



# **KAOS**

**Tokens, Classes and Metaclasses  
Entities and Relationships  
Actions and Time  
Agents, Goals and Constraints  
The KAOS Methodology**



## **Goal-Directed Requirements Acquisition (KAOS)**

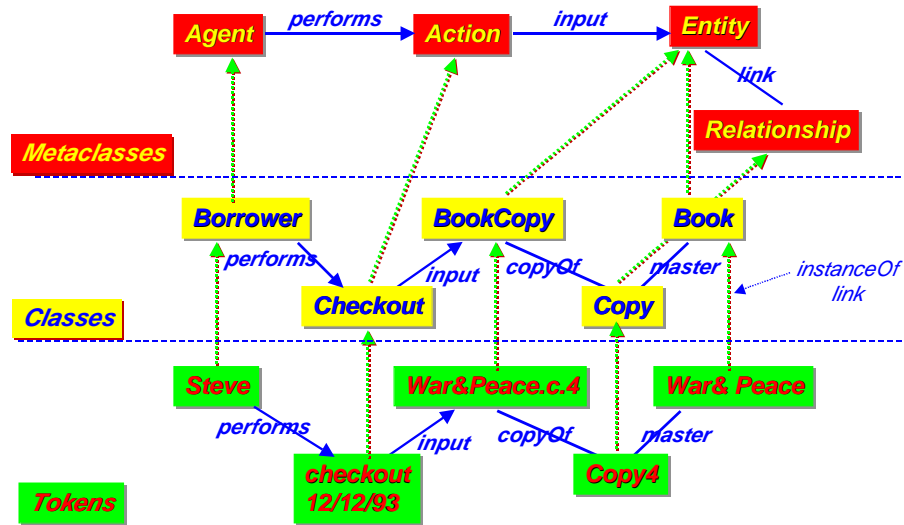
- (Organizational) goals lead to requirements.
- Goals justify and explain requirements which are not necessarily comprehensible by stakeholders.
- Goals can be used to assign responsibilities to agents so that prescribed constraints can be met.
- Goals provide basic information for detecting and resolving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

[Dardenne93]





# The Meta, Domain and Token Level



# 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 BorrowedBy, Card 0::1]

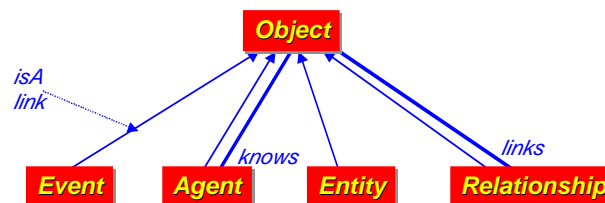
**Invariant** ( ∃ lib: Library, bor: Borrower, b: Book, bc: BookCopy )  
 [Borrowing(bor, bc) ∧ bc ∈ lib.collection ⇒  
 bc ∈ lib.checkedOut

∧ ◀ Requesting(bor, b) ∧ Copy(bc, b)]

...  
**end** Borrowing



## Entities and Relationships at the Metalevel



*Object* has meta-attributes *name*, *informalDef*, also:  
*exists* -- takes values *true*, *false* at the instance level,  
*invariant* -- its values are assertions at the domain level



## More Entities and Relationships

### Entity Meeting

**Has** when: *TimeInterval*, where: *Location*, feasible, scheduled: *Boolean*

**Invariant** ( $\forall m: \text{Meeting}$ ) ( $\text{feasible}(m) \Leftrightarrow \dots \wedge$   
 $\text{scheduled}(m) \Leftrightarrow \text{when} \neq \text{none} \wedge \text{where} \neq \text{none}$ )

**end Meeting**

### Relationship Requesting

**Links** Initiator [**Role** *Requests*, Card 0::N]

Meeting [**Role** *RequestedBy*, Card 1::N]

**Has** *dateRange*: *TimeInterval*, *participantList*: *setOf[Participant]*

**Invariant** ( $\forall m: \text{Meeting}, p: \text{Participant}$ )  
 $(\text{Requesting}(m,i) \wedge m.\text{when} = t \wedge p \in \text{Requesting}(m,i).\text{participantList}$   
 $\Rightarrow \text{available}(p, t)$

...

**end Requesting**



## Events

- An event is an instantaneous object.
- $\text{Occurs}(e) == e.\text{exists} = \text{true}$
- Here is an example of an event:

**Event** ReminderIssued

**Has** ToWhom: Bor, What: BookCopy, Message: text;

**Invariant** ( $\forall rs: \text{ReminderIssued}$ )

$(\text{Occurs}(rs) \Leftrightarrow (\exists p: \text{Staff})(\text{Performs}(p, \text{IssueReminder}))$

...

**end** ReminderIssued

- Events have a frequency meta-attribute which indicates how frequently they occur; its value is a time interval.



## Actions

**Action** CheckOut

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

**Output** Library [**Res**: lib], Borrowing

**Precondition**  $bc \in \text{lib.available}$

**Postcondition**  $\neg(bc \in \text{lib.available}) \wedge bc \in \text{lib.checkedOut}$   
 $\wedge \text{Borrowing}(\text{bor}, bc)$

...

**Action** IssueReminder

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

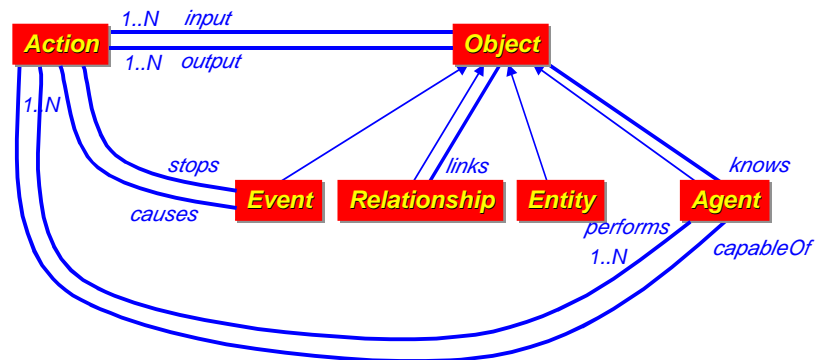
**Output** Reminder

**Triggercondition**

$\ll >2\text{wks} \text{Borrowing}(\text{bor}, bc) \wedge \neg(\ll \leq 1\text{wk} \exists r: \text{IssueReminder Occurs}(r))$   
 /\* bc has been borrowed for at least 2wks and there hasn't been a  
 reminder within the last week \*/



## The Action Metamodel



## More on Actions

- The meta-attributes of Action include:
  - Precondition -- must be true before execution;
  - Postcondition -- will be true after execution;
  - TriggerCondition -- triggers an execution of the action
- In addition, actions participate in four meta-relationships:
  - Input(*act*, *obj*) -- *obj* is in the domain of *act*;
  - Output(*act*, *obj*) -- *obj* is in the codomain of *act*;
  - Causes(*ev*, *act*) -- *ev* causes *act*;
  - Stops(*ev*, *act*) -- *ev* stops *act*;
- Modify and Inspect are two specializations of Action. Modify changes the state of one or more objects, while Inspect does not.



## Representing Time in KAOS

$\bigcirc\phi$	- $\phi$ is true in the next state
$\bullet\phi$	- $\phi$ is true in the previous state
$\blacktriangleright\leq x\phi$	- $\phi$ will be true sometime (within x)
$\blacktriangleleft\leq x\phi$	- $\phi$ was true sometime (within x)
$\blacksquare\leq x\phi$	- $\phi$ will be always true (after some time)
$\blacksquare\leq x\phi$	- $\phi$ was always true (until some time)
$\phi \cup \psi$	- $\phi$ is true until $\psi$ becomes true
$\phi \text{ S } \psi$	- $\phi$ has been true since $\psi$ became true

### Notation:

circle - previous/next state  
 star - sometime in the past/future  
 square - always in the past/future



## Agents

### Agent Staff

**Has** competenceAreas: setof[Competence]

**Invariant** ( $\forall st:Staff$ )

$InstanceOf(st, LibrarianStaff) \vee InstanceOf(st, ClerkStaff)$

**Load** .... /\* describes the agent's work load \*/

**CapableOf** CheckIn, CheckOut, IssueReminder, Reference, Cataloguing

**Performs** assigned: Action

**Knows** Borrowing [Interface: BorrowingSheet]

- 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  $\Rightarrow$  ( $\exists bc:$   
BookCopy) (Copy(bc, b)  $\wedge$  Borrowing(bor, bc)))

**ReducedTo** EnoughCopies, RegularAvailability, AvailabilityNotified

**ReducedTo** AsManyCopiesAsNeeded

**SystemGoal** Maintain[SafeTransportation]

**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.
- The pattern of a goal identifies what is to be done with the goal; five patterns are allowed:
  - **Achieve** -- achieve the goal at some point in the future  
 $P \Rightarrow \blacktriangleright Q$
  - **Cease** -- undo a goal at some point in the future  
 $P \Rightarrow \blacktriangleright \neg Q$
  - **Maintain** -- maintain a goal for some time  
 $P \Rightarrow \square Q$
  - **Prevent** -- prevent a goal from becoming true  
 $P \Rightarrow \square \neg Q$
  - **Optimize** -- maximize or minimize some measure  
max(fcn) or min(fcn)



## Goal Categories

Goals also have associated categories (these are specializations of the `Goal` metaclass), such as:

- **SatisfactionGoal** -- satisfying agent requests
- **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 heuristics for decomposition and operationalization (satisfaction).



## Subgoals

**SystemGoal** Maintain[RegularAvailability]

**InstanceOf** SatisfactionGoal

**Concerns** Library

**Definition** ( $\forall bor: Borrower, b: Book, bc: BookCopy, lib: Library$ )

$(bc \in lib \Rightarrow \Box[\neg(bc \in lib.available) \Rightarrow (\blacktriangleright_{\leq Nwks} bc \in lib.available)])$

/\* N is a parameter \*/

**SystemGoal** Achieve[AvailabilityNotified]

**InstanceOf** InformationGoal

**Concerns** Borrower, Library

**Definition** ( $\forall bor: Borrower, b: Book, bc: BookCopy, lib: Library$ )

$(Requesting(bor, b) \wedge \bullet(\neg\exists bc: BookCopy)(Copy(bc, b) \wedge bc \in lib.available) \wedge (\exists bc: BookCopy)(Copy(bc, b) \wedge bc \in lib.available))$

$\Rightarrow \blacktriangleright Knows(bor, lib.available)$

/\* 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** ( $\forall \text{bor: Borrower, b: Book, bc: BookCopy}$ )

$\Box[\text{Borrowing}(\text{bor}, \text{bc}) \wedge \text{Copy}(\text{bc}, \text{b}) \wedge \text{ONeed}(\text{bor}, \text{b})$   
 $\Rightarrow \text{OBorrowing}(\text{bor}, \text{bc})]$

*/\* 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. Constraints may be **hard** or **soft**.

**SoftConstraint** Maintain[LimitedBorrowingPeriod]

**Definition** ( $\forall \text{bor: Borrower, bc: BookCopy}$ )

$(\text{Borrowing}(\text{bor}, \text{bc}) \Rightarrow \neg \neg \text{Borrowing}(\text{bor}, \text{bc}))$

- Constraints operationalize goals

**SystemGoal** Maintain[RegularAvailability]

**Concerns** ...

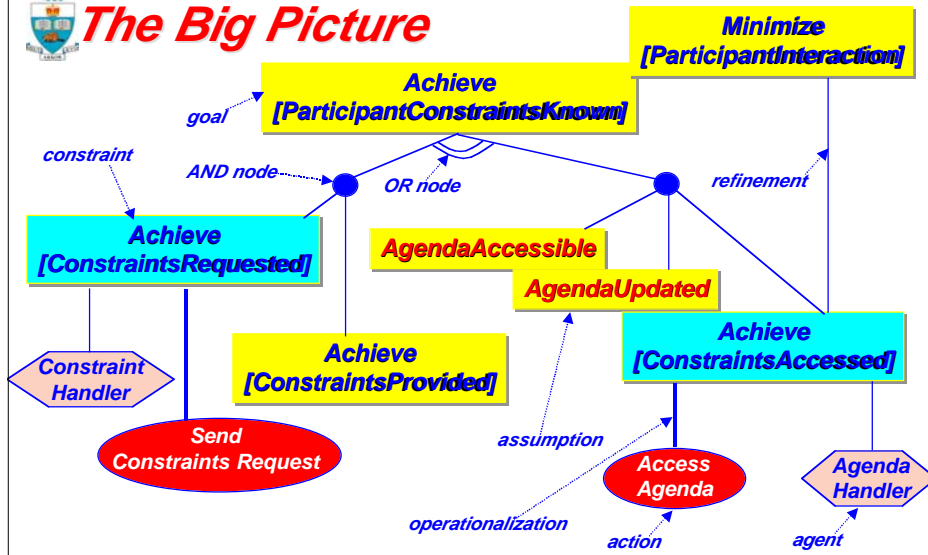
**Definition** ...

**OperationalizedBy** LimitedBorrowingPeriod, NoLostCopies,...

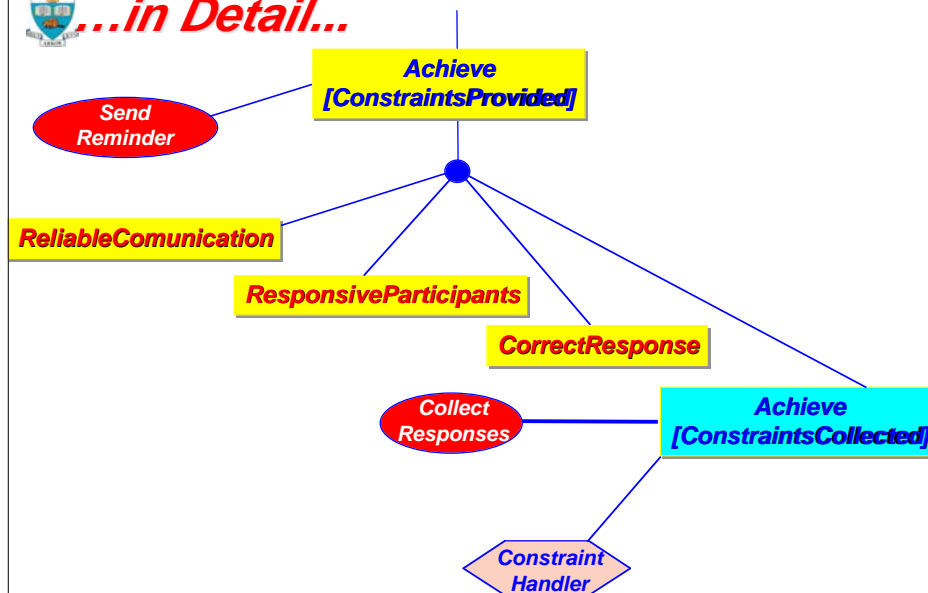
- Constraints are **ensured** by restricting existing actions and objects (through strengthened preconditions, invariants, etc.) or through the introduction of new actions and objects.



# The Big Picture



# ...in Detail...





## ***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
- Use 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 Acquisition 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 to someone doing organizational design, rather than software development!*



## References

- [Dardenne93] Dardenne, A., van Lamsweerde, A. and Fickas, S., "Goal-Directed Requirements Acquisition", in *The Science of Computer Programming 20*, 1993.
- [KAOS00] <http://www.ingi.ucl.ac.be/research/projects/AVL/ReqEng.html>.
- [vanLamsweerde98] A. 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, November 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," *Proceedings IWSSD'98 - Ninth International Workshop on Software Specification and Design*, IEEE, Isobe, Japan, April 1998.