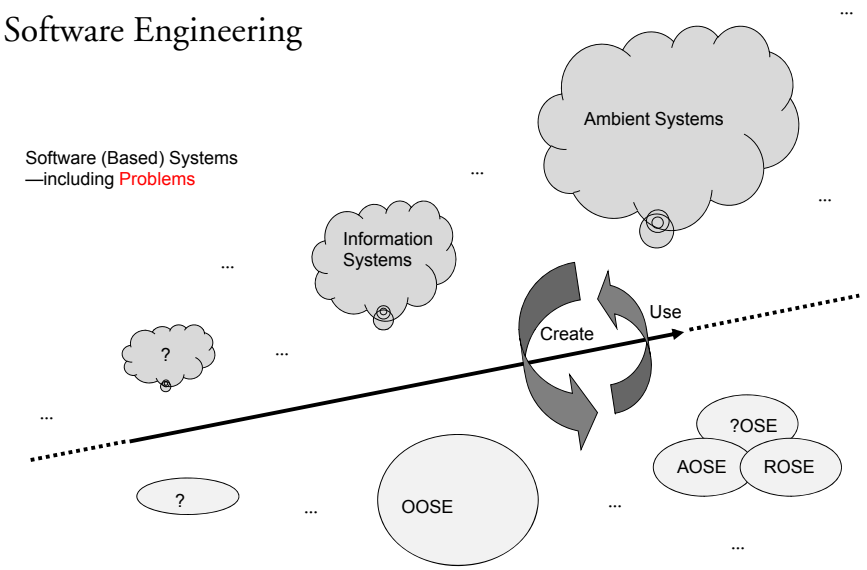
SSE2-PLDE_7

Contents:
- Conceptual Modeling & Programming

Literature:
- Kristensen.
  Associative Programming and Modeling: Abstractions over Collaboration.

Software Engineering



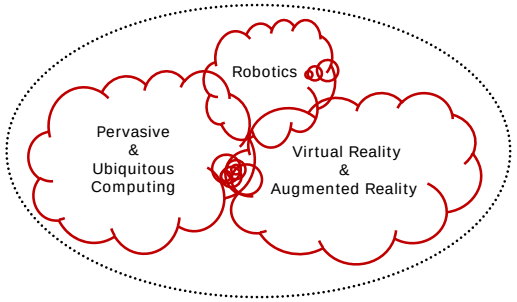
Software Engineering
—including Languages

AOSE ≡ Agent Oriented Software Engineering
ROSE ≡ Robot Oriented Software Engineering

History: Characterization …

	Tools	Information Systems	Ambient Systems
Paradigm	• Procedural/functional • Advanced, abstract computer	• Object-oriented • Domains, concepts/phenomena	• ?-oriented • System with life—autonomous, intelligent
Ingredients	• Procedure/function operations and data structures	• GUI, database, client/server • Architecture,	• People and agents collaborating—living in systems
Usage	• Use of a calculator system • Input and output, results	• Users of an information system • User, customer, developer	• "Effect" domain • Who/what is affected • Support, surveillance, assistance
Context	• Tool among other tools • Workbench	• Application domain • How the system is used	• Devices, locations, time, agents, …
Methodology	• Flowcharts • Structured programming	• UP/UML, OOAD, …	• ?

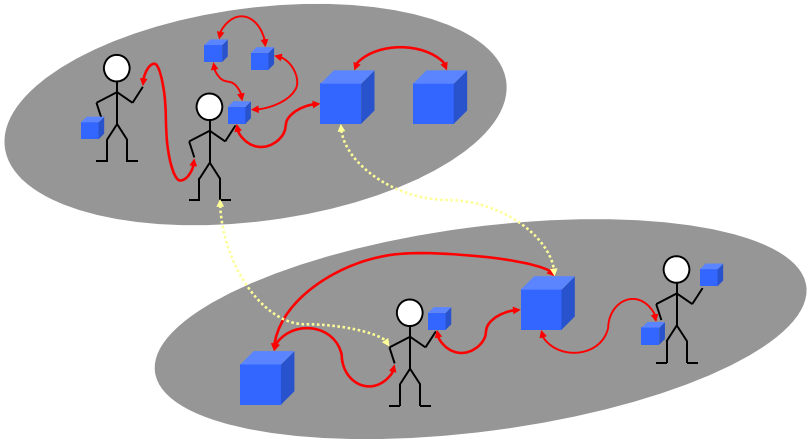
Ambient Systems



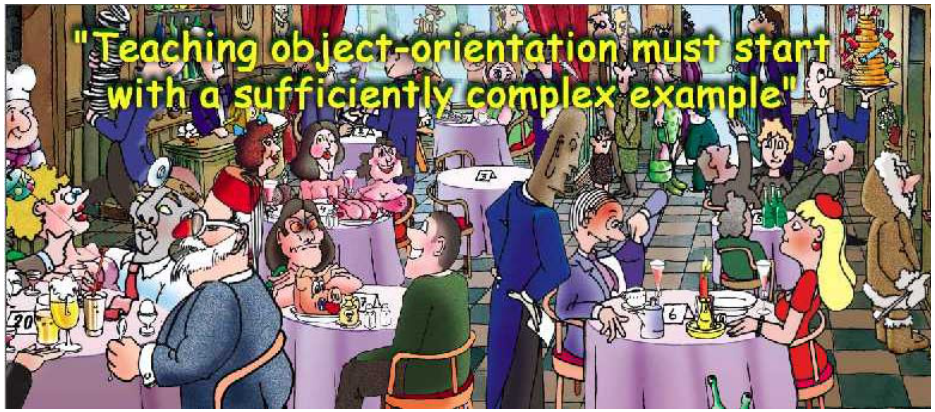
- … a natural consequence of the **evolution** of software-based computer systems involving additional equipment and also location and time
- … based on **ubiquitous** behavior and implemented through techniques from **pervasive** computing
- … During interaction between users only various devices will be **explicitly** present, but the system will be **implicitly** present only
- … users participate **simultaneously** in several such probably **overlapping** cases
- Users **"live"** in these cases over time, changing between being passive and active for one or more case
- Several users play **roles** … in order to participate in ongoing **interactions**

Ambient Systems

- Involve agents/users, data, locations, time, …
- Are organized dynamically, simultaneously, …
- May be based on the notion of activities …

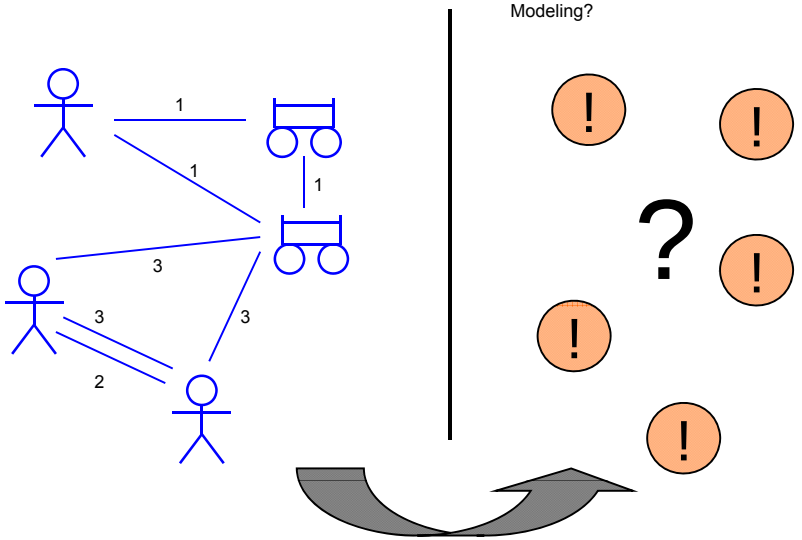


Conceptual Programming

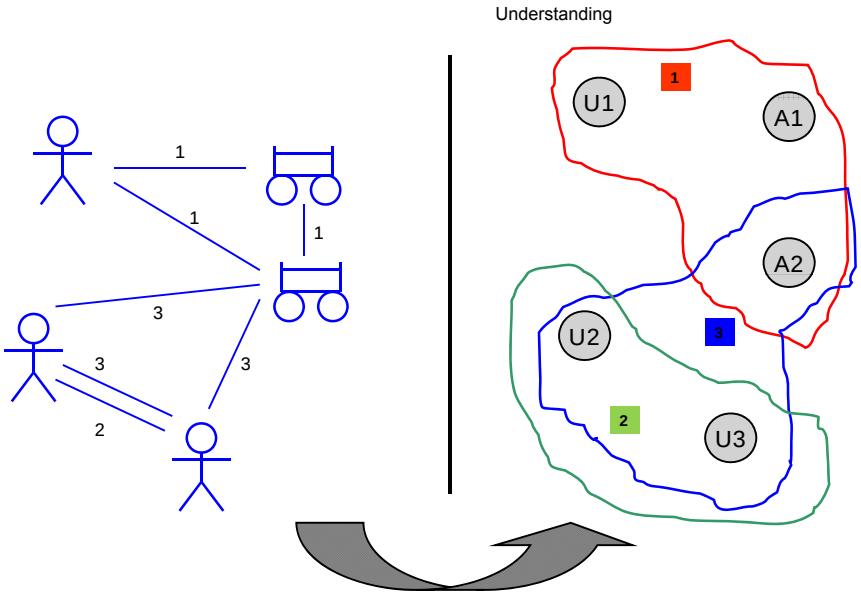


What goes on?

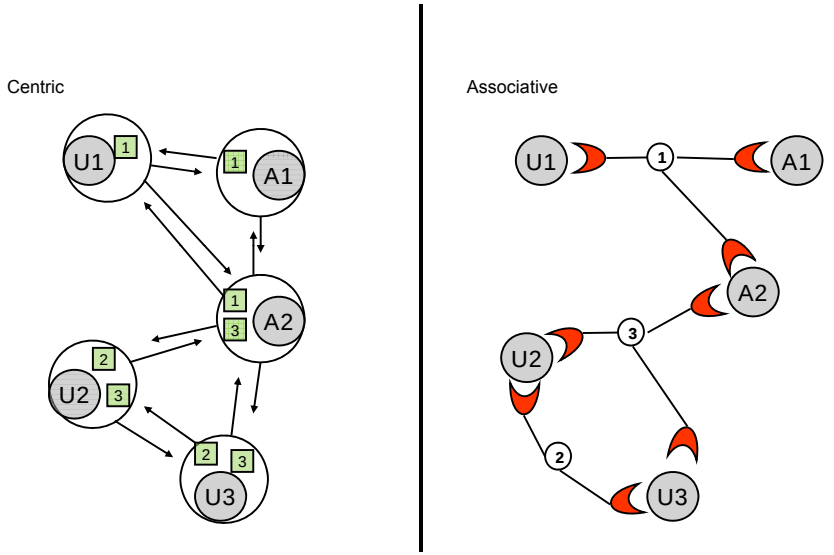
Collaboration



Collaboration



Collaboration



Conceptual Programming

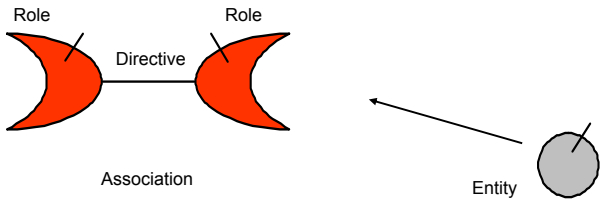
- Design of programming languages is based on human conceptualization in a general sense:
  - **Alternative kinds of concepts** and selected ingredients of these concepts are included in programming languages
  - Object orientation is seen as a specialized use of this approach, where the focus mainly is on **"things"** and their modeling in terms of classes and objects

- The intention is that certain additional kinds of general (but not application area specific) concepts may enrich programming languages:
  - The **purpose** is to limit the gap between understanding, designing and programming also in order to reduce the amount of software
  - The **advantage** of the approach is that because humans already use various alternative kinds of concepts, the modeling process is efficient and the model becomes understandable
  - The **challenge** is that any given potential kind of concept had to be understood and interpreted, and did not immediately comply with the typical understanding of programming languages
  - Each candidate concept should therefore be adjusted to fit with and slightly modify the expectations and possibilities at the programming level including implementation techniques.

Associations & (Autonomous) Entities

Associative programming:

- Associations are abstractions over collaboration
- Both modeling and programming
- Roles and directive



Association = Activity + Roles

Dynamic creation and deletion of instances of associations:

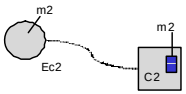
1: No associations exist for Ec2 of class C2

2: An association instance Ax with roles with properties n1 and n2 is created:

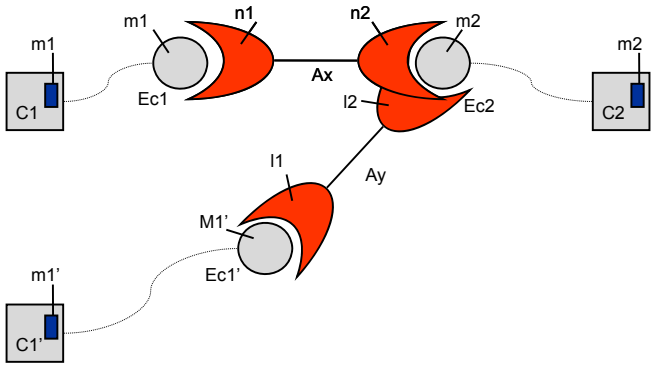
- Ec2 is associated by means of Ax with entity Ec1 of class C1
- Entity Ec2 of class C2 function as one participant in the association Ax

3: The association Ax no longer exists

4: Ec2 is associated by means of an association instance Ay with roles with properties l1 and l2 with entity Ec1' of class C1'



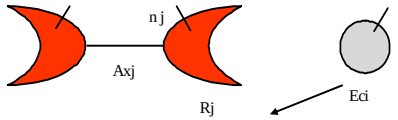
Entities with Associations



Programming

```
association Xj {

  role Rj for Ci {
    method nj (...)   ...
  }
  ...
  directive {
    ... Rj::nj(...) ...
  }
}

entity Axj of Xj
```

```
class Ci {

  action_part {...}

}

entity Eci of Ci

Eci enters Axj as Rj
```



Programming: Interleaved Execution

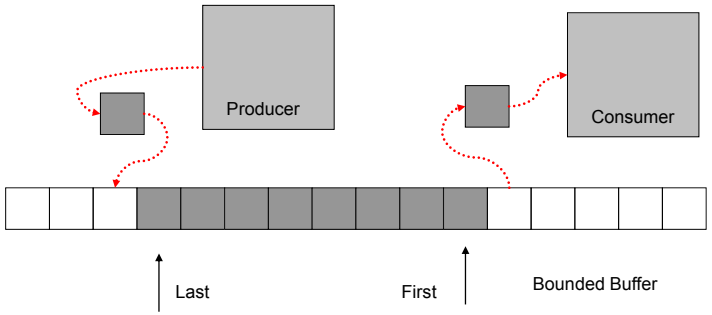
```
class Ci {
  ...
  action_part {... mi(...) ...}
}

entity Eci of Ci

Eci enters Axj as Rj


association Xj {
  role Rj for Ci {...}
  ...
  directive {... Rj::nj(...) ...}
}

entity Axj of Xj
```

Assumption:

1. A given Eci of Ci is engaged as role Rj of Xj in a collection of association instances Axj of Ax
2. the next individual action for Eci is mi(…)
3. for the collection of Ax's the next action to be executed for Axj with Eci in role Rj is Rj::nj(…)

Interleaved execution of Eci means, that exactly one out of mi and the collection of nj's is selected randomly and executed.

Example: Bounded Buffer



- `Producer` produces artifacts and `Consumer` consumes artifacts—concurrently
- Production times and consummation times are not related
- `Producer` delivers each artifact produced to `Bounded_Buffer`.
- `Consumer` retrieves each artifact for consummation from `Bounded_Buffer`
- `Producer` and `Bounded_Buffer` are synchronized during the transfer of an artifact. `Consumer` and `Bounded_Buffer` are synchronized during the transfer of an artifact
- `Bounded_Buffer` is bounded, i.e. a maximum number of elements may be kept in the buffer. If `Bounded_Buffer` is full no more elements may be added to the buffer and `Producer` has to wait for `Bounded_Buffer` not to be full. If `Bounded_Buffer` is empty no elements can be retrieved from the buffer and `Consumer` has to wait for `Bounded_Buffer` not to be empty.

Rendezvous in CSP …

By an input command in `Receiver`
`S?x`
the `Receiver` accepts an assignment to its target variable `x` from a `Sender S`

By an output command in `Sender`
`R!y`
the `Sender` requests an assignment from its expression `y` to a `Receiver R`

If the input and the output commands *correspond*, these "are executed simultaneously, and their combined effect is to assign the value of the expression of the output command to the target variable of the input command":

`y` is evaluated in `Sender` and assigned to `x` in `Receiver`

R

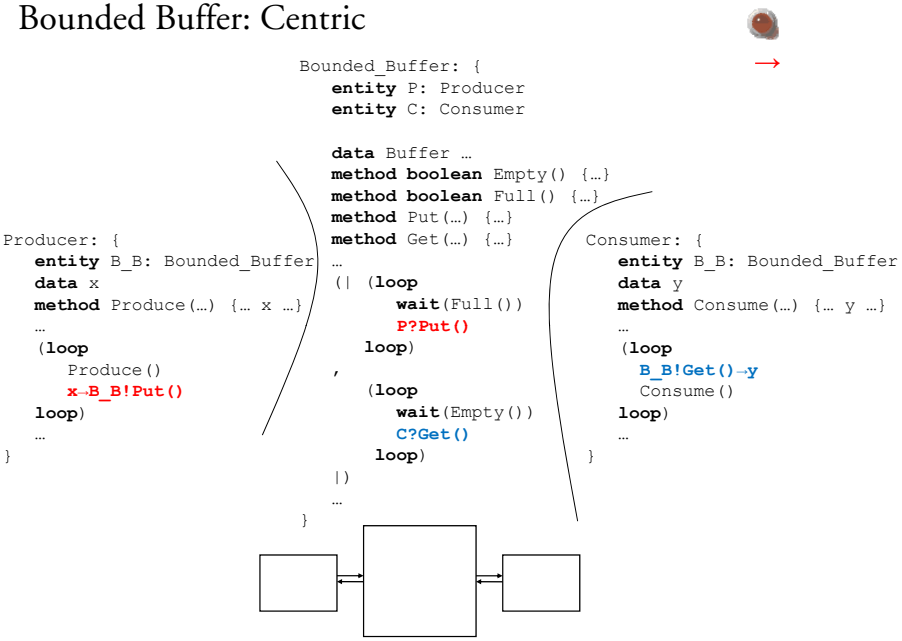
```
x: …

…
S?x
…
```

S

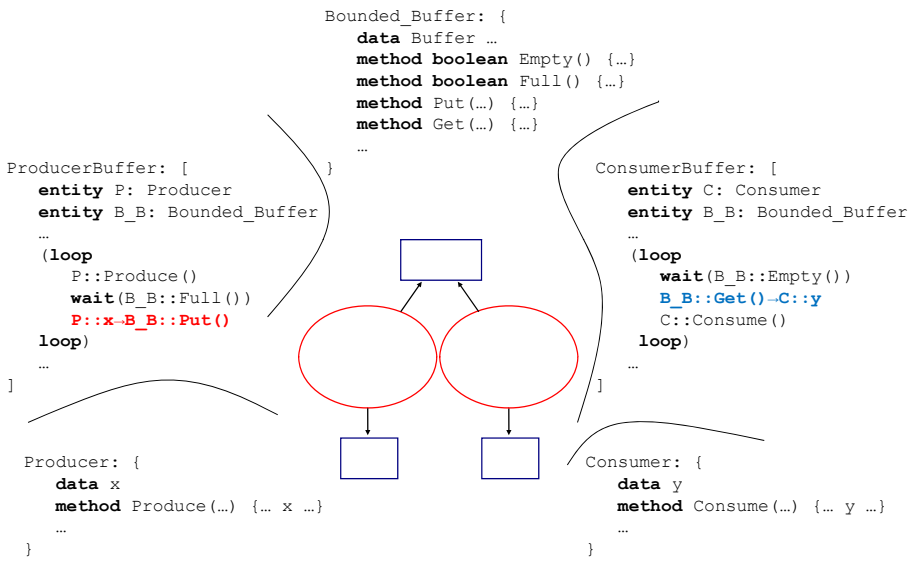
```
y: …

…
R!y
…
```

[Hoare, C. A. R.: *Communicating Sequential Processes*, Communications of the ACM, vol. 21, 1978]

Bounded Buffer: Centric



Bounded Buffer: Associative



Bounded Buffer: Figures