9th Workshop for PhD Students in Object Oriented Systems

Awais Rashid[1], David Parsons[2], and Alexandru Telea[3]

[1] Computing Department, Lancaster University, UK
marash@comp.lancs.ac.uk
[2] The Object People, Epsilon House, Chilworth Science Park, Southampton, UK
davidp@objectpeople.com
[3] Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands
alext@win.tue.nl

Abstract. The PhDOOS workshop covered a wide scope, as its over 20 participants were PhD students in all areas of object orientation. The presentations covered topics such as databases, languages, software engineering and artificial intelligence, components and generative programming, analysis and design, frameworks and patterns, aspected oriented programming, distribution, and middleware. Several topics of shared interest were identified and targeted in separate discussion groups on meta-information, the success or failure of OODBMS, and a general theme on the future of object orientation. As the participants had various research interests covering practically all the OO spectrum, we can confidently state that these topics reflect actually the concerns and needs of the OO community, and emerge from its concrete needs. This document is to be complemented by a workshop proceedings document which will publish the full versions of the presented papers.

1. Introduction

The 9th workshop for PhD Students in Object Oriented Systems (PhDOOS '99) was held on June 14-15, 1999 in Lisbon, Portugal in association with the 13th European Conference on Object Oriented Programming (ECOOP). The workshop was part of the series of PhDOOS workshops held in conjunction with ECOOP each year. The PhDOOS workshops differ from usual workshops. The scope of the presentations is wide. This is because the participants are PhD students and topics are derived from the areas of interest of the participants. The workshops serve as a forum for lively discussion between PhD students doing research in similar areas. For each participant, this is an opportunity to present his/her research to a knowledgeable audience who are working in a similar context. In particular, the presenter may learn about new points of view on this research or about related work, and future research collaboration may be initiated. The workshops also feature invited speakers talking about interesting future research topics in object orientation. This provides the participants an opportunity to have an "unplugged" discussion with well-known personalities in the field. The workshops also aim at strengthening the international Network of PhD Students in Object-Oriented Systems (PhDOOS[1]), which was initiated at the 1st workshop during ECOOP '91 in Geneva, Switzerland. PhDOOS '99 was organised by Awais Rashid, David Parsons and Alexandru Telea and followed the patterns of its predecessors. The participants were divided into three different categories. First, it was possible to submit a (3-8 page) position paper for review, and to give a 30 minutes presentation at the workshop. Second, it was possible to submit a one-page abstract for review and to give a 15 minutes presentation. Finally, anticipating some last-minute participants a "guest" status was defined for them, including a very short presentation if they wanted to give one. The workshop featured three keynote speakers: Ian Sommerville [2], Gregor Kiczales[3] and Ulrich Eisenecker[4]. The workshop received a total of 30 submissions from 14 countries in 3 continents. Of these 19 were position papers while 11 were abstracts. For the first time in the series of PhDOOS workshops a review process was introduced. Submissions were reviewed by PhD students almost two-thirds into their PhD. The review process was not designed to select the few very best papers, but to ensure that every participant was able to present some relevant material, and was sincere and well prepared. As a result, 17 position papers and 6 abstracts were selected for presentation at the workshop. Accepted papers and abstracts are available on the workshop web site at: http://www.comp.lancs.ac.uk/computing/users/marash/PhDOOS99. They will also be included in the workshop proceedings to be published by University of Eindhoven, The Netherlands.

2. Workshop Structure

The workshop was divided into sessions based on the presenters' areas of interest. These were as follows:

- Databases
- Languages, Software Engineering and Artificial Intelligence
- Components and Generative Programming
- Analysis and Design
- Frameworks and Patterns
- Aspect Oriented Programming
- Distribution and Middleware

The following sections summarise the discussion that took place in the various sessions.

[1] http://www.ecoop.org/phdoos/
[2] Computing Department, Lancaster University, Lancaster, UK
[3] Xerox PARC, USA
[4] University of Applied Sciences, Heidelberg, Germany

2.1. Databases

Juan Trujillo, Awais Rashid, Isabella Merlo, Marlon Dumas and Radovan Chytracek presented their work in this session. Juan Trujillo discussed the recent increased interest in multidimensional databases (MDB) and On-line Analytical Processing (OLAP) scenarios. He pointed out that OLAP systems impose different requirements than On-line Transactional Processing (OLTP) systems, and therefore, different data models and implementation methods are required for each type of system. There have been several different multidimensional data models proposed recently. However, there are certain key issues in multidimensional modelling, such as derived measures, derived dimension attributes and the additivity on fact attributes along dimensions, that are not considered by these proposals. He presented the GOLD model, an Object Oriented (OO) multidimensional model in which all the above-mentioned issues are taken into consideration. Since the GOLD model is based on the OO paradigm, data functionality and behaviour are easily considered, which allows one to encapsulate data and its operations (especially useful when referring to OLAP operations). The GOLD model, therefore, takes advantage of some OO issues such as inheritance and polymorphism and allows one to build complex multidimensional models. Finally, another main advantage of the GOLD model is that it is supported by an OO formal specification language (GOLD Definition Language, GDL) that allows one to define multidimensional conceptual schemes. More concretely, this GDL is an extension of the OASIS formal specification language (developed in the Technical University of Valencia, Spain) to capture more precisely the features linked to multidimensional databases. In this way, the requirements of the multidimensional conceptual schema can be validated, which allows one to check whether the system properties captured in the specification are correctly defined or not. Awais Rashid proposed a novel hybrid technique for impact analysis in complex object database schemata. He argued that like most database applications, object databases are subject to evolution. Evolution, however, is critical in OO databases since it is the very characteristic of complex applications for which they provide inherent support. These applications not only require dynamic modifications to the data residing within the database but also dynamic modifications to the way the data has been modelled (i.e. both the objects residing within the database and the schema of the database are subject to change). Furthermore, there is a requirement to keep track of the change in case it needs to be reverted. Object database schemata designed to fulfil the above set of requirements can become very large and complex. The large amount of information and complex relationships between the various entities in these schemata combine to make the process of assessing the effect of change expensive, time consuming and error-prone. However, without proper assessment, it is impossible for developers and maintainers to fully appreciate the extent and complexity of proposed changes. For maintainers this makes cost estimation, resource allocation and change feasibility study impractical. For developers, a lack of adequate impact analysis can lead to difficulties in ensuring that all affected entities are updated for each change to the conceptual structure of the database. Impact analysis has been employed to determine the extent and complexity of proposed changes during the various stages of the software life cycle. Although many of these techniques have been suggested for analysing the impact of changes to OO design and code level artefacts, inherent deficiencies in such methods render them unsuitable for performing change impact analysis in an object database schema. The hybrid technique Awais presented combined traditional impact analysis approaches with experience based capabilities in order to support change impact analysis in complex object database schemata. Isabella Merlo was of the view that object database systems (both the pure object oriented systems and the object-relational ones) are the systems that in the next few years will replace conventional relational databases systems, or even older generations systems (such as the hierarchical and the network ones). She pointed out that although many approaches have been proposed in the past to extend object database systems with innovative features and interesting results have been achieved, there is a lack of uniformity and standardization across those approaches. In her opinion one of the reasons is that, the standard for object-oriented databases, ODMG, is recent and not well-established. However, ODMG provides the basis for extending object-oriented databases with new capabilities. Among them, the introduction of temporal and active capabilities in ODMG is an important issue that research in the database area has to address. Her research has introduced temporal and active features in the ODMG standard. The introduction of time and active rules was addressed separately. In future she intends to investigate problems related to their integration in the same model. Marlon Dumas also discussed data models and languages for temporal OO database management systems. He indicated that research in this area has been prolific regarding temporal extension proposals to data models and languages. Whereas in the relational framework these works have led to the consensus language TSQL2, and two proposals to the SQL3 standardization committee, equivalent results are missing in the object-oriented framework. Early attempts to define temporal object data models failed to become widely accepted due to the absence of a standard underlying data model. As the ODMG proposal was released and adopted by the major object database vendors, several temporal extensions of it were defined. Marlon pointed out that these neglect at least some of the following important aspects:

1. migration support, as to ensure a smooth transition of applications running on top of a non-temporal system to a temporal extension of it
2. representation-independent operators for manipulating temporal data, as to exploit the abstraction principle of object-orientation
3. formal semantics

One of the main goals of his work is to propose a general framework for designing DBMS temporal extensions integrating the above features, and to apply it to the ODMG. The design and formalization of the main components of this framework are almost finished, leading to a temporal database model named TEMPOS. In addition to providing temporal extensions of ODMG's object model, schema definition and query languages, TEMPOS includes a language for describing patterns of histories. The feasibility of the proposal was validated through a prototype implementation on top of the O2 DBMS, which has been used to experiment on applications from various contexts. Radovan Chytracek discussed the great importance of database systems in any HEP (High Energy Physics) experiment. HEP community in LHC (Large Hadron Collider) era is in transition from FORTRAN to C++ and from data streams to persistent objects. Together with that a new data management will be necessary, which would allow transition from "files and tapes" approach towards the access to data in the form of objects by selection of their required physics contents. Data volumes of the LHC experiments are expected in the PB (1015 bytes) order of magnitude and this fact makes the job much harder to do. In order to conform to the object-oriented paradigm, LHCb (Large Hadron Collider Beauty; precision measurements of CP-Violation and rare decays) had to heavily investigate the design and development of object databases for both the on-line (data acquisition and real-time processing) and off-line (simulation, reconstruction and analysis) computing environments, e.g. the Event Store, Detector Description Database (DDDB), Calibration and Alignment Database etc. For that purpose the Gaudi framework at LHCb experiment is being developed to cover all stages of physics data processing. The design choices taken at the time of creating the Gaudi architecture take into account specifics of physicists work in order to provide access to object persistency technologies in a transparent way and proper data abstractions to make the physics data handling natural to physicists. Very important part of the framework is DDDB, which holds data describing detector apparatus structure and environment.

2.2. Languages, Software Engineering and Artificial Intelligence

The languages, software engineering and artificial intelligence stream included contributions from Stefan Chiettini, Moritz Schnizler, John Flackett and Cristian Sminchisescu. Stefan Chiettini opened the session by proposing a technique for the documentation of object interaction. He described how the documentation of object-oriented systems usually consists of two parts: First there is the static part with the description of classes and methods. This part usually contains information about interfaces, inheritance relations and aggregations. The second part, which was the topic of his presentation, describes the dynamic behaviour of the system in a certain situation at run time. Common design and documentation techniques like OMT or UML introduce event trace diagrams (OMT) and sequence diagrams (UML) to visualize run time behaviour of interacting objects. These diagrams show the message sequence in a certain situation at run time. Their major weakness is that they are themselves static and therefore capable of illustrating only one special case, typically called a 'scenario', not the general behaviour of objects. Stefan proposed behaviour diagrams as an extension of existing diagrams to meet the requirements of modern documentation: structured documents with hypertext and multimedia capabilities extended with the possibility to interactively explore the documentation. Behaviour diagrams enable the user to describe general situations in object-oriented systems like conditional message sequences or dynamically bound method calls. Moritz Schnizler followed with his work on a testing approach for program families. Today a popular (because cost efficient) software development approach is the deployment of program families, sometimes called product lines. A program family evolves over time from a successful program. Its main characteristic is that its members have many properties in common, especially their functionality, so object-oriented framework technology is well suited for their implementation. In practice, efficient testing of a program family member remains a problem, often meaning that new tests have to be developed for every single program. The aim of Moritz's work is to develop a more efficient process for testing them. The model for his approach is test benches from other engineering disciplines, e.g. when a car engine is developed. The principle idea is to transfer this approach to the area of software development with object-oriented frameworks. The main problem of this approach is the lack of approved testing methods for object-oriented software. Most testing techniques have their roots in imperative programming and are of little help in testing the interaction of collaborating classes which are a characteristic of object-oriented software. Moritz is investigating the possibility of testing the correct collaboration of classes in the context of frameworks, so that test cases for collaborating classes are developed from the originally intended behaviour for their collaboration. An example for such a testable collaboration is the MVC pattern where, for example, a test case is a state change to the model object requiring appropriate updates from its observing view objects. Based on this, built-in tests are included in the framework that automatically test the correct implementation and use of such pre-implemented collaborations. The final goal is to have built-in tests for all characteristic collaborations that comprise the functionality of a framework. So, using this built-in testing infrastructure, a developer can easily retest the framework's core functionality, when he adapts or extends it, in the context of a new program. For the classes under test that means they need to be more testable, implementing a special test interface that contains, for example, additional inspection methods. John Flackett continued this session with a description of his ConnECT (Connectionist/Symbolic Engine for High-Level Cognitive Tasks) system. ConnECT is concerned with the development of an object-oriented software tool which brings about a synergy of existing knowledge representation techniques, the focus of which is to model an aspect of Natural Language Processing (NLP) by automating text indexing and retrieval. ConnECT exploits object-oriented programming techniques in order to provide a flexible and robust architecture within which to model encapsulated matrices and their operations. Fundamentally, the system is controlled through the use of an envelope class, which in turn utilises object parameter passing as the means for synergising the distinct modular processes. An underlying data class forms the knowledge base upon which extraction operations operate to provide the built in intelligence required for the high level cognitive task proposed. The implementation differs from that of normal object parameter passing, as part of a variable aggregation, in as much as the data object being passed does not simply provide an extension to the receiving objects attributes, rather, it encapsulates all of the required attributes. Concluding this session with a language related presentation, Cristian Sminchisescu described his object-oriented approach to C++ compiler technology. Compilers of language translators front-ends comprise traditionally well-delimited stages like lexical, syntactical, and semantic analysis. Traditional compiler architecture is based on the separate design and implementation of these stages, using tools such as LEX and YACC. Although many text books for compiler design, formal languages, and parsing exist, there are few detailed descriptions regarding the design and implementation of a complete language processor for a complex language like C++. In particular, the C++ language has an inherently ambiguous grammar. This implies that no direct grammar transformation can transform its grammar into a nonambiguous one. Consequently, the traditional lexical-syntactic analysis pipeline will not be effective if one desires to implement the two stages in a modular, decoupled fashion. Most existing C++ compilers (such as the GNU g++ compiler for example) couple the two stages intimately by letting them share and modify complex data structures. The resulting product is monolithic and hard to understand and to maintain from a developer's perspective. Cristian has addressed the above problem by introducing a new, separate stage between the usual lexical and syntactical stages. The stage, called LALEX (lookahead LEX) takes over the C++ context dependency by special processing and introduction of disambiguation tokens. The resulting program pipeline can be built using tools such as LEX and YACC, is modular, and is simple to understand and maintain. Furthermore, the usage of OO techniques in the semantic analysis stage design is made possible by the simplification of its structure due to the LALEX stage. Inside this stage, a C++ program is represented as an (OO) abstract syntax graph whose nodes are classes that model the C++ language semantic constructs. The leaf subclasses of this hierarchy map to the C++ language terminals. The other nodes map to C++ syntactic, C++ semantic, or intermediate 'door' constructs. Modelling the parsed language's constructs as an OO type hierarchy has several advantages. First, semantic rules for constructs can be written as specific class methods. Second, the semantic stage's control mechanism can be written independently of the actual language being parsed, as a generic control algorithm that uses the Visitor design pattern on the syntax graph. Finally, the OO approach to C++ compiler construction has proven efficient in the implementation of the ambiguity resolution mechanisms needed for C++. The interface between the special LALEX stage and the usual parser is kept as simple as possible. LALEX is actually called back by the YACC-style parser to provide tokens. These are provided in a nonambiguous manner by calling back on the classic LEX stage and by using the disambiguation information provided by a specially maintained symbol table. In contrast to other compiler implementations, this symbol table is encapsulated in the LALEX stage and thus differs from the full-scale symbol table used by the parser stage. The above distinction helps for a clear design and implementation of the C++ compiler.

2.3. Components and Generative Programming

The components session was begun by Anthony Lauder, who introduced 'event ports'. Explicit invocation across collaborating components in component-based systems leads to tight component coupling. This diminishes component maintainability, flexibility, and reusability. The implicit invocation model, wherein components register their message interests with a broker, de-couples components and hence reduces inter-component dependencies. This, however, may ignore the historically determined nature of the flow of component message interests. This leads to implementations of message receipt functions polluted with guard code that rejects out-of-sequence messages in order to enforce components' time-ordered protocols. Statecharts, however, are ideally suited to expressing such protocols. By combining statecharts with implicit invocation, direct realization of time-ordered component protocols is achieved without code pollution, offering the potential for a cleaner, more adaptable component collaboration strategy. Anthony presented the development of 'event ports', which reflect this combination and encapsulate a promising new component model. Andreas Speck presented his OO real time (component based) control system. He explained how the rapid evolution of standard hardware such as workstations and PCs has made it possible to develop standard hardware-based universal control systems. Currently the traditional proprietary device-specific controller systems (e.g. robot controls, numeric controls) are ported to this new standard hardware. However, such control systems are still proprietary and device dependent. Andreas posed the question, how can we now build universal and flexible control systems? He has evaluated three approaches that are based on each other: an object-oriented architecture that may be used as an architectural pattern, a conventional object-oriented framework and a component-based framework. In contrast to the today's available control systems all these approaches are much more flexible and can be used to implement different control functionalities. The pattern provides no semi-finished code. However it is very useful when the universal control system should be realized on non standard platforms (e.g. industrial PCs with special real-time operating systems). Both framework approaches (conventional and component-based) already contain reusable base-code which may by adjusted to the user's needs (e.g. to the required control functionality and desired standard platform). Compared with the conventional framework the component framework is more flexible since it is not restricted to predefined flexible hot spots. The free exchange of components leads to a highly flexible system. Moreover the development of a component-based framework needs no specific existing architecture - generic architectural guidance is enough. Ian Oliver argued that animation has been shown to be a useful tool for the validation of the behavioural properties of a model. Animation can be thought of as the 'halfway' house between the specification and the final executable code, relying on some form of execution of the specification. He then discussed how the Object Constraint Language (part of the Unified Modelling Language) may be executed in some sense to provide the basis of an animation environment for OO modelling. Ian's work is based around formulating a mapping between OCL statements and a sequence of atomic operations that perform some form of basic modification to the UML object-diagram. The concepts he is interested in are the class, link and attribute value and so he has defined five operations: modify (value), create/delete (object) and link/unlink (links) that can be employed to modify these components on the diagram. The various presentations were followed by a keynote speech from Professor Ulrich Eisenecker who presented a detailed view of components and generative programming. He discussed how most software-engineering methods focus on singlesystem engineering. This also applies to object-oriented methods. In particular, developing for and with reuse are neither explicit activities nor are they adequately supported. Furthermore, there is no explicit domain scoping, which would delineate the domain based on the set of existing and possible systems. Current methods also fail to differentiate between intra-application and inter-application variability. In particular, inter-application variability is often implemented using dynamic variability mechanisms, even if static ones would be more efficient. Analysis and design patterns, frameworks, and components struggle for improving reuse and adaptability, but do not provide a complete solution. For example, despite the fact that frameworks are created in several iterations, there is still a high chance that they contain unnecessary variation points, while important ones are missing. He argued that Domain Engineering overcomes the deficiencies of single-system engineering. It includes a domain scoping activity based on market studies and stakeholder analysis. Analysing commonalities, variabilities, and dependencies lies at the heart of domain engineering. The results of domain engineering (i.e. engineering for reuse) are reusable assets in the form of models, languages, documents, generators, and implementation components. These results represent the input to application engineering (i.e. engineering with reuse). An extremely useful means for capturing features and variation points are feature diagrams, which were originally introduced by the FODA method (Feature-Oriented Domain Analysis). They are augmented by additional information including short descriptions of features, dependencies, rationales for features, default values, etc. Two kinds of languages are then derived from feature models, namely domain specific configuration languages and implementation components configuration languages. The former is used to describe the requirements for a specific system from an application-oriented point of view. The latter is used to describe the implementations of systems in terms of composing components. Configuration knowledge is used to map from requirements specifications to configurations of implementation components. Manual coding of implementation configurations for a large number of variants is a tedious and error prone process. Therefore, generative programming introduces configuration generators translating requirements specifications into optimised configurations of implementation components. An adequate support for implementing such generators requires the ability to define domain-specific languages and representations (e.g. graphical representations), domain-specific optimisations, type systems, and error detection. Furthermore, it is important to be able to implement domain-specific debugging and editing facilities for entering, manipulating, and rendering program representations, as well as domain-specific testing and profiling facilities. A library of domain abstractions, which also contains code extending a programming environment in the above-mentioned areas, is referred to as an active library.

2.4. Analysis and Design

The analysis and design session covered a wide range of subjects and included contributions from Akos Frohner, Glenn Lewis, Christoph Steindl and Fabio Kon. A keynote speech was also delivered by Professor Ian Sommerville. Akos Frohner began by describing layered design visualisation. Designing an object-oriented system is a process that is well supported by a great number of notations and design techniques such as UML. Although UML provides notation for almost all aspects of object-oriented software design, it lacks features for describing aspects that are outside of the design domain or require information from different diagrams. For example, there are no good notations for the visualisation of frameworks, friendship relationships, components, meta-level aspects or security considerations. As a possible solution Akos proposes to use dynamic multi-layer diagrams, in addition to passive, paper oriented diagrams. Such diagrams allow the user of an object-oriented CASE tool to concentrate on the specific feature that she or he is interested in, and filter out the remaining parts. The basic idea is to place related elements of a diagram on to one layer and stack these layers up. If all the layers are used, the final diagram will contain all details, but one may hide any unnecessary layers to focus on a small and comprehensible subset of the components. In an active CASE tool layers can be locked to disable the modification of some elements. Akos also explained the task of framework documentation, showing only the skeleton of a hot-spot, and letting the user add more details by uncovering hidden layers. One may also extend the hot-spot in place on a new layer without modifying the original diagram. Akos gave further examples using the layered structure to support the design of complex systems and their three-dimensional visualisation. The layering technique adds some new notational features to the existing possibilities of UML, but the main impact is on the design work itself. Using layers to associate elements allows users to express their own way of thinking above the logical structure of the model (i.e. package boundaries). Akos' work is part of an ongoing research to use non object-oriented features in the design of large programs, including the storage and visualisation of such information. This presentation was followed by Glenn Lewis, describing a practical approach to behavioural inheritance in the context of coloured Petri Nets. Inheritance means one can begin with an abstract representation of an object that is easy to understand and clutter-free, and incrementally change that to a more concrete representation. In other words, inheritance provides support for abstraction, which is the most common and effective technique for dealing with complexity. The principle of substitutability has been proposed in various forms to give the expectations that an incrementally changed component should comply with if it is to be substituted for a component. One possibility, which is known as weak substitutability, relates to the compatibility of method parameter and result types - it does not require behavioural compatibility. Many consider that weak substitutability is not enough: substitution may still lead to incorrect behaviour even if the weak substitutability principle is satisfied. Another version of the substitutability principle, referred to as strong substitutability, requires behavioural compatibility between the type and subtype. There are a number of proposals for substitutability in the context of concurrent object-oriented systems, but it is unclear whether these proposals are overly constrained for practical application. Glenn presented a discussion of substitutability, and in the context of coloured petri nets he presented a set of three incremental modifications which lie somewhere between weak and strong substitutability. The constraints that he imposes can be checked statically and they have the property that if the refinement is at least as live as the abstraction, then strong substitutability holds (this property cannot be checked statically.) The incremental changes are presented informally. Formal definitions of the proposed increment changes can be found elsewhere, as can an examination of case studies in the literature that suggests the above forms of incremental change are applicable in practice. Christoph Steindl followed with a presentation of static analysis of object-oriented programs, specifically describing his work on program slicing in Oberon. Static analysis derives information by inspection of the source code, and this information must be valid for all possible executions of the program. Conservative assumptions must be taken if the program uses conditional branches and iteration since it is not known at compile time which branches will be taken at run time and how many iterations there will be. Static information is necessarily less precise than dynamic information (obtained by monitoring one specific execution of a program) but it can be computed once for all possible executions, whereas dynamic information must be computed again and again. Two main concepts of object-oriented programming are polymorphism and dynamic binding. These dynamic aspects are difficult to integrate into static analysis, e.g. in most cases the exact destination of polymorphic call sites cannot be determined by static analysis. Additionally, data flow analysis for heap allocated objects is difficult. Since the number of objects is unbounded, they cannot be handled individually. If they are classified into groups, then all objects of a group are aliases for the data flow analysis. Christoph has developed a program slicer that models dynamic aspects of object-oriented programs correctly . Starting from conservative assumptions about dynamic binding and aliases, new user guidance techniques are used to reduce these assumptions. In this way, static analysis can be enriched with user-supplied knowledge to yield information with a precision similar to dynamic information. Fabio Kon presented a framework for dynamically configurable multimedia distribution. Multimedia applications and interfaces will radically change how computer systems will look in the future. Radio and TV broadcasting will assume a digital format and their distribution networks will be integrated with the Internet. Existing hardware and software infrastructures, however, are unable to provide all the scalability and quality of service that these applications require. In previous work, Fabio has developed a framework for building scalable and flexible multimedia distribution systems that greatly improves the possibilities for the provision of quality of service in large-scale, wide-area networks. This framework was successfully deployed in different situations including the live broadcast of a long-term, live audiovisual stream to more than one million clients in dozens of countries across the globe. In his presentation, he identified some significant problems that limited the usability of the previous framework. He proposed mechanisms for attacking these problems and described how he was using mobile configuration agents and a CORBA-based framework for providing efficient code distribution , dynamic reconfiguration, and fault-tolerance to the multimedia distribution framework. The work is based on the infrastructure for dynamic configuration and management of inter-component dependence provided by the 2K Distributed Operating System. 2K is a component-based system that uses a dynamically configurable CORBA communication layer to support on-the-fly adaptation of component-based applications. In his keynote speech Professor Ian Sommerville discussed integration of social and OO analysis. Most methods of analysis focus on technical aspects of the system to be developed and provide little or no support for understanding human, social and organisational factors that may influence the design of a system. While techniques such as use-cases represent an important recognition of the importance of people, there is still the key issue of determining where use-cases come from , what are critical use-cases, etc. His talk presented an overview of a method called Coherence that has been specifically designed to support social analysis of a work setting and to represent this analysis in UML. The motivation for this work was a conviction of the importance of social analysis and the need to take this to the software engineering community in terms that they could understand. The outcome of the social analysis is a set of use-cases that can then be the starting point for more detailed object-oriented analysis.

2.5. Frameworks and Patterns

Nathalie Gaertner, Alexandru Telea, Markus Hof and Aimar Marie led the discussion in this session. Nathalie Gaertner presented her experiences with working with business patterns and frameworks. She defined frameworks as generic applications, described by a set of abstract classes and the way instances of their subclasses collaborate. She pointed out that although frameworks allow a rapid development of new applications through customisation there are two main problems. First, designing a framework is a highly complex , time-consuming work and secondly, understanding the overall architecture and how to use it is difficult. She argued that one way to improve this situation is to include business and design patterns in the framework's architecture since each pattern provides a concise and useful architectural guidance to a related problem. Moreover, the reuse of patterns in software development allows the integration of flexible modular adaptable well-engineered solutions at a higher level than classes . Business patterns are domain-specific patterns. Integrating these patterns into frameworks - both related to the same business - makes it possible to exploit the generic architecture of frameworks along with the high level abstractions , business knowledge and documentation of the patterns. Nathalie presented a fuzzy logic control framework as an example to demonstrate the synergetic approaches of business patterns and frameworks. Alexandru Telea described the VISSION Simulation System which combines OO and dataflow modelling. He discussed that scientific visualisation and simulation (SimVis) is mostly addressed by frameworks using data and event flow mechanisms for simulation specification, control, and interactivity. Even though OO powerfully and elegantly models many application domains, integration of existing SimVis OO libraries in such systems remains a difficult task. The elegance and simplicity of the OO design usually gets lost in the integration phase, as most systems do not support the combination of OO and dataflow concepts. Practically no SimVis system addresses the needs of its component developers, application designers, and end users in a uniform manner. His proposed solution, VISSION, is a general-purpose visualisation and simulation OO system which merges OO and dataflow modelling in a single abstraction. This abstraction, called a metaclass, extends non- intrusively a C++ class with dataflow notions such as data inputs, outputs, and update operation, to promote it to a higher, more reusable level . VISSION uses a C++ interpreter to execute glue code that connects the metaclasses representing the system's components. Components can be loaded, instantiated and connected dynamically without recompiling or relinking VISSION. The needs of the three user groups mentioned above are addressed extensively and uniformly. Component designers get the full power of C++ to design new components or reuse existing C++ class libraries without having to change them. Application designers get a graphics user interface (GUI) in which component iconic representations can be assembled to build the desired SimVis application as a dataflow network. End users can easily steer a running simulation by the GUIs that VISSION automatically constructs for each component, or by typing C or C++ code that is interpreted dynamically . Alex also presented screenshots of several simulations and visualisations successfully constructed in VISSION.

Markus Hof presented a framework for arbitrary invocation semantics. He first discussed how most object-oriented languages for distributed programming offer either one fixed invocation semantic (synchronous procedure call), or a limited number of invocation semantics. At best, they support a default mode of synchronous remote invocation, plus some keywords to express asynchronous messaging. The very few approaches that offer rich libraries of invocation abstractions usually introduce significant overhead and do not handle the composition of those abstractions. He then described an approach for abstracting remote invocations. Invocation semantics, such as synchronous, asynchronous, transactional, or replicated are all considered first class abstractions. Using a combination of the Strategy and Decorator design patterns, he suggested an effective way to compose various invocation semantics. This technique allows different semantics on different objects of the same class. It is even possible to have several different views of one and the same object simultaneously. To reduce the overhead induced by the flexibility of the approach, just-in-time stub generation techniques are used. With the help of the semantic information supplied by the programmer, the necessary stub and skeleton code pieces are generated only on demand. This allows for late optimisations and adaptations. The work distinguished between two kinds of invocation abstractions . First, actual abstractions responsible for the execution of the method (synchronous, asynchronous, delayed, etc), and second, invocation filters that decorate an abstraction or other filters (at-most-once, transactional, logging, etc). Aimar Marie 's discussion focused on problems linked to the design of medical diagnostic systems. She pointed out that nowadays, computer-aided systems cannot be black boxes which contain a monolithic process. Systems must contain all components useful to store information, to search for a specific disease, to consult validated clinical tables and to compare the results for several diseases. This suggests the production of strongly inter-connected process modules, sharing a common database of information. Sharing information is possible when the knowledge base is structured regardless of the treatment used. She has adopted an object-oriented architecture to design the knowledge base and a generic model of collaboration between several treatment modules. In this context, she has tested how the use of patterns can help develop such a model and to improve the design of the system. In her opinion the different problems to solve are: to model in the same way pathologies which have heterogeneous signs, to identify generic behaviour into the various procedures of treatment and to design an interface for these procedures to guarantee the communication through the system. She has added to the object model, four patterns which give a solution to these problems. The pattern Composite keeps hidden the complexity of signs and allows treating all of them as simple sign. The pattern Iterator is used to define the generic task common to all diagnostic procedures to access the description elements and give them to a specific diagnostic engine. The pattern State saves the information of "presence" or "absence" of signs without taking into account which treatment is done, numerical calculus, symbolic evaluation and so on. Finally, the pattern Strategy defines a class of reasoning method, all diagnostic procedures are design to respect this interface. The four patterns define four strategic points of the diagnostic system architecture which are not given by the semantic analysis of the domain.

2.6. Aspect Oriented Programming

Gregor Kiczales gave a talk on aspect-oriented programming and the AspectJ tool, an aspect-oriented extension to Java. Using the SpaceWar Game as an example, he explained how 'cross cutting concerns' can appear across other modularised components of a system. For example, issues of game 'look and feel' can be spread amongst many otherwise loosely coupled classes. Given that such cross cutting concerns are inevitable, and that they cause tangled code and difficulties in maintenance, we can usefully modularise them into 'aspects '. An aspect encapsulates a cross cutting concern, 'introducing' fields and methods to classes and 'advising' (extending) existing processes . Gregor went on to describe how aspect orientation is at a stage where empirical testing is required along with theoretical and practical developments analysis to prove its validity and usefulness. Building a user community is essential to researching this approach and showing what results it can produce. Specific issues for research include software engineering (finding aspects, process, program modularity), language design (support for both static and dynamic cross-cuts), tools (programming environments, aspect discovery, refactoring) and theory (language and program semantics, cross-cutting). Gregor concluded his talk with a question and answer session, describing various characteristics of syntax and comparing his work with other approaches such as subject oriented programming.

2.7. Distribution and Middleware

In this session presentations were made by Fabio Costa and Christoph Peter. Fabio Costa talked about middleware platforms, an effective answer to the requirements of open distributed processing. However, in his opinion existing middleware standards do not fulfil important requirements of new application areas like multimedia and mobile computing, which require dynamic adaptability of the underlying platform. He was of the view that such requirements can be met by the adoption of an open engineering approach, based on computational reflection. Reflection offers a principled way to open up both the structure and the behaviour of a system by providing a causally connected self-representation of its implementation, and allowing its inspection and manipulation. He presented his ongoing research on the design and implementation of a reflective architecture for multimedia middleware, which allows the run-time reconfiguration of the components and services of the platform . The design is based on a multi-model reflection framework, whereby the different aspects in the engineering of the platform are identified and each one is represented by a distinct and orthogonal meta-model. There are currently four such meta-models:

- encapsulation (exposes the constitution of interfaces)
- composition (represents the configuration of compound components)
- environment (exposes the mechanisms for message handling at interfaces boundaries)
- resource management (represents the reservation and allocation of resources)

At run-time, meta-objects of any of these meta-models can be created and assigned to individual components of the platform. This makes explicit the corresponding aspect and allows the programmer or some controlling mechanism to dynamically reconfigure the internals of the platform. Christoph Peter argued that languages like C++ and Java have shown that strong, static typing is a good basis for programming. In distributed environments, there are well known calculi like the ss-calculus or Vasconcelos' TyCO. Static type systems for these calculi also exist. But their expressiveness is limited, as none of them can express the sequencing of messages which is an important part of the behaviour of objects. He suggested use of (Static) Process Types, based on a concept which allows to express the sequencing of messages in the type information. This is done by providing changeable state information in the types. When a message is sent, information about the state change of the message's receiver is provided by the type. The sender of the message can update its information about the state of the receiver. His presentation concentrated on applications of the process type concept:

- Examine a matching relationship for process types: Process types provide the subtyping relationship and genericity for re-use. But binary methods can be re-used only with a higher-order subtyping mechanism like matching.
- Using process types for deadlock detection: With asynchronous message passing, an object is blocked while waiting for an answer (message) from another object. If there are cycles of blocking objects, a deadlock occurs. An extension of process types allows to express that a request shall imply an answer. This property can be guaranteed statically and therefore, an important reason for deadlocks can be detected.
- Integrate process types into CORBA: The IDL of CORBA provides little information about the behaviour of an object. A goal of Christoph's research is to examine what possibilities of process types may be used to enhance the interface information but still provide static type checking.

3. Workshop Discussions

Three questions of key interest to the workshop participants were identified during the various sessions. These were as follows:

- OODBMS: Industrial failure or next generation technology?
- Using meta-information
- New approaches in object orientation - Where will we be tomorrow?

The above questions were discussed by the interested participants who arrived at the following conclusions.

3.1. OODBMS: Industrial Failure or Next Generation Technology?

Discussion Participants: Radovan Chytracek, Marlon Dumas, Anthony Lauder, Awais Rashid, Juan Trujillo

Object-oriented programming languages, systems, and methodologies have experienced tremendous industrial success. Object-Oriented Database Management Systems (OODBMS), however, are lagging far behind relational DBMSs in at least three dimensions: market penetration, sales revenue, and consumer awareness. The purpose of the discussion was to elucidate some of the reasons behind this apparent ”industrial failure” and to draw some possible conclusions on the subject . In the discussion, participants considered OODBMS to be those which provide classical DBMS services (persistence, transactions , concurrency, etc), under a data model supporting the basic concepts of currently used object-oriented languages (e .g. Java, C++). In particular, Versant, ObjectStore, O2 and Jasmine are considered to be OODBMS while Oracle v8 is not, since it supports neither inheritance nor encapsulation. Some of the OODBMS suppliers (e.g. ObjectStore and Jasmine) are not actually experiencing any commercial or industrial failure (in a financial sense). Nevertheless their visibility and market penetration remain limited. As a result, their long-term prospects are not very clear. Other OODBMS suppliers are currently experiencing severe financial and/or commercial problems. OODBMS suppliers emphasize object-orientedness, and its benefits over the relational paradigm. Typical claimed benefits include reduction of the impedance mismatch between the programming languages and the DBMS, performance advantages (due to navigation from roots and sophisticated caching and swizzling technologies), and transparent support for complex user-defined types. This latter feature has actually enabled OODBMSs to make major in-roads in some niche markets around specialized fields needing complex data such as computer-aided design and computer-aided software engineering. On the other hand, RDBMS suppliers emphasize scalability, reliability, security, and other hard-won DBMS features that are currently (arguably) missing in OODBMS. As a result, RDBMSs have gained (and continue to gain) massive penetration into almost all markets. Furthermore, RDBMS are currently trying to integrate some OO features into their products. Although this will not transform them into fully-fledged OODBMS, it will reduce to some extent their limits regarding complex data management. From these observations we may say that OODBMSs, in contrast to RDBMSs, are a relative failure (compared to early promises in terms of market penetration). There are two viewpoints with respect to what the future may bring. The "optimistic" one states that OODBMS are simply "hibernating", and that their time will come. The contrasting view is that, in the future, a major shakeout will occur, eliminating all but a few OODBMS suppliers addressing small niche markets , with no major penetration ever occurring. Below we enumerate some of reasons underlying the current industrial failure of ODBMSs, and it is, we believe, the way in which these issues are addressed by ODBMS vendor that will determine their future prospects:

– There are many OODBMS suppliers, whereas the RBDMS marketplace has undergone waves of mergers, buyouts, and bankruptcies, leaving a small number of players. This leaves potential buyers of ODBMSs with the uneasy feeling that a similar future shakeout could eliminate any supplier they committed to now.
– Despite the existence of an OMG-sanctioned supposed standard, detailed studies have revealed many inconsistencies within it, and each of the OODBMS companies has followed their own charter leading to portability problems across products.
– Strong OODBMS research in academic labs has had limited commercial pickup, so that OODBMS vendors are having to resolve fundamentally difficult issues on very tight budgets with limited resources.
– OODBMSs lack important features required by industry: e.g. performance with large amounts of data (this is a contentious issue), security, and scalability.

- RDBMSs have proven their reliability though hard-won debugging in the field over many years. OODBMSs have not yet gone through this, leading to uncertainty over their reliability.
- Users are already committed to an RDBMS, with their own layers to resolve impedance mismatches, with mountains of application code, and with staff highly trained and experienced in RDBMS technology.
- It is not clear to what extent the touted advantages of OODBMSs are needed for mainstream applications.

3.2. Using Meta-Information

Discussion Participants: Stefan Chiettini, Akos Frohner, Markus Hof, Christoph Steindl, Alexandru Telea.

In the past, systems have been monolithic and static. In the future, systems will be modular and dynamic. The use of meta-information helps to perform this evolutionary step or is an enabling technology for it. When we look at the historical development of programming languages, we see a continuing rise of the level of abstraction of machine details: from machine instructions, via assembly languages and higher programming languages to structured and object-oriented languages. The next step in this development is platform independence. Combined with network transparency, it allows writing programs that can run on any computer in a networked environment. The composition of new software out of existing components is another promising application area: The programmer composes programs in a visual programming environment by sticking components together or glues them together using scripting languages. The components shall then cooperate, which they only can do if they have some knowledge about each other: their interfaces, properties, facilities and so on. Meta-information seems to be the bridge between abstraction on one hand and knowledge about each other on the other hand. Meta-information makes some knowledge explicit that was previously only implicit. It is also a means to make information available at run time that was usually only available at compile time. Meta-information is also the key to systems that are really extensible where only the required components are active at a time and where additional functionality can be added on demand. Meta-programming can exploit meta-information to several degrees. It can use metainformation to:

- observe and manipulate itself and other running programs (introspection).
- explicitly call functionality that is normally hidden in the run-time system, e.g. creation of new objects, dynamic loading, linking, and unloading of components (interception).
- change the behaviour of language primitives at run time, e.g. object creation and destruction, method dispatch, and access to simple attributes (invocation) The participants agreed that it will be crucial for every computing system and especially programming language to offer a standardized access to meta-information. Many do so already (Lisp, Smalltalk, CLOS, Beta,

Oberon-2, Java), and the implications and principles are apparently well understood. However, in the participants' view it is vital that this access is efficient in memory usage as well as in its run-time behaviour. They saw a wide field for further research and projects in this area: retrieving meta-information from legacy systems in an automatic or semiautomatic way; extending popular languages to handle meta-information; creating visual environments to support various aspects of this field.

3.3. New Approaches in Object Orientation - Where will we be tomorrow?

Discussion Participants: Andreas Speck, Fabio Kon, Ian Oliver, Aimar Marie, Moritz Schnizler, John Flackett, Martin Geier (guest participant)

Currently new trends in object-orientation are rising such as aspect-oriented programming (introduced by G. Kiczales' and C. Lopes' group at Xerox PARC), the component generators (U. Eisenecker, K. Czarnecki and D. Batory's work), component-based approaches (C. Szyperski), intentional programming (C. Simonyi) and adaptive programming (K. Lieberherr). The states of these approaches are quite different. While components are already in use and supported by many commercial systems (e.g. CORBA implementations, COM or Java Beans) others are still in evaluation. This great variety of approaches leads to many questions: What is their impact in the future? Do they bear interesting research challenges? Which of them will supersede in the future? Within the PhDOOS community many PhD candidates are doing research in this area. Special crucial points are aspect-oriented programming (especially in connection with object distribution), reflective architectures, component generators, component-based software development including the development of generic architectures for component-based frameworks, and dependence management in component-based distributed systems, dynamically configurable middleware systems as well as secure ports for inter-component communication.

4. Acknowledgements

We gratefully acknowledge the financial support of AITO and Object Technology International Inc. This support made possible the presentce of many participations at the workshop, as well as the initiative to produce separate workshop proceedings containing the full versions of the presented papers.

5. Participants List

1. Stefan Chiettini, Institut fur Praktische Informatik, Johannes Kepler University Linz, A-4040 Linz, Austria, email: stefan.chiettini@ssw.uni-linz.ac.at
2. Radovan Chytracek, CERN, Geneva, Switzerland, email: Radovan.Chytracek@cern.ch
3. Fabio Costa, Computing Department, Lancaster University, Lancaster LA1

4YR, UK, email: fmc@comp.lancs.ac.uk
4. Marlon Dumas, LSR-IMAG Lab, University of Grenoble, France, email: Marlon.Dumas@imag.fr
5. John C. Flackett, Systems Engineering Faculty, Southampton Institute, East Park Terrace, Southampton, SO14 0YN, UK, email: John.Flackett@solent.ac.uk
6.Akos Frohner, Eotvos Lorand University, Institute of Informatics, Budapest, email: Akos.Frohner@elte.hu
7.Nathalie Gaertner, Laboratoire EEA, Groupe LSI, Universite de Haute-Alsace, 12 rue des freres Lumiere, 68093 Mulhouse Cedex, France, email: n.gaertner@evhr.net
8.Markus Hof, Institut fur Praktische Informatik, Johannes Kepler University Linz, A-4040 Linz, Austria, email: hof@ssw.uni-linz.ac.at
9.Fabio Kon, Department of Computer Science, University of Illinois at Urbana Champaign, USA, email: f-kon@uiuc.edu
10. Anthony Lauder, Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent, CT2 7NF, UK, email: Anthony@Lauder.u-net.com
11. Glenn Lewis, Electrical Engineering and Computer Science, University of Tasmania, email: Glenn.Lewis@utas.edu.au
12. Marie Beurton-Aimar, University of Bordeaux, France, email: Marie.Aimar@dim.ubordeaux2.fr
13. Isabella Merlo, Dipartimento di Informatica e Scienze dell'Informazione, University of Genova, Italy, email: merloisa@disi.unige.it **14.**Ian Oliver, University of Kent at Canterbury, England, UK, email: ian.oliver@bcs.org.uk
15. David Parsons, The Object People, Epsilon House, Chilworth Science Park, Southampton SO16 7NS, UK, email: davidp@objectpeople.com
16. Christof Peter, Technische Universitat Wien, Institut fur Computersprachen, Argentinierstrasse 8, A-1040 Vienna, Austria, email: fchristofg@complang.tuwien.ac.at
17. Awais Rashid, Computing Department, Lancaster University, Lancaster LA1 4YR, UK, email: marash@comp.lancs.ac.uk
18. Moritz Schnizler, Aachen University of Technology, Department of Computer Science III, Software Construction Group, Ahornstr. 55, 52074 Aachen, Germany , email: moritz@informatik.rwth-aachen.de
19. Cristian Sminchisescu, Department of Computer Science, Rutgers University, USA, email: crismin@paul.rugers.edu
20. Andreas Speck, Wilhelm-Schickard-Institut fur Informatik, Universitaet Tuebingen, 72076 Tuebingen, Germany, email: speck@informatik.uni-tuebingen.de
21. Christoph Steindl, Institut fur Praktische Informatik, Johannes Kepler University Linz, A-4040 Linz, Austria, email: steindl@ssw.uni-linz.ac.at
22. Alexandru Telea, Department of Mathematics and Computing Science, Eindhoven University of Technology, Den Dolech 2, 5600 MB Eindhoven, The Netherlands , email: alext@win.tue.nl
23. Juan Trujillo, Research Group of Logic Programming and Information Systems, Dept. of Financial Economics, University of Alicante, E-03071, Alicante, Spain , email: juan.trujillo@ua.es
24. Martin Geier, University of Erlangen/Nuremberg, Germany (guest participant)