A Short Graduate-Level Course on Requirements Engineering (RE)

John Mylopoulos
University of Ottawa

Federal University of Pernambuco (UFPE),
Recife, November 11-20, 2019

Acknowledgements

- This short graduate-level course touches on research conducted over the past 45 years at the Universities of Toronto and Trento with colleagues and students, including Sol Greenspan (NSF), Alex Borgida (Rutgers), Eric Yu (UToronto), Lawrence Chung (UTexas, Dallas), Jaelson Castro (UFPE), Manuel Kolp (UNamur), Neil Ernst (UVic, Canada), Ivan Jureta (UNamur, Belgium), Vitor Souza (UFES), Roberto Sebastiani, Paolo Giorgini, Kostas Angelopoulos, Fatma Aydemir, Mai Chi Nguyen (Trento, Italy); also, most recently, Yehia ElRakaiby (LERO) and Alessio Ferrari (CNR).

Course objectives

- The course consists of two 1hr and three 2hr lectures; it is intended to introduce the topic of Requirements Engineering (RE), focusing on Goal-Oriented RE (GORE), discuss its importance and history.
- It also covers three research threads each of which spans more than a decade of research. The topics are:
  - ✓ The requirements problem (Jackson&Zave, GORE);
  - ✓ Modelling and analyzing requirements (NFR, i*);
  - ✓ A requirements-based software development methodology for multi-agent systems (Tropos).

Lecture I
Requirements Engineering: Motivation, Definitions and History

John Mylopoulos
University of Ottawa

Federal University of Pernambuco (UFPE),
Recife, November 11, 2019

Requirements Engineering (RE)

- Concerned with the elicitation, analysis and refinement of ***stakeholder requirements*** in order to produce a ***specification*** for a system-to-be.
- Founded on seminal works by Douglas Ross, Michael Jackson and others in the mid-70s.
- Unique research area within Computer Science because its task is to ***define*** software engineering problems (in terms of a specification), rather than solve ones.
- Interesting area, because stakeholder ("early") requirements are almost always vague, informal, conflicting, unattainable, and more (... in short, "scruffy"), but they constitute the input to the RE process none-the-less.

RE is both hard and expensive

- Major cause of project failure: Survey of US software projects by the Standish group

	1994	1998
Successful	16%	26%
Challenged	53%	46%
Cancelled	31%	28%

Top 3 success factors:
1) User involvement
2) Executive management support
3) Clear statement of requirements

Top 3 factors leading to failure:
1) Lack of user input
2) Incomplete requirements & specs
3) Changing requirements & specs

(for a more recent survey, see http://calleam.com/WTPF/?page_id=1445)

- Cost of fixing errors: A requirements error found during testing costs 100 times more to fix than a programming error found during testing.

RE example

- Suppose we are asked to specify requirements for a meeting scheduler running in a university department and serving its members.
- Here our top stakeholder requirement for the system-to-be is

  r0 := “System shall schedule meetings”
- We ask the department head if there are other requirements, and she proposes

  r1 := “Timetable information shall remain confidential”

  r2 := “System shall schedule good meetings”
- We also ask potential users, and they add another requirement

  r3 := “Selection of a time slot shall be done by an administrator”

RE example (cont'd)

- r0 is a ***stakeholder requirement***. To address it, we decide that there are two functions needed

  r4:= "Collect timetable constraints from participants for the preferred period of the meeting"

  r5 := "Select a time slot"

- We also decide that r4 will be done by the system, so we now have a ***functional requirement***

  r6 := "System shall collect timetables"

- However, r5 should not be the system's responsibility, according to r3. So, we introduce new functional requirements for the system

  r7 := "System shall display timetable details for administrator"

  r8 := "System shall record administrator's decision"

RE example (cont'd)

- But what about r1 and r2? These are ***quality requirements***. To deal with them, we decide that

  r9 := "Users shall only have access to their personal schedule"

, leading to

  r10 := "Access to meeting database is limited to personal data"

, while r2 leads to

  r11 := "80% of meetings shall have >70% participation"

, a ***quality requirement,*** but also a functional requirement

  r12 := "Systems shall send reminders for all meetings"

RE example (final)

- A ***(requirements) specification*** consists of functional and quality requirements for a system-to-be.
- For our example, the specification has as follows

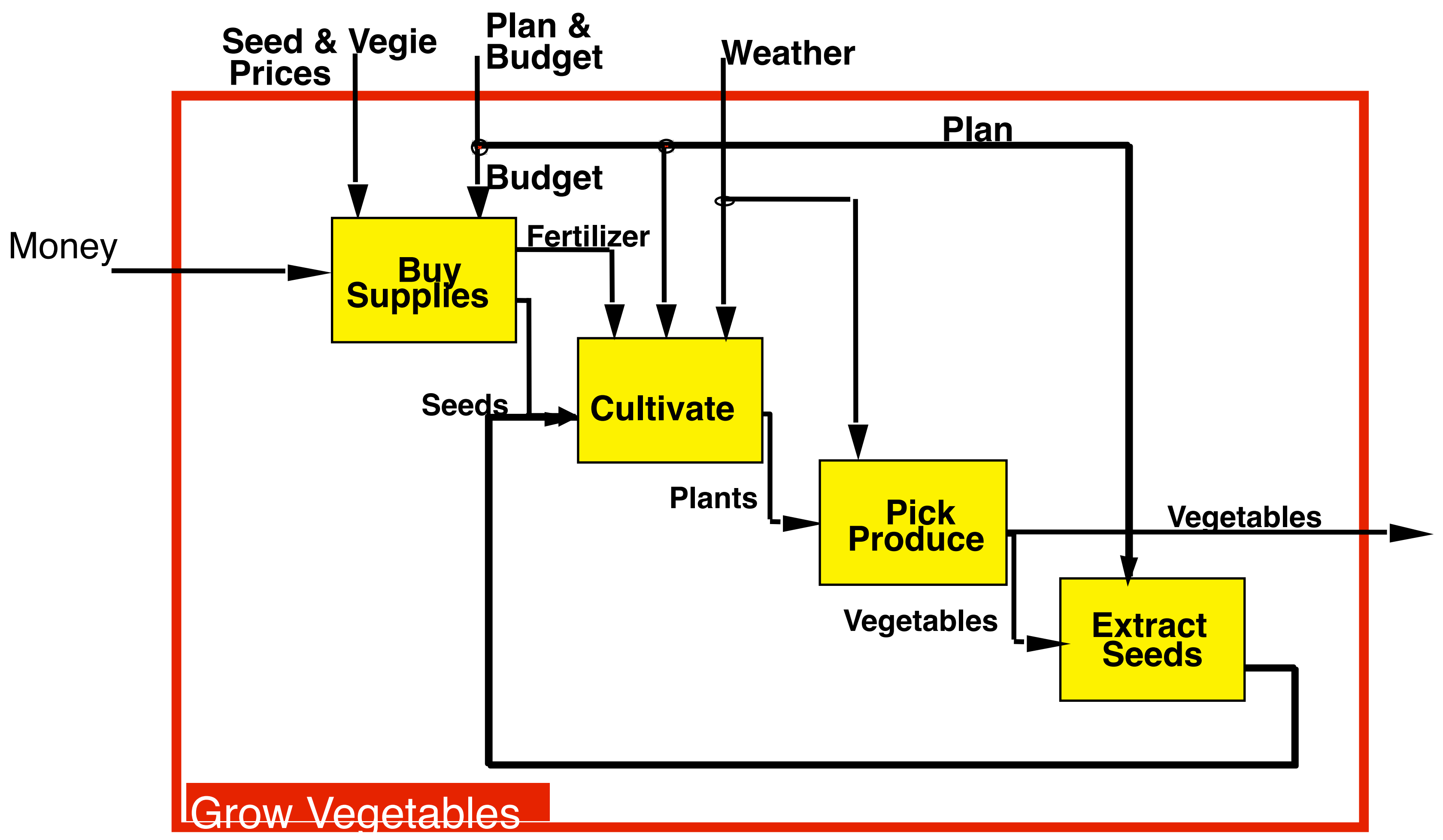
  spec := {r6, r7, r8, r10, r11, r12}
- More terminology:
  - ✓ Functional and quality requirements ***operationalize*** stakeholder requirements.
  - ✓ Stakeholder requirements are ***refined*** into other stakeholder requirements until they are simple enough to be operationalized.
- You can think of stakeholder requirements as defining a ***requirements problem***, while a specification defines a ***solution*** thereof.

… Once upon a time …

- In the early days of Computer Science, people didn't develop software; they simply programmed ⋯
- Software Engineering (SE) was born in 1968 and promised to turn software development into an engineering discipline.
- But unlike other engineering disciplines, research and practice on software development remained oblivious to all activities but one: programming.
- Then Doug Ross came along in the mid-70s as a SE researcher and practitioner …!

Structured Analysis and Design Technique -- SADT (~1975)

[Ross77]



SADT: Novel ideas and impact

- ✧ SADT models consist of activity and data/entity boxes, interconnected through input/output/control arrows.
- ✧ Models represent the operational environment of a system, and how the system's functions fit in that environment.
- ✧ The purpose of SADT models is to describe the ***functional requirements*** of a system.
- ✧ SADT was used in practice since the mid-seventies.
- ✧ It led to Data Flow Diagrams (DFDs) that were taught in universities around the world since the late 70s under the label "Systems Analysis and Design", e.g., [Kendall88].
- ✧ As with other structured techniques, SADT assumed a hierarchically structured problem domain.

Formal requirements modelling languages

- In the ‘80s there were several research threads focusing on formalizing requirements modelling languages, such as SADT and DFDs.
- Among these projects we note:
  - ✓ Janis Bubenko, Royal Institute of Technology [Bubenko80]
  - ✓ Eric Dubois et al, University of Namur [Dubois86]
  - ✓ Sol Greenspan et al, University of Toronto [Greenspan82]

Requirements modelling language (RML)

```
EntityClass Patients with
  necessary, unique, part
     record: MedicalRecords
  association
     location:  NursingHomes;  room:  Ro
    producer
     register: AdmitPatients(per<-this)
  modifier
     assessment: Assess(patient<-this)
  consumer
     release: Discharge(patient<-this) ...
  initially
     rightPlace?: record.place = location
     startClean?: paymentDue = 0
end Patients
```

```
ActivityClass AdmitPatients with
    input
      per: Persons
    control
      home: NursingHome
      doc: Doctors
    output
      pat: Patients
    initially
      alreadyIn?: not(p in Patients)
    finally
                        ...
    part
      getBasicInfo: Interview(whom<-per)
      place: AssignRoom(...)
                        ...
end AdmitPatients
```

Remarks on RML

- RML was formalized by using ideas from semantic networks and (early) description logics.
- Both entity and activity classes were organized into taxonomies of generalization hierarchies.
- Our attempt to formalize a ***structured*** analysis notation ended up with us stumbling on an ***object-oriented*** analysis one [Booch94]!
- The RML paper presented at ICSE 1982 won the most influential paper award 10 ICSEs later precisely for this reason.

Requirements as goals

- Goal-oriented analysis focuses on ***early*** (aka ***stakeholder***) requirements, when a problem (= stakeholder needs) are identified, and alternative solutions are explored and evaluated.
- During goal-oriented analysis, we start with initial stakeholder goals such as "Fulfill every book request", or "Schedule meetings" and keep refining them until we have reduced them to alternative collections of functional and quality requirements that each satisfy initial goals.
- Initial goals may be conflicting, ambiguous, incomplete, invalid (don't represent a stakeholder need) and more.

Goals in KAOS [Dardenne93]

- (Organizational) goals lead to requirements.
- Goals justify and explain the presence of requirements that are not necessarily comprehensible by stakeholders.
- Goals provide basic information for detecting and resolving conflicts that arise from multiple viewpoints [Dardenne93].
- Example goal:

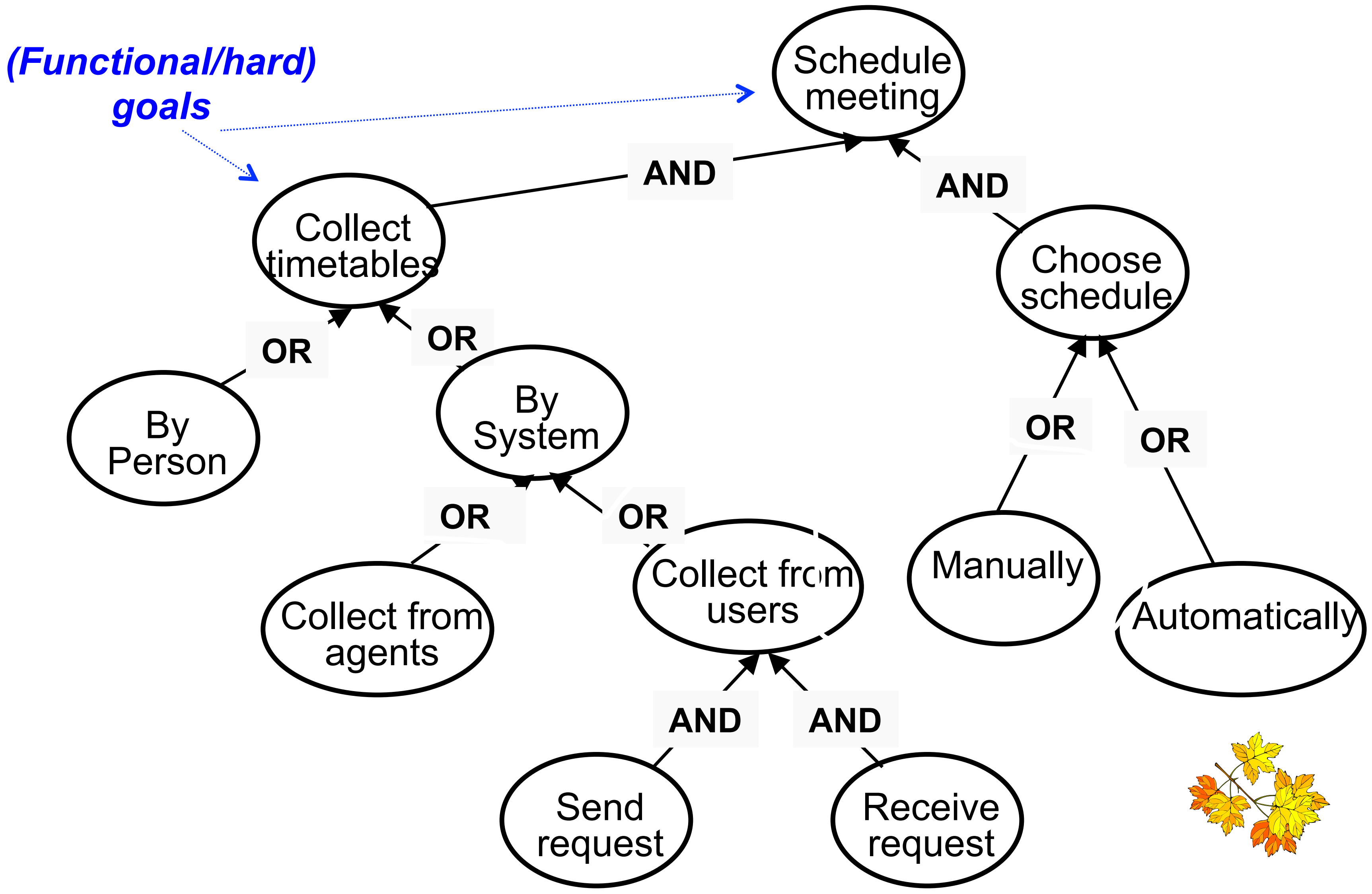
SystemGoal Achieve[BookRequestSatisfied]

InstanceOf SatisfactionGoal

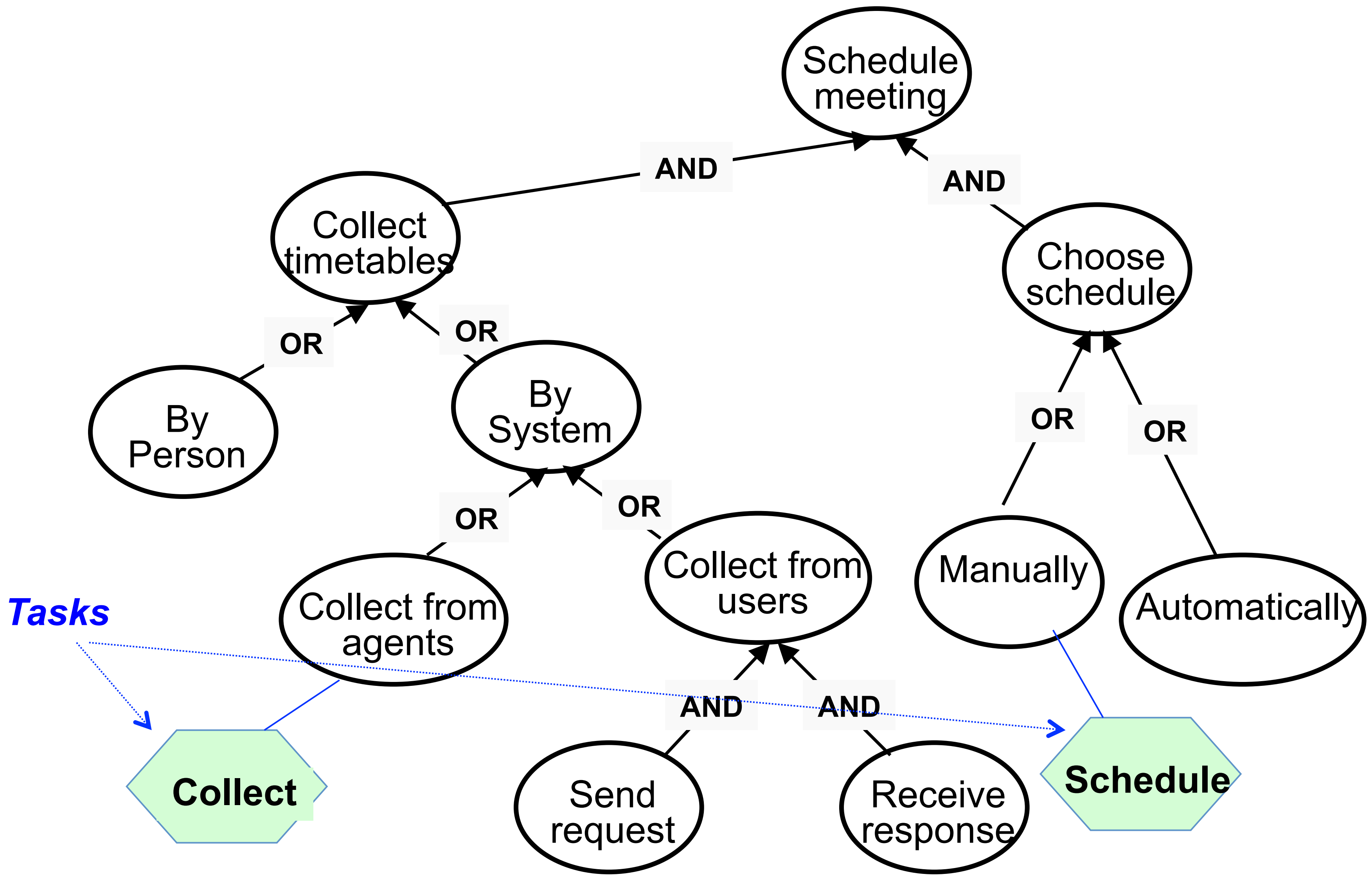
Concerns Borrower, Book, Borrowing, ...

Definition ($\forall$bor: Borrower, b: Book, lib: Library)
(Requesting(bor, b) $\wedge$ b.subject $\in$ lib.coverageArea $\Rightarrow$
$\Diamond$[($\exists$bc: BookCopy) (Copy(bc, b) $\wedge$ Borrowing(bor, bc)))]

Goal analysis leads to alternatives



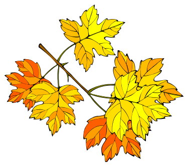
Alternatives lead to designs/plans

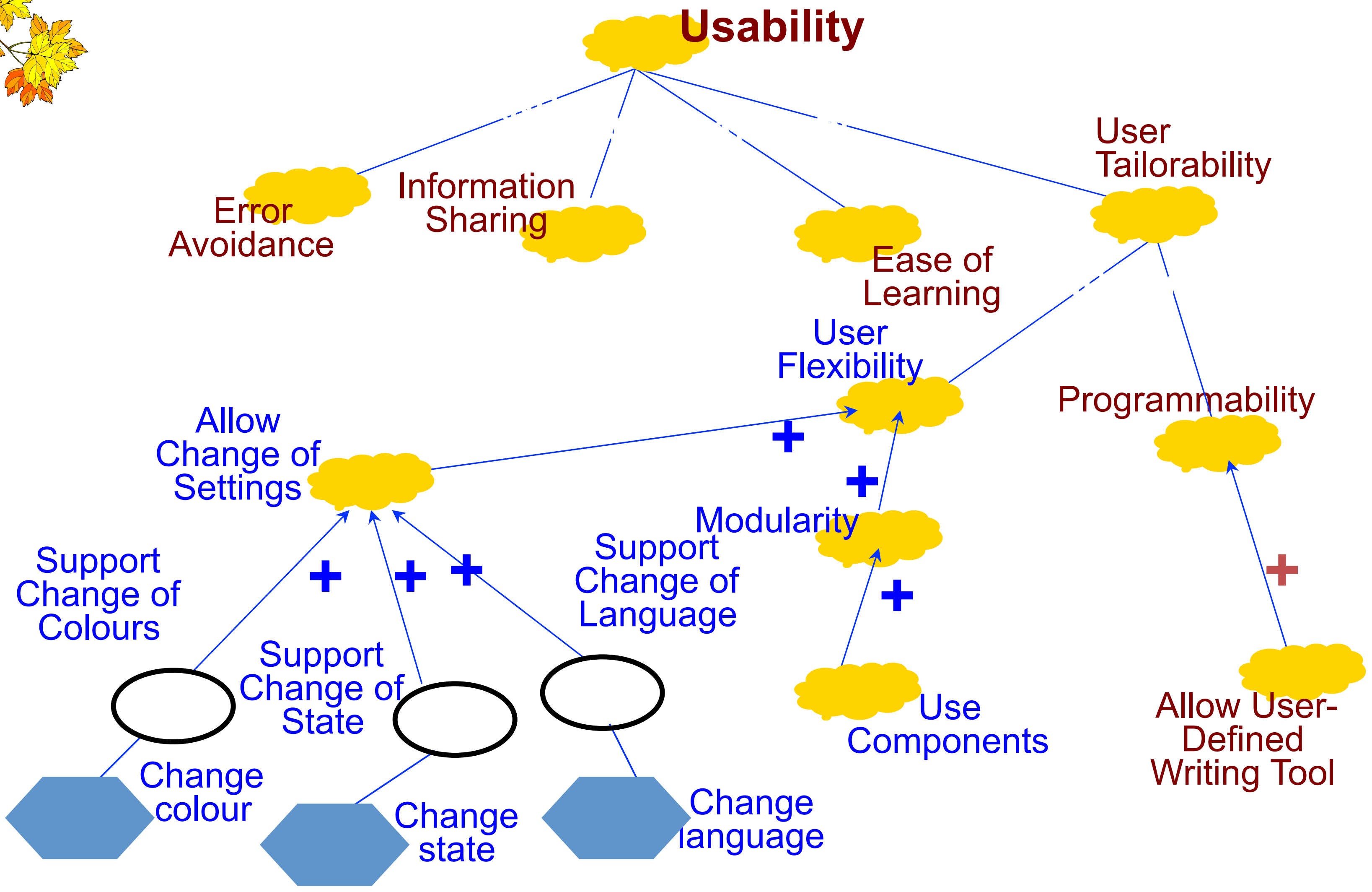


Softgoals

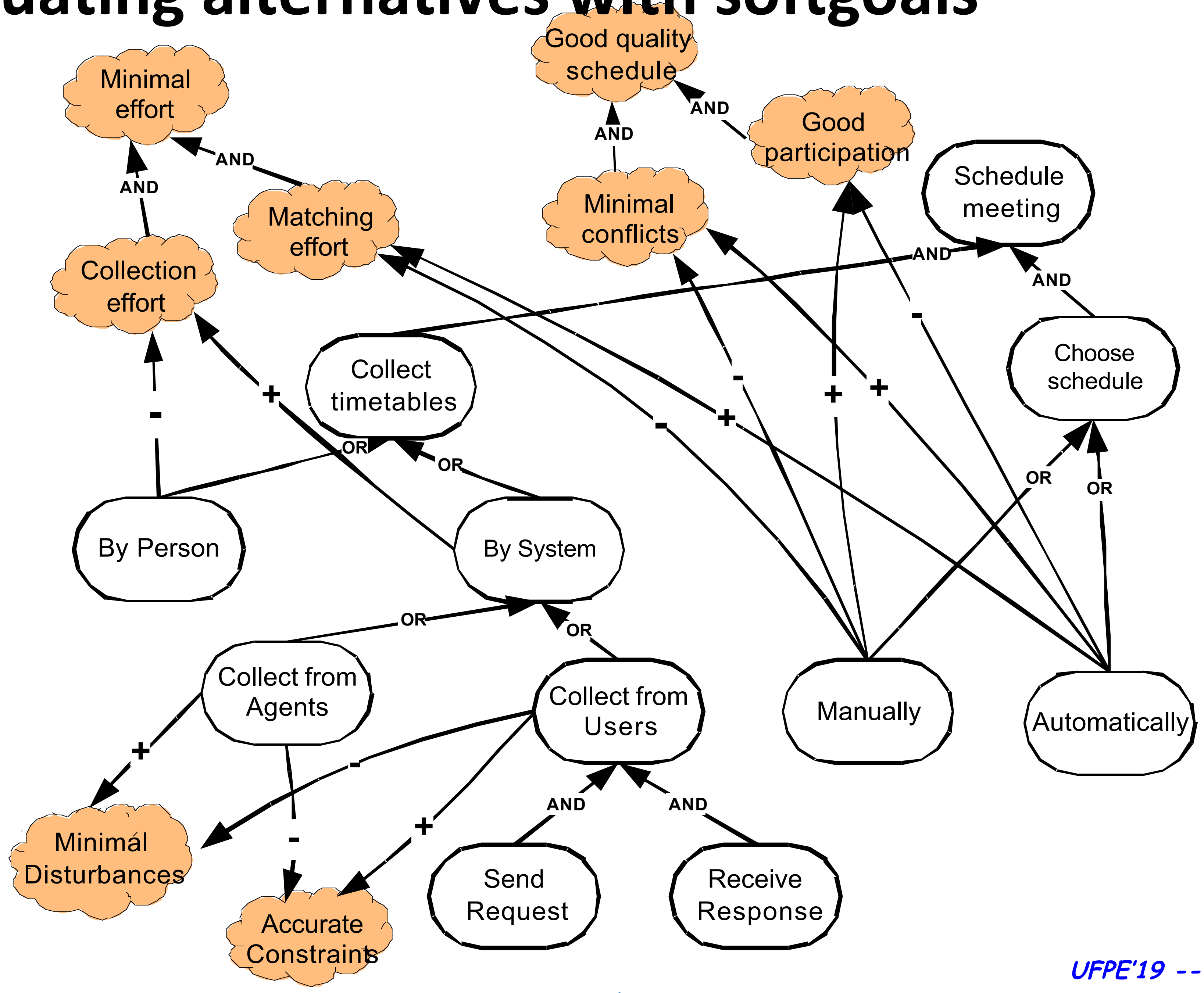
- Functional goals, such as “Schedule meeting” are well defined in the sense that they admit a formal definition.
- ***Quality*** (aka ***non-functional***) ***goals***, e.g., “higher profits”, “satisfied customers” or “easily maintainable system”, specify ***qualities*** a system-to-be should adhere to.
- Such qualities usually admit no generally agreed upon definition, are inter-related and often conflicting.
- Such qualities are represented as *softgoals*.
- Softgoals can be thought as undefined concepts, with no clear-cut criteria for satisfaction; hence softgoals are *satisficed*, rather than satisfied (NFR framework, [Mylopoulos92], [Chung93]).

Usability as a softgoal





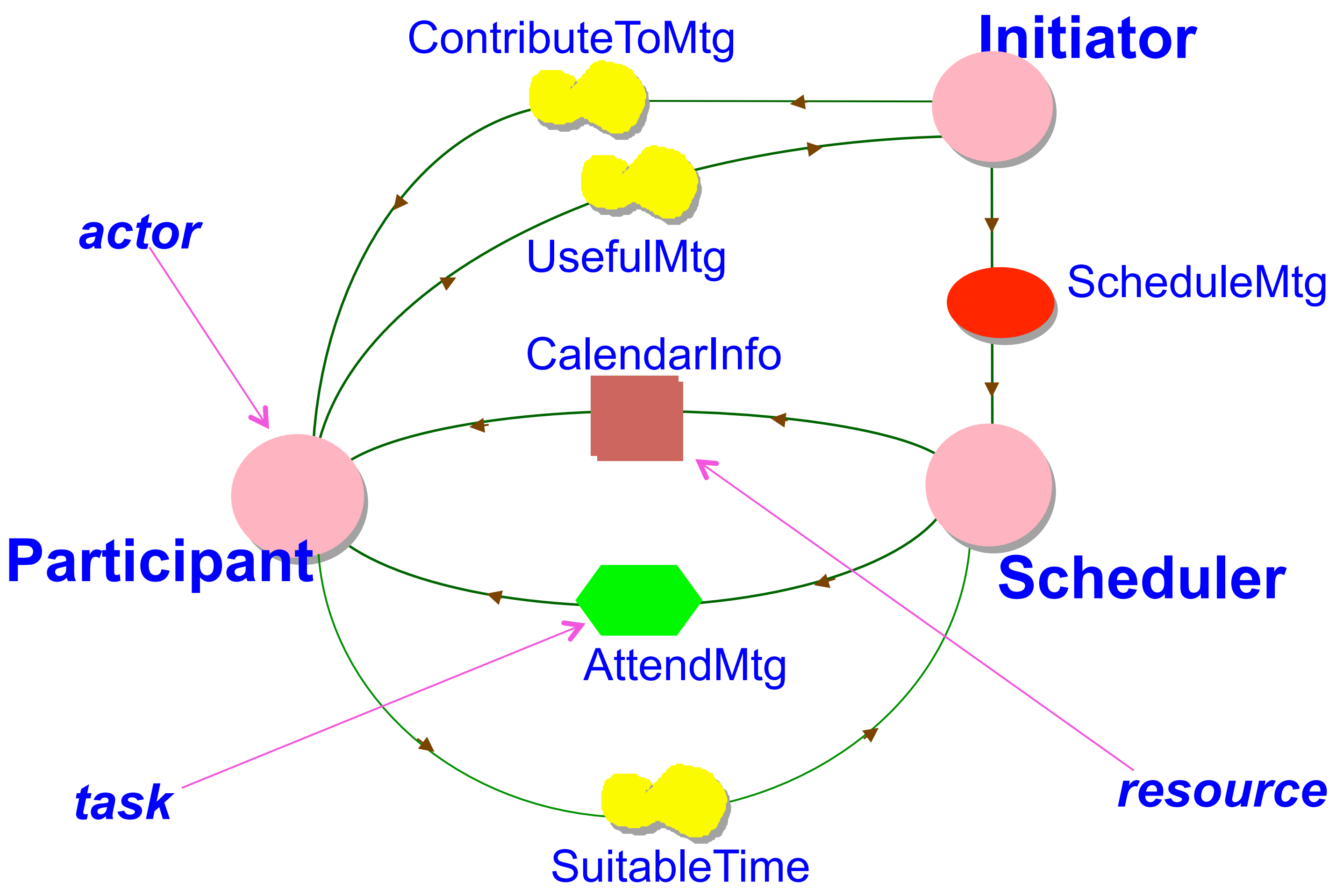
Evaluating alternatives with softgoals



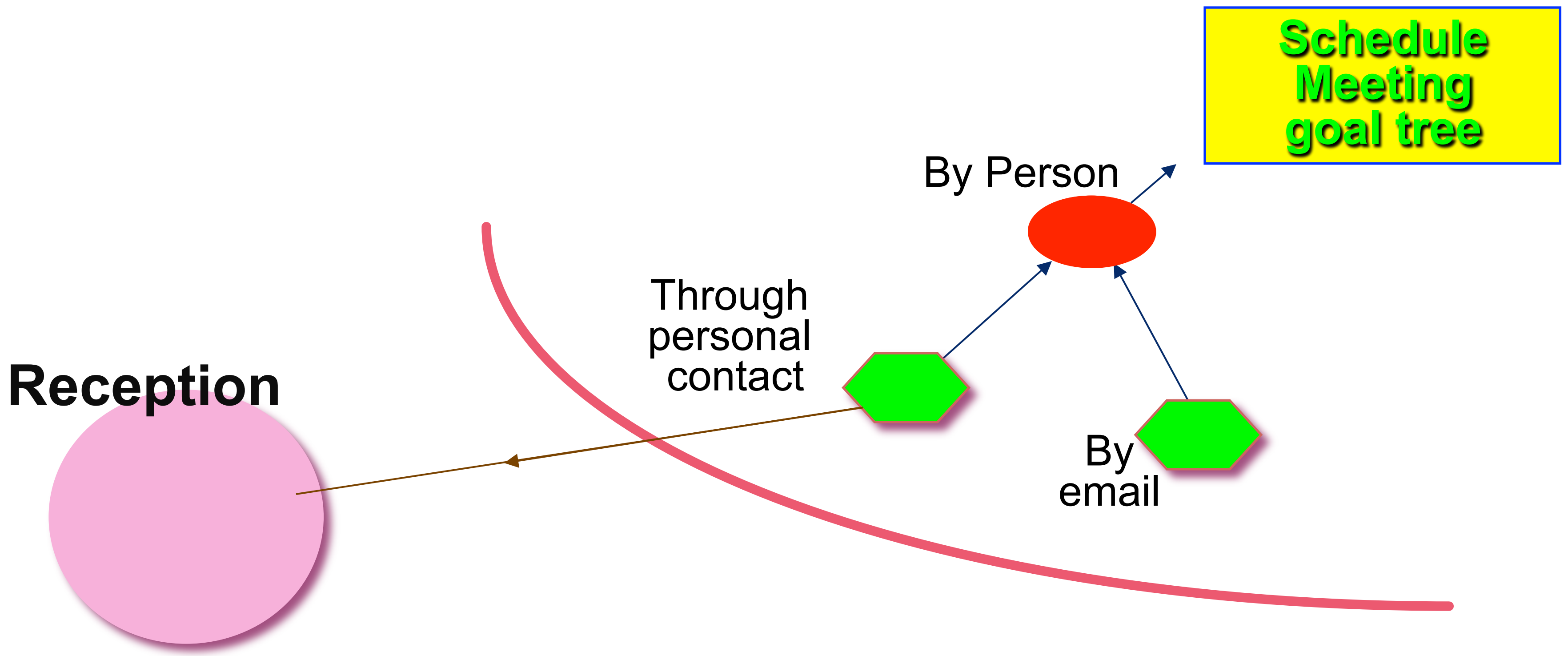
Stakeholders and their goals

- In KAOS, goals are global objectives for the system-to-be.
- In *i** [Yu93], goals are desired by ***actors*** and are ***delegated*** to other actors for fulfillment.
- According to *i**, early requirements analysis involves identifying stakeholders and their goals, analyzing these goals, delegating them to other actors etc.
- The result of this process consists of ***actor dependency*** and ***actor rationale*** models.

An Actor Dependency Model



An actor rationale model



Actor dependencies are ***intentional***: One actor ***wants*** something, another is ***willing*** and ***able*** to deliver on that something.

Goals in requirements analysis

- KAOS, the NFR proposal, as well as *i** advocate the use of goals in RE.
- KAOS uses goals to go from stakeholder needs to functional specifications.
- NFR uses them to represent and analyze quality requirements. Quality requirements lead to criteria for evaluate functional alternatives (… ***and*** specifications).
- *i** relates goals to the actors who want them and keeps track of delegations.

Lecture II
KAOS: Keep All Objectives Satisfied

John Mylopoulos
University of Ottawa

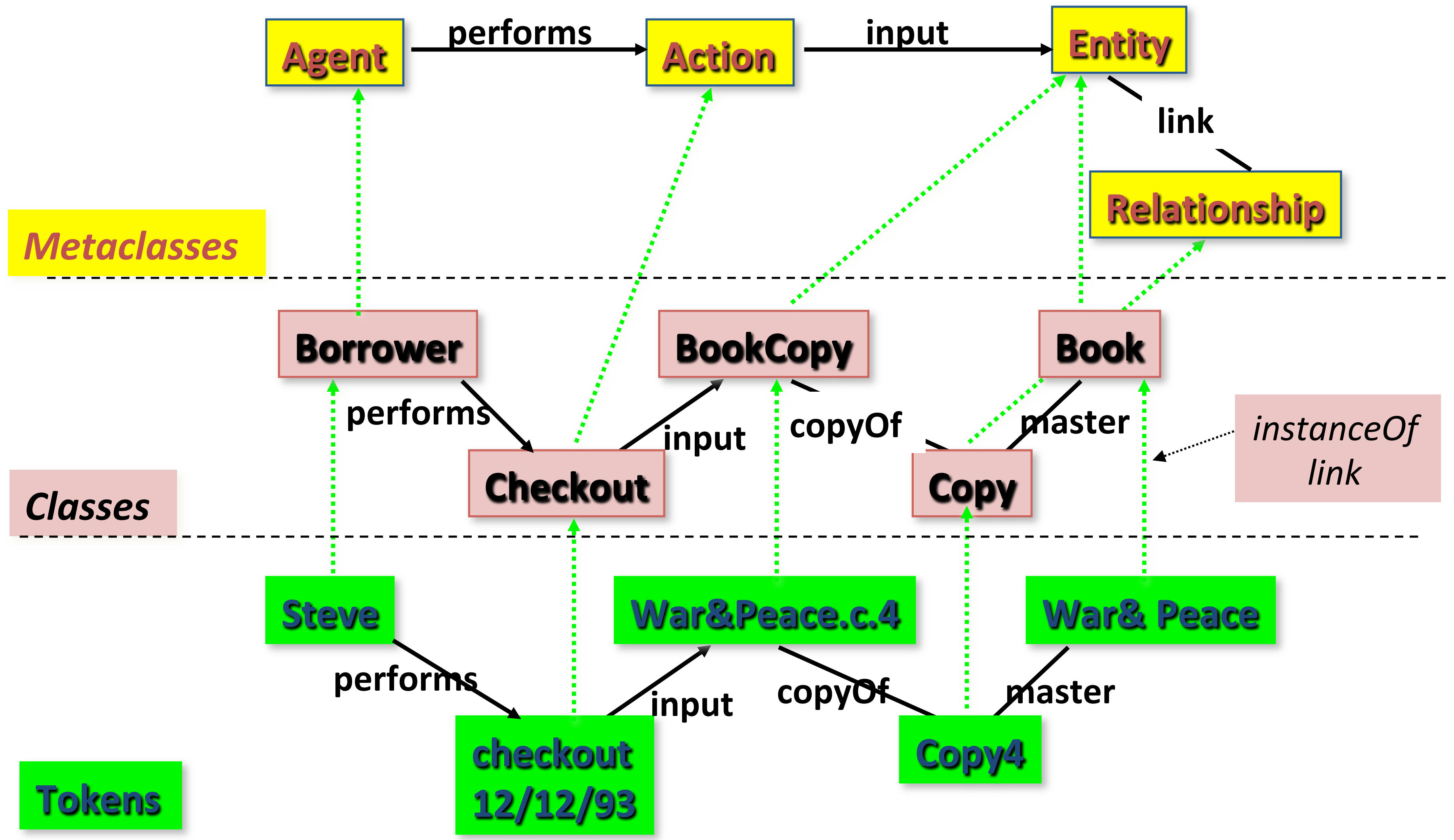
Federal University of Pernambuco (UFPE),
Recife, November 11, 2019

KAOS: Why goals?

- (Organizational) goals lead to requirements.
- Goals justify and explain requirements which are not necessarily comprehensible by stakeholders.
- Goals can be used to assign responsibilities to agents so that prescribed constraints can be met.
- Goals provide basic information for detecting and resolving conflicts that arise from multiple viewpoints

[Dardenne93]

The meta, domain and token levels



Entities and relationships

Entity Library
Has collection, available, checkedOut, lost: setOf[BookCopy]
    coverageArea: setOf[Subject]
Invariant collection = available ∪ checkedOut ∪ lost
 ∧ available ∩ checkedOut = ∅ ∧ available ∩ lost = ∅
 ∧ checkedOut ∩ lost = ∅) ...
end Library

Relationship Borrowing
Links Borrower [Role Borrows, Card 0::N]
    BookCopy [Role Borrowed, Card 0::1]
Invariant (∀lib: Library, bor: Borrower, b:Book, bc: BookCopy)
 [Borrowing(bor,bc) ∧ bc ∈ lib.collection ⇒
   bc ∈ lib.checkedOut ∧ ♦ Requesting(bor,b) ∧ Copy(bc,b)] ...
end Borrowing

Events

- An event represents an instantaneous happening.
- Occurs(e) == e.exists = true
- Here is an example of an event:

```
  Event ReminderIssued
    Has toWhom:Bor, what:BookCopy, mes: text;
    Invariant  (∀rm: ReminderIssued)
(Occurs(rm(toWhom,what,mes)) ⇒
       (∃p: Staff)(Performs(p,IssueReminder(what,toWhom)))
    …
 end ReminderIssued
```

Actions

Action CheckOut

Input BookCopy [Arg: bc], Library [Arg: lib], Borrower [Arg: bor]

Output Library [Res: lib], Borrowing

Precondition bc ∈ lib.available

Postcondition ¬(bc ∈ lib.available) ∧ bc ∈ lib.checkedOut ∧
    Borrowing(bor,bc)

Action IssueReminder

Input BookCopy [Arg: bc], Borrower [Arg: bor]

Output Reminder

Triggercondition

$\blacklozenge_{>2wks}$ Borrowing(bor,bc) ∧
        ¬($\blacklozenge_{\leq 1wk}$ ∃rm: IssueReminder Occurs(rm(bc,bor)))

/* bc has been borrowed by bor for at least 2wks and there hasn't been a reminder within the last week */

Representing time in KAOS

$\circ\phi$ - ϕ is true in the next state

$\bullet\phi$ - ϕ is true in the previous state

$\Diamond_{\leq x}\phi$ - ϕ will be true sometime (within x)

$\blacklozenge_{\leq x}\phi$ - ϕ was true sometime (within x)

$\Box_{\leq x}\phi$ - ϕ will be always true (until some time)

$\blacksquare_{\leq x}\phi$ - ϕ was always true (before some time)

ϕ U ψ - ϕ is true until ψ becomes true

ϕ S ψ - ϕ has been true since ψ became true

Notation:

circle - previous/next state

diamond - sometime in the past/future

square - always in the past/future

Agents

Agent Staff
Has competenceAreas: setof[Competence]
Invariant (∀st:Staff)
 InstanceOf(st,LibrarianStaff) ∨ InstanceOf(st, ClerkStaff)
Load ....   /* describes the agent' s work load */
CapableOf CheckIn, CheckOut, IssueReminder, Reference, Cataloguing
Performs Checkout
Knows Borrowing [Interface: BorrowingDB]

- Agents may be humans, organizational units, or software.
- Agents may be composed from other agents through a Cartesian product construction.
- An agent performs only actions she is capable of.
- Knows(ag,obj) means that ag can observe the state of obj through some interface.

Goals

SystemGoal Achieve[BookRequestSatisfied]
InstanceOf SatisfactionGoal
Concerns Borrower, Book, Borrowing,...
Definition ($\forall$bor: Borrower, b: Book, lib: Library)
(Requesting(bor,b) $\wedge$ b.subject $\in$ lib.coverageArea
$\Rightarrow \Diamond$ ($\exists$bc: BookCopy) (Copy(bc,b) $\wedge$ Borrowing(bor,bc)))
ReducedTo EnoughCopies, RegularAvailability,
AvailabilityNotified
ReducedTo AsManyCopiesAsNeeded

SystemGoal Maintain[SafeTrasportation]
InstanceOf SafetyGoal
Concerns Passenger
InformalDef ...

Goal patterns

- A goal is a ***non-operational objective*** in that there is no single action that an agent can perform to achieve it.
- Patterns identify what can be done with a goal:

Achieve – achieve a goal at some point in the future

$$P \Rightarrow \Diamond Q$$

Cease -- undo a goal at some point in the future

$$P \Rightarrow \Diamond \neg Q$$

Maintain -- maintain a goal for some time

$$P \Rightarrow \Box Q$$

Prevent -- prevent a goal from becoming true

$$P \Rightarrow \Box \neg Q$$

Optimize -- maximize or minimize some measure

max(fcn) or min(fcn)

Goal metaclasses

- Goals also have associated categories defined by metaclasses, such as:
  - ✓ *SatisfactionGoal* -- satisfying agent needs
  - ✓ *InformationGoal* -- informing agents
  - ✓ *RobustnessGoal* -- recovering from failures
  - ✓ *ConsistencyGoal* -- maintaining consistency
  - ✓ *SafetyGoal, PrivacyGoal*, maintain agents in states that are safe and observable under restricted conditions.
- These categories are useful because each one has its own heuristics for decomposition and operationalization (satisfaction).

Subgoals

SystemGoal Maintain[RegularAvailability]
InstanceOf SatisfactionGoal
Concerns Library
Definition (∀bor: Borrower, b: Book, bc: BookCopy, lib: Library)
(bc ∈ lib ⇒ □[¬(bc ∈ lib.available) ⇒ ($\Diamond_{\leq 2wks}$ bc ∈ lib.available)])

SystemGoal Achieve[AvailabilityNotified]
InstanceOf InformationGoal
Concerns Borrower, Library
Definition (∀bor: Borrower, b: Book, bc: BookCopy, lib: Library)
(Requesting(bor,b) ∧ ●(¬∃bc:BookCopy (Copy(bc,b) ∧ bc ∈ lib.available)) ∧ (∃bc:BookCopy (Copy(bc,b) ∧ bc ∈ lib.available)) ⇒ ◊Knows(bor, lib.available))
/* If a borrower requests a book, and the book just became available, the borrower will be informed */

Conflicting goals

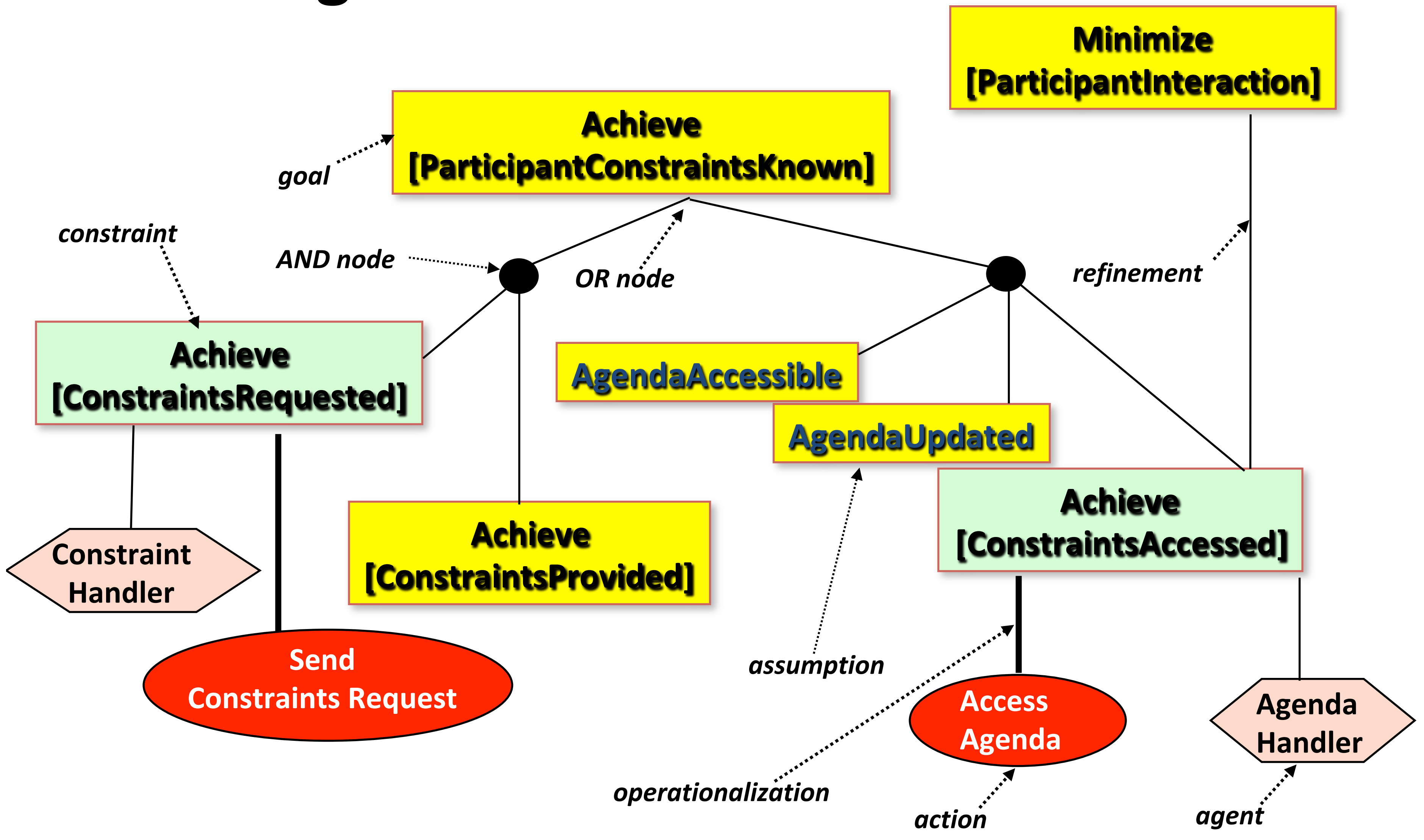
PrivateGoal Maintain[LongBorrowingPeriod]
InstanceOf SatisfactionGoal
Concerns Borrower, Borrowing
Definition ($\forall$bor: Borrower, b: Book, bc: BookCopy)
$[\text{Borrowing}(bor, bc) \land \text{Copy}(bc, b) \land \circ\text{Need}(bor, b)$
$\Rightarrow \circ\text{Borrowing}(bor, bc)]$
/* If a borrower has borrowed a book and she still needs it, she can continue to borrow it */
Conflicts with RegularAvailability

This goal is in conflict with RegularAvailability and can be declared so explicitly.

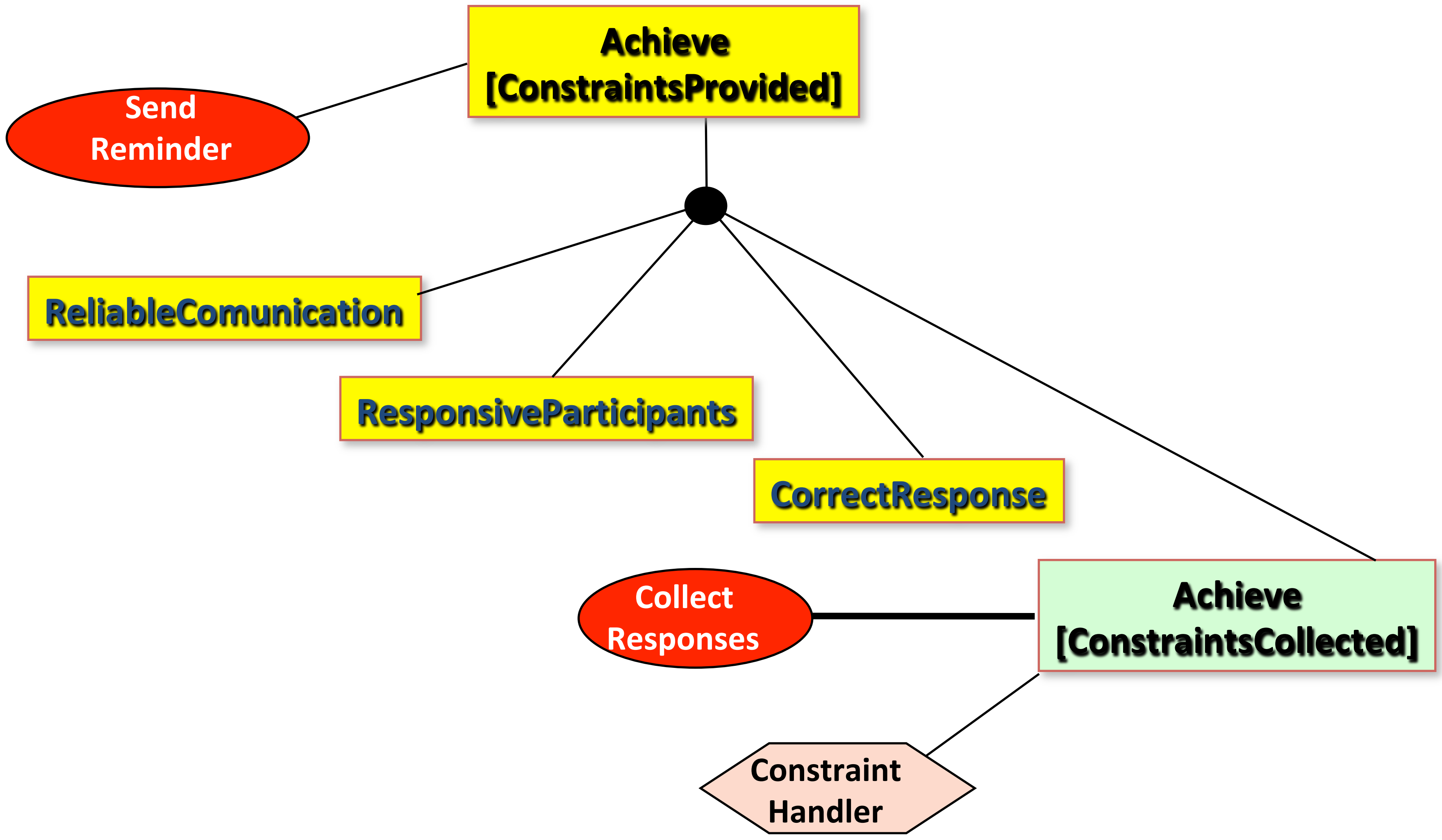
Constraints

- Constraints are ***operational objectives*** in that there are particular actions that agents can perform to achieve them.
  SoftConstraint Maintain[LimitedBorrowingPeriod]
   Definition (∀bor: Borrower, bc: BookCopy)
        (Borrowing(bor, bc) ⇒ ◇ ¬ Borrowing(bor, bc)
- Constraints operationalize goals
   SystemGoal Maintain[RegularAvailability]
   Concerns ...
   Definition ...
   OperationalizedBy LimitedBorrowingPeriod, NoLostCopies,...
- Constraints are ***ensured*** by restricting existing actions and objects (through strengthened preconditions, invariants, etc.) or through the introduction of new actions and objects.

The Big Picture



… more detail …


Achieve [ConstraintsProvided]
Send Reminder
ReliableComunication
ResponsiveParticipants
CorrectResponse
Collect Responses
Achieve [ConstraintsCollected]
Constraint Handler

The KAOS modeling philosophy

- Modeling a social setting involves a variety of concepts, including ***goals***, ***agents***, ***concerned objects***, ***actions***, ***constraints*** and ***responsibilities***.
- Goals lead to assignments of responsibilities and designs of actions and artifacts
- Uses a metamodel to support reuse of generic domain modeling patterns [Dardenne93]
- For example, the library domain is an instance of the "resource allocation" meta-domain, which also covers car/room/dwelling rental and is similar to airline/hotel reservation, class registration etc.

The KAOS requirements analysis process

- Identify goals, and their concerned objects.
- Identify potential agents and their capabilities.
- Operationalize goals into constraints.
- Refine objects and actions.
- Derive strengthened objects and actions to ensure constraints.
- Identify alternative responsibilities.
- Assign actions to responsible agents.

- All this could be just as useful for organizational design as it is for software development!

References for lectures I and II

- [Booch94] Booch, G., *Object-Oriented Analysis and Design,* Benjamin-Cummings, 1994.
- [Bubenko80] Bubenko J., "Information modeling in the context of system development", Proceedings IFIP Congress 1980. 395-411, 1980.
- [Chung93] Chung L.*, Representing und Using Non-Functional Requirements: A Process-Oriented Approach*. PhD. thesis, Dept. of Computer Science, University of Toronto, 1993.
- [Dardenne93] Dardenne A., van Lamsweerde A., Fickas S.,"Goal-Directed Requirements Acquisition", in *The Science of Computer Programming 20*, 1993.
- [Darimond96] Darimond R., van Lamsweerde A., "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", 4th ACM Symposium on Foundations of Software Engineering, 179-190, San Francisco, 1996.
- [Dubois86] Dubois E., Hagelstein J., Lahou E, Ponsaert F., and Rifaut A. " A knowledge representation language for requirements engineering", Proc. of the IEEE, 74(10), 1986.
- [Greenspan82] Greenspan S., Mylopoulos J., and Borgida A.. "Capturing more world knowledge in the requirements specification", Proceedings 6th Internationl Conference on Software Engineering, Tokyo, 1982.
- [KAOS00] http://www.ingi.ucl.ac.be/research/projects/AVL/ReqEng.html.
- [Kendall88] Kendall K., Kendall J., *Systems Analysis and Design*, Prentice Hall, 1988.
- [Mylopoulos92] Mylopoulos J., Chung L., Nixon B., "Representing and using non-functional requirements: A process-oriented approach", IEEE Transactions on Software Engineering, 18(6), 1992.

References (cont'd)

- [Ross77] Ross D., "Structured Analysis: A Language for Communicating Ideas," *IEEE Transactions on Software Engineering 3(1)*, Special Issue on Requirements Analysis, January 1977, 16-34.
- [vanLamsweerde98] van Lamsweerde, A., Darimont, R., Letier, E., "Managing Conflicts in Goal-Driven Requirements Engineering," *IEEE Transactions on Software Engineering*, Special Issue on Managing Inconsistency in Software Development, IEEE, Nov. 1998.
- [vanLamsweerde98a] A. van Lamsweerde, A., Willemet, L., "Inferring Declarative Requirements Specifications from Operational Scenarios," *IEEE Transactions on Software Engineering*, Special Issue on Scenario Management, IEEE, December 1998.
- [vanLamsweerde98b] A. van Lamsweerde, A., Letier, L., "Integrating Obstacles in Goal-Driven Requirements Engineering," Proceedings ICSE'98 - 20th International Conference on Software Engineering, IEEE-ACM, Kyoto, April 98.
- [vanLamsweerde98c] Feather M., Fickas S., van Lamsweerde A., Ponsard C., "Reconciling System Requirements and Runtime Behaviour", 9th International Workshop on Software Specification and Design (IWSSD'98) IEEE, Isobe, Japan, April 1998.
- [Yu93] Yu E., "Modeling organizations for information systems requirements engineering", IEEE International Symposium on Requirements Engineering, 34-41, San Diego, 1993.