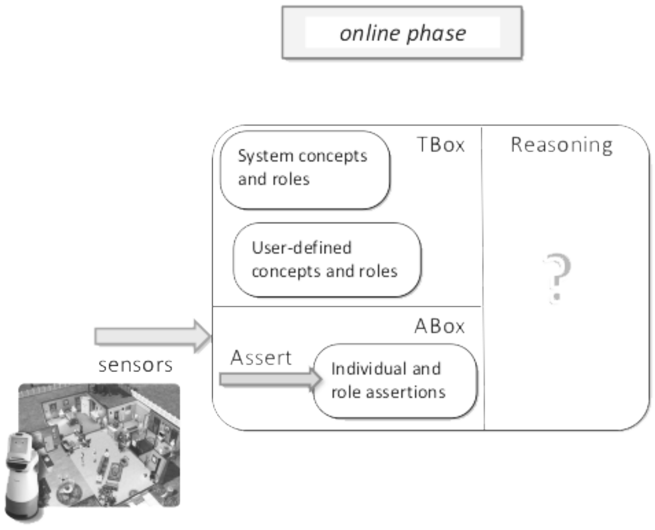
The system architecture comprises a TBox, an ABox and Reasoning.



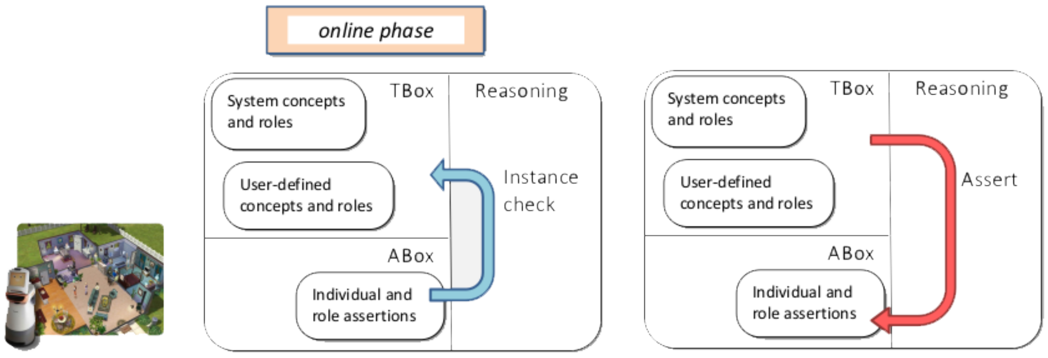
During the *offline phase*, the designer defines system concepts and roles by designing the TBox:



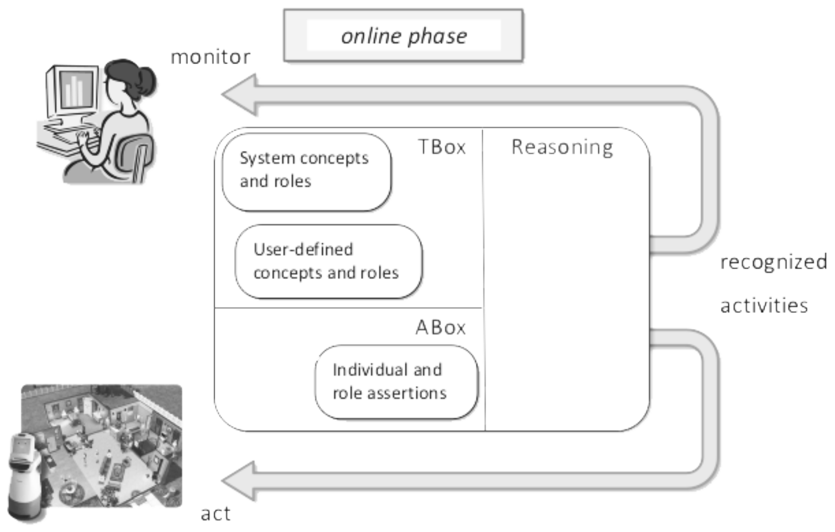
During the *online phase*, sensors asserts individuals and role assertions by populating the ABox:


online phase
System concepts and roles
TBox
Reasoning
User-defined concepts and roles
?
ABox
sensors
Assert
Individual and role assertions

During the reasoning in the online phase, **instance checking** (e.g. Merlin is an instance of Mage and he has a domicile → he is also an instance of Resident) and **assertion** from TBox to ABox are performed.


online phase
System concepts and roles
TBox
Reasoning
User-defined concepts and roles
Instance check
ABox
Individual and role assertions
System concepts and roles
TBox
Reasoning
User-defined concepts and roles
Assert
ABox
Individual and role assertions

The reasoning leads to the activities recognition and to actions in the environment while the designer can monitor the behavior of the system.


online phase
monitor
System concepts and roles
TBox
Reasoning
User-defined concepts and roles
recognized activities
ABox
Individual and role assertions
act

4.3.4 Offline phase: designing the TBox

When we define the TBox during the offline phase, we need to design:

1. system concepts and roles;

2. user-defined concepts roles.

Defining system concepts means formalizing generic ideas like :

- **sensor** as any physical or virtual source of information

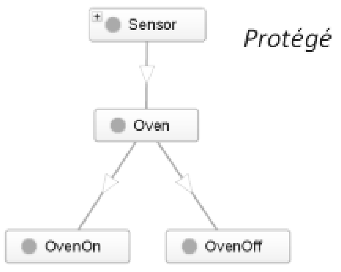
- **sensor state** as the most recent value returned by a sensor, which can result from a non-uniform sampling

- the definition of **event**

- the definition of **time intervals**,

- etc..

For example, in a smart home system, different types of sensors (*presence sensors, clocks, item sensors, etc..*) can be included in the system concept of *Sensor*.
User-defined concepts are derived from system concepts and are related to the application use case:



For example, an *Oven* and its states *OvenOn* and *OvenOff* are user-defined concept included in the system concept of *Sensor*.

System roles formalize generic properties (e.g. $\exists StartsAt.Integer$) and relationships (e.g. $\exists before^{+}$) between concepts. User-defined roles are related to user-defined concepts.

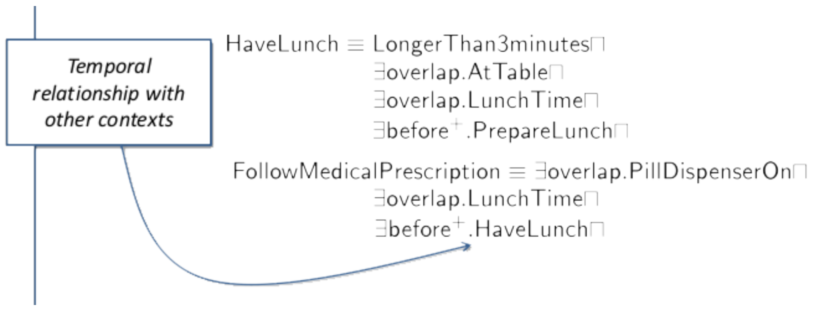
$$\begin{aligned}\mathsf{Event} \equiv\ & \exists \mathsf{overlap.Sensor} \sqcap \exists \mathsf{startsAt.Integer} \sqcap \\ & \exists \mathsf{before}^{+}.\mathsf{Event}\end{aligned}$$

$$\begin{aligned}\mathsf{PrepareLunch} \equiv\ & \mathsf{Event} \sqcap \\ & \exists \mathsf{overlap.OvenOn} \sqcap \\ & \exists \mathsf{overlap.LunchTime}\end{aligned}$$

For our system, the **context** is a configuration of sensor states holding in a **time interval**. In the previous figure, we can notice that *Prepare Lunch* is equivalent to the configuration of sensor states where *OvenOn* and *LaunchTime* are observed. By introducing temporal constraints concepts in our system such as:

$$\begin{aligned}&\mathsf{LongerThan} \sqsubseteq \mathsf{Event}\\ &\mathsf{LongerThan1minute} \sqsubseteq \mathsf{LongerThan}\\ &\mathsf{LongerThan5minutes} \sqsubseteq \mathsf{LongerThan1minute}\\ &\mathsf{LongerThan10minutes} \sqsubseteq \mathsf{LongerThan5minutes}\\ &\ldots\\ &\mathsf{ShorterThan} \sqsubseteq \mathsf{Event}\\ &\mathsf{ShorterThan10minutes} \sqsubseteq \mathsf{ShorterThan}\\ &\mathsf{ShorterThan5minutes} \sqsubseteq \mathsf{ShorterThan10minutes}\\ &\mathsf{ShorterThan1minute} \sqsubseteq \mathsf{ShorterThan5minutes}\\ &\ldots\end{aligned}$$

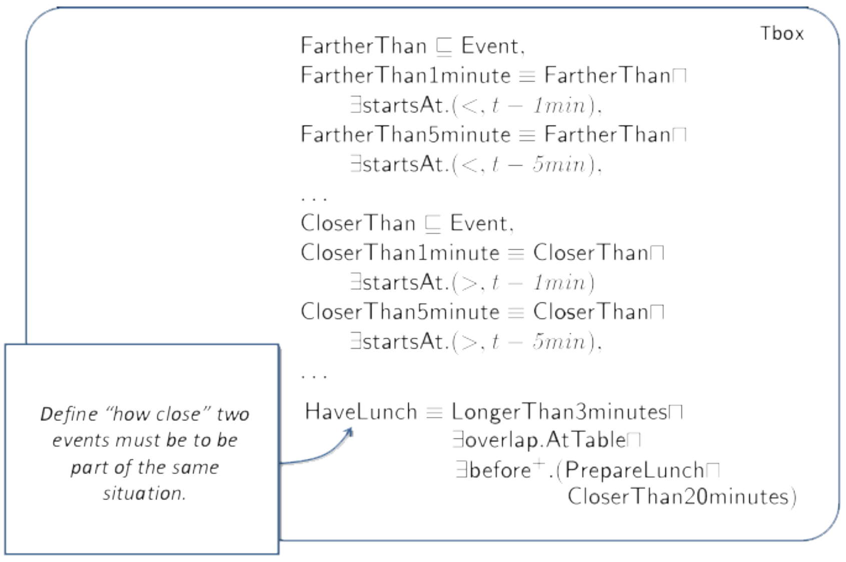
We can define temporal relationships between concepts, being able to define a **situation** as a temporal sequence of contexts:



$$\begin{aligned}\mathsf{HaveLunch} \equiv\ & \mathsf{LongerThan3minutes} \sqcap \\ & \exists \mathsf{overlap.AtTable} \sqcap \\ & \exists \mathsf{overlap.LunchTime} \sqcap \\ & \exists \mathsf{before}^{+}.\mathsf{PrepareLunch} \sqcap\end{aligned}$$

$$\begin{aligned}\mathsf{FollowMedicalPrescription} \equiv\ & \exists \mathsf{overlap.PillDispenserOn} \sqcap \\ & \exists \mathsf{overlap.LunchTime} \sqcap \\ & \exists \mathsf{before}^{+}.\mathsf{HaveLunch} \sqcap\end{aligned}$$

We can also define system concepts using concrete domains:


Tbox
Define "how close" two events must be to be part of the same situation.

$$\begin{aligned}&\mathsf{FartherThan} \sqsubseteq \mathsf{Event},\\ &\mathsf{FartherThan1minute} \equiv \mathsf{FartherThan} \sqcap\\ &\quad \exists \mathsf{startsAt}.(<, t - 1min),\\ &\mathsf{FartherThan5minute} \equiv \mathsf{FartherThan} \sqcap\\ &\quad \exists \mathsf{startsAt}.(<, t - 5min),\\ &\ldots\\ &\mathsf{CloserThan} \sqsubseteq \mathsf{Event},\\ &\mathsf{CloserThan1minute} \equiv \mathsf{CloserThan} \sqcap\\ &\quad \exists \mathsf{startsAt}.(>, t - 1min)\\ &\mathsf{CloserThan5minute} \equiv \mathsf{CloserThan} \sqcap\\ &\quad \exists \mathsf{startsAt}.(>, t - 5min),\\ &\ldots\end{aligned}$$

$$\begin{aligned}\mathsf{HaveLunch} \equiv\ & \mathsf{LongerThan3minutes} \sqcap \\ & \exists \mathsf{overlap.AtTable} \sqcap \\ & \exists \mathsf{before}^{+}.(\mathsf{PrepareLunch} \sqcap \\ & \qquad \mathsf{CloserThan20minutes})\end{aligned}$$

Time instants (e.g. t-5min) must be computed on the basis of the current time t, meaning that we need to updated such definitions every t.

Moreover, we can generalize concepts:

$$\mathsf{HaveLunch} \sqsubseteq \mathsf{HaveMeal}$$
$$\mathsf{HaveDinner} \sqsubseteq \mathsf{HaveMeal}$$

4.3.5 Online phase: populating the ABox

The first step to populate the ABox is to assert individuals and roles. The second step is to record the changes in sensor states configuration (considering also their time duration) as events.
Such changes will be represented with a rising edge on the corresponding concept. When one or more rising edges happen or when some time relationship is satisfied, they will trigger a new event. The event will last as long as there are no falling edges in any of the concepts defining the event: one falling edge is enough to trigger the end of the event.
For example, think an event E_{ab}, triggered at instant t_i by rising edges in concepts A and B. Let's analyze some cases:

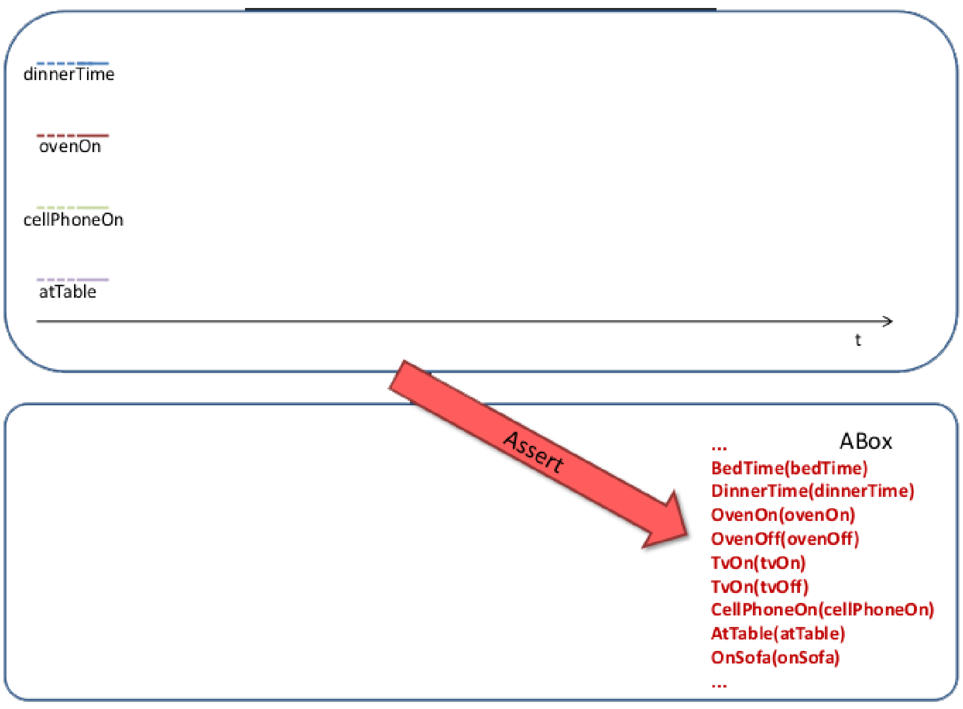
- Suppose that at time t_{i+4} A isn't observed anymore and it has a falling edge: the event lasted 4 instants.

- Suppose that at time t_{i+4} A,B,C are observed, a new event E_{abc} characterized by concepts A, B and C is added. Again, suppose that at instant t_{i+7} C has a falling edge: E_{abc} will end with a length of 3 instants. Suppose, instead that at instant t_{i+7} A has a falling edge: both events E_{ab} and E_{abc}, will end, the former with length of 7 instants, while the latter with a length of 3 instants.

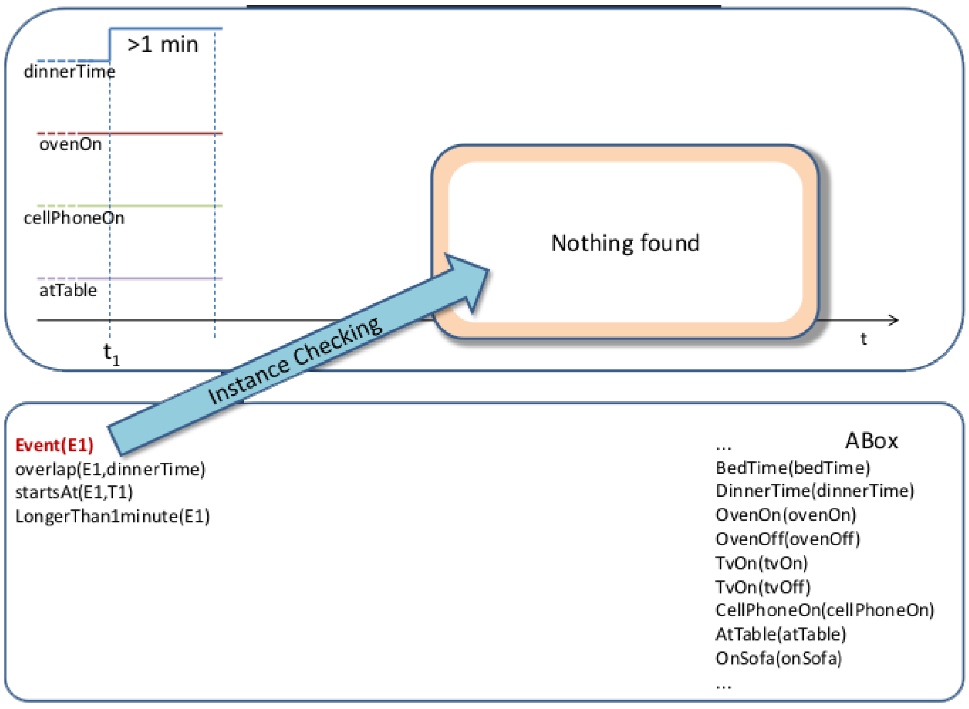
- Suppose that at instant t_{i+10} A and B are still observed: 10 instants are passed with A observed, this trigger a new event E_{longA} which started at t_i.

After each event, an instance checking is run to see if it's possible to assert additional context information based on the concepts defined by the user.

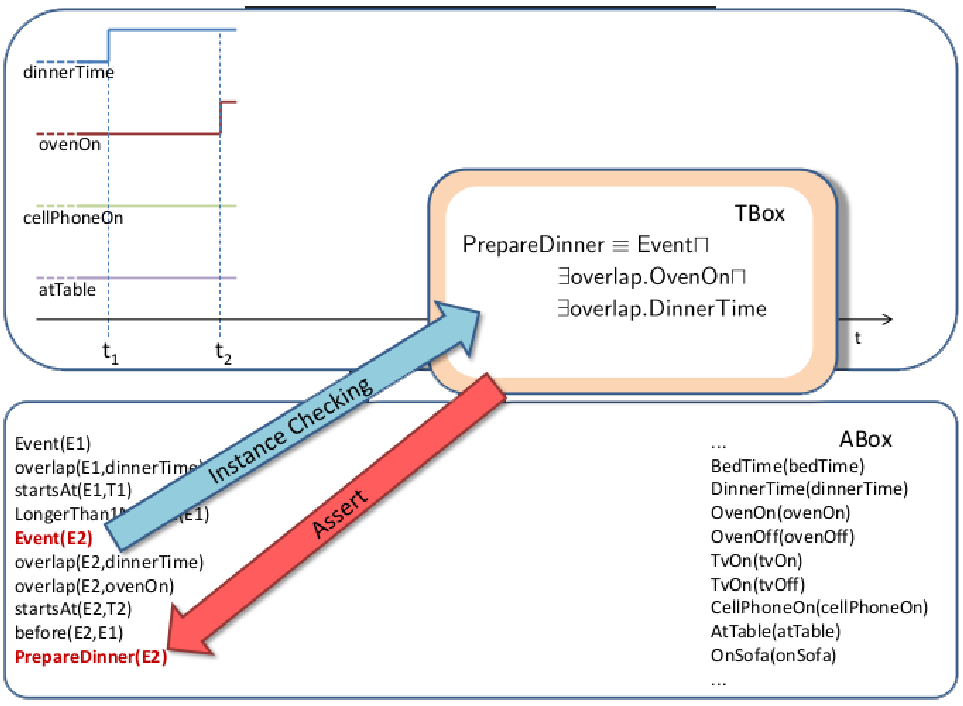
Let's see a practical example.
Initial assertions:



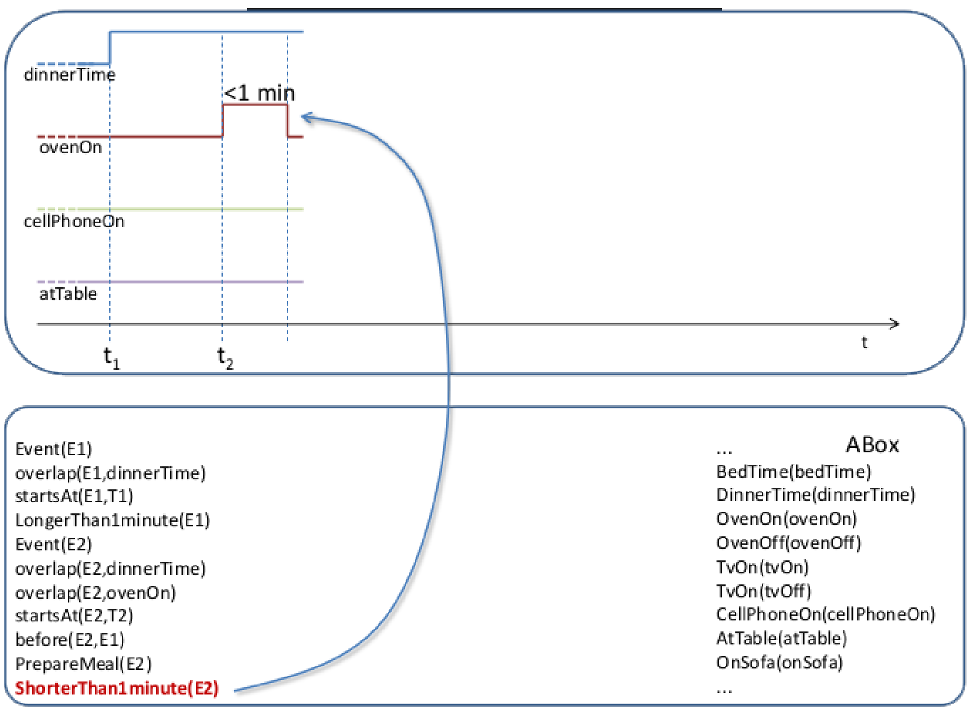
At instant t_1 the event $E1(\mathit{DinnerTime})$ is recorded, but instance checking doesn't produce any result because there isn't a corresponding concept:



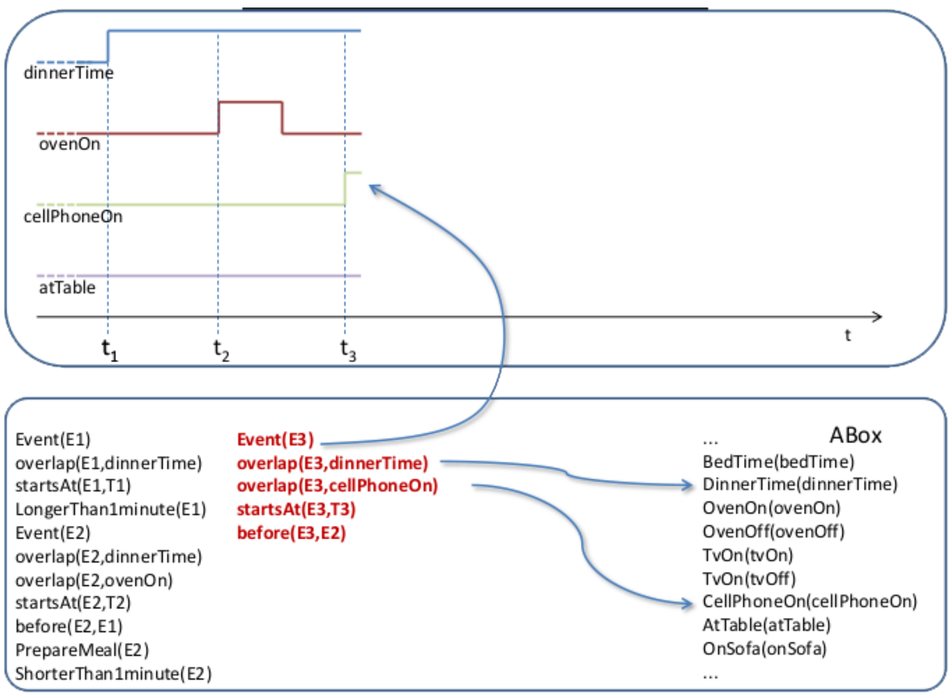
At instant t_2 the event E1 is followed by the event E2(*DinnerTime, OvenOn*): this knowledge is added to the ABox as *before(E2,E1)*. This time, instance checking asserts that, given the sensor and the temporal data, *PrepareDinner* has occurred and add it to the ABox:



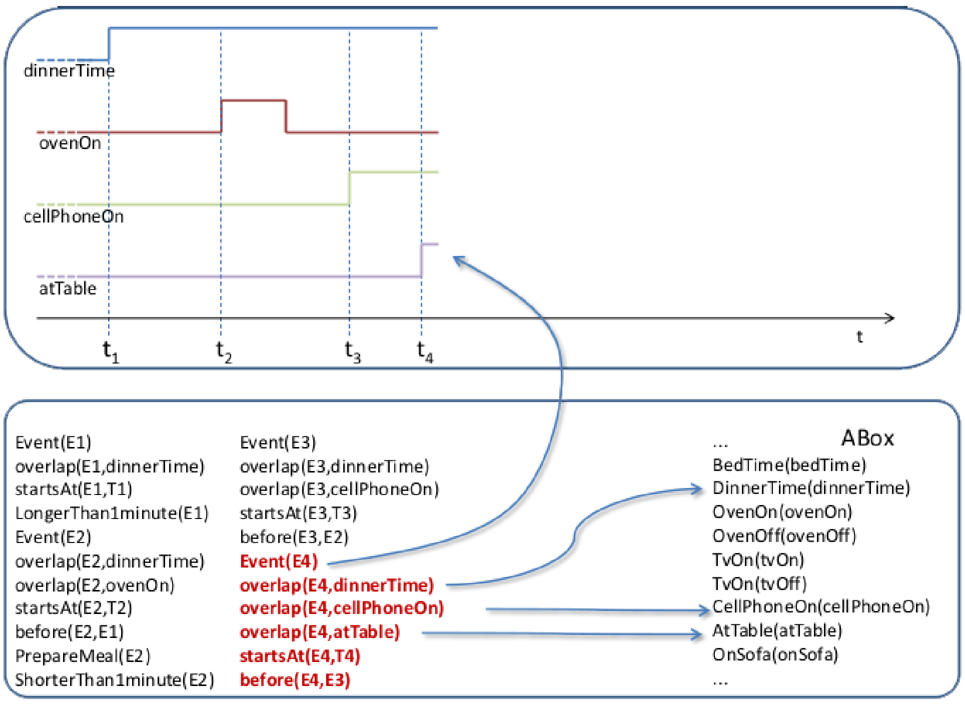
After less than one minute, *OvenOn* deactivates, causing event E2 ending while acknowledging its temporal duration:



At instant t_3, *cellPhoneOn* is observed, triggering event E3(*DinnerTime, cellPhoneOn*):



At instant t_4, *AtTable* is observed, triggering event E4(*DinnerTime, cellPhoneOn,AtTable*):

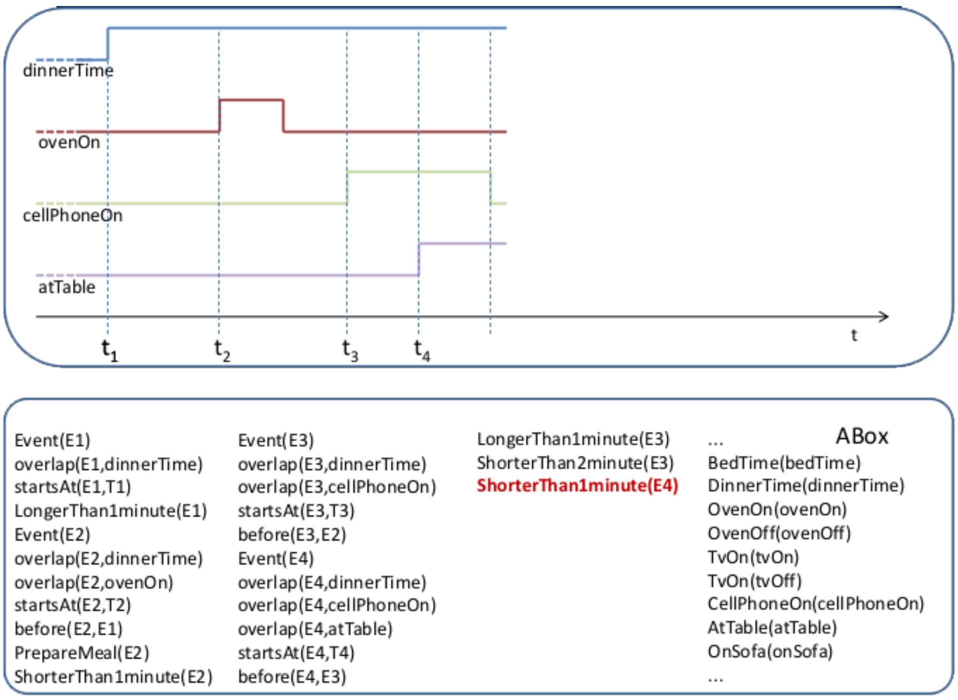


Event(E1)
overlap(E1,dinnerTime)
startsAt(E1,T1)
LongerThan1minute(E1)
Event(E2)
overlap(E2,dinnerTime)
overlap(E2,ovenOn)
startsAt(E2,T2)
before(E2,E1)
PrepareMeal(E2)
ShorterThan1minute(E2)

Event(E3)
overlap(E3,dinnerTime)
overlap(E3,cellPhoneOn)
startsAt(E3,T3)
before(E3,E2)
Event(E4)
overlap(E4,dinnerTime)
overlap(E4,cellPhoneOn)
overlap(E4,atTable)
startsAt(E4,T4)
before(E4,E3)

...
BedTime(bedTime)
DinnerTime(dinnerTime)
OvenOn(ovenOn)
OvenOff(ovenOff)
TvOn(tvOn)
TvOn(tvOff)
CellPhoneOn(cellPhoneOn)
AtTable(atTable)
OnSofa(onSofa)
...

After a while, *cellPhoneOn* is not observed, so there is a falling edge which "ends" event E3 and E4 (adding knowledge about their temporal length to the ABox):



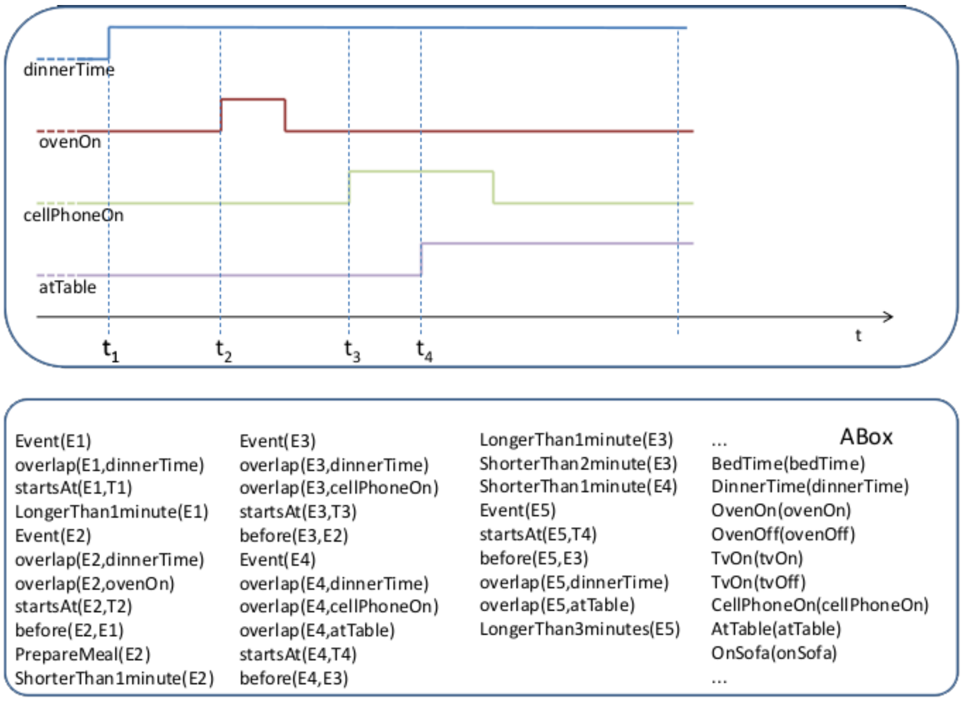
Event(E1)
overlap(E1,dinnerTime)
startsAt(E1,T1)
LongerThan1minute(E1)
Event(E2)
overlap(E2,dinnerTime)
overlap(E2,ovenOn)
startsAt(E2,T2)
before(E2,E1)
PrepareMeal(E2)
ShorterThan1minute(E2)

Event(E3)
overlap(E3,dinnerTime)
overlap(E3,cellPhoneOn)
startsAt(E3,T3)
before(E3,E2)
Event(E4)
overlap(E4,dinnerTime)
overlap(E4,cellPhoneOn)
overlap(E4,atTable)
startsAt(E4,T4)
before(E4,E3)

LongerThan1minute(E3)
ShorterThan2minute(E3)
ShorterThan1minute(E4)

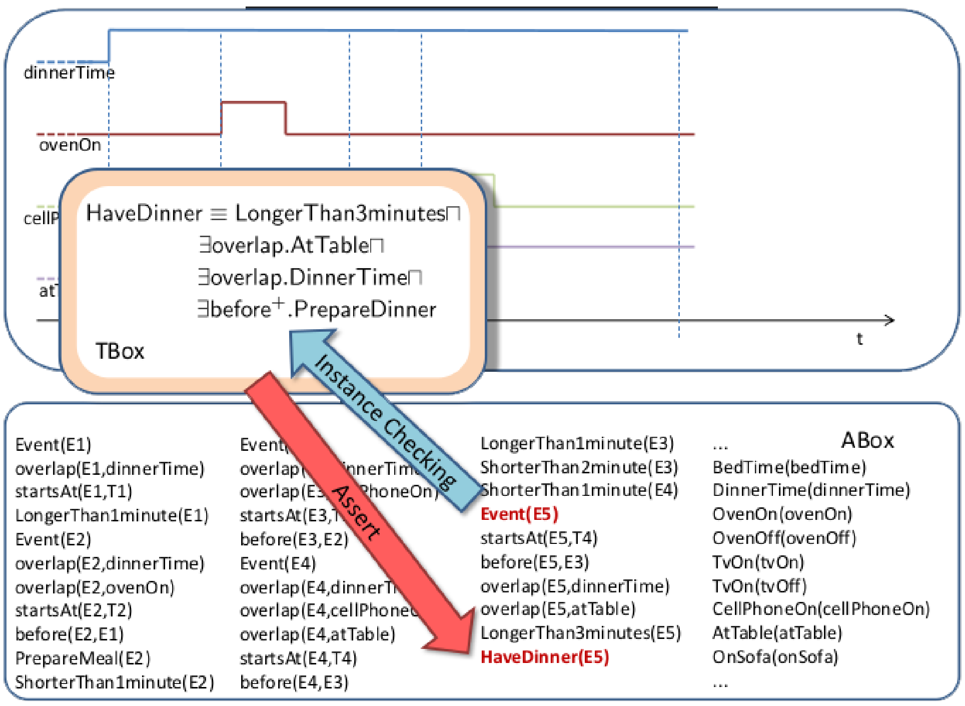
...
BedTime(bedTime)
DinnerTime(dinnerTime)
OvenOn(ovenOn)
OvenOff(ovenOff)
TvOn(tvOn)
TvOn(tvOff)
CellPhoneOn(cellPhoneOn)
AtTable(atTable)
OnSofa(onSofa)
...

A new event E5(*DinnerTime,AtTable*) is added to ABox in order to keep track of those two observations during time. It starts at t_4 because is when *AtTable* starts.



```
Event(E1)                 Event(E3)                 LongerThan1minute(E3)    ...
overlap(E1,dinnerTime)    overlap(E3,dinnerTime)    ShorterThan2minute(E3)   BedTime(bedTime)
startsAt(E1,T1)           overlap(E3,cellPhoneOn)   ShorterThan1minute(E4)   DinnerTime(dinnerTime)
LongerThan1minute(E1)     startsAt(E3,T3)           Event(E5)                OvenOn(ovenOn)
Event(E2)                 before(E3,E2)             startsAt(E5,T4)          OvenOff(ovenOff)
overlap(E2,dinnerTime)    Event(E4)                 before(E5,E3)            TvOn(tvOn)
overlap(E2,ovenOn)        overlap(E4,dinnerTime)    overlap(E5,dinnerTime)   TvOn(tvOff)
startsAt(E2,T2)           overlap(E4,cellPhoneOn)   overlap(E5,atTable)      CellPhoneOn(cellPhoneOn)
before(E2,E1)             overlap(E4,atTable)       LongerThan3minutes(E5)   AtTable(atTable)
PrepareMeal(E2)           startsAt(E4,T4)                                    OnSofa(onSofa)
ShorterThan1minute(E2)    before(E4,E3)                                      ...
```

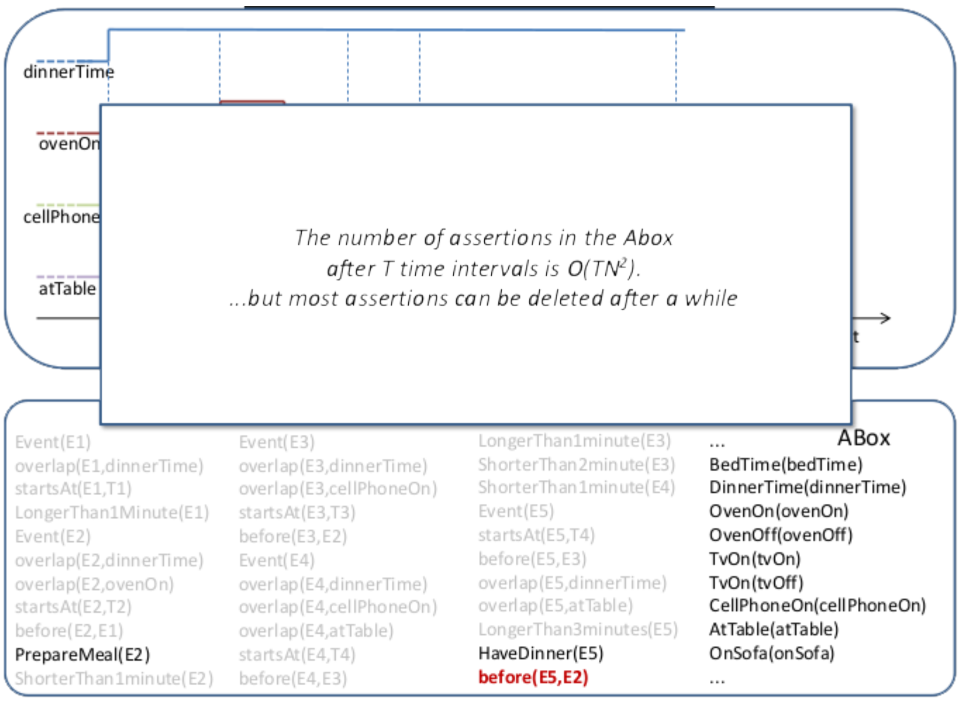
Now, instance checking detects that the configuration of sensor states (*DinnerTime,AtTable*) and the temporal relationship (E5 happened after E3 which happened after E2 which was recognized as *PrepareDinner*) related to *HaveDinner* is satisfied and so it adds this knowledge to the ABox:



$$\text{HaveDinner} \equiv \text{LongerThan3minutes} \sqcap \exists\text{overlap.AtTable} \sqcap \exists\text{overlap.DinnerTime} \sqcap \exists\text{before}^{+}.\text{PrepareDinner}$$

```
Event(E1)                 Event(                    LongerThan1minute(E3)    ...
overlap(E1,dinnerTime)    overlap(   innerTim       ShorterThan2minute(E3)   BedTime(bedTime)
startsAt(E1,T1)           overlap(E   honeOn)       ShorterThan1minute(E4)   DinnerTime(dinnerTime)
LongerThan1minute(E1)     startsAt(E3,T             Event(E5)                OvenOn(ovenOn)
Event(E2)                 before(E3,E2)             startsAt(E5,T4)          OvenOff(ovenOff)
overlap(E2,dinnerTime)    Event(E4)                 before(E5,E3)            TvOn(tvOn)
overlap(E2,ovenOn)        overlap(E4,dinnerT        overlap(E5,dinnerTime)   TvOn(tvOff)
startsAt(E2,T2)           overlap(E4,cellPhone      overlap(E5,atTable)      CellPhoneOn(cellPhoneOn)
before(E2,E1)             overlap(E4,atTable)       LongerThan3minutes(E5)   AtTable(atTable)
PrepareMeal(E2)           startsAt(E4,T4)           HaveDinner(E5)           OnSofa(onSofa)
ShorterThan1minute(E2)    before(E4,E3)                                      ...
```

The maximum number of new assertions with N individual sensors at time t is N^2 in the worst case. The number of assertions in the ABox after T time interval is $O(TN^2)$:



The number of assertions in the Abox after T time intervals is $O(TN^2)$. ...but most assertions can be deleted after a while

Event(E1)	Event(E3)	LongerThan1minute(E3)	...
overlap(E1,dinnerTime)	overlap(E3,dinnerTime)	ShorterThan2minute(E3)	BedTime(bedTime)
startsAt(E1,T1)	overlap(E3,cellPhoneOn)	ShorterThan1minute(E4)	DinnerTime(dinnerTime)
LongerThan1Minute(E1)	startsAt(E3,T3)	Event(E5)	OvenOn(ovenOn)
Event(E2)	before(E3,E2)	startsAt(E5,T4)	OvenOff(ovenOff)
overlap(E2,dinnerTime)	Event(E4)	before(E5,E3)	TvOn(tvOn)
overlap(E2,ovenOn)	overlap(E4,dinnerTime)	overlap(E5,dinnerTime)	TvOn(tvOff)
startsAt(E2,T2)	overlap(E4,cellPhoneOn)	overlap(E5,atTable)	CellPhoneOn(cellPhoneOn)
before(E2,E1)	overlap(E4,atTable)	LongerThan3minutes(E5)	AtTable(atTable)
PrepareMeal(E2)	startsAt(E4,T4)	HaveDinner(E5)	OnSofa(onSofa)
ShorterThan1minute(E2)	before(E4,E3)	**before(E5,E2)**	...

Chapter 5

BDI Agents

5.1 Introduction

Classical planning systems have goals, and plan sequence of actions to reach goals. BDI stands for Belief, Desire and Intentions, where **Belief** consists in the knowledge about the state of the world, **Desire** constitutes the goals to be reached and **Intentions** represents the commitment to achieve goals.

5.2 Procedural Reasoning System

A Procedural Reasoning System has the ability to:

- Reason upon its own beliefs, desires, and intentions.
- Deal with multiple goals and intentions.
- When in a multi-agent context, it can reason on other agents beliefs, desires and intentions.

All examples shown up to know require only simple reactive behaviours, such as turning on the heating system when the person gets home, switching off the lights in other rooms if the person is cooking or turning on the boiler to heat the water for a shower if the person went out for jogging. However, when you think about a more complex situation such as cooking itself, a sequence of activity to be planned is involved.

- Show a recipe book and allow the user to choose a recipe;
- Check which ingredients are present/missing/passed by in the fridge (through RFID tags on the food);
- Suggest the user where to go buying missing ingredients, depending on which shops are known to be open/closed (possibly showing driving direction on a portable device embedded with GPS);
- Suggest the user how to correctly refill the fridge;
- Assist the user while performing the 4 basic tasks of cooking: preparation, concocting, cooking, and arranging.

During the execution, if the situation changes, the system must be able to abandon the plan and possibly formulate a new one, therefore simple planning is not sufficient. (i.e: at some time, the person gets bored and decide to make a phone call to the Pizza delivery service; the shop closes before the user has reached it; two users want to use the kitchen at the same time for different purposes; somebody refills the fridge (with required ingredients) while the person is outside for buying them at the closest shop and so on...) Classical planning system (like STRIPS) were not designed to deal with such situations, on the other side, reactive systems (behaviour based, subsumption) do not allow to consider complex plans.

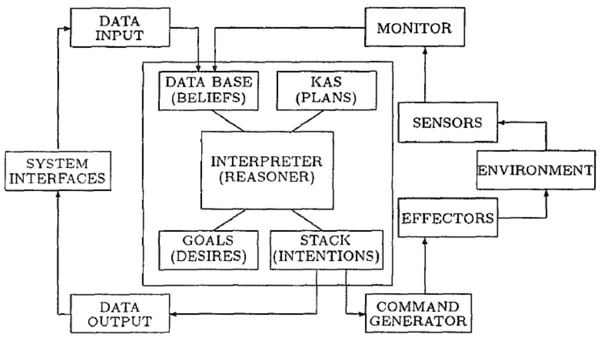
A typical approach consists in employing a **plan constructor** and a **plan executor**. The former formulates an entire course of action before commencing execution of the plan while the latter calls low level routines to execute the primitive actions. The plan is composed of these action and it is monitored to verify that their execution produce the desired effect. If there are some problem the plan constructor is invoked again.

Some problems with this approach rise when too many decision are taken at the beginning. Since planning is computationally expensive, and often in contrast with real-time requirements, it is better to defer decisions as much as possible, when the system will have more information to take the best decision. In any case, replanning is required if something changes.

In traditional system the focus is on plan generation, where the plan can be simply stored in the system as procedures to achieve a given result, learnt, shown by demonstration or even taught to the system. The most important part is not planning, but "dealing" with plans (which can be multiple, must be correctly integrated, abandoned, re-considered, etc.).

A Procedural Reasoning System is made up of:

- Beliefs: database containing current beliefs or facts about the world.
- Goals: current goals or desires to be realized.
- Knowledge Areas: procedures describing how certain sequences of actions and tests may be performed to achieve given goals or to reach particular situations.
- Intentions: process stack (containing all currently active KAs) which can be viewed as the system's current intentions for achieving its goals or reacting to some observes situation.
- Interpreter (inference mechanism) to manipulate these components.



Skipped slides "BDI Agents" from 15 to 31

5.3 AgentSpeak

AgentSpeak is a formal language implemented in Jason. The alphabet consist of

- Variables;
- Constants;
- Function symbols;
- Predicate symbols;
- Action symbols;
- Connectives;
- Quantifiers;
- Punctuation symbols;

Apart from first order connectives (and, or, not), we also use

- ! (for achievement)
- ? (for test)
- ; (for sequencing)
- $\leftarrow$ (for implication).

Standard first-order definitions of terms, first-order formulas, closed formulas, and free and bound occurrences of variables are used.

Definition 1: beliefs

- If b is a predicate symbol and $t_1, ..., t_n$ are terms, then $b(t_1, ..., t_n)$ or $b(t)$ is a **belief atom**.
- If $b(t)$ and $c(s)$ are belief atoms, $b(t) \wedge c(s)$ and $\neg b(t)$ are **beliefs**.
- A belief atom or its negation will be referred to as a **belief literal**.
- A ground belief atom will be called a **base belief**.

In our case,

$$cooking\_tem(x, y) \qquad in(x, y) \qquad \Rightarrow \quad \text{belief atoms}$$

$$to\_be\_cooked(bread, 180deg) \qquad in(oven, bread) \qquad \Rightarrow \quad \text{base belief}$$

A **goal** is a state of the system which the agent wants to bring about. We consider two types of goals: an achievement goal and a test goal:

- An **achievement goal**, written as
$$!g(t)$$
states that the agent wants to achieve a state where $g(t)$ is a true belief.
- A **test goal**, written as
$$?g(t)$$
states that the agent wants to test if the formula $g(t)$ is a true belief or not.

In our example, putting the bread in the oven can be stated as an achievement goal, i.e., $!in(oven\_bread)$, whereas seeing if the bread is in the oven can be stated as a test goal, i.e., $?in(oven, bread)$.

Definition 2: goals

- If g is a predicate symbol, and $t_1, .., t_n$ are terms then $!g(t_1, .., t_n)$ or $!g(t)$ and $?g(t_1, .., t_n)$ or $?g(t)$ are **goals**.

When an agent acquires a new goal or notices a change in its environment, it may trigger additions or deletions to its goals or beliefs. We refer to these events as **triggering events**. We consider the addition/deletion of beliefs/goals as the four triggering events:

- Addition is denoted by the operator $+$
- Deletion is denoted by the operator $-$

$\rightarrow$ in our example, noticing that the user wants to cook the bread is a triggering event

Definition 3: triggering events

- If $b(t)$ is a belief atom, $!g(t)$ and $?g(t)$ are goals, then $+b(t), -b(t), +!g(t), -!g(t), +?g(t), -?g(t)$ are **triggering events**.

The purpose of an agent is to observe the environment, and **based on its observation and its goals, execute certain actions**. These actions may change the state of the environment. For example, if *put* is an action symbol, putting an object X in Y, written as $put(X, Y)$, is an action. This action results in an environmental state where the object X is inside Y.

Definition 4: actions

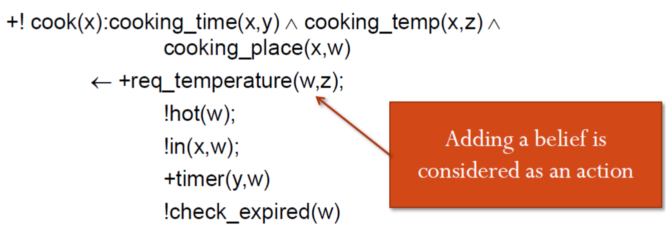
- If a is an action symbol, and $t_1, .., t_n$ are terms then $a(t_1, .., t_n)$ is an **action**.

An agent has **plans** which specify the means by which an agent should satisfy an end. A plan consists of a head and a body:

- The **head of a plan** consists of a triggering event and a context, separated by a :, the triggering event specifies why the plan was triggered, i.e., the addition or deletion of a belief or goal.
$$triggering\_event : context$$
The context of a plan specifies those beliefs that should hold in the agent's set of base beliefs, when the plan is triggered.

- The **body of a plan** is a sequence of goals or actions and it specifies the goals that the agent should achieve or test, and the actions that the agent should execute.

For example, if we want a plan that is executed when the user wants to cook bread (x):

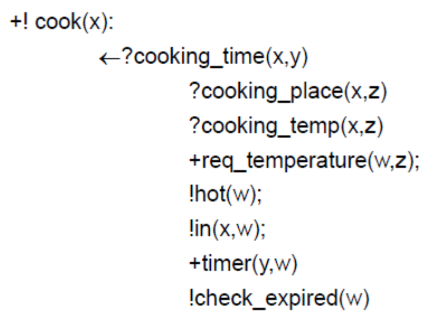


```
+! cook(x):cooking_time(x,y) ∧ cooking_temp(x,z) ∧
           cooking_place(x,w)
     ← +req_temperature(w,z);
           !hot(w);
           !in(x,w);
           +timer(y,w)
           !check_expired(w)
```

The plan is triggered, for example when the goal $!cook(bread)$ is added in the database. The relevant unifier is $\{x/bread\}$, which is applied everywhere in the plan. The plan is applicable if it is possible to find also an unifier for y, z, and w (specified in the context) within the belief database.

If the cooking time, temperature, and place are not among the belief, the plan is not applicable.

An alternative definition could be:



```
+! cook(x):
      ←?cooking_time(x,y)
            ?cooking_place(x,z)
            ?cooking_temp(x,z)
            +req_temperature(w,z);
            !hot(w);
            !in(x,w);
            +timer(y,w)
            !check_expired(w)
```

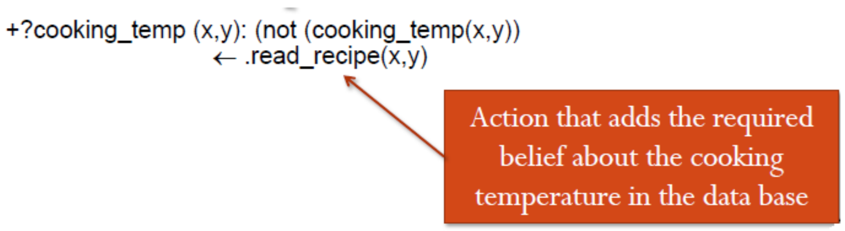
In this case the plan is always applicable. When it execute the 3 test goals at the beginning of the body $(?cooking\_time(x,z), ?cooking\_place(x,z), ?cooking\_temp(x,z))$:

- It checks if they are a logical consequence of the base beliefs, otherwise, it adds the triggering event of the type $+?g$ to the set of events.

- If there are no applicable plans for such event, then the plan fails (fails "normally", i.e., for the "no applicable plans" reason). We will see later what happens when a plan fails.

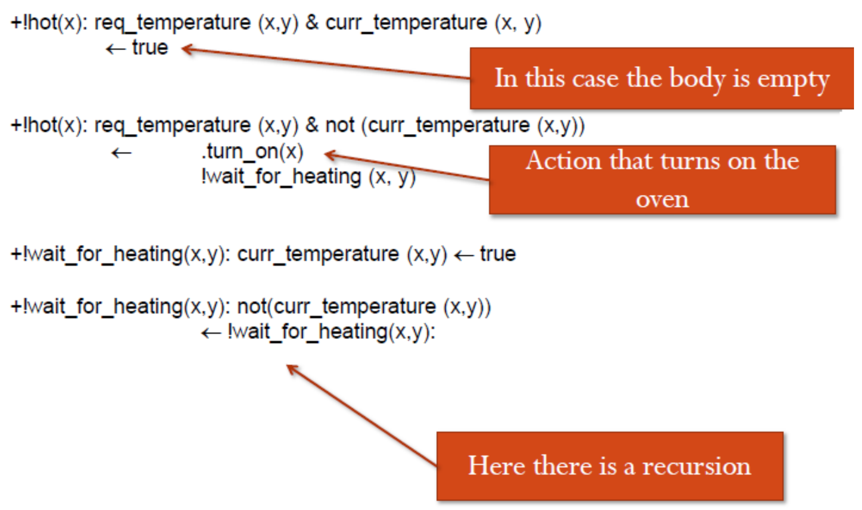
- Analogously, if there is no applicable plan for a triggering event of the type $+!g$ or an action within the plan fails, we say that the plan fails. Failures are backpropagated.

As an example, this is the plan if the temperature is not in the database:



```
+?cooking_temp (x,y): (not (cooking_temp(x,y))
                    ← .read_recipe(x,y)
```

Action that adds the required belief about the cooking temperature in the data base



```
+!hot(x): req_temperature (x,y) & curr_temperature (x, y)
        ← true

+!hot(x): req_temperature (x,y) & not (curr_temperature (x,y))
        ←       .turn_on(x)
                !wait_for_heating (x, y)

+!wait_for_heating(x,y): curr_temperature (x,y) ← true

+!wait_for_heating(x,y): not(curr_temperature (x,y))
                ← !wait_for_heating(x,y):
```

In this case the body is empty
Action that turns on the oven
Here there is a recursion

Notice also that we are assuming that *current_temperature* is a belief that is updated by somebody else.

Definition 5

- If e is a triggering event, $b_1, ..., b_n$ are belief literals, and $h_1, ..., h_n$ are goals or actions then $e : b_1 \wedge ... \wedge b_n \leftarrow h_1, ..., h_n$; is a plan.
- The expression to the left of the arrow is referred to as the head of the plan and the expression to the right of the arrow is referred to as the body of the plan.
- The expression to the right of the colon in the head of a plan is referred to as the context. For convenience, we shall rewrite an empty body with the expression true.

Some of the major differences between a logic program and an agent program are as follows:

- In a pure logic program there is no difference between a goal in the body of a rule and the head of a rule.
- In an agent program the head consists of a triggering event, rather than a goal. This allows for a more expressive invocation of plans by allowing both data-directed (using addition/deletion of beliefs) and goal-directed (using addition/deletion of goals) invocations.
- Rules in a pure logic program are not context-sensitive as plans.
- Rules execute successfully returning a binding for unbound variables; however, execution of plans generates a sequence of ground actions that affect the environment.
- While a goal is being queried the execution of that query cannot be interrupted in a logic program. However, the plans in an agent program can be interrupted.

Informally, an agent consists of:

- A set of base beliefs, B;
- A set of plans, P;
- A set of events, E;
- A set of actions, A;
- A set of intentions, I;
- Three selection functions, SE, SO, SI.

When the agent notices a **change in the environment** or an external user has asked the system to adopt or discard a goal or the system itself has added/deleted a belief or a goal, an appropriate **triggering event** is generated.

Moreover, **events**, both internal or external, are **asynchronously added** to the set of events E, at the same time, the selection function SE selects an event to process from the set of events E. This event is removed from E and is used to unify with the triggering events of the plans in the set P.

The plans whose triggering events so unified are called relevant plans and the unifier is called the relevant unifier.

Then, the **relevant unifier** is applied to the **context condition** and a correct answer substitution is obtained for the context, such that the context is a logical consequence of the set of base beliefs, B.

Such plans are called **applicable plans** or options and the composition of the relevant unifier with the correct answer substitution is called the **applicable unifier**. For each event there may be many applicable plans or options.

The **selection function SO** chooses one of these plans. Applying the applicable unifier to the chosen option yields the intended means of responding to the triggering event.

Each intention is a **stack of partially instantiated plans** or intention frames. In the case of an external triggering event (i.e., generated by a new belief in the data base or added by the user) the intended means is used to create a new intention, which is added to the set of intentions I.

In the case of an **internal event** (i.e., generated by another plan) the intended means is pushed on top of an existing intention that triggered the internal event.

Then, the **selection function SI** selects an intention to execute. When the agent executes an intention, it executes the first goal or action of the body of the top of the intention. **Executing an achievement goal** is equivalent to **generating an internal event** to add the goal to the current intention. **Executing a test goal** is equivalent to **finding a substitution for the goal** which makes it a logical consequence of the base beliefs.

If such a substitution is found **the test goal is removed** from the body of the top of the intention and the substitution is applied to the rest of the body of the top of the intention. Executing an action results in the action being added to the set of actions, A, and it being removed from the body of the top of the intention. The agent now goes to the set of events, E, and the whole cycle continues until there are no events in E or there is no runnable intention.

What exactly are the cases where negative events of the forms "-!p" and "-?p" are generated? If an action fails or there is no applicable plan for a subgoal in the plan being executed to handle an internal event with a goal addition "+!p" or "+?p", then the whole failed plan is removed from the top of the intention and an internal event for "-!p" or "-?p" associated with that same intention is generated. This allows the programmer to specify backup plans!

It is claimed that open-world assumption is available. **What does this mean?**

- There is a "strong negation" operator "∼".
- When assuming open world, the user models the environment with a set of propositions known to be explicitly true and a set of propositions known to be explicitly false of the environment at a certain moment in time (the latter are literals preceeded by the operator).
- Of course, there is still default negation (as usual in logic programming languages), so you can say, in the context of a plan, "not p(t) & not ∼p(t)" to check if the agent is uncertain about "p(t)".
- Note that it's completely up to the users to prevent inconsistency (or to use it, if they so wish). You could add internal beliefs (or have beliefs from perception of the environment) that p(t) is true and that ∼ p(t) is also true
- Finally, note that strong negation can also appear in the triggering events, plan body, and anywhere a literal can appear.

What's the difference between "!" and "!!" ? The difference between "!" and "!!" is that the latter causes the goal to be pursued as a separate intention. Within the body of a plan in one intention, if you have "!g1; !g2" the agent will attempt to achieve g2 only after achieving (or finishing executing a plan for) g1. If you say "!!g1; !g2" the agent will then have another separate intention to achieve g1 and can immediately start attempting to achieve g2. What will be done first (executing a bit of the intention with g1 or the old intention with g2) will depend on the choices that the intention selection function makes.

5.4 Exercise

You are requested to design a system to control resources within the port of Genova. In particolar, resources can be

- Human workers (truck drivers, forklift drivers, crane drivers, dockers, security staff, administrative staff)
- Means of transport (trucks, forklifts, cranes)
- Sensors distributed in the environment (cameras, PIR sensors, laser scanners)
- Computers (server, PDA devices and desktop computers)

Security people are equipped with a PDA which is able to suggest a course of action to take depending on the specific situation. In particular, this is used to encode best practices in the security domain, e.g., when security personnel find a person in an area where it is not allowed to be in. Consider the situation above, i.e., when somebody is detected by sensors in a restricted area. The security staff is supposed to follow the procedure A:

- Reach the area of interest
- Check the identity of the person
- Transmit the identity of the person to the server

Assume that the following actions are available, described as operator in first order predicate logics

- "goto_area(X)" (the system provides guidance to reach the area X)
- "id_person(X)" (the user is required to insert manually the identity of the person X, which is "unknown" if the person is not recognized)
- "transmit_id (X)" (the system transmit automatically the identity of X to the remote server)

Moreover, the system is able to infer the current position of the user and to add a corresponding belief in the knowledge base with "user_position(X)"

Describe procedure A as a plan in Agent Speak, by introducing all the predicate symbols that you need.

Chapter 6

universAAL

6.1 Introduction

The universAAL platform derives from a (like named) European Union funded R&D project. The American computer scientist Mark Weiser predicted that computers would soon become a lot smaller, cheaper and much, much more numerous. Thanks to these developments, computer technology could move away from the desks and into the surroundings of their users, thus becoming what Weiser had called a "calm technology" a tool that supports its users without requiring their ongoing attention. Weiser called it "Ubiquitous Computing", others used other terms such as "Pervasive Computing", "Ambient Intelligence", or "Ambient Assisted Living", but more or less, all of these refer to the same concept: distributed networks of technical devices that, for the most part invisibly, surround us and assist us in many aspects of everyday tasks without us even noticing. As a simple example, imagine that your TV automatically turns down the volume if you pick up the ringing phone (as famously described by Tim Berners Lee in his article "The Semantic Web"). A system to which both the TV and the phone are connected could "understand" that when talking on the phone, you prefer a low ambient noise, and thus turn out the sound for you.

6.2 The software

universAAL is a software that helps to build so called "assistive systems" by connecting various, heterogeneous technical devices to a single, unified network. universAAL also delivers the means to control this distributed system (and the devices that it is based on). In this regard, universAAL is not unlike an Operating System for AAL. Operating systems help to connect hardware components (such as, among others, a processor, a graphic card, a screen and a keyboard) to a single system and give the user the means to control this system. One of the main differences is, however, that most of the components that work together within a universAAL based system weren't originally meant to do so (such as your TV and your fridge). In the universAAL world, a single device that is connected to an assistive system is referred to as being a "node". There are two ways of how to integrate a device into an universAAL based assistive system, assuming that the device in question is networked.

- The first way is to install a specific piece of the universAAL platform on the device, the so called "**Middleware**".
- The second way does not require a given device to run the universAAL Middleware. The device in question is rather connected to a node that runs this Middleware, and this node is used as an intermediary by the system in order to control the additional device.

6.3 Middleware

The **middleware** shall be capable of hiding the distribution and heterogeneity of the diverse devices that make up the system at its core, and manages communication between nodes.

From an application programmer's perspective, this means that you do not need to worry about the fact that some of the applications that your appliance talks to may actually be running on other devices than the one your application is. You will simply forward all of your application's messages and requests to the middleware component, which will take it from there, making sure that each message reaches its recipient, thus greatly simplifying your life as a programmer.

The middleware deals with situations in which it either loses the connection to one or more nodes, or situations in which additional nodes try to (re –)join the system. The latter is handled by the discovery and peering mechanisms of the middleware that automatically recognize and integrate all qualified nodes within reach. The real challenge, however, lies in handling the loss of nodes, as this kind of unpredictable behavior makes it difficult to organize a reliable system.

For example, users may turn certain devices off (by accident or intentionally) or carry them away (as will happen with mobile phones on a regular basis). Some devices may run out of battery while others might simply fail. This results in a considerable degree of uncertainty, whether about a specific node, meant are both the device and the software running on it, will really be available when talked to by the system. The solution to this problem is called "goal based interoperability". It is based on the principle idea to formulate requests in a semantic and not in a syntactical way, thus stating what is supposed to be done: the "goals" that are to be achieved rather than how this should be done (which would include the specification of an addressee).

The universAAL platform achieves this through the use of ontological descriptions. The task of finding the appropriate recipient for a message is then left to a mediator that needs to know of all possible recipients that are currently available and must be able to decide, which one of them (if any) is the right one for this specific request.

On the downside, however, this means that applications also require well thought out strategies for fault tolerance as there can be no guarantees that the dynamic resolution of dependencies through the mediators will actually be successful. The universAAL Middleware has three mediators , and each of them is responsible for the delivery of a certain message category: the buses.

6.4 The three buses

The three buses form the heart of the universAAL platform. All communication between universAAL based applications should happen only in a round about way via one of them, even if physically, these applications are located on the same node. Each of the buses handles a specific type of message/request, and the way that a bus operates is based on the characteristics of this category of information.

6.4.1 Context Bus

The so called Context Bus, is an "event based" communication channel: an application forwards information to this bus without taking interest in the existence of actual recipients. A typical "context provider", an application that publishes context information onto the Context Bus, would be an application talking to a sensor (such as a temperature sensor).

At regular intervals, this application forwards the values it receives from the sensor to the Context Bus, maybe after having preprocessed them in a certain way.

Such an application does not need to know who is receiving its messages, that is, who the "context subscribers" are that have registered themselves to the Context Bus as recipients for this specific kind of sensor related messages (if there are any at all).

6.4.2 Service Bus

The so called Service Bus, on the other hand, is "call based". In this context, the term "service" shall mean the provision of a functionality, or in other words, that someone is doing something for somebody else.

- An application that offers a service (= can do something) announces this by registering a corresponding service profile, that is a description of what it is capable of doing, with the Service Bus. This kind of application is called a "service callee" (because it gets called by another application).

- An exemplary service callee would be an application that can control a lamp, maybe one of those in the living room, and it would register the appropriate service profile with the Service Bus (stating something like "I can turn off one of the lights in the living room").

- The counterpart to the service callees are applications that require a service, the "service caller". They send a service request to the Service Bus, asking for a specific service (as in "I need someone to turn off the lights in the living room, please").

- It is then up to the Service Bus to find one or more matching service profiles to the service request and, if a match is available, to forward the request to the corresponding service callee(s).

As mentioned before, the Service Bus is different to the Context Bus in that it is call based, meaning that the caller (= the service requester) expects an answer to its message and might temporarily cease its execution to wait for that.

- Once the service callee has completed the job it was asked to do, it puts his answer, the "service response", on the Service Bus (as in "Lights should be off now, you're welcome").

- The Service Bus makes sure that the answer really reaches the service caller application, which can then proceed with its execution (knowing that the lights in the living room should be off).

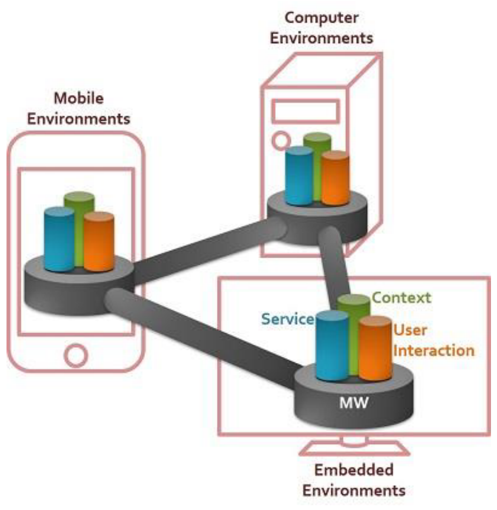
6.4.3 UI Bus

There is one more bus you need to know of, the so called UI Bus (UI for user interaction).

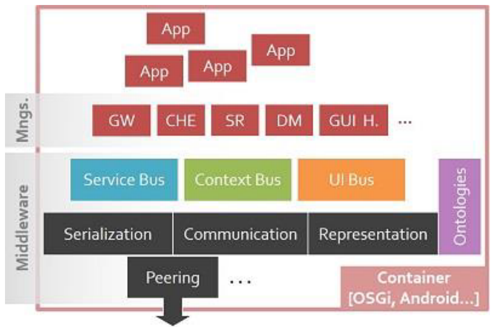
The purpose of the UI Bus is to deliver messages that are somehow related to explicit user interaction. For example, an application that wants the user to be notified about a certain event would use the UI Bus. If the user reacts to this notification by issuing an explicit command to the system, then this information would also go over the UI Bus.

6.5 Functionalities

The Middleware is the core part of universAAL platform and takes care that all universAAL nodes in a Space can cooperate with each other. To do so it establishes peer to peer communications between them so that they can share the different kinds of universAAL semantic communication: Context, Service and User Interaction, following the shared Ontological Model.

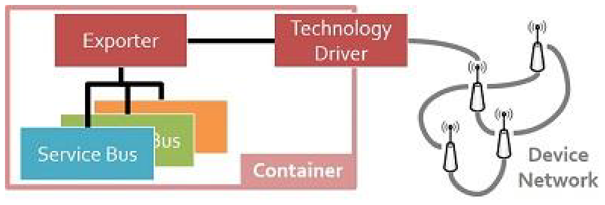


6.5.1 Layer composition



- The **Container** is the part that lets the Middleware logic execute in different environments. There are different Containers so that the Middleware can run on devices with plain Java, computers or embedded systems running OSGi, Android smartphones, and so on. The Container determines how the components of universAAL and the applications that use it are coded and built. In OSGi they would be Bundles, in Android they would be APKs, and so forth. Currently universAAL only officially supports these two.
- The **Peering** part is responsible for interconnecting and communicating the instances of the Middleware. More technologies can be added modularly. A hypothetic UPnP connector could discover other nodes with this connector, and use bridging options for using multiple technologies.
- The Communication part is the one holding the ultimate logic of the Middleware that enables the flow of universAAL semantic information across peers, by defining specific purposes Buses (Context, Service, UI).
- The representation model defines how ontologies are handled.
- The serialization component encrypts and parses messages across nodes.

6.5.2 universAAL application

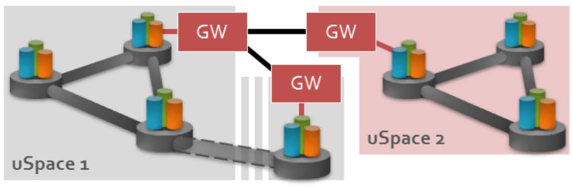


A universAAL Application is any piece of software that can run on the Container and that makes use of the universAAL Buses or Managers, whether by consuming them or providing into them, in order to provide a Service or a part of it. Regarding hardware sensor and actuator devices, these are connected through "exporters", which are just like an application exporting the devices interfaces and information into universAAL platform. There would be a different exporter per technology (KNX, ZigBee...).

6.5.3 uSpace

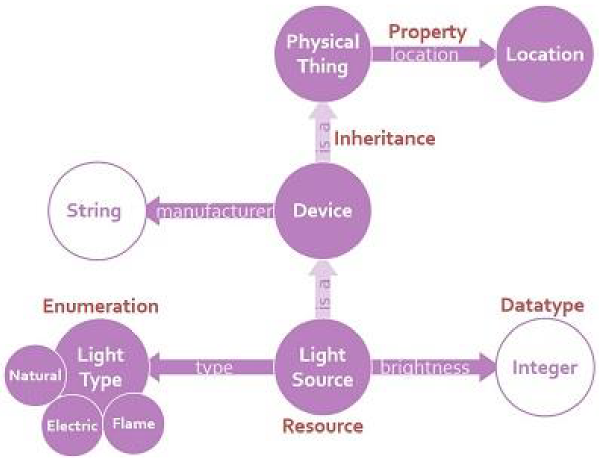
The universAAL platform defines the concept of uSpace as a logical environment in which all universAAL qpplications can communicate to each other and the platform through the same buses, seamlessly, regardless of the node or container they run within and the physical network beneath.

- The Services provided by these applications are oriented to a specific human user (the Assisted Person) or set of users.
- Communication across different Spaces is not possible except through a special Manager called uSpace Gateway, which takes care of message exchanges and authorization between different Spaces. Moreover, it takes also care of connecting remote nodes that belong to the original Space, accomplishing a similar function to that of a VPN, when networked connection through Peering is not possible.



6.6 Ontological Model

In universAAL knowledge is shared in the form of Ontologies.



6.6.1 What ontologies are made of

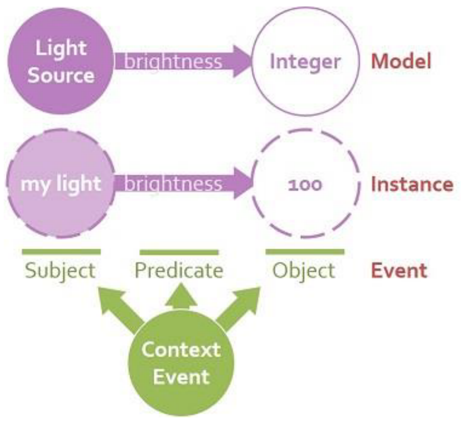
- **Resources** are how the concepts are represented. They are the nodes in the mesh.
  - They are identified by a URI.
  - They can inherit from other resources, and have properties that link to other resources or datatypes.
- **Properties** are links between the concepts.
  - They are also identified by URIs and can also inherit from other properties.
  - They can have restrictions upon them, like cardinality. resources or datatypes.
- **Datatypes** are the native data formats, like Boolean, Integer and so on. They are always present by default and don't have properties.
- **Enumerations** are sets of instances of Resources, representing different specific values that a property can point to.

6.6.2 Context

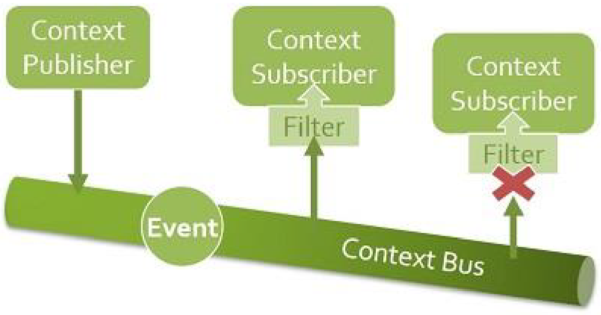
Context information represents the environment of the system, from physical surrounding, including users, to system information. Context communication is event based, forwarding updates of the context. This is done in the form of Context Events, which are sent by Context Publishers, and can be consumed by Context Subscribers.

6.6.3 How context information is shared

Context Events are the minimal unit of context information sharing and are built on the Ontological model of universAAL. The minimal context information that can be extracted from an Ontological model is a link of two concepts through a property, modelled as a triple with subject, predicate and object. This is known as a statement.



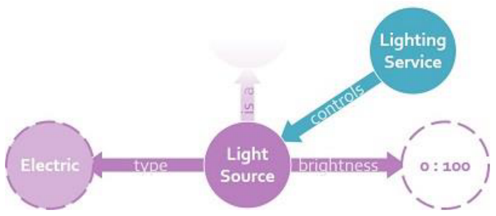
- **Context Publishers** are applications that are capable of sending Context Events. They build these events with the Ontological model and broadcast them.
- **Context Subscriber** are any application interested in consuming Context Events. They define a filter to restrict which types of Events they are exactly interested in.



6.6.4 Services

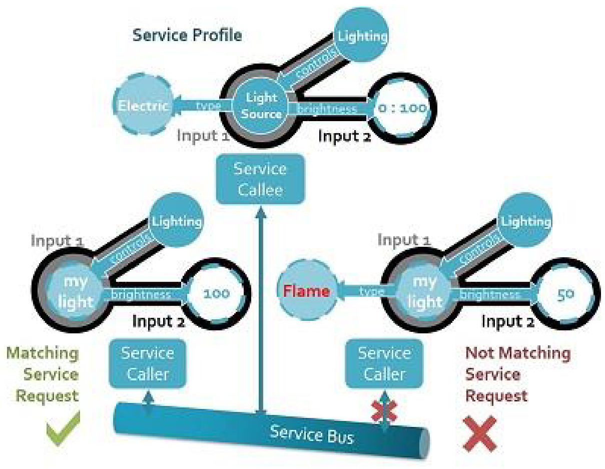
Services are request/response interactions between applications. They are semantic, which means that you don't need to call a service like "turn on a light by ID" (which you can) but you can ask to for services that "turn on all lights in a given location", without knowing them in advance. This is possible because services are described with the Ontological model.

A **Service Ontology** is needed since it is the shared model between requester and provider. It works as an anchor to the Ontological model and also allows restrictions over the original model to make services more specific.



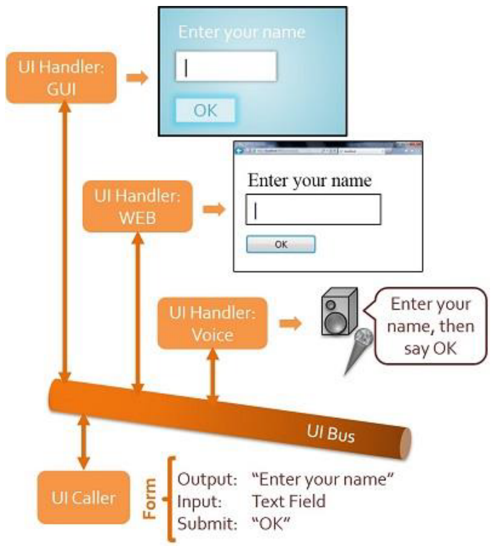
- **Service Callees** are those applications that provide services of certain Service Ontology. They do so by registering Service Profiles which are the equivalent to methods. They represent the operation to perform. Starting at the Service Ontology they describe arguments as a Path to a concept on which an Effect is expected.
- **Service Callers** are the applications that request the execution of a service. Service Requests are the counterpart of Service Profiles, they are built the same way but declare what the Caller wants

to execute. The Requests are matched to registered Profiles and if they are ontologically equivalent, the Callee(s) that registered them will be called and will give an answer.



6.7 User Interaction

Direct User Interaction is achieved by universAAL applications in a decoupled fashion. They do not handle how this information is presented to the user, only what is being presented (and handled in return). To do so the interaction information is abstracted by representing it with an Ontological model.



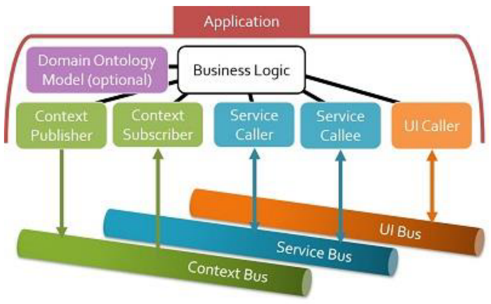
- User **Interaction Callers** are the applications that want to have some kind of direct interaction with the user. They build a Form that represents exactly what they want to show to the user and what they want in return. **Forms** are the ontological representation of the typical user interaction components, like textual inputs, multiple selections, buttons, and so on. Forms are created by UI Callers and sent to UI Handlers to be rendered, filled by user, and sent back to UI Callers to be processed.
- User **Interaction Handlers** are special types of applications in charge of translating the Forms sent by UI Callers to a physical rendering that a user can interact with, such as a GUI, a sound output or Web page. Then interpret the user responses to fill in the information requested by UI Callers into the Form and send it back. There can be several UI Handlers in different locations, with different modalities, and the UI Callers are oblivious to them, thus achieving multi modal and multi location interaction.

6.8 Applications

An application, in addition to its own business logic, and regardless of its structure, needs one or more of the universAAL "wrappers" presented until now:

- Context Publisher
- Context Subscriber
- Service Caller
- Service Callee
- User Interaction Caller

Because Ontologies are used as data model, the Application must make use of an Ontology describing its information domain. An application can define its own ontological model but it is strongly recommended that an existing implementation is reused (if it exists), for interoperability purposes.


Application
Domain Ontology Model (optional)
Business Logic
Context Publisher
Context Subscriber
Service Caller
Service Callee
UI Caller
UI Bus
Service Bus
Context Bus