Modularity

**Collaboration Modules**

Team classes – modules larger than classes

Teams combine certain properties from classes (SOP) and packages (define code organization). As classes (scripts) are separable, user applications and libraries are shared separately.

Role classes – relative entities

Each role interface lives in a class and is an example of a polymorphic relationship (see below).

Team Inheritance

Applying inheritance to compound modules (thread) only results in separate classes. Complex inheritance, e.g., a passenger may play the role of the FlightBooking system.

Long-term maintenance and evolution

The relationships between the above problems could be handled. If a software system is supposed to be used over a long period of time, it is required to prevent development costs.

Role Instantiation

The implementation of the sub-team. So within the team, there is an implicit adjustment to consistently connect roles per needs.

Adaptability

A role class can be bound to a base class via the role, too.

Role Binding

Making base method/field accessible per role.

Dynamic filtering

Avoid weaving extra code.

Flattish-Bonus Example

The example shows two requirements encapsulated in two completely independent modules.

- **Flight**
  - Class
  - Contains properties of the flight
  - Provides methods for handling the flight

- **Bonus**
  - Class
  - Contains properties of the bonus
  - Provides methods for handling the bonus

Team Instantiation

The implementation of the team, theBonus, is bound to the role of the flightBooking system.

Sustainability of Software Engineering Requires

**Object Teams provide**

- Adaptability
- Code reusability
- Scalability
- Modularity

**Limitations of Object-Oriented Programming**

- **Scale**
  - When object-oriented programming works well in the small, creating a system from 100s of classes does not sufficiently support a good modular structure.
  - Hierarchical structure is not clustered towards the goal of scalable modules.

- **Complexity**
  - Any single change in the domain logic does not require the modification of any other parts of the system.
  - Complexity increases in crosscutting concerns.

- **Adaptability**
  - Existing objects can be extended by crosscutting concerns.
  - Facade patterns allow to refactor modules and adapt them.

- **Modularity**
  - All concerns should be encapsulated in modules with well-defined boundaries.

- **Scalability**
  - Any system is scalable if you add a wrapper around it.

**Object Teams**

ObjectTeams/Java extends the Java programming language for role-based aspect-oriented software development to support the requirements for sustainable software engineering.

Object Teams/Java

- **Modularity**
  - All concerns should be encapsulated in modules with well-defined boundaries.
  - As packages teams support nesting which is optionally realized by a hierarchy of directories and files.

- **Adaptability**
  - Existing objects can be extended by crosscutting concerns.

- **Scalability**
  - Any system is scalable if you add a wrapper around it.

- **Long-term maintenance and evolution**
  - The relationships between the above problems could be handled.

Tools

- **Development Tools**
  - Team Modeler
  - Team Monitor

- **Runtime Environment**
  - OTRE

- **Language definitions**
  - OTML

- **Integration**
  - JDT

- **Softwares**
  - IDE for developers

- **Components**
  - Case studies (GEBIT Solutions)
  - Students (diploma theses)

Scalability

Programming language features allow the following differentiations:

- **Modeling**
- **Language**
- **Decompilation**
- **Layering**
- **Compositionality**

Object Teams Ecosystem

- **Plug-in C**
- **Plug-in B**
- **Plug-in A**

Objects can be accessed as usual, but not as pure objects.

The relationships between the above problems could be handled.

Evolution

As software matures over time, efforts are required to prevent development costs.

Adaptation

The relationships between the above problems could be handled.

Implementation

- **Language**
  - OTML

- **Modeling**
  - OTModeler

- **Language Integration**
  - JDT

- **Language Definition**
  - OTLD

- **Evaluation**
  - Performance

- **Tools**
  - Code contribution

Contact:

Stephan Herrmann (project lead)

Christine Hundt (OTRE, embedded systems)

Marco Mosconi (OT Modeler, components)