Applying the BDI paradigm in Communications Systems

Romelia Plesa
PhD Candidate
OCICS - SITE, University of Ottawa

November 7, 2006

Presentation Overview

- Proposed architecture
- BDI basics
- Applying BDI
- AgentSpeak(L) basics
- Example

Motivation

Presence, Context and the personalization of services

- Most of today's telephony communication services could be characterized as *context free*.
  - They provide no real-time context regarding the purpose or the circumstances of a phone call that one is receiving.
- Context information is needed in order to manage the use of the phone or other communication services.
- Context-aware support provides the application relevant knowledge about the environment in which it functions.
  - The applications analyze available context information and together with user preferences determine the best way to communicate at any given point in time.

A vision of Context-based scenarios

The advocates of presence technology and contextual services promise a world where people will be connected ***when they want, how they want, and with whom they want*** and their communication will be tailored on specific desires and preferences.

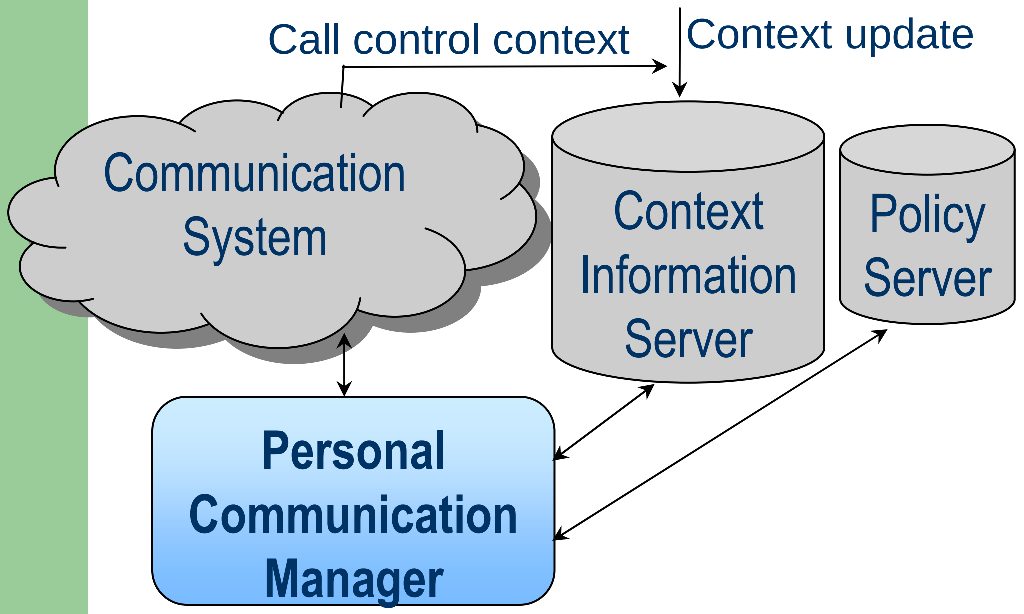
Goal

- to propose an architecture that supports presence and contextual services in telecom and allows context-aware call handling based on information about the environment (context) and individual policies.

New Services

- **Context-based services**
  - *All calls from my students will have announcement X played out.*
- **Availability services**
  - *Secretaries are not available to answer enquires during lunchtime*
- **Notification services**
  - *Remind me of the 3 pm meeting if I am not already in the meeting room.*
- **Personal addressing services**
  - *If the call is from a person involved in project X, redirect it to the team leader*

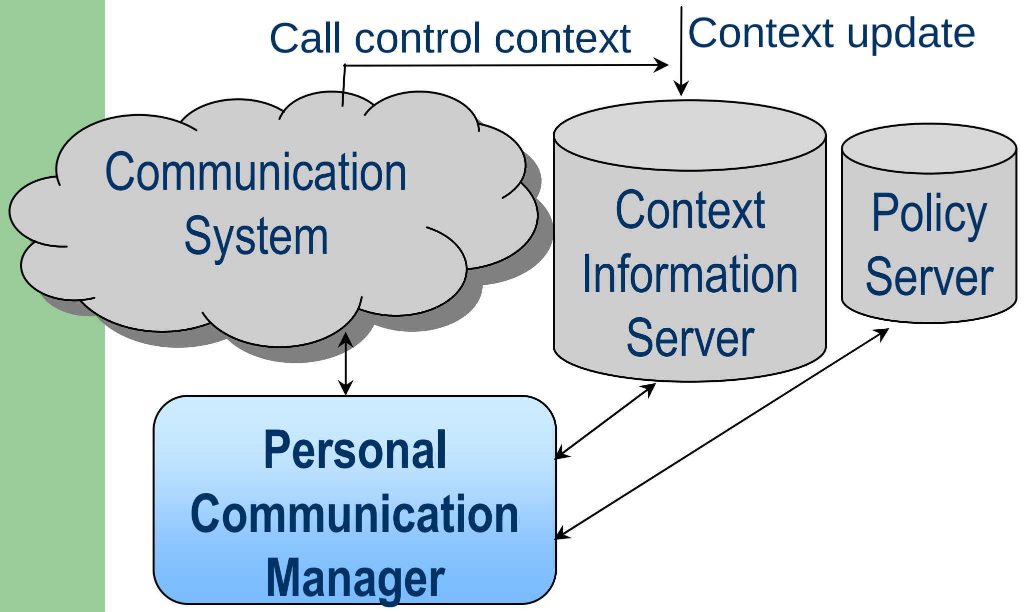
The Architecture



Functional Requirements:

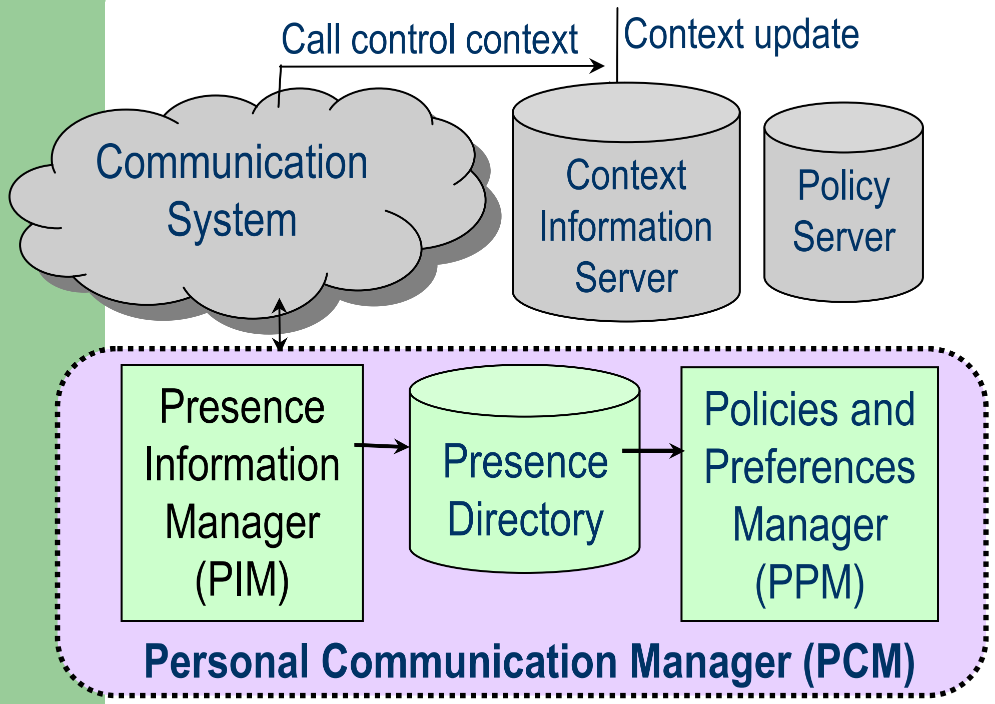
- collection of context information using sensors
- dissemination of context information
- publishing of presence information from users and their devices
- description of user policies and preferences
- user preferences-based handling of communication

The Architecture



- architecture independent of the communication protocol (SIP, H.323 or other session protocol).
- **Context Information Server** updates, stores and distributes the context information.
- **Policy Server** manages the user’s policies.
  - *Personal policies* allow users to establish preferences about how their calls should be handled.
  - *Subscription/ Notification policies* allow users to project different presence to different persons.

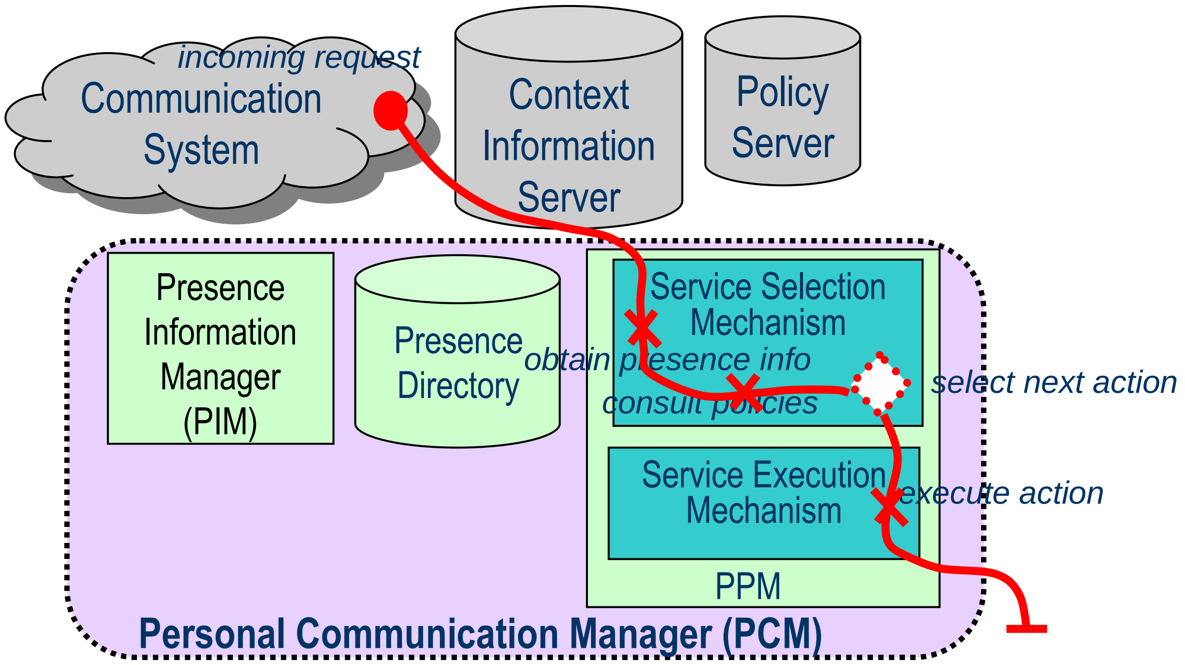
Personal Communication Manager


Call control context
Context update
Communication System
Context Information Server
Policy Server
Presence Information Manager (PIM)
Presence Directory
Policies and Preferences Manager (PPM)
Personal Communication Manager (PCM)

- a *software agent* that represents each user.
- PCM receives request messages (such as INVITE for a SIP-based architecture) and decides how they should be handled.

- PCM has three components
  - **Presence Information Manager** - a rule-based process that builds the "consolidated presence information".
  - **Presence Directory** - a repository in which all known and deduced presence information is deposited.
  - **Policies and Preferences Manager -** contains the preferences logic to respond to requests to contact an entity.

The Call Model


incoming request
Communication System
Context Information Server
Policy Server
Presence Information Manager (PIM)
Presence Directory
Service Selection Mechanism
obtain presence info
consult policies
select next action
Service Execution Mechanism
execute action
PPM
Personal Communication Manager (PCM)

- Includes context update, service selection based on context information and user personal policies as well as service execution.
  - The service selection and execution mechanisms will be incorporated into the Personal Communication Manager (in the **Policies and Preferences Manager (PPM)** component).

The BDI Model

- Belief, Desire, Intention (BDI) is an architecture for modeling Intelligent Software Agents
- BDI agents can solve problems in dynamic and real-time environments with little or no human intervention
- The BDI architecture is used in a variety of applications ranging from robots that play soccer to air traffic controllers in airports

BDI Agents

- Systems that are situated in a changing environment
- Receive perceptual input from the environment
- Take actions to affect their environment

From the various options and alternatives available to it at a certain moment in time, the agent needs to select the appropriate actions or procedures to execute.

The ***selection function*** should enable the system to achieve its objectives, given

- the computational resources available to the system
- the characteristics of the environment in which the system is situated.

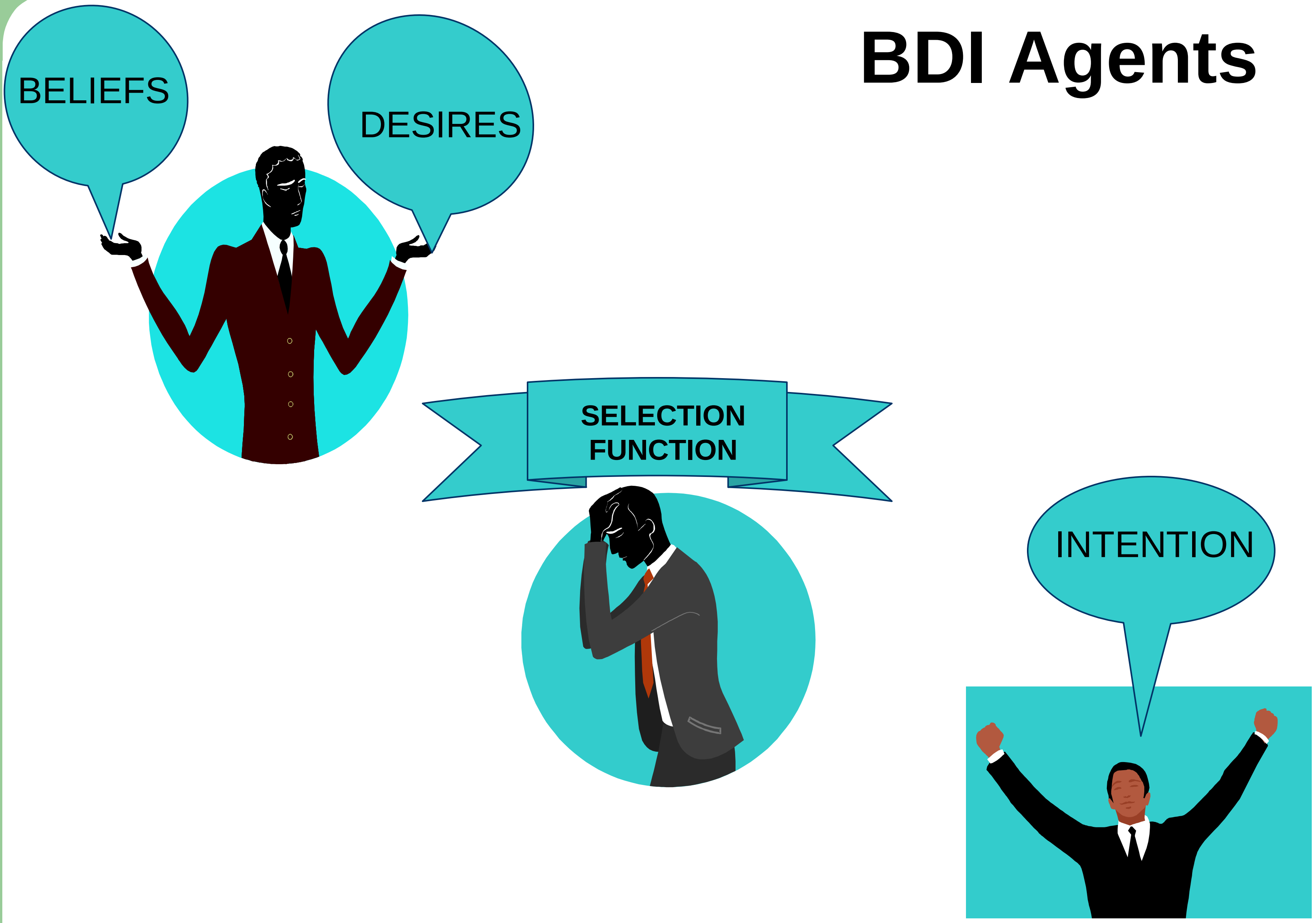
BDI Agents

- two types of input data required for the selection function:
- **Beliefs:**
  - represent the characteristics of the environment
  - are updated appropriately after each sensing action.
  - can be viewed as the ***informative*** component of the system.
- **Desires**
  - contain the information about the objectives to be accomplished, the priorities and payoffs associated with the various objectives
  - can be thought as representing the ***motivational*** state of the system.

BDI Agents

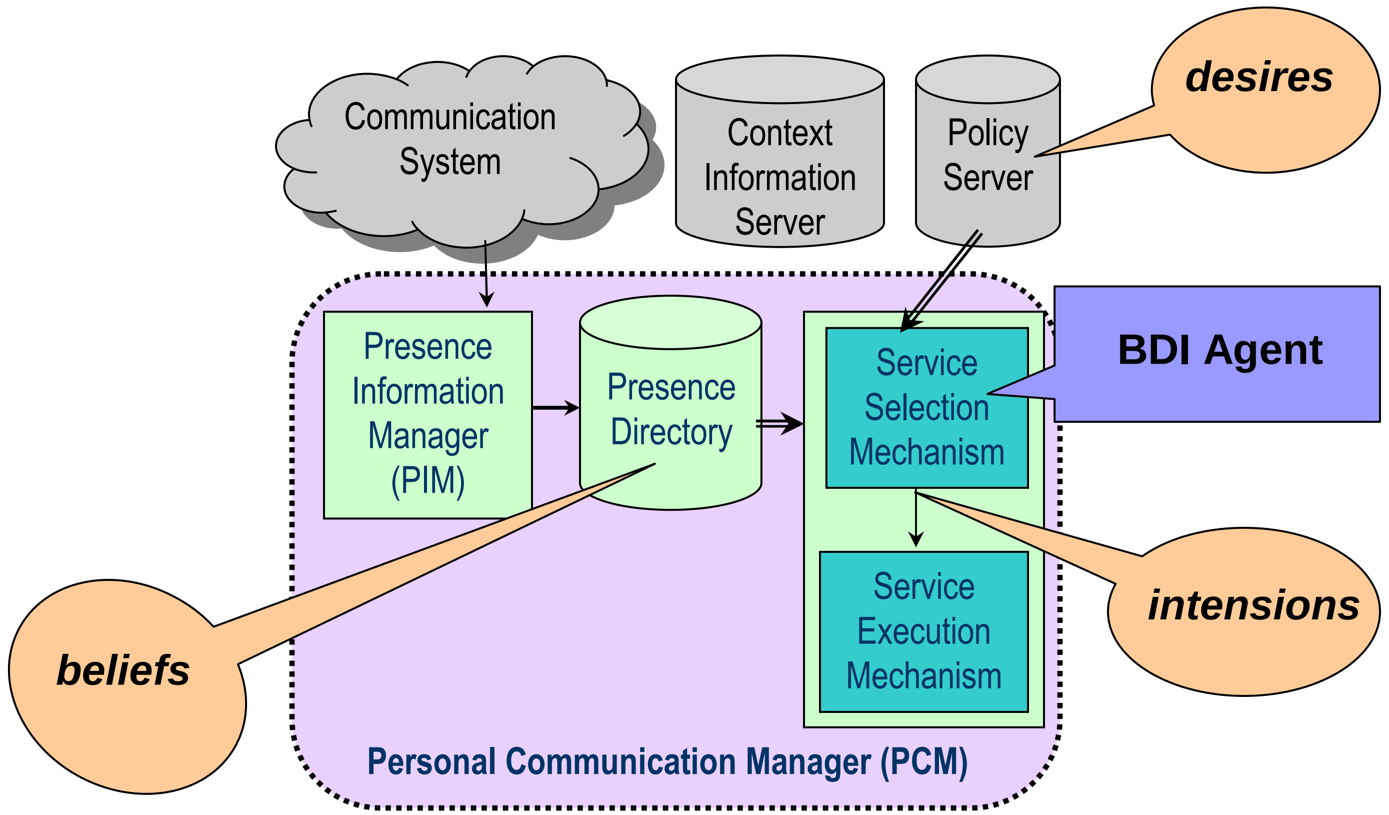
- **Intentions**
  - represent the currently chosen course of action (the output of the most recent call to the selection function)
  - capture the *deliberative* component of the system.

BDI Agents



BDI Mapping

- implementing PPM as a BDI agent


Communication System
Context Information Server
Policy Server
desires
Presence Information Manager (PIM)
Presence Directory
Service Selection Mechanism
BDI Agent
Service Execution Mechanism
intensions
beliefs
Personal Communication Manager (PCM)

Representing Context and Policies

- implement PPM as a BDI agent that conforms to the AgentSpeak) formalism.
- An AgentSpeak(L) agent consists of a ***set of beliefs*** and a ***set of plans***.
  - Beliefs → the content of the Presence Directory
  - Plans → Policies (stored in the Policy Server).
- For implementation, we use **Jason**, an interpreter for AgentSpeak(L)

AgentSpeak(L)

- attempt to bridge the gap between theory and practice
- a model that shows a one-to-one correspondence between the model theory, proof theory and the abstract interpreter.
  - provides an elegant abstract framework for programming BDI agents.
  - natural extension of logic programming for the BDI agent architecture
  - based on a restricted first-order language with events and actions.
  - the behavior of the agent (i.e., its interaction with the environment) is dictated by the programs written in AgentSpeak(L).

AgentSpeak(L) - Basic Notions

- The specification of an agent in AgentSpeak(L) consists of:
  - a set of base ***beliefs***
    - facts in the logic programming sense
  - a set of ***plans***.
    - context-sensitive, event-invoked recipes that allow hierarchical decomposition of goals as well as the execution of actions with the purpose of accomplishing a goal.

AgentSpeak(L) - Basic Notions

- ***belief atom***
  - is a first-order predicate in the usual notation
  - belief atoms or their negations are termed ***belief literals***.

AgentSpeak(L) - Basic Notions

- ***goal***
  - is a state of the system, which the agent wants to achieve.
- two types of goals:
  - ***achievement goals***
    - predicates prefixed with the operator “!”
    - state that the agent wants to achieve a state of the world where the associated predicate is true.
    - in practice, these initiate the execution of *subplans*.
  - ***test goals***
    - predicates prefixed with the operator‘?’
    - returns a unification for the associated predicate with one of the agent’s beliefs; it fails if no unification is found.

AgentSpeak(L) - Basic Notions

- ***triggering event***
  - defines which events may initiate the execution of a plan.
  - an *event* can be
    - internal, when a subgoal needs to be achieved
    - external, when generated from belief updates as a result of perceiving the environment.
  - two types of triggering events:
    - related to the *addition* ('+') and *deletion* ('-') of attitudes (beliefs or goals).

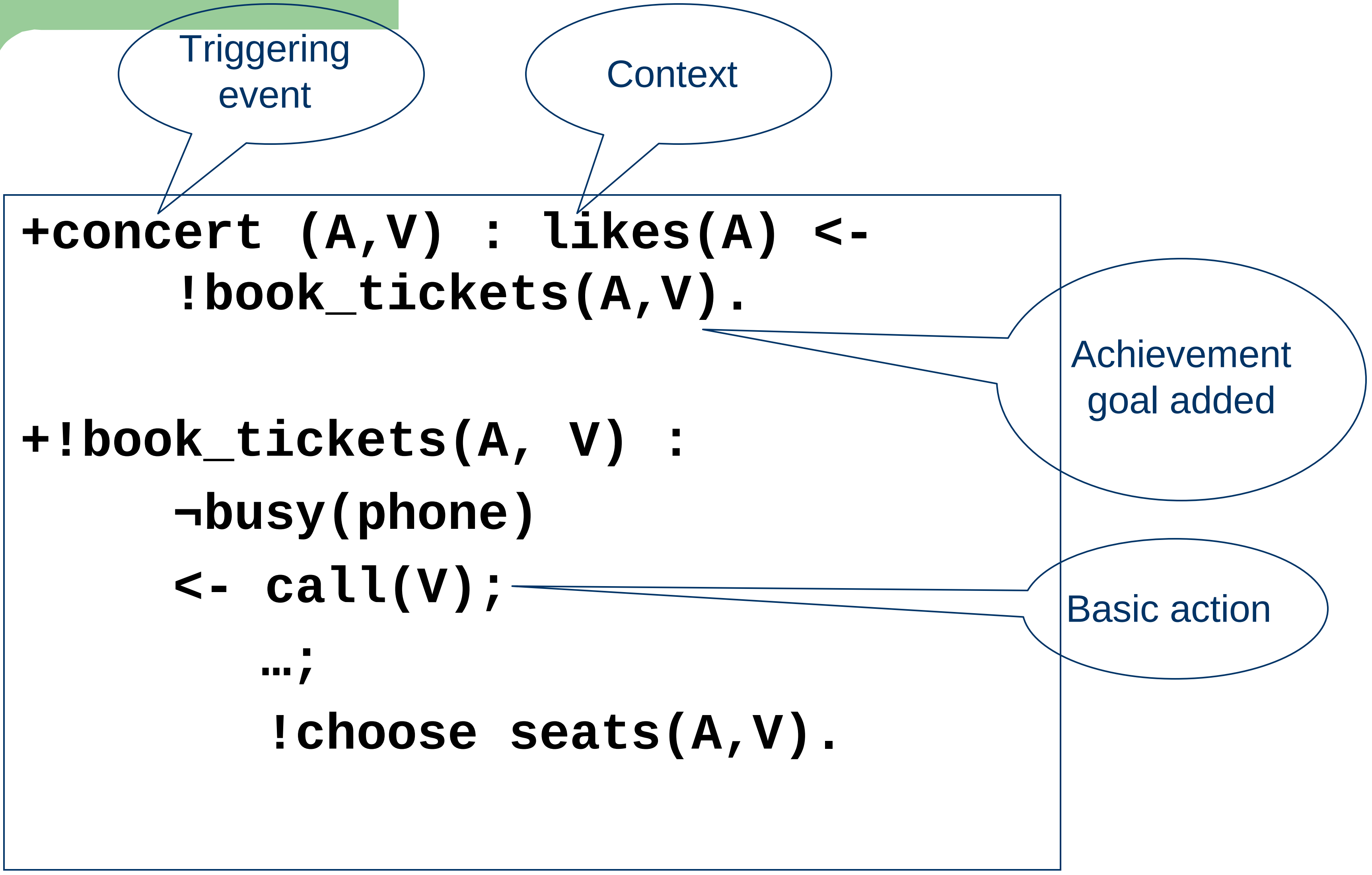
AgentSpeak(L) - Basic Notions

- ***Plans***
  - refer to the *basic actions* that an agent is able to perform on its environment.

```
p ::= te : ct <- h
```

Where:

- **`te`** *- triggering event (denoting the purpose for that plan)*
- **`ct`** *- a conjunction of belief literals representing a context.*
  - *The context must be a logical consequence of that agent's current beliefs for the plan to be applicable.*
- ***h*** *- a sequence of basic actions or (sub)goals that the agent has to achieve (or test) when the plan, if applicable, is chosen for execution.*



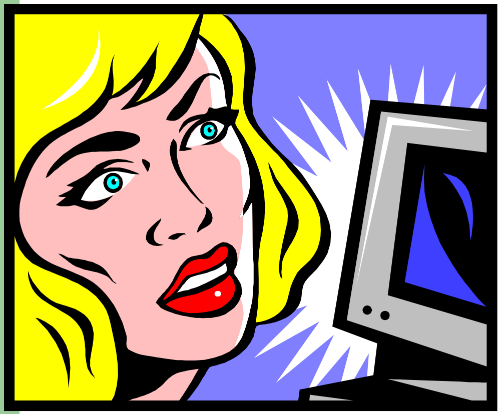
```
+concert (A,V) : likes(A) <-
     !book_tickets(A,V).

+!book_tickets(A, V) :
     ¬busy(phone)
     <- call(V);
          …;
          !choose seats(A,V).
```

AgentSpeak(L) - Basic Notions

- ***Intentions***
  - plans the agent has chosen for execution.
  - Intentions are executed one step at a time.
  - A step can
    - query or change the beliefs
    - perform actions on the external world
    - suspend the execution until a certain condition is met
    - submit new goals.
  - The operations performed by a step may generate new events, which, in turn, may start new intentions.
  - An intention succeeds when all its steps have been completed. It fails when certain conditions are not met or actions being performed report errors.

AgentSpeak(L) Example


ALICE

- During lunch time, forward all calls to Carla.
- When I am busy, incoming calls from colleagues should be forwarded to Denise.

AgentSpeak(L) Example Beliefs

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time(“11:30”).
```

AgentSpeak(L) Example Plans

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time("11:30").
```

"During lunch time, forward all calls to Carla".

```
+invite(X, alice) : lunch_time(t)   ←
   !call_forward(alice, X, carla). (p1)
```

"When I am busy, incoming calls from colleagues should be forwarded to Denise".

```
+invite(X, alice) :
     colleague(X)  ←
   !call_forward_busy(alice,X,denise).
                                   (p2)
```

```
+invite(X, Y): true   ←   connect(X,Y).
                                   (p3)
```

AgentSpeak(L) Example Plans

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time(“11:30”).
+invite(X, alice) : lunch_time(t)   ←   !call_forward(alice, X, carla).   (p1)
+invite(X, alice) :     colleague(X)  ← call_forward_busy(alice,X,denise).(p2)
+invite(X, Y): true   ←  connect(X,Y).                                     (p3)
```

```
+!call_forward(X, From, To) : invite(From, X)
   ← +invite(From, To), - invite(From,X)          (p4)
```

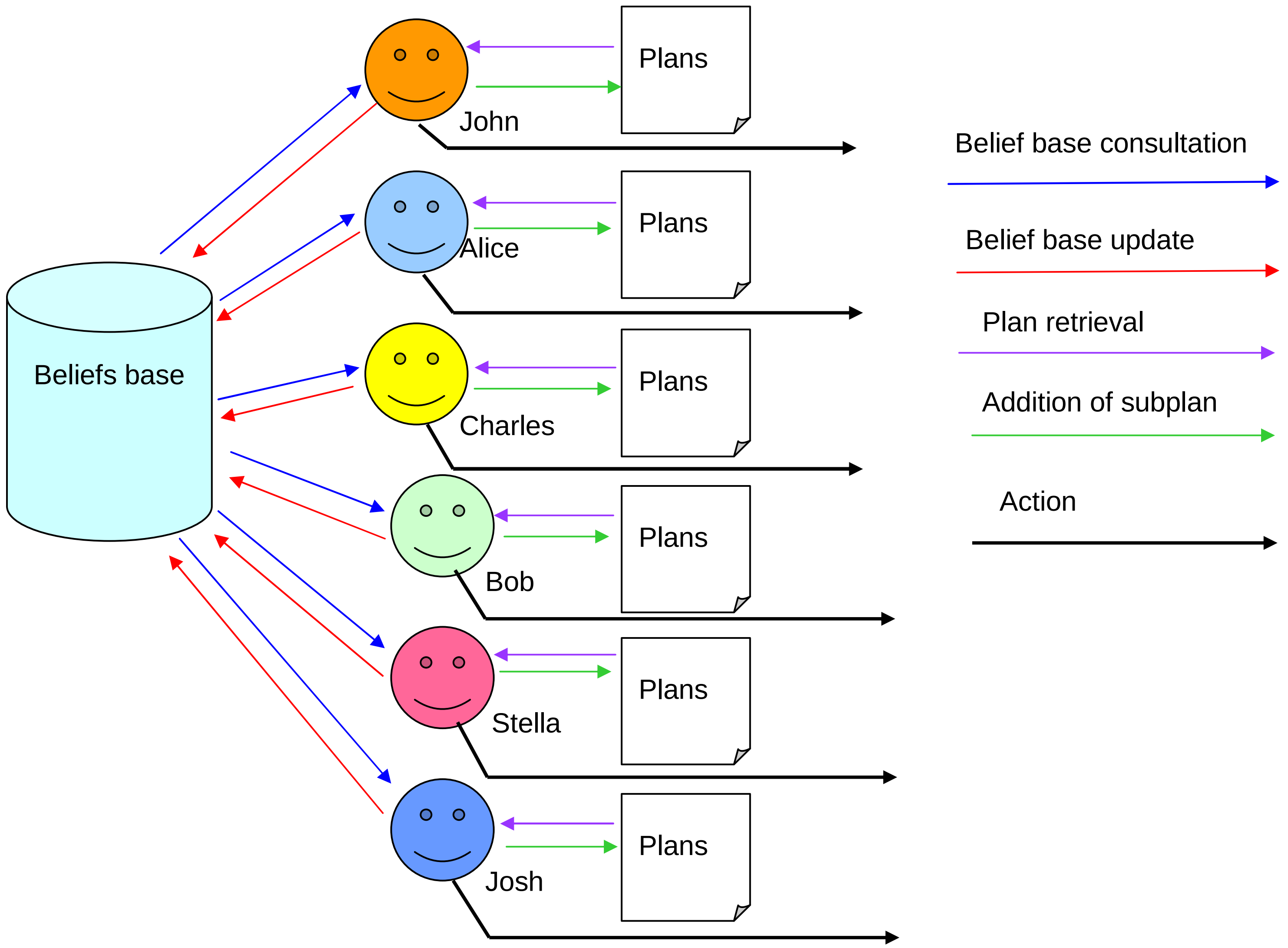
```
+!call_forvard_busy(Y, From, To) : invite(From, Y)&
   not(status(Y, idle)))
         ← +invite(From, To), - invite(From,Y).     (p5)
```

AgentSpeak(L) Example

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time(“11:30”).

+invite(X, alice) : lunch_time(t)
               ← !call_forward(alice, X, carla).           (p1)
+invite(X, alice) :    colleague(X)
               ← call_forward_busy(alice,X,denise).        (p2)
+invite(X, Y): true   ←  connect(X,Y).                   (p3)
+!call_forward(X, From, To) : invite(From, X)
   ← +invite(From, To), - invite(From,X)                        (p4)
+!call_forvard_busy(Y, From, To) : invite(From, Y)& not(status(Y,
   idle)))
       ← +invite(From, To), - invite(From,Y).              (p5)
```

Simulation


Beliefs base
John
Alice
Charles
Bob
Stella
Josh
Plans
Plans
Plans
Plans
Plans
Plans
Belief base consultation
Belief base update
Plan retrieval
Addition of subplan
Action

Essential features (1)

- **Plan selection**
  - In case there are a number of alternative plans for achieving the same goal, the agent is able to make a choice based on some comparison of the different plans.
  - may depend on the time needed, the overall cost, the risk factor, the user preferences, etc.
  - Appropriate decision procedures must therefore be supplied for supporting plan selection.
- **Context-sensitivity**
  - Planning must take into account the current context in which the user is situated (the current user’s physical location, the latest changes in his schedule, etc.).
  - The beliefs base is updated with all the changes in the environment using the AgentSpeak mechanism of event perception.

Essential features (2)

- **Plan failure recovery**
  - If a plan fails at some stage, the agent is able to retract properly and select another alternative plan.
- **Conflict resolution and goal selection**
  - the user might have a number of goals that cannot be achieved simultaneously.
  - In such cases, the agent must be able to make a decision about which goals to try to achieve.
  - In making such decisions, it needs to take into account the importance of the goals as well as the costs of executing the plans.

Conclusions

- the BDI agent paradigm, although originally developed for other purposes, is particularly suited to the user communication domain.
- The actions that the agent decides to take arise from the instantiation of partially specified plans, selected to fulfill the user's goals, given the beliefs that it has at that point in time.
- The details of the plan are filled in as the plan progresses, which allows for a wide range of possible courses of action.