06302 Abstracts Collection
Aspects For Legacy Applications
— Dagstuhl Seminar —

Siobhán Clarke[1], Leon Moonen[2] and Ganesan Ramalingam[3]

[1] Trinity College - Dublin, IE
Siobhan.Clarke@cs.tcd.ie
[2] CWI - Amsterdam, NL
leon.moonen@computer.org
[3] IBM TJ Watson Research Center, US
grama@microsoft.com

Abstract. From 26.07.06 to 29.07.06, the Dagstuhl Seminar 06302 "Aspects For Legacy Applications" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Aspect orientation, software evolution, program analysis, reverse engineering, aspect identification, software reengineering

06302 Summary – Aspects For Legacy Applications

This paper provides a summary of the objectives, structure, and the outcome of Dagstuhl seminar #06302 on Aspects For Legacy Applications, held from July 26th to July 29th 2006 at Schloss Dagstuhl, Germany. The goal of the seminar was to bring together researchers from the domains of aspect oriented software development, software reengineering (with a focus on reverse engineering, program comprehension, software evolution and software maintenance) and program analysis to investigate how aspects can help us to understand, maintain, and transform legacy software systems.

Keywords: Aspect orientation, software evolution, program analysis, reverse engineering, aspect identification, software reengineering

Joint work of: Moonen, Leon; Ramalingam, Ganesan; Clarke, Siobhán

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/879

Face-off:AOP+LMP vs. legacy software

Bram Adams (Gent University, B)

Our presentation relates on a first attempt to see if aspect-oriented programming (AOP) can really help with the revitalisation of legacy business software. By means of four realistic case studies covering reverse engineering, restructuring and integration, we discuss the applicability of the aspect-oriented paradigm in the context of two major programming languages for such environments: Cobol and C. For each case, we consider both advantages and disadvantages.

Keywords: AOP, legacy software, reverse-engineering, re-engineering, Cobol, C

Joint work of: Adams, Bram; De Schutter, Kris

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/888

Mining Aspects from Version History

Silvia Breu (Cambridge University, GB)

As software evolves, new functionality sometimes no longer aligns with the original design, ending up scattered across a program. Aspect mining identifies such cross-cutting concerns in order to then help migrating a system to a better design, maybe even to an aspect-oriented design. We address this task by applying formal concept analysis to a program's history: method calls added across many locations are likely to be cross-cutting. By taking this historical perspective, we introduce a new dimension to aspect mining. As we only analyse changes from one version to the next, the technique is independent of a system's total size and scales up to industrial-sized projects such as Eclipse.

Keywords: Aspect mining, formal concept analysis, mining software repositories

Joint work of: Breu, Silvia; Zimmermann, Thomas

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/880

Full Paper:
http://www.cl.cam.ac.uk/ sb586/Site/Publications%20:%20ASE%202006_files/ase2006.pdf

See also:

See also: Silvia Breu, Thomas Zimmermann, Christian Lindig. Mining Eclipse for Cross-Cutting Concerns. In Proceedings of the Third International Workshop on Mining Software Repositories (MSR 2006), Shanghai, China, May 2006, pp. 94-97.

HAM: Cross-cutting Concerns in Eclipse

Silvia Breu (Cambridge University, GB)

As programs evolve, newly added functionality sometimes does no longer align with the original design, ending up scattered across the software system. Aspect mining tries to identify such cross-cutting concerns in a program to support maintenance, or as a first step towards an aspect-oriented program. Previous approaches to aspect mining applied static or dynamic program analysis techniques to a single version of a system.We leverage all versions from a system's CVS history to mine aspect candidates with our Eclipse plug-in HAM: when a single CVS commit adds calls to the same (small) set of methods in many unrelated locations, these method calls are likely to be cross-cutting. HAM employs formal concept analysis to identify aspect candidates. Analysing one commit at a time makes the approach scale to industrial-sized programs. In an evaluation we mined cross-cutting concerns from Eclipse 3.2M3 and found that up to 90% of the top-10 aspect candidates are truly cross-cutting concerns.

Keywords: Aspect Mining, Aspect-Oriented Programming, CVS, Eclipse, Formal Concept Analysis, Java, Mining Version Archives

Joint work of: Breu, Silvia; Zimmermann, Thomas; Lindig, Christian

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/884

Mining Additions of Method Calls in ArgoUML

Silvia Breu (Cambridge University, GB)

In this paper we refine the classical co-change to the addition of method calls. We use this concept to find usage patterns and to identify cross-cutting concerns for ArgoUML.

Keywords: Aspect Mining, Aspect-Oriented Programming, CVS, Eclipse, Formal Concept Analysis, Java, Mining Version Archives

Joint work of: Breu, Silvia; Zimmermann, Thomas; Lindig, Christian; Livshits, Benjamin

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/886

Mining Eclipse for CrossCutting

Silvia Breu (Cambridge University, GB)

Software may contain functionality that does not align with its architecture. Such cross-cutting concerns do not exist from the beginning but emerge over time.

By analysing where developers add code to a program, our history-based mining identifies cross-cutting concerns in a two-step process. First, we mine CVS archives for sets of methods where a call to a specific single method was added. In a second step, simple cross-cutting concerns are combined to complex cross-cutting concerns. To compute these efficiently, we apply formal concept analysis-an algebraic theory. Unlike approaches based on static or dynamic analysis, history-based mining for cross-cutting concerns scales to industrial-sized projects: For example, we identified a locking concern that cross-cuts 1284 methods in the open-source project Eclipse.

Joint work of: Breu, Silvia; Zimmermann, Thomas; Lindig, Christian

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/885

Info

Magiel Bruntink (CWI - Amsterdam, NL)

Hi, I'm with the CWI in Amsterdam. I work in a project focussing on the improvement of crosscutting concerns in legacy software. We're cooperating with a manufacturing company in Holland (ASML) which maintains a huge (15 MLOC) C sourcebase. That system is suffering from crosscutting concerns that have been implemented idiomatically, i.e., manually based on non-formal descriptions. Problems include faulty error handling, code duplication and inconsistencies. I'm interested in hearing other people's experiences with dealing with crosscutting concerns in industrial settings, especially reports on actual code migrations towards AOP. I'll also give a talk myself sharing our experience with migrating a trivial-looking-but-not-quite-so-trivial concern, that is, tracing.

Some of our previous work on the subject:
Discovering Faults in Idiom-based Exception Handling (ICSE06)
http://homepages.cwi.nl/~bruntink/papers/icse2006.pdf
Isolating Idiomatic Crosscutting Concerns (ICSM05)
http://homepages.cwi.nl/~bruntink/papers/icsm05.pdf
See you at Dagstuhl!
Magiel

Adding Monitoring, Tuning and Autonomic Control Aspects to Legacy Applications in the Absence of AOP

James R. Cordy (Queens University - Kingston, CA)

One part of autonomic computing is the ability to identify, separate and automatically tune parameters related to performance, security, robustness and other properties of a software system. Often the response to events affecting these properties consists of adjusting tunable system parameters such as table

sizes, timeout limits, restart checks and so on. One can think of these tunable parameters as a set of knobs that can be tweaked or switched to adapt the system to environmental or usage changes. In many ways these tunable parameters correspond to the switches and potentiometers on the control panel of many hardware devices.

While modern software systems designed for autonomic control may make these parameters easily accessible, in legacy systems they are often scattered or deeply hidden in the software source, cross-cutting both structure and semantics of the application. Software Tuning Panels for Autonomic Control (STAC) is a system for automatically re-architecting legacy software systems to separate this aspect and facilitate autonomic control. STAC works to isolate tuneable system parameters into one visible area of a system, producing a resulting architecture that can be used in conjunction with an autonomic controller for self-maintenance and tuning.

If AOP were available, then an obvious solution to this challenge would be to use AOP to specify and weave the monitoring, tuning and control aspect into the legacy program. However, our solution is constrained to work without AOP, implementing the aspect using traditional pure Java contructs instead. Somewhat surprisingly, the non-AOP implementation of this traditional aspect-oriented example turns to be not only feasible, but to have significant engineering advantages over the AOP solution as well, leading to the question of whether aspects may work better as a design concept than as a programming technique.

References

E. Dancy and J.R. Cordy, "STAC: Software Tuning Panels For Autonomic Control", Proc. CASCON'06, 16th IBM Centre for Advanced Studies International Conference on Computer Science and Software Engineering, Toronto, October 2006, 15 pp. (to appear)

Joint work of: Cordy, James R.; Dancy, Elizabeth

Recovering Code Categories from Legacy Source Code

Rui Correia (University of Leicester, GB)

A short overview over a methodology for reengineering Software Systems and change its architecture was presented. After that, a more detailed explanation was given about our existing code categories and the ongoing work on how to categorize Legacy Source Code. Finaly, some future wrok ideas and the hope that Aspect Mining tecnhiques can be used in our work was expressed.

Rule-based Model Extraction from Source Code

Rui Correia (University of Leicester, GB)

In the context of an approach for reengineering legacy software systems at the architectural level, we present in this paper a reverse engineering methodology that uses a model defined as a type graph to represent source-code subject to a code categorization process. Two alternative methods for referencing the source code are discussed: native vs. graphical. To represent the code, the native representation uses the abstract syntax tree while the graphical uses a programming language metamodel. Two options regarding the way that the graph can relate to the source code reference model are also considered: association model vs. direct link. The extraction of the program representation, complying to the type graph, is based on rules that categorize source code according to its purpose. The techniques to address this process, such as the code categorization rules, are shown together with examples.

Keywords: Reverse engineering, Code categorization, Program representation

Joint work of: Correia, Rui; Matos, Carlos; El-Ramly, Mohammad; Heckel, Reiko

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/881

Toward a Compositional Model for Data Update/Query in Reactive Applications: A Datalog-Based Approach

John H. Field (IBM TJ Watson Research Center - Hawthorne, USA)

We describe preliminary work on candidate constructs for data query and data update in _Collage_, a programming model under development at IBM Research. Collage is intended for implementation of distributed, event-driven ("reactive") applications, e.g., web commerce sites, supply chain management, portals/aggregators, auctions, multi-party financial transactions, or workflow. Collage is organized around a collection of autonomous processes, or _services_. The state of a service is updated in response to external _events_, such as pushing a button on a web form, or receiving a message from a remote process.

Normally, independently-specified state updates are difficult to compose due to sequencing and data dependence issues. In this talk, we describe a variant of datalog that allows composition of independently-developed rules defining distinct aspects of an application's behavior. In addition to defining reactive state updates in a compositional way, rules can also be used to specify invariant properties of a service state. Such properties are either checked after every state update, or maintained constructively.

Joint work of: Field, John; Marinescu, Maria-Cristina; Stefansen, Christian

Issues and Topics for AOP in the Context of Model-Driven Software Engineering

Kostas Kontogiannis (University of Waterloo, CA)

Model-driven Development (MDD) and Model-driven Evolution (MDE) are emerging areas in the software engineering community. Techniques pertaining to the formalization of models for various software artifacts at different levels of abstraction (requirements, architecture, design, code, testing), the infrastructure for model transformation and model synchronization, as well as, techniques for code generation from higher level of abstraction models, are issues currently being investigated.

In this respect, Aspect Oriented Programming (AOP) is posing new challenges in the context of MDD and MDE. Some of the topics and issues to discuss in this workshop could relate to the problems stemming from where and how Model-driven techniques meet AOP for reverse engineering, and reengineering purposes.

More specifically, interesting issues that arise from considering MDD/MDE within the context of Aspect Oriented Programming, include the recovery and modeling of Aspects from legacy systems, the transformation of Aspect models to new forms to support evolution or reengineering, consistency checking of Aspect models while these evolve as consequence of maintenance or evolution activities

Keywords: Model Driven Software Software Evolution, Model Driven Software Development, Model Tranformations, Aspect Oriented Programming

Mining Control Flow Graphs for Crosscutting Concerns (But Finding Delegation)

Jens Krinke (FernUniversität in Hagen, D)

There exist some approaches that analyse execution relations to identify crosscutting concerns. Execution relations express some recurring invocation pattern between two methods. Depending on the specific relation, the approaches mine specifically for join points that are already present in the system. However, all approaches need some kind of filter to distinguish crosscutting delegation from crosscutting concerns. A simple filter that ignores non-void methods results in much better results with a low number of false positives. In a case study based on JHotDraw, a set of identified crosscutting concerns are discussed and compared with previous work. Delegation has a major role in aspect mining and the relation of delegation to crosscutting concerns can be compared to the relation of delegation to inheritance.

Keywords: Aspect mining

abstract

Kiarash Mahdavi (King's College - London, GB)

Background and Interests:

My background is in Software clustering as a means for software analysis and comprehension. I also have an interest in Heuristic search algorithms. I currently work as a Research Associate at Kings College London, UK, under the CONTRACTS project(Concept Assignment to Raise the Abstraction Level of Slicing). This project is headed by Professor Mark Harman and Dr Nicolas Gold. It involves exploring comprehension techniques such as Concept Assignment as a means to improve Slicing/Slicing criteria.

Aspects and concepts in software seem closely related, therefor one of our current interest is in finding ways to relate these two ideas. We think this maybe achieved by finding ways to identify aspects in software by using Concept Assignment techniques or Alternatively by employing Aspect identification techniques to help us more accurately find Concepts in software.

Questions:

-Is there a Relationships between Concept and Aspects in code?

-If so, can this relationship be used to help with identification of aspects or concepts?

-Further more can techniques used to identify aspects and concepts be combined to create more powerful concept or aspect identification techniques? The following refrences provide a good indication of our overall view and interest in CONTRACTS:

Useful References:

[1] Mark Harman and Nicolas Gold and Robert M. Hierons and David Binkley Code Extraction Algorithms which Unify Slicing and Concept Assignment WCRE, pp. 11-21, IEEE Computer Society, 2002.

[2] N. E. Gold and M. Harman and D. Binkley and R. M. Hierons Unifying program slicing and concept assignment for higher-level executable source code extraction, SoftwareÛPractice and Experience, 35(10), pp. 977-1006, August 2005.

[3] Nicolas Gold and Mark Harman and Zheng Li and Kiarash Mahdavi Allowing Overlapping Boundaries in Source Code using a Search Based Approach to Concept Binding to be published in the upcoming ICSM 2006 conference

[4] Dave Binkley and Nicolas Gold and Mark Harman and Zheng Li and Kiarash Mahdavi An Empirical Study of Executable Concept Slice Size to be published in WCRE 2006

The following are related to my previouse work in software clustering:

1. Kiarash Mahdavi and Mark Harman and Robert M. Hierons A Multiple Hill Climbing Approach to Software Module Clustering ICSM, pp. 315-324, IEEE Computer Society, 2003.

2. Kiarash Mahdavi and Mark Harman and Robert Hierons Finding Building Blocks for Software Clustering, Genetic and Evolutionary Computation Ű GECCO-2003, pp. 2513-0, Springer-Verlag, 2003.

3. Mark Harman and Stephen Swift and Kiarash Mahdavi An empirical study of the robustness of two module clustering fitness functions GECCO, pp. 1029-1036, ACM, 2005.

Steps towards Consistent Mining and Migration of Crosscutting Concerns

Marius Marin (TU Delft, NL)

Crosscutting concerns are often defined and explained through a variety of examples ranging from logging to design patterns, to business rules, to policy enforcement, etc. These examples illustrate crosscutting behavior at different levels of granularity and give a rather intuitive definition of the crosscutting concerns. Such heterogeneous examples and implicit definition reflect on aspect mining and migration efforts, as they lack required consistency in describing (common) findings and wider applicable solutions.

This presentation proposes a discussion about how to ensure consistency in defining and describing crosscutting functionality, about how to provide compatibility between aspect mining results so we can compare and combine these results and aspect mining techniques. It builds on previous and ongoing efforts on identifying common idioms in implementation of crosscutting concerns, and describing these idioms as atomic, generic concerns, called *sorts*.

Keywords: Aspect identification; crosscutting concern sorts; framework

Joint work of: Marin, Marius; Moonen, Leon; van Deursen, Arie

Full Paper:
http://arxiv.org/ftp/cs/papers/0606/0606113.pdf

A common framework for aspect mining based on crosscutting concern sorts

Marius Marin (TU Delft, NL)

The increasing number of aspect mining techniques proposed in literature calls for a methodological way of comparing and combining them in order to assess, and improve on, their quality. This paper addresses this situation by proposing a common framework based on crosscutting concern sorts which allows for consistent assessment, comparison and combination of aspect mining techniques. The framework identifies a set of requirements that ensure homogeneity in formulating the mining goals, presenting the results and assessing their quality.

We demonstrate feasibility of the approach by retrofitting an existing aspect mining technique to the framework, and by using it to design and implement two new mining techniques. We apply the three techniques to a known aspect mining benchmark and show how they can be consistently assessed and combined to increase the quality of the results. The techniques and combinations are implemented in FINT, our publicly available free aspect mining tool.

Keywords: Aspect identification, crosscutting concern sorts, framework

Joint work of: Marin, Marius; Moonen, Leon; van Deursen, Arie

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/882

Delving source code with formal concept analysis

Kim Mens (Univ. cath. de Louvain, B)

Getting an initial understanding of the structure of a software system, whether it is for software maintenance, evolution or reengineering purposes, is a nontrivial task.

We propose a lightweight approach to delve a system's source code automatically and efficiently for relevant concepts of interest: what concerns are addressed in the code, what patterns, coding idioms and conventions have been adopted, and where and how are they implemented. We use formal concept analysis to do the actual source-code mining, and then filter, classify and combine the results to present them in a format that is more convenient to a software engineer. We applied a prototype tool that implements this approach to several small to medium-sized Smalltalk applications. For each of these, the tool uncovered several design pattern instances, coding and naming conventions, refactoring opportunities and important domain concepts. Although the tool and approach can still be improved in many ways, the tool does already provides useful results when trying to get an initial understanding of a system. The obtained results also illustrate the relevance and feasibility of using formal concept analysis as an efficient technique for source code mining.

Keywords: Source-code mining, formal concept analysis, software classification.

Joint work of: Mens, Kim; Tourwé, Tom

Full Paper:
http://www.info.ucl.ac.be/~km/MyResearchPages/publications/journal_article/JA_2005_CLSS.pdf

See also: K. MENS & T. TOURWE. Delving source code with formal concept analysis. Elsevier Journal on Computer Languages, Systems & Structures, 31(3-4) : 183-198. Special Issue: Smalltalk. Elsevier, October-December 2005. (Early draft of the published version.)

Aspect Mining : An emerging research domain

Kim Mens (Univ. cath. de Louvain, B)

This invited presentation offers a first, in-breadth survey and comparison of current aspect mining tools and techniques. It focuses mainly on automated techniques that mine a program's static or dynamic structure for candidate aspects. We present an initial comparative framework for distinguishing aspect mining techniques, and assess known techniques against this framework. The results of this assessment may serve as a roadmap to potential users of aspect mining techniques, to help them in selecting an appropriate technique. It also helps aspect mining researchers to identify remaining open research questions, possible avenues for future research, and interesting combinations of existing techniques

Keywords: Aspect mining

Cross-fertilisation between Software Evolution and AOSD

Tom Mens (Université de Mons, B)

My main research interest is the investigation of novel techniques, formalisms, mechanisms and tools to gain a better understanding of, and improve support for software evolution. (And this both at programming and modelling level.)

The main reason for me to attend this Dagstuhl Seminar is to establish a cross-fertilisation between the research domain of Software Evolution on the one hand, and AOSD on the other hand. In software evolution research, a whole range of formal techniques are being explored and used (graph transformation, description logics, dependency analysis, and many more).

It is likely that the same mechanisms can also be very helpful in AOSD research as well.

The other way around, techniques, tools and formalisms used in AOSD may be exploited to support software evolution. This can be summarised in the following two research questions:

- How and which techniques and formalisms that have already proven their use in software evolution research can we apply to an AOSD context?
- How and which techniques and formalisms that have already proven their use in AOSD research can we apply to a software evolution context?

The material I submitted are some papers I have written in the context of (model-driven) software evolution. During the seminar, I hope to be able to discuss on how these ideas can be applied in an AOSD context as well.

Another important issue that bothers me in AOSD research (and to a certain extent also in OO research) is its usefulness. Until now, I didn't find a single empirical study that effectively shows the benefits of AOSD as compared to other programmingg paradigms. Just like in the good old OO days, everyone

claims that aspects are supposed to make software more reusable, adaptable and evolvable, but I have my doubts that this is actually the case. (I am deliberately playing advocate of the devil here.) A scientific validation of these claims has not yet been made, so I think it is important for the community to undertake efforts to do this. After all, what is the reason of migrating to aspect-oriented software, if we don't even know whether it will be beneficial? This is a topic I would like to see discussed during the seminar as well.

Keywords: Software evolution, formal support, AOSD

Analysing dependencies between aspects (using transformation technology)

Tom Mens (Université de Mons, B)

Analysing dependencies between aspects, or between the crosscutting concerns they address is a major challenge in AOSD research. This is clearly exemplified by the recent call for submissions of a special issue for the Transactions on AOSD journal (see excerpt below).

In previous work, I have gained expertise in dependency analysis. In particular, I have explored the use of graph transformation theory and technology for the purpose of analysing dependencies between model transformations (in the context of inconsistency management) and refactoring. More specifically, the formal notion of critical pair analysis was used for this purpose (see attached papers).

It is my convictiopn that the same techniques can also be used and exploited in an AOSD context, in order to analyse and reason about dependencies between aspects.

=== EXCERPT FROM CALL FOR SUBMISSIONS === TRANSACTIONS ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT SPECIAL ISSUE ON DEPENDENCIES AND INTERACTIONS WITH ASPECTS

Special issue web site: http://www.aosd-europe.net/events/dia_taosd/

Crosscutting concerns exist throughout software development cycle - from requirements through to implementation. Dedicated modularization units representing these concerns are termed aspects. While crosscutting other concerns, aspects often exert broad influences on these concerns, e.g. by modifying their semantics, structure or behaviour. These dependencies among both aspectual and non-aspectual elements may lead to either desirable or (more often) unwanted and unexpected interactions. We encourage submissions investigating the problems of such dependencies and interactions and handling them at all levels:

- starting from the early development stages (i.e., requirements, architecture, and design), looking into dependencies among requirements (e.g. positive/ negative contributions between aspectual goals, etc.) and interactions or interference caused by aspects (e.g. quality attributes) in requirements, architecture, and design;

- analysing these dependencies and interactions both through modelling and formal analysis;
- considering language design issues which help to handle such dependencies and interactions (e.g. 'dominates' mechanism of AspectJ), and,
- studying such interactions in applications.

Keywords: Dependency analysis, critical pair analysis, aspects , model transformation, crosscutting concerns

Using AOSD technology to support UML language extensibility

Tom Mens (Université de Mons, B)

An interesting topic I would like to discuss the use of AOSD techniques to support metamodeling. In particular, I would like to see how AOSD can be used to better support the built-in language extensibility mechanism of the UML modeling language.

In the UML 2.0 infrastructure document, UML Profiles are defined as a standard language extension mechanism to extend or further constrain the UML syntax and semantics. In many ways, these profiles can be considered as a kind of aspects.

Indeed, each profile is a kind of independent stand-alone extension of the UML metamodel. But since profiles can be applied together, all of the problems apparent in combining and weaving aspects also appear when trying to combine profiles:

- profiles can be dynamically applied or retracted from a model
- rofiles can be dynamically combined
- it is not foreseen at profile definition time which profiles will be applied together
- the order in which profiles are applied may be important
- some profiles may be in mutual conflict and cannot be applied together

All of these issues appear to be very important, but are not addressed at all in the UML document. I believe that techniques exploited in AOSD research may also be applicable to address the above problem.

Keywords: UML profiles, metamodeling, language extensibility, modeling

The Problem with Legacy Applications

Hausi Müller (University of Victoria, CA)

In the first part of the talk, we briefly summarize major avenues of legacy systems research and highlight selected success stories. Despite all these research results, dealing with (legacy) software systems does not seem to get easier. In fact with the event of the Web, legacy systems seem to be morphing into systems of systems. According to a June 2006 SEI study, the software systems will evolve into Ultra-Large-Scale (ULS) systems and socio-technical ecosystems [http://www.sei.cmu.edu/uls]. The notion of an ecosystem connotes complexity, decentralized control, hard-to-predict reactions to disruptions, and difficulty of monitoring and assessment. Thus in many ways, legacy systems have already morphed into ecosystems.

The presentation then concentrates on the software complexity problem inherent in ULS systems and how to use autonomic computing technology to alleviate this problem.

References:

[1] L. Northrop et al., SEI; Ultra-Large-Scale Systems, 134 pages, June 2006 http://www.sei.cmu.edu/uls/
[2] A. Kluth. Information Technology, The Economist, Oct 28, 2004
[3] M. Shaw, Everyday Dependability for Everyday Needs, Procs 13th Inter. Symposium on Software Reliability Engineering, pp. 7-11,2002
[4] IBM Systems Journal, Special Issue on Autonomic Computing, 24(1), 2003 Kephart, Chess, IEEE Computer, 36(1):41-50, Jan 2003
[5] IBM Research AC Web Site; http://www.research.ibm.com/autonomic/
[6] IBM, An Architecture Blueprint for Autonomic Computing, White Paper, June 2005
[7] H. Müller, L. O'Brian, M. Klein, and B. Wood; Autonomic Computing, SEI Tech Report, 61 pages, June 2006
[8] M. Shaw, Beyond Objects: A Software Design Paradigm based on Process Control, ACM SEN, 20(11):27-38, 1995
[9] P. Bennett, In Rome's Basement, National Geographic, July 2006

Keywords: Legacy systems, systems of systems, ultra-large-scale systems, ecosystems, autonomic systems

Managing Concern Interfaces

Martin Robillard (McGill University - Montreal, CA)

Programming languages provide various mechanisms to support information hiding. One problem with information hiding, however, is that providing a stable interface behind which to hide implementation details involves fixing in advance

the services offered through the interface. We introduce a flexible approach to define and manage interfaces to achieve separation of concerns in evolving software.

Our approach involves explicitly specifying interface and implementation classes for individual concerns, and automatically classifying implementation classes based on their relation to the interface. Our approach is supported by JMantlet, a tool that provides advanced interface management within an integrated development environment. We report on a case study of a large system that provides evidence that flexible interface management is desirable and adequately supported by our approach.

Keywords: Separation of Concerns, Dependency Anaysis, JMantlet, Application Programming Interfaces

Joint work of: Boulanger, Jean-Sebastien ; Robillard, Martin

Full Paper:
http://www.cs.mcgill.ca/~martin/papers/icsm2006b.pdf

See also: Jean-Sébastien Boulanger and Martin P. Robillard. Managing Concern Interfaces. In Proceedings of the 22nd IEEE International Conference on Software Maintenance, pages 14-23, September 2006.

Aspects - from Promise to Reality

Sabhah, Daniel

The concepts underpinning aspect oriented software development have been with us for many years. The last couple of years have been particularly exciting, with much of the promise brought into sharp reality. The timing for our industry couldn't be more critical; urgent help is needed to address the growing software complexity crisis. Deployment of uniform implementations of cross-cutting concerns into a range of software products is now feasible, and large and complex software can be factored and recomposed into simpler, better targeted, higher quality offerings. In this talk we describe how IBM plans to put this technology into production to: simplify the delivery and service of high quality software, deliver new solutions for our customers' development requirements, create opportunities for customers to add value to their software, and to accelerate new initiatives at the heart of IBM's software strategy.

Mining Aspects in Legacy System Documentation

Americo Sampaio (Lancaster University, GB)

Aspect-Oriented Requirements Engineering (AORE) provides separation of concerns at the requirements level. In order to cope with concern identification and structuring into different requirements models, tool support is vital to effectively reduce the burden of performing various AORE tasks such as aspect mining.

The EA-Miner tool provides automated support for mining various types of concerns from a variety of early stage requirements documents (e.g., legacy specifications, user manuals, interview transcripts, etc.) and structuring these concepts into specific aspect-oriented requirements models (e.g., viewpoints-based, use-case-based).

The key insight for early-stage requirements automation is the use of natural language processing to reason about properties of the requirements as well as the utilization of semantics revealed by the natural language analysis in building the models.

Keywords: Early Aspects, Aspect Mining, AORE, NLP

Joint work of: Sampaio, Americo; Awais, Rashid

All about Andy...

Andy Zaidman (University of Antwerp, B)

Hi, I'm a PhD student at the University of Antwerp, Belgium. I'm in the last phew months of my PhD, after which I will move to the Delft University of Technology in the Netherlands.

My main research topic is trying to stimulate program comprehension by extracting and analyzing run-time information from systems. This work is carried out within the ARRIBA research project, where a number of universities and industrial partners try to learn from each others needs and solutions. In this framework we carried out an experiment in 2005 where we wanted to collect runtime information from an industrial legacy C system. For this we used aspects, more precisely Aspicere. This work was carried out in close collaboration with Bram Adams and Kris De Schutter from the Ghent University, both of whom are also attending this seminar.

I have two points of interest that are particularly relevant to this seminar, nl: - the problems that arise when you try to deploy an aspect solution in a legacy environment. How do you cope with legacy compilers, with the build process, etc. - how to cope with a distributed, multi-system, multi-language environment in which you want to deploy aspects.

See you soon at Dagstuhl!

Andy

Keywords: Abstract

Introduction to Aspects

Oege de Moor (Oxford University, GB)

This talk provides a gentle introduction to aspect-oriented programming. With aspects, one can intercept events at runtime by writing patterns called "pointcuts"; it is also possible to add new members to existing classes.

We illustrate these mechanisms via a simple example, namely that of checking that the use of the Enumeration interface is failsafe (Enumeration is an outdated version of Iterator; while Iterator is expected to be failsafe, Enumeration is not). After a brief discussion of the advantages and disadvantages of aspects, I'll zoom in on two recent developments, namely pointcuts that match on complete program traces, and pointcuts that capture semantic properties (as opposed to pointcuts that are entirely name-based).

The attached "Paper.pdf" is a draft paper to be published at FATES/RV 2006 - it contains much of the introductory material I shall cover in my talk. "Other.pdf" is a very recent paper, submitted to POPL, on the topic of giving a semantics to pointcuts via a highly restricted form of logic programming called "Datalog".

Full Paper:
http://aspectbench.org