Object-Oriented Metamodelling

In The MetaEdit+ MetaCASE Environment

Steven Kelly

Demonstration D5, ECOOP’97, Jyväskylä
June 10th Tue 17:30
June 11th Wed 11:30

Outline

Metamodelling

MetaEdit+

GOPRR

O-O features

Method Engineering

Metamodelling and MetaCASE

Many Information System Development methods

- Structured, Object-Oriented, BPR

Not all methods can be supported by CASE tools

- Too many methods, evolving too fast

Separate CASE tool for each method → problems

- Isolates different parts of company, different projects, different phases

…Need a metaCASE tool to support *any* method

- Model methods = metamodelling, CASE tool follows metamodel
- Support and integrate multiple methods at once
- Easy addition of new methods, changes to existing ones

MetaEdit+ History

Based on experience gained with MetaEdit

- First non-textual metaCASE environment
- Thousands of users in over 30 countries

Completely new system: language, db, graphic library

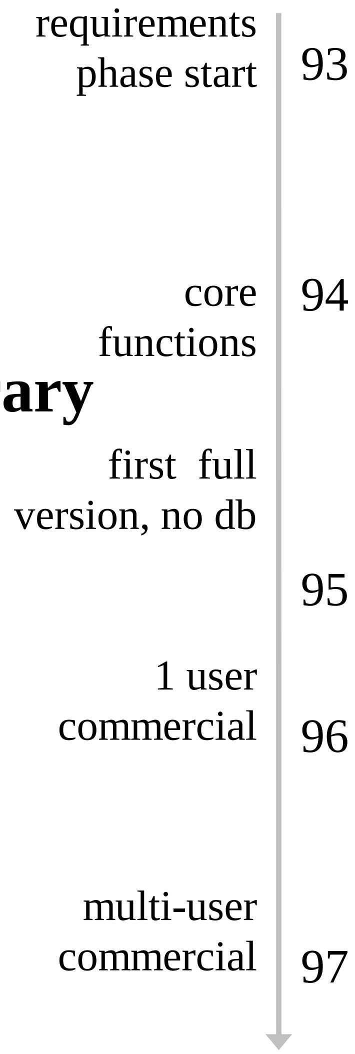
- 3 major versions of each during project!

Core of 4 software engineers, about 10 others

- Envy code management system
- Later modelled MetaEdit+ with MetaEdit+

Own code: 140 classes, 30.000 lines

- Total: 1450 classes, 396.000 lines, i.e. >90% reuse



requirements phase start 93

core functions 94

first full version, no db

95

1 user commercial 96

multi-user commercial 97

MetaEdit+ Features

Multi-user

True repository allows both multi-user modelling and metamodelling

Multi-tool

Diagram, Matrix, Table, plus browsers

Multi-method

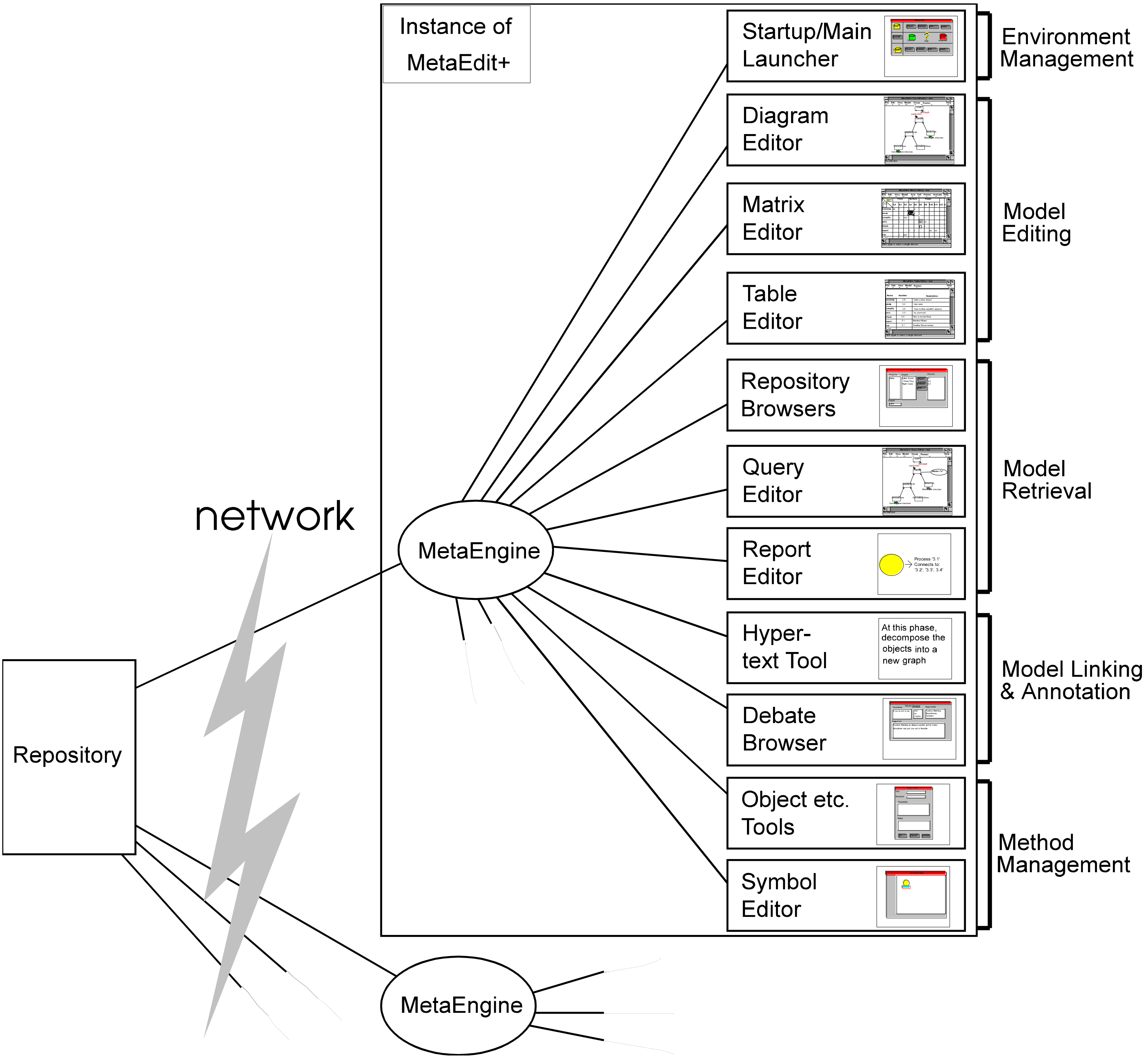
Multiple, integrated methods, several O-O, structured, BPR etc.

Multi-level

Metamodelling and modelling within same toolset

✓ Usability ✓ Flexibility ✓ Extensibility

General Architecture



Instance of MetaEdit+

Startup/Main Launcher

Diagram Editor

Matrix Editor

Table Editor

Repository Browsers

Query Editor

Report Editor

Process '3.1'
Connects to:
'3.2', '3.3', '3.4'

Hyper-text Tool

At this phase, decompose the objects into a new graph

Debate Browser

Object etc. Tools

Symbol Editor

Environment Management

Model Editing

Model Retrieval

Model Linking & Annotation

Method Management

network

MetaEngine

Repository

MetaEngine

Repository: ArtBase Object Store

Adds persistence to Smalltalk

- Orthogonal: per instance not per class
- Transparent: persistent objects behave just like normal ones

Classes are first-class citizens in repository

- Even metaclasses treated mostly as first-class citizens

Late binding and caching enhance performance

- Performs as Smalltalk during transactions except on first fetch
- Memory management transparent to programmer

No special database language

- Just need to mark objects as persistent once

Metametamodel: GOPRR

Metatypes: Graph, Object, Relationship, Role, Property

Graph:	DFD,	Booch Class Diagram
Object:	Process,	Class
Relationship:	Data Flow,	Inheritance
Role:	To,	Superclass
Property:	Process Number,	Class name

Same language for models and methods

- Sharpens learning curve for metamodellers
- Reduces conceptual mismatches between method and model levels

All method information in data

- No need to write code to metamodel

Graph

Has properties in the same way as an object

Contains sets of objects, relationships and roles

Specifies how these are bound together

Binding = relationship, collection of roles, for each role a set of objects

E.g.	Inherits	(1	Superclass	{Class, Class&Object})
		(1..N	Subclass	{Class, Class&Object})

Only place that links roles, relationships and objects:
avoid storing what relationships an object has etc.

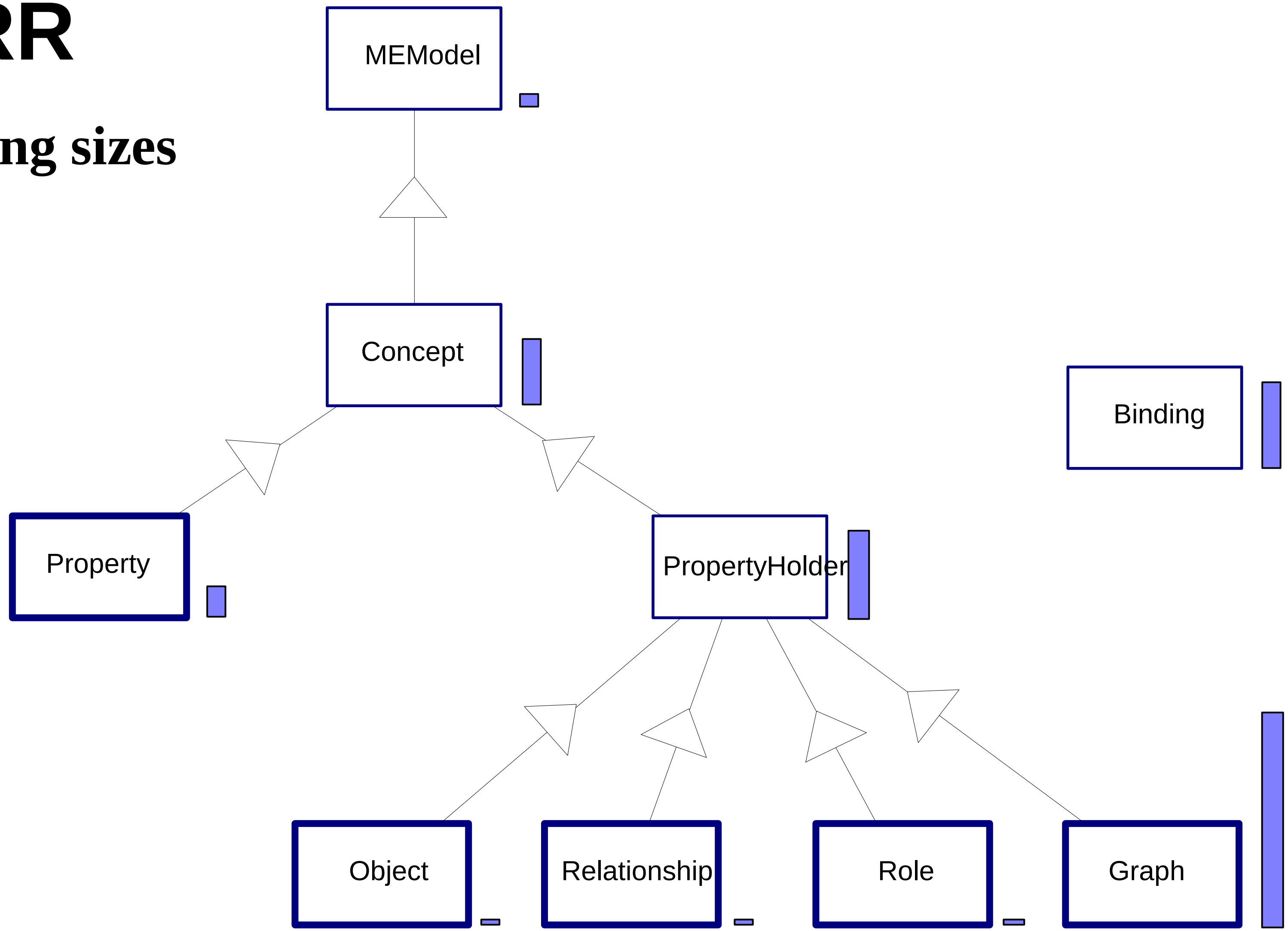
Method Integration

Decomposition (strict), Explosion (more free-form)

Reuse: add objects, properties from other graphs

GOPRR

Showing sizes



Metamodelling with GOPRR:

Subclass metatype classes to make types

- Define which properties to use etc. in form-based GUI
- Subclass created automatically:
  - basically only addition in subclass is a variable for each property
- Define symbol and property dialog (defaults created automatically):
  - these are stored in class instance variables

Instantiate subclasses for instances

- Instance created
- Initialised with default values for each property
- User asked to fill in own values for properties
- Representation of instance created (position & scaling info. etc.)

Object-Oriented Metamodelling

Inheritance

Encapsulation

Polymorphism

Abstraction

Reuse

Inheritance

Subtypes inherit supertype's properties

- Allows fast modelling of similar types, e.g. Class and Class&Object

Data type of complex property

- Most data types simple, e.g. String, but may also be Object type etc.
- Property can hold any instance of the given data type or its subclasses
- E.g. reference to a Class could also be to a Class&Object

Bindings: Legal relationships

- If an object type can take part in a binding, all its subtypes can too
- Simplifies definition of legal relationships

Inheritance (2)

Rules for legal values in property types

E.g. DFD Process number should be [0-9](.[0-9])*

Check supertype rules too: optional.

Reports

Can run reports from any supertype right up to Graph metatype

Encapsulation

- **Behaviour always accompanies data**
- **Rules, Symbols, and Dialogs stored with type**
  - Rule = Smalltalk code block, automatically generated
  - Symbol = pure data structure
  - Dialog = data structure encoded in automatically generated method
  - Changes to type automatically update symbols and dialogs
- **All other behaviour inherited from metatype**
  - Dependent on data in type, used by metatype’s methods

Polymorphism

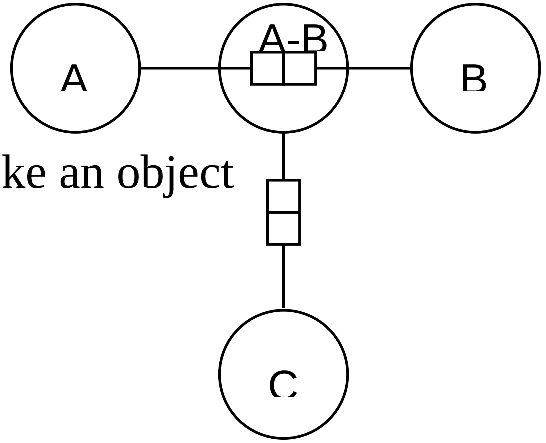
Object etc. as Property: 'Complex' property

Reference to another object

Way to hold complex piece of information

Relationship as Object

E.g. NIAM's objectified relationship:
relationship itself takes part in another relationship like an object



Overloading

Many operations identical for both types and instances

Overriding

Subtypes can override dialogs, rules, symbols

Abstraction

Abstract types

- Created but not used in Graph's types set
- Can still be used in Graph's binding set
- Useful when properties or legal relationships shared by several types
- Can also be used as way to organise types, with no other semantics

Complex Properties

- Can use without needing to know details of the object type
- Better than encoding several pieces of information into a string

Separation of Binding and Relationship

- Relationships carry no excess baggage to encumber their reuser

Reuse

- **Objects, Roles, Relationships, Graphs, Properties**
  - …both types and instances
  - Reuse on type level defines possible reuse on instance level
- **Reuse from different projects, graphs, objects**
  - Choose by current location or type
  - Search for reusable elements in browsers (wildcards)
- **View where element is used**
- **Over 30% of types in metamodels reused**
  - Much higher figures possible if methods not followed exactly

Method Engineering

software engineering ~ programming

method engineering ~ metamodelling

Reusable components

Integrate methods by reuse

Organise components, search and browse for reuse, view current users

Test metamodels by modelling immediately

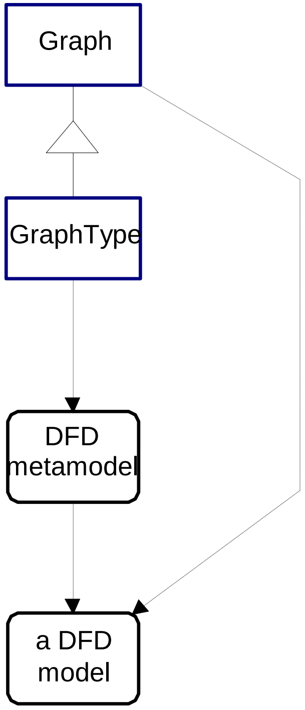
No compile-link-run cycle

Update metamodels even when instances exist

Automatically update models

Warn about or disallow changes that are most dangerous

Difficulties



Instance & inheritance hierarchies mixed?

E.g. Graph type behaves like a specialised Graph instance

Solution: Graph has instance methods duplicated as class methods

When to inherit behaviour

Don’t allow relationship subtypes in binding

Don’t inherit graph type contents, just copy them

Do we need inheritance for property types?

Need better tools for method engineer

Currently functionality divided among several tools

Improvements in searching for reusable types

Overview of methodology

From: Risto 'Rise' Pohjonen [rtpo@cc.jyu.fi]
Sent: 13. May 1997 14:06
To: 'mtilman@argo.be'; 'shai_by@NetVision.net.il'; 'stevek@jytkoson2.jyu.fi';
'buchner@pu.informatik.th-darmstadt.de'; 'klaus.makela@reserch.nokia.com';
'cimmino@rs0.corinto.interbusiness.it'; 'stadt@cs.utwente.nl';
'antero.taivalsaari@research.nokia.com'
Subject: Demonstrations

Dear ECOOP '97 demonstration proposers,

Here is my suggestion for the ECOOP '97 demonstration schedule. Most of you wanted to give your demonstrations twice and I have tried to fix everything up according to your preferences. Everyone has been given the opportunity to give his/her demo once during the main conference (Wed, Thu, Fri). During the main conference, demos will run simultaneously with paper sessions (although not with invited talks or the panel session). The "duplicate" demos have been allocated on Monday and Tuesday and they will run simultaneously with the late-afternoon tutorials.

Two demos have been allocated only once, so Mehmet & Richard and Antero, please contact me if you want to have another one and we can check things out - there are still some options.

What I need you to do now is to check out your demo time and the information provided about your demo in this message. If there are some problems with the scheduling, please let me know. The information given here about your demo is the same that I have planned to be included in the final programme. So, please check the titles of demos, the names of your organizations and the content of the abstracts.
IF YOUR ABSTRACT (= D3, D6 and D7) EXCEEDS 20 LINES, PLEASE SUPPLY ME A SHORTER ONE!

Best regards,

Risto Pohjonen
ECOOP '97 Demonstration Chair

ECOOP '97 Demonstration Schedule

June 9th	Mon	17:30	D1 (Shai Ben-Yehuda)
		18:30	D2 (Piergiorgio Cimmino)
		19:00	D3 (Klaus Mäkelä)
		19:30	D4 (Jürgen Buchner)
June 10th	Tue	17:30	D5 (Steven Kelly)
		18:30	D6 (Martine Devos, Michel Tilman)
June 11th	Wed	11:30	D5 (Steven Kelly)
		14:30	D7 (Mehmet Aksit, Richard van de Stadt)
		16:15	D8 (Antero Taivalsaari)
June 12th	Thu	11:00	D1 (Shai Ben-Yehuda)
		14:00	D3 (Klaus Mäkelä)
		14:30	D4 (Jürgen Buchner)
June 13th	Fri	9:30	D6 (Martine Devos, Michel Tilman)
		11:00	D2 (Piergiorgio Cimmino)

D1 Profiles in Java

Shai Ben-Yehuda, SELA Labs, Israel

The Profile paradigm suggests a way to develop user oriented software products, means to enable users to customize their software products, adapting them to their specific needs. The methodology guides the developers using it to synthesize the declarative section of the software out of the realm of the developer to a neutral zone, called the profiles section. The profiles section is used and maintained by both developers and users.The profiles paradigm is demonstrated in Java environment.

D2 PDA Client-Server or Client-Server and Mobile Computing

Piergiorgio Cimmino, CORINTO (Consorzio Ricerca Nazionale Tecnologia Oggetti), Italy

Mobile Computing is perhaps the final fronteer of the Client - Server paradigm. The demo is about a palmtop application (Newton Messages Pad 130) aimed at helping the Italian doctor to carry out their own patient management activities. Its most attractive side, relies in the communication module (which is called the Communication Toolbox), which allows communication, over the Internet (using a PPP server), with a Nokia GSM Cellular phone (but also normal phone lines can be used) to an HTTP server interfacing a C++, CLI CGI module which retrieves farmaceutical data and handle reservations with a DB2/6000 database. Additional information and a sketch of the proposed architecture can be found at: http://corinto.interbusiness.it/P6//P6_welcome.html

Corinto (http://corinto.interbusiness.it) is a joint venture among IBM, APPLE, and SELFIN.

D3 The training of the OCTOPUS method via Internet at Nokia

Klaus Mäkelä, Nokia Research Center / Software Technology Laboratory, Finland

Octopus is a method for developing object-oriented software particularly for embedded real-time systems. It has been developed at Nokia Research Center and it is widely used at different Nokia Business Units. Octopus method is teached to Nokians on different tutorials and this training program is supported by Nokia internal Internet courses. We demonstrate two of these courses here.

Visual Octocourse is an active map based Internet course. It gives a systematic introduction to Octopus method. Topics include:

1. structure of the development process of a software system
2. development sequences of requirements specification, system architecture, subsystem analysis, subsystem design and performance analysis.

All six topics are active maps with 5 - 10 references to the diagrams, scenarios and different datasheets of the method.

General Octopus Course is a textual Internet course. The course gives an outline of the OCTOPUS method. The course covers major concepts of the method: structuring the development process, system requirements and architecture phases and subsystem analysis, design and implementation phases. The texts are clarified with figures and tables. The course includes a glossary of the definitions and notations.

D4 HotDoc - A Framework for Compound Documents

Jürgen Buchner, Technische Hochschule Darmstadt, Germany

HotDoc introduces a new interpretation for the concept of a "document". A HotDoc document is not only a static sequence of text and pictures etc., a HotDoc document is a user interface for applications, which can be combined and placed freely by the user within a document.

HotDoc is a framework for the development of editors for compound documents. It is implemented in VisualWorks Smalltalk. Due to its object-oriented nature, many restrictions of other systems do not apply to HotDoc. It can be used by developers to implement application-specific editors or user interfaces.

D5 Object-Oriented Metamodelling in the MetaEdit+ MetaCASE Environment

Steven Kelly, MetaPHOR project, University of Jyväskylä

We will demonstrate the use of object-oriented metamodeling for fast, incremental definition and evolution of methods in MetaEdit+.

Recent studies suggest that a main reason for the perceived failure of CASE tools has been the narrowness of their approach, which forces designers to use a fixed set of methods supported. MetaCASE tools, which allow modification and definition of new method support into CASE tools, have however been difficult to use.

MetaEdit+ is a metaCASE tool which uses a simple browser- and form-based GUI for method definition. The methods are definedas a set of reusable object-oriented components, and method definition and use are accomplished within the same tool, enabling fast method development and incremental evolution. Existing models are automatically updating to reflect changes in the method. MetaEdit+ method modelling supports inheritance, encapsulation, polymorphism and abstraction.

D6 Evolutionary Application Development using a Meta-repository based Framework

Martine Devos, Argo, Belgium
Michel Tilman, Argo, Belgium

The Argo framework is an object-oriented framework developed inVisualWorks\Smalltalk for modeling the business logic of a wide variety of organizations and application domains, making as few assumptions as possible about particular business models. It offers a set of generic tools for defining, customizing, managing and maintaining the business objects pertaining the workings of a particular organization, be it in the context of database applications, electronic document management or workflow. Its main business areas are large administrations which need to cope with several applications sharing a common business model.

The key features of the architecture are its open-ended nature, relying on an extensible framework of re-usable components containing the generic application logic, and on a meta-repository containing descriptions of the actual business model.

The repository captures both formal and informal knowledge of the business model and of personal and shared work practices. High-level end-user and administration tools consult the repository at run-time, querying the meta-model for dynamic behavior. Changes to the repository can be made at-runtime and are immediately available to clients.

End-user applications can be developed interactively and incrementally through modeling and configuration. They are not hard-coded, neither are they generated. Instead, we build increasingly complete specifications of end-user applications that can be executed immediately. This way, applications can evolve more easily with changing needs or organization, be it at the level of functionality or the business model.

D7 Composable Solutions to Modeling Problems of the Object Model Using Composition Filters

Mehmet Aksit, TRESE project, University of Twente, The Netherlands, TRESE project, University of Twente, The Netherlands

The conventional object-oriented model as adopted by languages such as C++, Smalltalk and Java is not capable of modeling certain aspects of applications in an adaptable and reusable way. Some of these modeling problems are for example multiple views, dynamic inheritance, reusable synchronization specifications, and reusable constraints.

Since 1987 we have been carrying out research activities at the University of Twente to enhance the expression power of the object-oriented model.

We investigate solution techniques using the following criteria:

1. To solve the modeling problems, the object model must be enhanced modularly without losing its basic characteristics.
2. Since more than one problem can be experienced for the same object, enhancements must be independent from each other.

As a result, we have extended the conventional object-oriented model with the concept of composition filters. Composition filters can be attached to objects without modifying the basic object structure.

A number of different filter types have been defined, each addressing a certain concern. For example, the Dispatch filter can be used to express dynamic inheritance and delegation mechanisms. Constraints among objects can be modeled by using Meta filters. The Wait filter is suitable for defining reusable synchronization constraints [4,5]. The RealTime filter is useful for programming classes with a reusable real-time behavior.

We will illustrate solutions for the aforementioned modeling problems using Sina, an object-oriented programming language which adopts the composition filters model.

D8 TDE -- Collaborative Environment for Geographically Distributed Object-Oriented Software Design

Antero Taivalsaari, Nokia Research Center, Software Technology Laboratory, Finland

TDE (Telecom Design Environment) project builds visual software design and reverse engineering tools for Nokia's software developers. The project focuses especially on features that facilitate seamless, interactive, "live" collaboration between designers located in different buildings, cities or even countries. The TDE environment combines many challenging technology dimensions, including visual information and document management dimension, OOA/OOD CASE tool dimension, and "liveboard" collaboration and communication dimension. In this demonstration the capabilities of TDE are illustrated, and the future directions of the system are outlined.
