

UML-based tool for Constructing Component Systems via Component Behaviour Inheritance

Ella Roubtsova



The Netherlands

Serguei Roubtsov



Finland

ERCIM fellowship



Outline

- Component and Component Specification;
- The need of a tool support for component specifications reuse.
- Tool idea. Process semantics of component specification with process inheritance is the basis of reuse .
- Reuse of behavioral patterns and design debugging .



Component and Component specification

Component specification is an abstraction from component implementation keeping :

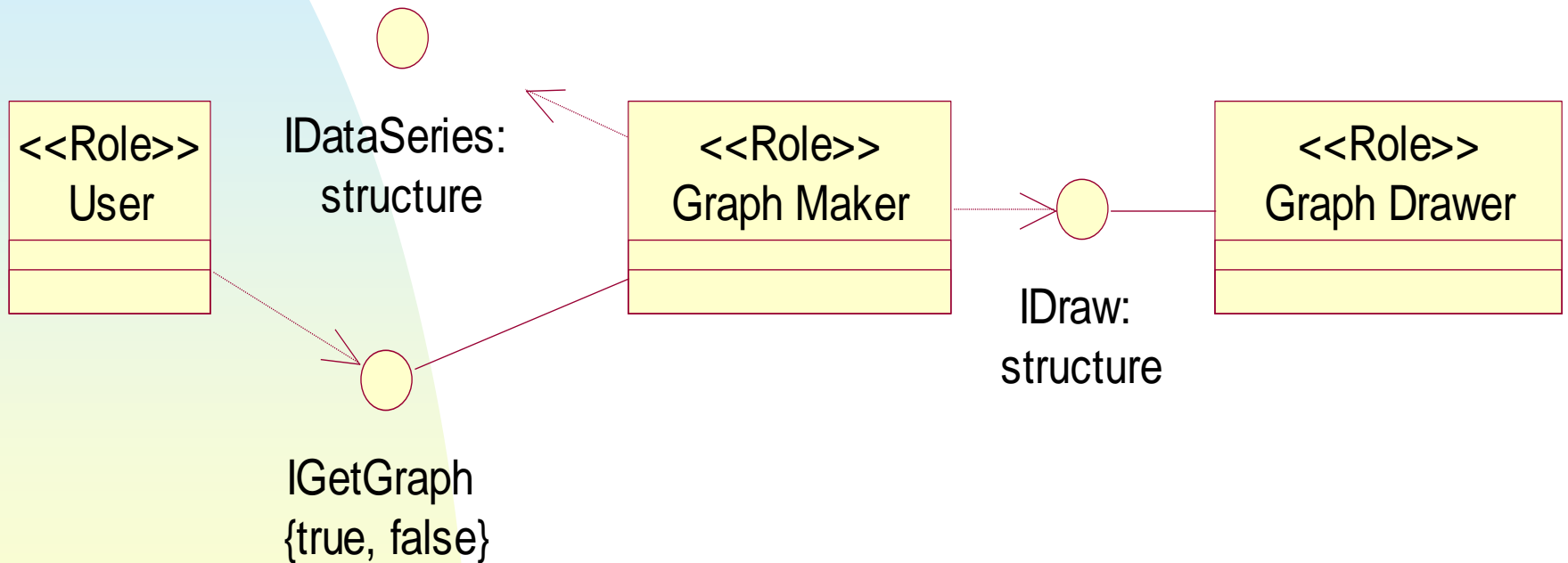
- Contractually specified **interfaces** and **explicit dependencies**
- **Independent** deployment
- **Composition** by third parties



Behavioural pattern

$$P = (A, z, z^*, z_F, T)$$

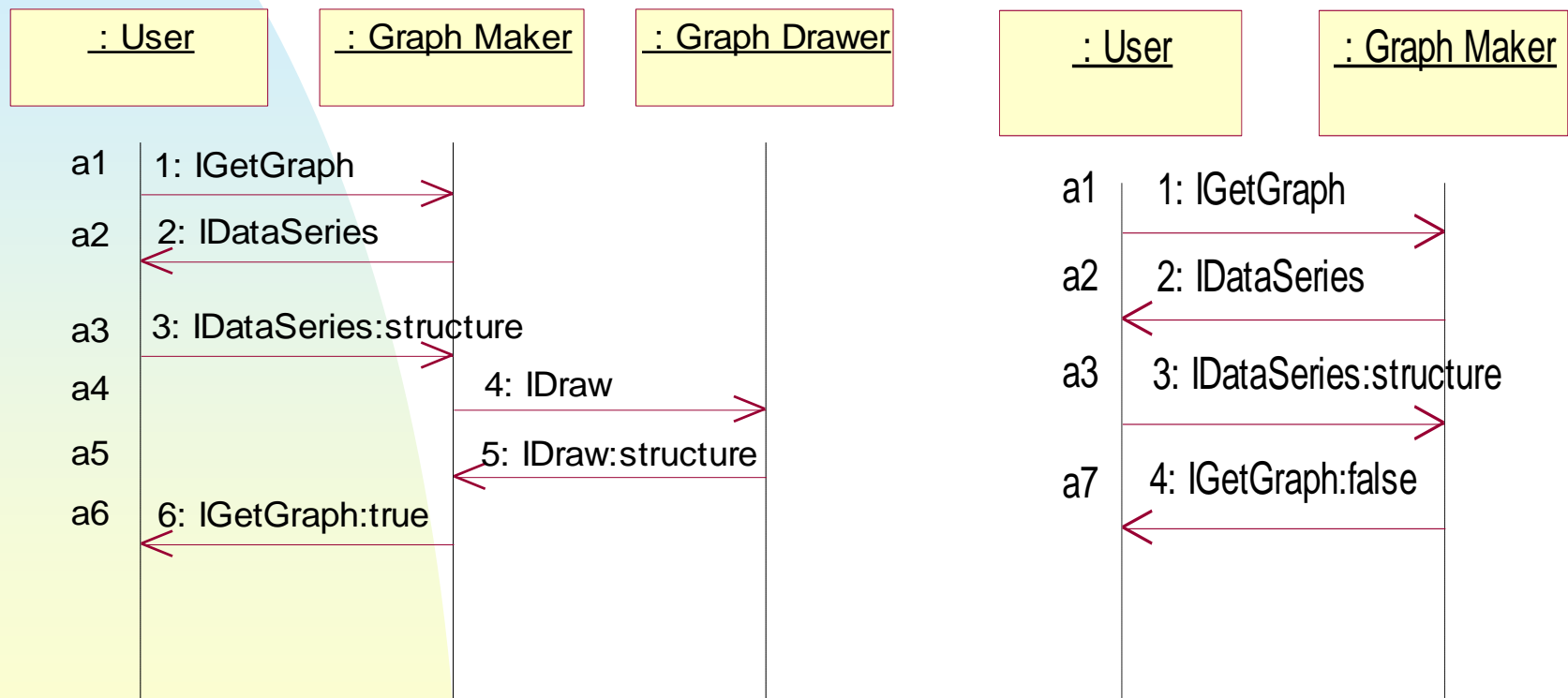
Interface-role diagram



$a = User.GraphMaker.IDataSeries : true$

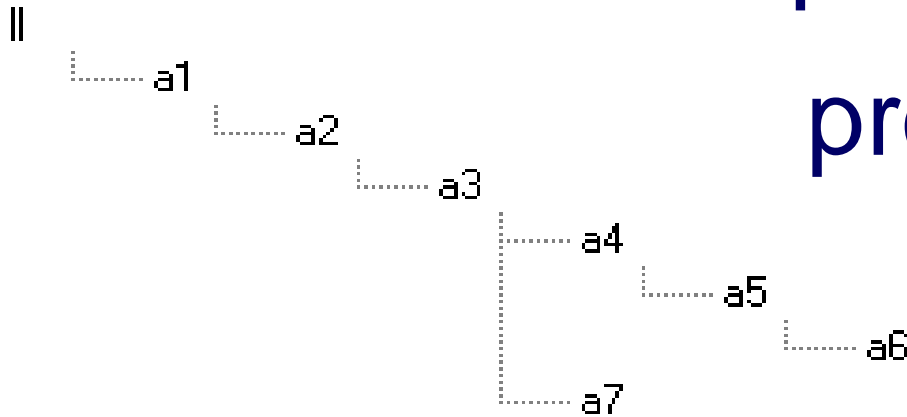


Set of sequence diagrams specifies the structure of the behavioural pattern



$$p = start \cdot a_1 \cdot a_2 \cdot a_3 \cdot (a_4 \cdot a_5 \cdot a_6 + a_7) \cdot final$$

Behavioural pattern =
process term =
process graph



$$p = start \cdot a_1 \cdot a_2 \cdot a_3 \cdot (a_4 \cdot a_5 \cdot a_6 + a_7) \cdot final$$

$a_1 = User. Graph\ Maker. IGetGraph$



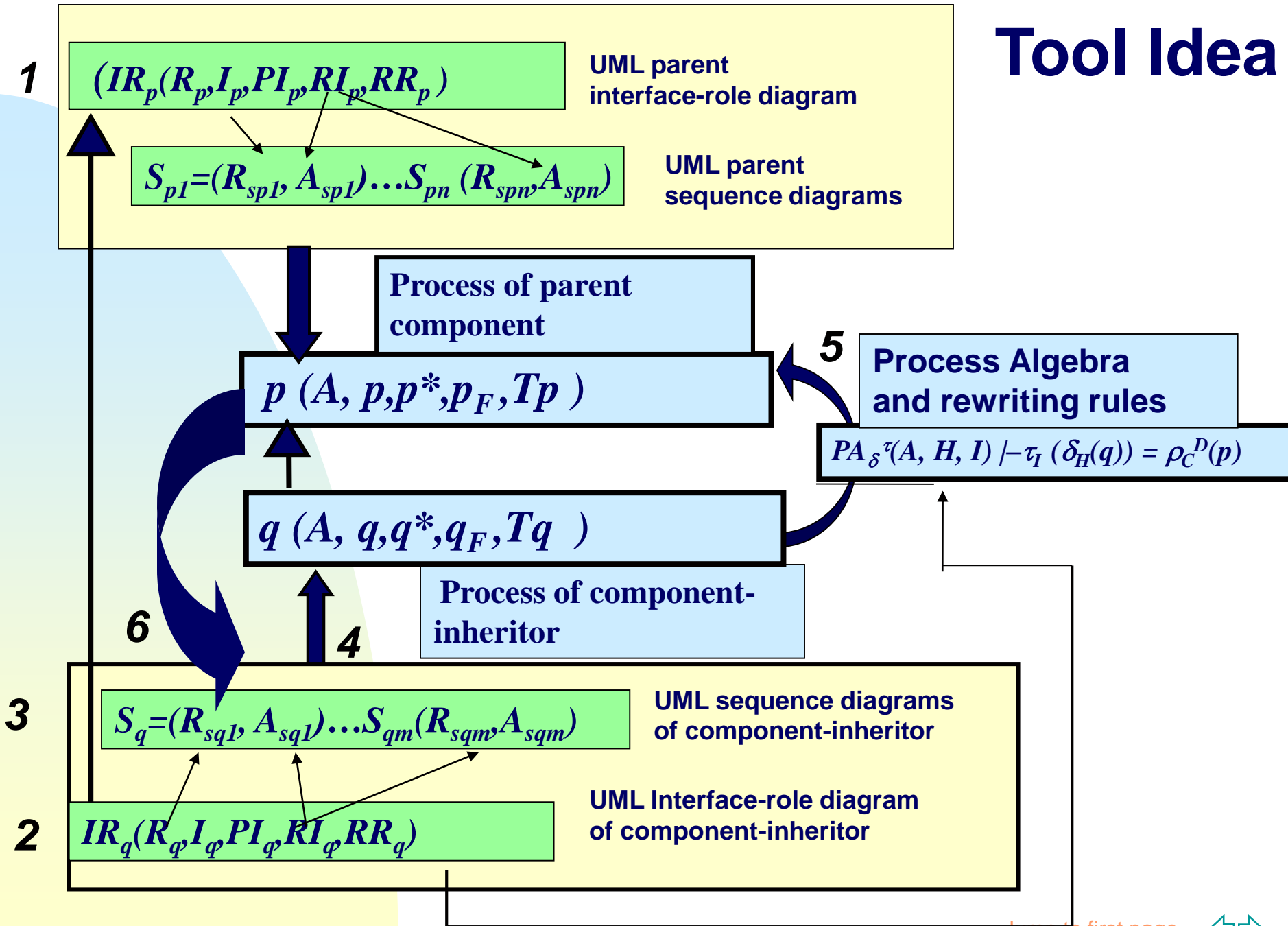
Motivation

Why do we need a UML based tool supporting constructing component systems?

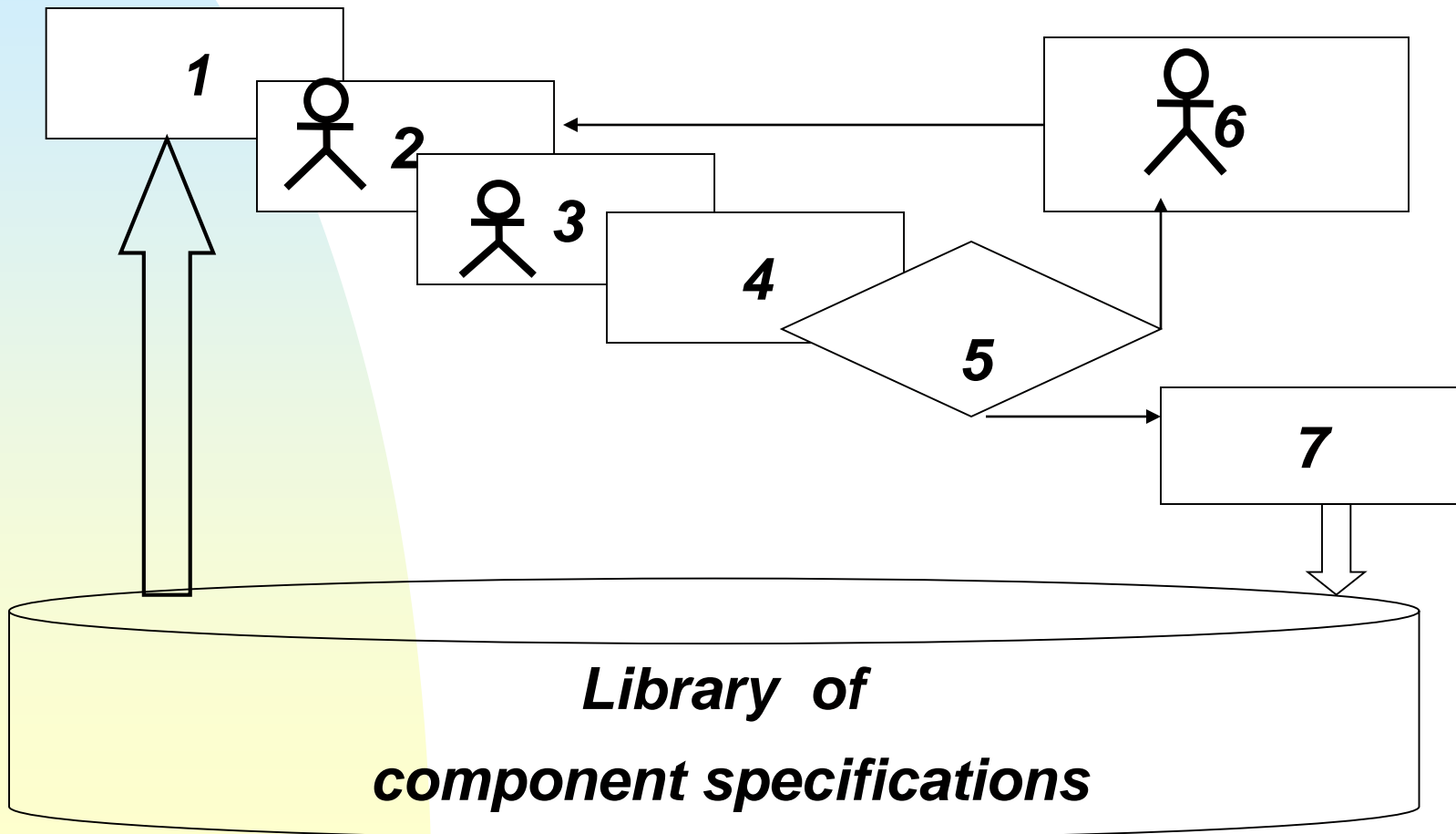
- **Consistent specification of a component is a time consuming task.**
- **Composition of component specifications should not spoil specified behaviour of components .**



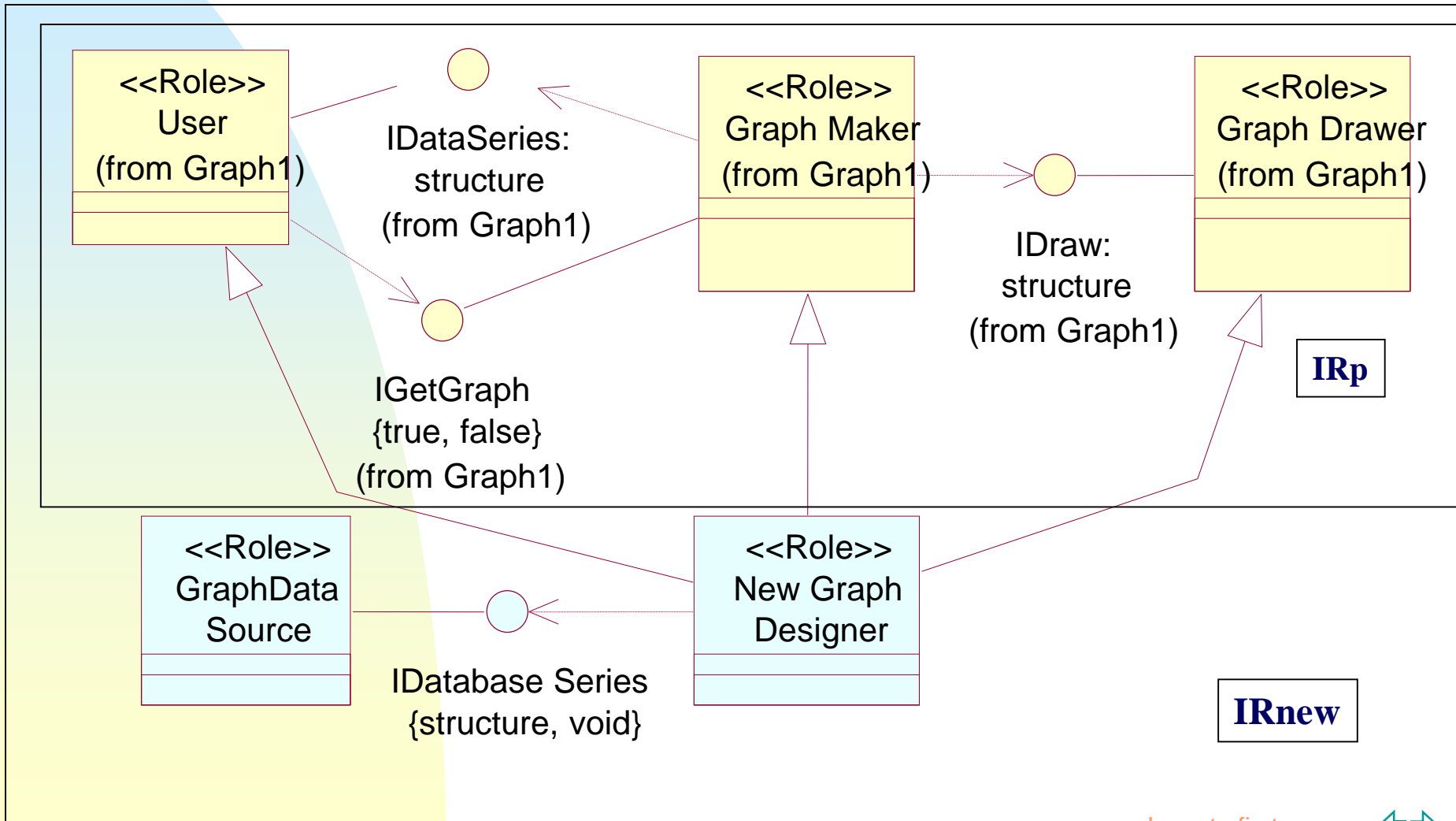
Tool Idea



Constructing component systems as a workflow



Reuse of role-views (1,2)



Rules for Interface-Role Diagram Reuse

1. Create new roles $R_p \cap R_{new} = \emptyset$

2. To inherit an element of communication, inherit the role which provides the interface and the role which requires the interface

$RR_{new} \neq \emptyset$, $RI_q = RI_p \cup RI_{new} \cup RI_d$;

3. Create new interfaces provided and required by new roles: $I_p \cap I_{new} = \emptyset$,



- (untitled)
- Use Case View
- Logical View
- Component View
- Deployment View
- Model Properties

Interface-Role Diagram Reuse (1,2)

Class Diagram: Logical View / Main

Eindhoven University of Technology, VTT Electronics

Component Specification Tool

Version 2.3.1
Copyright E.E. Roubtsova, S.A. Roubtsov

```

    graph LR
      Role1[Role 1] ..> IGet[IGet]
      Role2[Role 2] ..> IGive[IGive]
      IGet ..> IGive
  
```

Component Name:

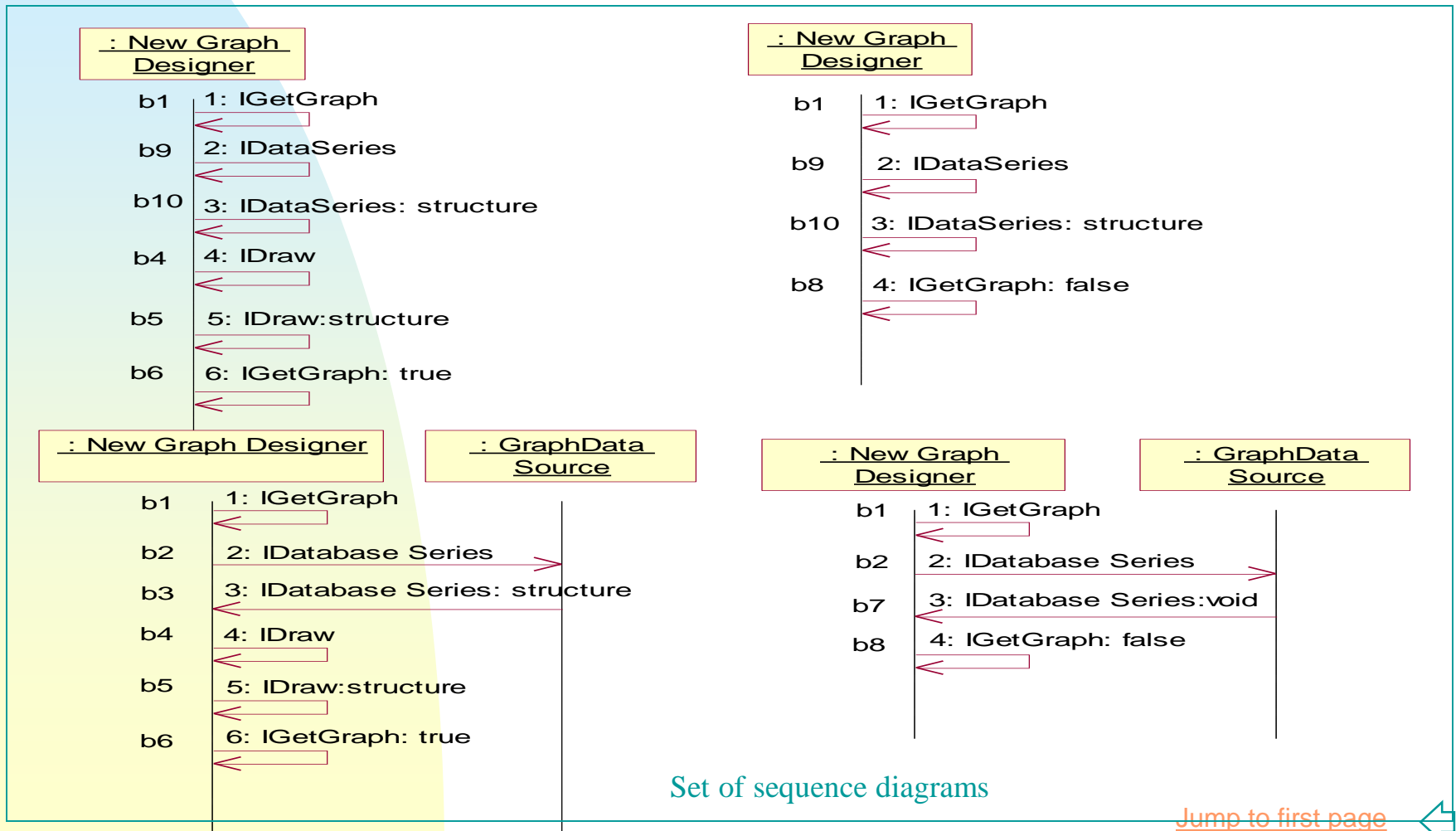
Imported Components:

Description:

Buttons: Import As a Parent..., Use Tool, Exit Tool

Log

Reuse of sequence diagram sets (3)

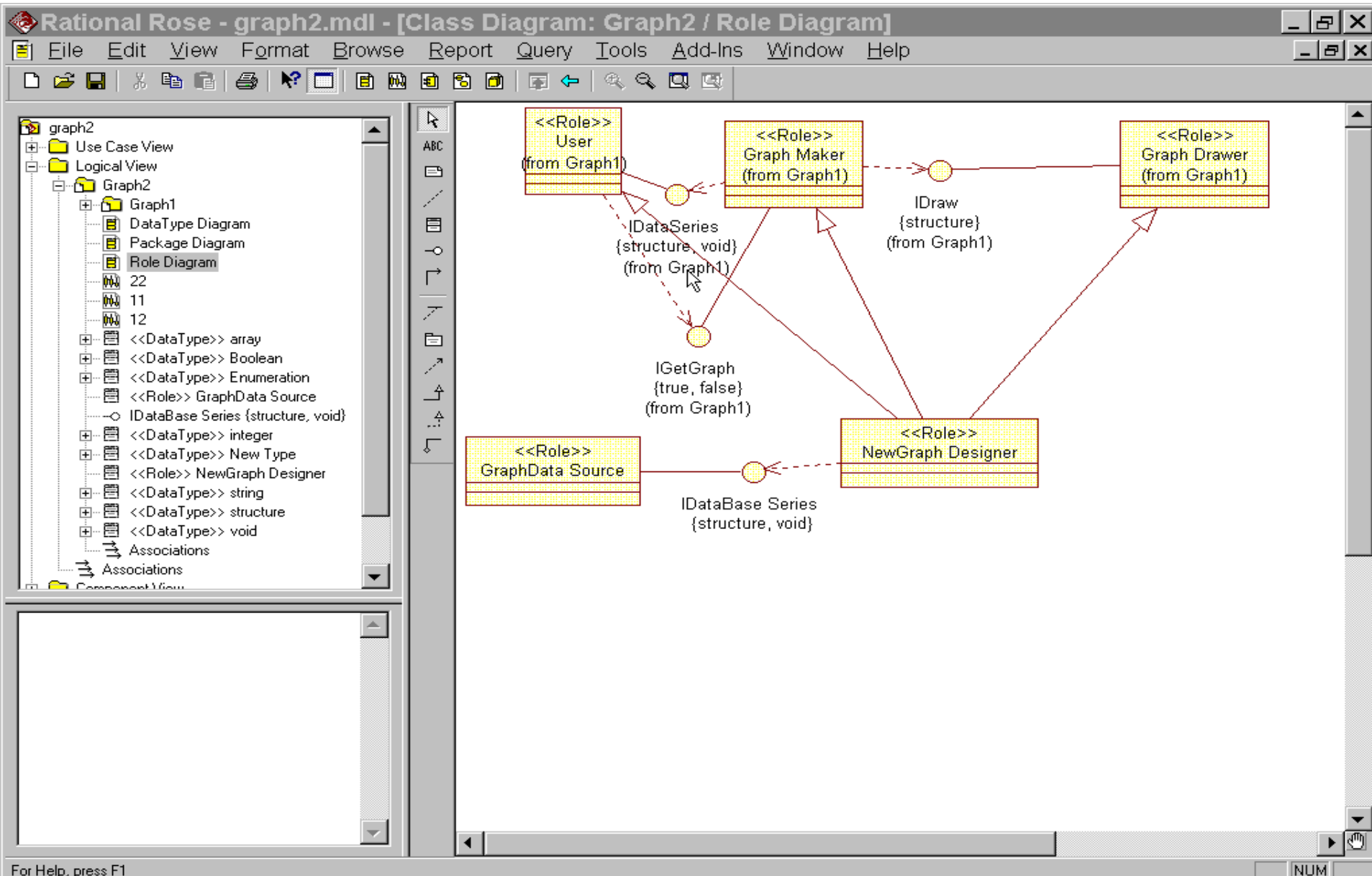


Rules for Sequence Diagram Set Reuse

1. Design a sequence diagram set from actions specified by interface-role diagram ;
2. Tool constructs the process corresponding the sequence diagram set ;
3. Choose the parent from the set of parents;
4. Tool checks the inheritance of the process of this parent;



Design a sequence diagram (3)



Practical use of process inheritance in the UML

$$PA(A, H, I+RN) \dashv\!\! \dashv \tau_I(\delta_H(q)) = \rho_p^d(p)$$

How to find from the interface-role diagram

- the renaming function for the parent process
- the set of all actions A for the process algebra
- the set of actions H to be blocked;
- the set of actions I to be hidden;



Tool support for inheritance checks (4)

The screenshot displays the Rational Rose software interface. The main window shows a class diagram titled "Class Diagram: Graph2 / Role Diagram". The diagram includes the following elements:

- Roles:** `<<Role>> User (from Graph1)`, `<<Role>> Graph Maker (from Graph1)`, `<<Role>> GraphData Source`, and `<<Role>> New Graph Designer`.
- Interfaces:** `IDataSeries: structure (from Graph1)`, `IGetGraph {true, false} (from Graph1)`, and `IDraw: structure (from Graph1)`.
- Relationships:**
 - `GraphData Source` inherits from `User`.
 - `New Graph Designer` inherits from `Graph Maker`.
 - `New Graph Designer` implements `IGetGraph`.
 - `New Graph Designer` implements `IDraw`.
 - `User` is associated with `IDataSeries`.
 - `Graph Maker` is associated with `IDataSeries`.
 - `Graph Maker` is associated with `IDraw`.
 - `GraphData Source` is associated with `IDataSeries`.

The left sidebar shows a project tree for "graph2" with folders for "Use Case View", "Logical View", and "Graph2". Under "Graph2", there are sub-folders for "Graph1", "Data Type Diagram", "Package Diagram", and "Role Diagram". A list of data types and roles is visible below these folders.

At the bottom of the window, there is a "Log" window and a status bar with the text "For Help, press F1" and a "NUM" indicator.

Debugging of Specifications (5)

The screenshot displays the Rational Rose software interface for a project named 'LocalAccess.mdl'. The left-hand side contains a project tree with the following structure:

- LocalAccess
 - Use Case View
 - Logical View
 - Local Access
 - Internet Provider
 - DataType Diagram
 - Package Diagram
 - Role Diagram
 - SeqD1
 - SeqD2
 - <<DataType>> array
 - <<DataType>> Boolean
 - <<DataType>> Enumeration
 - IConnect
 - <<DataType>> integer
 - IPassword1
 - <<DataType>> New Type
 - <<Role>> Sequire Provider
 - <<DataType>> string
 - <<DataType>> structure
 - <<Role>> User
 - <<DataType>> void
 - Associations
 - DataType Diagram
 - Package Diagram

The main workspace shows a 'Class Diagram: Internet Provider / Role...' with the following elements:

- Two role classes: `<<Role>> User` and `<<Role>> Sequire Provider`.
- Two interface classes: `IPassword1` and `IConnect`.
- Associations: `User` is associated with `IPassword1` and `IConnect`. `Sequire Provider` is associated with `IPassword1` and `IConnect`.

The bottom of the window shows a 'Log' window and a status bar with the text 'For Help, press F1' and 'NUM'. The Windows taskbar at the bottom includes the Start button, 'Trondheim', 'Microsoft Po...', 'Rational Ros...', and the system tray with the time '19:51'.

Debugging of Specifications (6)

The screenshot displays the Rational Rose software interface for a project named 'LocalAccess.mdl'. The main window shows a 'Class Diagram: Local Access / Role Diagram'. The diagram features four roles: 'User', 'Employee', 'Administrator', and 'Secure Provider'. 'User' and 'Secure Provider' are generalizations of 'Employee' and 'Administrator' respectively. Two interfaces are shown: 'IConnect {true, false}' and 'IPasswrd1 {recognized, not Recognized}'. 'User' and 'Secure Provider' implement 'IConnect', while 'Employee' and 'Administrator' implement 'IPasswrd1'. A third interface, 'IPasswrd2 {true, false}', is also present but not implemented by any role shown. The left sidebar contains a project tree with folders for 'Use Case View', 'Logical View', and 'Local Access', listing various models and data types. The bottom of the window shows a taskbar with the Start button, open applications like 'Microsoft Po...', 'ASet2', and 'Rational Ros...', and the system clock showing '13:23'.

Conclusion

- Tool does not change the current practice of component specification, it does not demand any specific skills from designers (formal model and methods are built into the tool).
- Advantages and results of the tool methodology:

- Consistent component specifications in the UML;
- Correct reuse of component specifications;
- Decreasing of the design time;

E.Roubtsova@tue.nl

ext-Serguey.Roubtsov@vtt.fi

