Rudi Studer, Mark A. Musen (editors):

Shareable and Reusable Problem Solving Methods

Dagstuhl-Seminar-Report; 113
08.05.-12.05.95 (9519)

ISSN 0940-1121

Das Internationale Begegnungs- und Forschungszentrum für Informatik (IBFI) ist eine gemeinnützige GmbH. Sie veranstaltet regelmäßig wissenschaftliche Seminare, welche nach Antrag der Tagungsleiter und Begutachtung durch das wissenschaftliche Direktorium mit persönlich eingeladenen Gästen durchgeführt werden.

Bezugsadresse: Geschäftsstelle Schloss Dagstuhl
Universität des Saarlandes
Postfach 15 11 50
D-66041 Saarbrücken, Germany
Tel.: +49 -681 - 302 4396
Fax: +49 -681 - 302 4397
e-mail: office@dag.uni-sb.de
url: http://www.dag.uni-sb.de

Sharable and Reusable Problem-Solving Methods

Mark A. Musen, Stanford University
Rudi Studer, University of Karlsruhe

For several years, researchers have argued that there are significant advantages when intelligent systems are constructed so that the procedural knowledge used to solve problems is made explicit. In the 1980s, a number of researchers experimented with single, abstract problem-solving methods (such as "heuristic classification" or "propose and revise") that not only provided a unifying architecture for a class of knowledge-based systems at runtime, but that also defined precise "knowledge roles" that could facilitate the process of knowledge acquisition: Building a new knowledge-based system became a matter of providing domain knowledge to satisfy each of the "knowledge roles" associated with the method. The explicitness of these knowledge roles made it possible to build both automated knowledge-acquisition tools that could create systems that incorporated these unitary problem-solving methods, and knowledge-acquisition-tool generators that could create interactive knowledge-acquisition tools that were specialized for different classes of application areas.

Although monolithic problem-solving methods such as "propose and revise" can be shared among developers and reused to generate new applications, they have significant limitations. They often do not fit the problem requirements of the application tasks for which they are employed, and lack flexibility at runtime in adapting their control strategies to unexpected situations. In recent years, investigators have begun to experiment with sharable and reusable problem-solving methods that may be composed from smaller grained building blocks, and that may provide more versatility in system development and operation.
Furthermore, several research groups have begun to investigate how domain models also might be made reusable and sharable. These investigations have resulted in various proposals for formal ontologies that make explicit the conceptualizations on which domain models are based, and for providing the means to define different views on such domain models.

This Dagstuhl seminar brought together investigators and practitioners from Europe, the United States, and Japan, all of whom were involved in development of intelligent systems from reusable components. Intensive discussions made it evident that, despite continued progress by the research community, terms used to describe problem-solving methods and the domain knowledge on which those methods operate are not always used in a consistent manner, thus sometimes making it difficult to compare approaches. An important benefit of the seminar was to understand alternative frameworks in more detail.

Although the use of formal languages to represent problem-solving behavior in a declarative fashion offers the best opportunity to clarify semantics, seminar participants were keenly aware of the difficulty of formalizing knowledge that is inherently procedural in nature. Much discussion focussed on the limitations both of formal descriptions of problem-sovling methods, and of situations in which problem-solving components have only operational semantics.

Additional discussion concentrated on the relationships between problem-solving methods and domain knowledge. Seminar participants debated approaches to representing as explicit ontologies both domain knowledge and the data on which problem-solving methods operate.

Seminar participants demonstrated the importance of developing modularized, component-based architectures for intelligent systems. There was a belief that researchers were developing a more shared perspective, but that differences in approach clearly remain. However, now that the development and definition of reusable problem-solving methods has become a maturing field, there is an urgent need for empirical results to validate many of the assumptions that investigators have made. The difficulty of carrying out many of the necessary empirical studies was well appreciated by seminar participants.

Contents

Frank Puppe
Problem Solving Methods for Configurable Expert System Shells

Wilhelm Schäfer
A Component-oriented Approach Towards Software Reuse

Guus Schreiber
Issues in Ontology Construction and their Connection to Problem-Solving Methods

Nigel Shadbolt
Are Tasks and Problem Solving Methods Indispensable?

Bill Swartout
Method Representations in EXPECT

Walter van de Velde
Reuse in Cyberspace

Is Tacit Knowledge Sharable?
by Hans Akkermans,
University of Twente

In the area of knowledge management, a distinction is often made between tacit and explicit knowledge in the organization. Tacit knowledge refers to the hard-to-pin-down skills rooted in experience, and is seen as hard to manage. Ontologies, however, are a means to make tacit background knowledge explicit.
This is illustrated by discussing various ontologies underlying a database library of reusable and sharable physical simulation models, currently under development in order to speed up mechatronic systems design. Ontologies represent different relevant, but separate, viewpoints on the world of the application. These viewpoints are connected by ontology mapping rules, which appear themselves to be quite elaborate and containing a lot of knowledge. Although ontologies represent tacit background knowledge and thus often have a meta-level flavour (aboutness), they are not necessarily meta-level specifications of domain knowledge in the technical-formal sense. Nevertheless, they are quite different from conceptual schemata of applications. Also, it appears that the conceptualizations they characterize cannot be seen as semantic points; rather, conceptualizations are fuzzy areas.
Thus, ontological engineering is about describing underlying viewpoints on the world as well as about different shades of meaning within a single viewpoint. A variety of mechanisms to structure sharable ontologies is therefore needed.

Reuse within a Formal Compositional Framework
by Frances Brazier and Jan Treur,
Vrije Universiteit Amsterdam

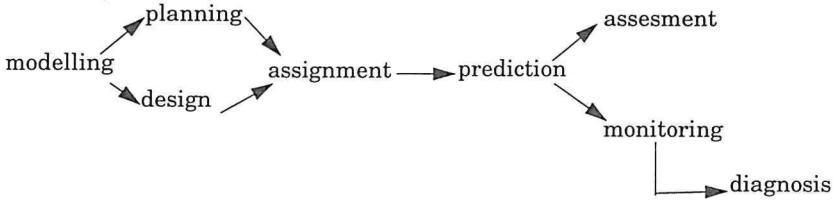
The focus of interaction between knowledge engineers and experts is to devise a shared task model on which they both agree. The acquisition of this agreed (mediating) model is often structured on the basis of existing abstract, domain-independent task models. Such mdoels are specialised and instantiated for the task at hand on the basis of knowledge of (1) specification of tasks and subtasks as components and sub-components, (2) information exchange between tasks and subtasks as information links, (3) sequencing of tasks as task control, (4) domain knowledge as knowledge structures and (5) delegation of tasks between agents as task assignment. The result of knowledge and task modelling is a task model with a compositional structure, formally specifi ed within a compositional architecture. Task models for which domain knowledge of the specific domain of application has been left unspecified, are, in principle, reusable problem solving methods. The advantage of this formal compositional approach is that control and strategic knowledge are explicitly (declaratively) defined, together with the conceptual, formal and operational semantics of both static and dynamic aspects of system behaviour. By defining the formal semantics in temporal logic a basis is provided for verification and validation. By defining a formal language automated support can be provided (e.g. automated prototype generation (currently available), automated redesign of compositional architectures (current research)). This behaviour-oriented approach has been particularly useful in modelling multi-agent tasks in which cooperation and flexible distributed control are essential. DESIRE, a formal compositional framework, developed over the past 7 years within the AI group at the VU (10-15 researchers), has been successfully applied in both academic environments and industry.

The Problems with Tasks and Methods

by Joost Breuker,
SWI, University of Amsterdam

For the CommonKADS methodology for building knowledge based systems a "library of reusable problem solving" components has been constructed. These components are (parts of) problem solving methods (PSMs). However a major problem is the indexing of these PSMs by problem types or tasks. This simple questions got not a simple answer. It has lead to revisiting the notions of solution, problem and PSM. Full solutions consist of a case model and a related argument structure. The latter part is constructed from tests applied to (parts of) proposed case model. (Parts of) case models are constructed or selected. This explains the "generate & test" paradigm that can be identifi ed in all PSM. Looking at case models, or rather their "conclusion" parts suggests 8 major types of problems. These conclusion or problem types have dependencies, which can be represented

as a suite:



Ontology Critiquing
by Henrik Eriksson,
Linköping University

The use of tools that support the knowledge-engineering process can be challenging. Developers often make modeling mistakes that could have been avoided in hindsight. Design-critiquing systems can assist the developer by highlighting potential problems with modeling decisions, and with the use of knowledge-engineering tools. Naturally, automated systems cannot detect all potential problems, but they help developers avoid many common mistakes, and they improve the quality of the target knowledge-based system.

PROTEGE-II is a knowledge-engineering environment that supports the developer in the reuse of problem-solving methods and the design of domain-specific knowledge-acquisition tools. PROTEGE-II provides a set of tools that allows the developer to combine problem solvers from reusable components, edit ontologies (which define concepts and relationships among concepts), and generate automatically knowledge-acquisition tools based on the ontologies. In our approach, the Critique Tool (CT) examines intermediate structures produced by PROTEGE-II, and presents a list of critiquing points to the PROTEGE-II user. Critiquing systems are useful in many phases in the PROTEGE-II design cycle.

Problem-Solving Methods: Programming at the Knowledge Level
by Dieter Fensel and Remco Straatman,
SWI, University of Amsterdam

The talk argues that we need a level between symbol and knowledge level where we can reason about rational problem solvers with limited resources. Given the rationality by the knowledge level of Newel, all computational problems can be solved by variants of generate and test. Problem-solving methods which are developed by the knowledge engineering community (like propose and revise for parametric designs etc.) do not improve the effect but the efficiency of the problem-solving process. That is, they enable problem solving in practice and not only in principle. In more detail, the talk has three main messages: First, problem-solving methods describe an efficient way to achieve a goal (i.e., to meet a desired functionality) by making assumptions about available knowledge. There-

fore, the description of a problem-solving method consists of three parts: First, the functionality of a problem-solving method must be described. This functionality must be matched with the goal (i.e., the task) which should be solved. Second, an operational description of a problem-solving method describes a heuristic algorithm which achieves the goal in an efficient manner. Third, the operational description of the method makes assumptions about the available domain knowledge which is required to find a solution efficiently. Such assumptions must be fulfilled by a given domain layer if a problem-solving method should be applicable. Second, the development of efficient problem-solving methods is not at all achieved by hierarchical refinement of its conceptual representation (i.e., by its inference structure and its control). That is, the conceptual structure of the problem-solving method will be changed and not only refined during such a process. In general, a more efficient variant of a problem-solving method is not achieved by refinement of a given method but by merging different inference steps into each other. Mainly, generation and test steps are merged into each other to come up with very efficient generation steps. Improving the efficiency of a method does therefore change its conceptual structure. Therefore, the conceptual structure can not be used as a guideline for this process. Instead, the assumptions made by a method can be used for this purpose. Improvement in efficiency is achieved by introducing new assumptions or by strengthening given assumptions. Developing efficient problem solvers is therefore an assumption-driven activity.
Third, the model of expertise does not describe a high-level design model of the system. We expect the same structure-destroying phenomena in order to achieve efficiency as observed when developing more efficient variants of a problem-solving method. Again, the conceptual structure of the specification becomes changed in order to achieve efficiency. Still, the model of expertise has a very important purpose. It describes the informal structure of the correctness and efficiency proof of a problem-solving process. It describes the way to deduce the functionality of the method based on its assumptions. Therefore, it describes the rational underlying the assumptions of the method.
An open issue for future research is to develop an appropriate notion to speak about efficiency at the knowledge level.

Formalizing Basic Ontological Distinctions
by Nicola Guarino,
National Research Council, LADSEB-CNR,

Formalizing the basic ontological distinctions which underly a particular knowledge base means offering a way to specify the intended meaning of its vocabulary by constraining the set of its models, giving explicit information about the intended nature of the modelling primitives and their a priori relationships.

We proposed an intensional definition of a conceptualization, aimed to capture the very basic ontological assumptions about the intended domain, related to issues such as identity and internal structure. To tackle such issues, a model framework endowed with mereological primitives has been adopted. Within such a

framework, we have introduced some ontological properties of unary predicates like countability, rigidity and dependence, which have been used to draw a rigorous distinction between concepts, types, properties, and roles. We also introduced some ontological distinctions between binary relations, in order to distinguish between internal relations (suitable to be used as attributes within objects) and external relations, which link together different objects.

Notations for Problem-Solving Methods
by Frank van Harmelen joint work with Annette ten Teije,
SWI, University of Amsterdam

Notations for problem solving methods have evolved from mostly informal to formal notations. However, currently available notations do not allow any formal representations of properties of these problem-solving methods. We present a novel notation for the functionality of problem-solving methods which allows us to write down such properties. This in turn enables us, for the first time, to formalise a theory about the composition and selection of (the functionality of) problem-solving methods.
Our notation can be summarised as problem-solving methods as axiom schemas. More precisely, a family of related problem-solving methods (e.g for diagnosis or design) is written as an axiom schema for first-order axioms. Such an axiom schema defines the schematic form of any individual problem-solving method that is a member of that family. The axiom schema contains a number of predicate variables. Different instantiations of these predicate variables correspond to different individual problem-solving methods. We can write the axiom-schema as a formula in a first-order meta-theory, with the predicate-variables as first-order variables ranging over names of object-predicates. Because of this, we can state relationships between problem-solving methods in this first-order meta-theory. This can be done for properties such as their relative strength, their (conditional) equivalence, the effects of substituting parameters, and how to obtain problem-solving methods with particular properties.
We have used our notation to formalise a large family of problem-solving methods for diagnosis, and we have begun to use this family for formulating a configuration theory and an approximation theory for (the functionality of) diagnostic problem-solving methods.

Object-Oriented Reuse of Problem-Solving Knowledge: Experiences in Scheduling Applications
by Masahiro Hori,
Tokyo Research Laboratory, IBM Japan Ltd.

The focus of recent research on development methodologies for knowledge systems has been shifting from composition issues toward context issues, with sharing and reuse of problem-solving knowledge as goals. This talk introduced our research on both types of issues, in the light of our experiences in developing

scheduling applications.

We have been working on configuring problem-solving methods from components, following the concept of Computer-Aided Knowledge Engineering (CAKE). The CAKE environment consists not only of processes for building knowledge systems, but also of processes for developing libraries of knowledge to be reused in the prospective systems. We are investigating the composition issues for a class of scheduling problems, called the job assignment task, by defining the task ontology and the task-specific components to be configured into methods. We are also exploring the context issues related to the component elicitation process in the development of knowledge systems. Our task formulation, however, is rather abstract, defining a broad range of scheduling problems solely as assigning given jobs to available resources while satisfying constraints. Therefore, the typology of basic inference steps, which we call method ontology, consists of generic set/structure manipulation, and component customization is not necessarily easy.
We then focus on production-scheduling problems, slightly limiting the scope of the target problems. In this way, we have revealed regularities in the application domain knowledge, and have remodeled the problem-solving knowledge by explicating its structural aspect. On the basis of this investigation, we have implemented a framework for production-scheduling applications in C++. It consists of three loosely coupled subsystems: a schedule model, a scheduling engine, and a graphical user interface. The talk described a couple of our experiences in applying the framework to the development of actual applications.

Finally, we point out that, in industrial applications, it is crucial to accumulate reusable knowledge contents and integrate them with existing software environments, and that moderate synergy between knowledge engineering and software engineering is indispensable for this purpose.

Maximizing Reusability and Usability in Problem-Solving Methods
by Todd R. Johnson and B. Chandrasekaran,
The Ohio State University

Many of the difficulties with reusing methods from task-specific architectures arise because they are designed around inflexible methods that overly limit the knowledge that can be used by a system and that make it difficult to build systems that smoothly integrate multiple problem-solving methods. To some extent, this inflexibility stems from a fundamental trade-off between reusability and usability. As the reusability (generality) of a method is increased, its usability (i.e., ability to constrain and guide system development) usually decreases. Reusability is limited whenever a method either expects knowledge that is not available, provides no means to make use of certain kinds of available knowledge, or cannot make use of special purpose control knowledge. When one or more of these occur, the method user must drop out of the knowledge level view provided by the method in order to modify the symbol-level implementation of the method. We propose that the problem space computational model (PSCM) can be used to

maximize reusability and usability because it enables us to define methods that: 1) Specify only necessary control knowledge without completely specifying the sequence of inference steps. 2) Afford dynamic component composition, based on the knowledge available at run time; 3) Permit a user to modify method behavior by adding additional domain and control knowledge without modifying the original method specification; and 4) Allow dynamic interleaving of subtasks from different methods without modifying the original methods. Although the PSCM provides the potential for maximizing reusability and usability, it does not provide any constraints for guiding system design. These constraints are provided by Task Structures and Generic Task Problem Spaces. Each task structure specifies for a given generic task the alternative methods for achieving that task, the subtasks of each method, methods for achieving each subtask, and so on, until primitive subtasks, which do not require further decomposition, are reached. Task structures are specified at the knowledge level, so that representational details are avoided. Each method in the task structure is implemented as a Generic Task Problem Space (GTPS), described using the PSCM language. Thus, the knowledge level and the PSCM are two levels for describing systems: the knowledge level is implemented using the PSCM. Task structures and GTPSs are content theories of tasks and methods described, respectively, at the knowledge level and the PSCM. In this approach, reusability is maximized because methods do not overly constrain the knowledge they can use. Usability is maximized because a problem-solving method specifies knowledge roles for required domain knowledge as well as specific roles for optional control knowledge.

Reuse in the VITAL workbench

by Enrico Motta,
Knowledge Media Institute, The OpenUniversity

Structured methodologies for knowledge-based system (KBS) development such as CommonKADS or VITAL view knowledge engineering as a process characterized by the construction of multiple models fulfilling different goals in the application life-cycle. In particular, the VITAL workbench supports the construction of conceptual models of problem solving, functional and technical design models, as well as providing a software architecture supporting the implementation of hybrid knowledge bases. From a general point of view there are two aspects concerning reuse-centred model development. The first is related to the fact that model-development should be reuse-based; the other concerns the outcome of a model development process, which should be a reusable product. In VITAL we provide methods, languages, and tools supporting both aspects of reuse-centred model-development. Model-based knowledge acquisition (KA) in VITAL is supported by the GDM methodology and tool, and is carried out through a process of model refinement, in which an initial, highly coarse-grained problem solving model is made increasingly fine-grained during various cycles of KA, using a generative grammar of model fragments. This grammar takes the form of rewrite rules, which replace an inference step with a finer-grained description. From a model reuse perspective the important aspect of the GDM methodology is that

the model construction process is reuse-based and that a comparatively small number of rewrite rules can provide an extensive library of problem-solving models. Moreover, because the rules in the GDM library have been designed with the aim of generating generic problem solving models, the outcome of the GDM analysis is itself a reusable product.
An important aspect of model-based KA is the formalization or operationalization of the conceptual model. The VITAL workbench supports the construction of operational conceptual models by means of the OCML language. A number of features of OCML facilitate reuse-centred conceptual modelling. The modelling framework underlying OCML distinguishes between problem solving and domain model and this distinction (common to many approaches to KBS development) facilitates the reuse of domain-independent problem solving methods and method-independent domain KBs. Recording the various steps in a model development process is crucial for reuse. To this purpose OCML comprises an expressive language for recording the argumentation process, OCML-DDL, which allows the knowledge engineer to record design choices at various levels of granularity. Finally, the VITAL workbench also comprises a software architecture, VITAL-KR which provides the basic substructure for integrating knowledge-based software components making use of diverse representation and inference paradigms. This architecture supports implementation-level reuse, as well as allowing experimentation in the design and implementation of hybrid systems.

Problem Solving Methods for Configurable Expert System Shells
by Frank Puppe,
Würzburg University

Problem-specific expert system shells with graphical knowledge acquisition components like ONCOCIN/OPAL or MED2/CLASSIKA are quite successful in enabling experts to build and maintain knowledge bases largely on their own. Their usefulness can be greatly increased, if they can be adapted to a specific domain. Adaption may include adding new or switching between available methods for a subtask like e.g. heuristic, statistic, case-based, set-covering or functional diagnostic evaluation inside classification or different ways for exchanging particular assignments between demand objects and supply objects violating some constraints inside assignment problem solving.
This is achieved by following some restrictions, in particular that all methods sharing the same knowledge have to represent it in the same form and that all methods solving the same subtask need to have the same interfaces.

The resulting flexibility greatly increases the programming effort necessary for developing the various graphical knowledge acquisition components. For reducing the effort tools for generating them are needed, which should be based on the specification of the knowledge representation of the methods. This is achieved by a library of parametrized graphical editors for e.g. hierarchies, forms, tables, graphs, together with a declarative specification language for the graphical design and the mapping between the graphical and the internal knowledge

representation.
The concepts proposed here are implemented in the shell-boxes D3 for classification, COKE for assignment, and in the generator META*KA for graphical knowledge acquisition components.

A Component-oriented Approach Towards Software Reuse
by Wilhelm Schäfer,
FB Mathematik/Informatik, Universität-GH Paderborn

The talk gave an overview about existing approaches for developing, assessing and retrieving reusable components. It describes the limits of type systems of current program languages concerning reuse and indicates that more large-grained components than classes or packages or modules have to be developed to make reuse an economic benefit. Consequently the notion of a subsystem is introduced. A tool supporting modular design based on graphical notations and subsystems is presented. This tool supports incremental and intertwined development of subsystems and maintains consistency of design and code.
This allows later reuse of design patterns together with their corresponding code pieces. Such sophisticated tools are not widely available and require application of reuse concepts to build them efficiently and not from the scratch. Consequently, a generator for those type of tools which support incremental and intertwined is discussed. Then the talk lists a few approaches for defining metrics to assess a component's quality and for building a controlled vocabulary to classify and retrieve a component. Finally, examples for reengineering existing legacy code are given which is another type of reuse than the previously presented development of components with having their later reuse is mind.

Issues in Ontology Construction and their Connection to Problem-Solving Methods
by Guus Schreiber,
University of Amsterdam

The term ontology" is currently fashionable in the context of knowledge sharing and reuse. However, no agreement exists on what an ontology is or how it can be used. In this talk we discuss the view taken by the European ESPRIT project KACTUS on what an ontology is and we illustrate through examples one particular way of using ontologies. The KACTUS project aims at the development of methods and tools for the reuse of knowledge about technical systems during their life-cycle. The project is application-driven: systems are being developed in the domains of preliminary-ship design, oil-production processes, and electrical networks.
In KACTUS, an ontology is defined as an "explicit, partial specification of a conceptualization that is expressible as a meta-level viewpoint on a set of possible domain theories for the purpose of modular design, redesign and reuse of knowledge-intensive system components". Ontologies can have a recursive structure,

meaning that an ontology expresses a viewpoint on another ontology. Such a viewpoint entails a reformulation and/or reinterpretation of the other ontology. This multi-level organization raises research questions such as the required expressivity of the mapping formalisms for expressing viewpoints between ontologies. The important consequence of this approach is that through the use of different ontologies with different generality and by partitioning the knowledge base accordingly, we can identify different classes of knowledge bases, with different scope, generality and reusability. These features are illustrated through an example of knwoledge-base reuse in the doamin of designing elevators.

Are Tasks and Problem Solving Methods Indispensable?
by Nigel Shadbolt,
University of Nottingham

This talk reviewed a number of issues associated with tasks and problem solving methods in Knowledge Engineering:
- What kinds of things can tasks and methods be?
- What utility do they have?
- What future do they have?

Tasks are classically defined as a description of *what* needs to be achieved to solve a problem. A method is a description of *how* to achieve a task. A number of positions were discussed with respect to the terms task and method.
The first of them regards examples of tasks (e.g. design) and methods (e.g. propose and revise) as having the status of psychological categories. It is still important to determine what sort of status this means: a functional equivalence, an algorithmic equivalence or an implementational one.
A second approach - characterized as "cheerful unconcern" - argues that the content of a theory of tasks and methods has only pragmatic utility. This instrumental view seems to be held by the majority of Knowledge Engineers.
A third position - psychological anti-realism - argues that a large proportion of the elements used in psychological accounts of behaviour are the unavoidable by-products of our interpretive acts. This does not mean that they are debased as explanatory accounts.
I defined the "Cognitive Interaction Problem". This is the recognition that all accounts of cognition will depend on the use to which such accounts are put by cognitive agents. Any explanatory account is given in the context of a set of goals and expectations. It was argued that for both Knowledge Engineering and Cognitive Psychology these are similar: decomposing the complexity of a phenomenon, facilitating communication and sharing of the domain content, reliably reproducing classes of behaviour given certain states. In addition, it appears that the content theory of "pragmatic" Knowledge Engineering can be exported as putative psychological explanations under the anti-realist interpretation.
Finally a number of deficiencies in our understanding of problem solving methods were enumerated. Including a lack of agreed ways of characterising the control aspects of methods and the absence of empirical studies across subjects of the cost/benefit of using our task structure and problem solving methods.

Method Representations in EXPECT
by Bill Swartout,
USC/Information Sciences Institute

Our long term research goal is to develop tools that will guide end users in augmenting and modifying knowledge based systems so that they can make changes without having to understand the details of how a system is implemented, or its underlying structure. In current knowledge acquisition tools much of the knowledge needed to guide acquisition is inflexible because it is either built into the tool itself or fixed when the system is first created. This limits the range of systems that can be built and supported with these tools.

To create more powerful knowledge acquisition tools and systems, we not only need better tools, but we also need to change the architecture of the knowledge based systems themselves so that their structure will provide better support for acquisition. The EXPECT framework for knowledge based systems makes the structure of knowledge based systems more explicit and declarative, thus enabling its knowledge acquisition tools to analyze a system's structure and thereby guide acquisition. This allows more flexible acquisition than is possible with current tools.
In this talk, we focused on one aspect of EXPECT's more explicit architecture: its representation of goals and the capabilities of problem solving methods. In many frameworks, goals are just symbols that have little explicit semantics associated with them. In EXPECT, the key idea for capturing more of the semantics is to represent goals and method capabilities using the Loom knowledge representation language. Using the Loom Classifier, a hierarchy of methods can be automatically created which helps both in locating methods that can be used for achieving goals and in identifying related methods. In addition, because this representation captures the semantics of the goals and is decomposable, it is possible to reformulate goals (that is, change them into other forms) in the event a method for achieving a goal cannot be found. This helps ease the reuse of methods and makes them applicable in a broader context.

Reuse in Cyperspace
by Walter van de Velde,
Artificial Intelligence Laboratory, Free University of Brussels

My aim here is to provoque some thought on present and future directions for reuse. My starting point is that the reality of computing is rapidly changing. Wide area networks, distributed application and multi-medial systems are a the component technologies of new computing environments that classical techniques for reuse may no longer be well adapted to.

Libraries, standard interface and interchange formats are key elements in the

classical reuse toolbox. Libraries rely on the hypothesis that a model can be reused over a characterizable range of situations. The solution for reuse therefor seems to be clear: one or more appropriatly indexed libraries of reusable elements must be agreed on, maintained and used. One has to ask three questions: Does it work? Is it useful? Is it possible? For all three arguments can be provided that cast doubt on the standard positive answer.

But also the kind of applications that we will be building will have very different characteristics from the present more stand-alone and custom made applications. They will execute on different computers in an extended time-span, trying to achieve real tasks that go beyond functional computation (e.g. spending your money to search for and acquire your favorite collector's item). Such software agents, much like human agents, go out to search for the information and resources (including people) that they require to achieve the tasks that they were send out to do. What will be needed is a new model of computing altogether. Societies of software agents will cooperate and compete in their virtual world of networks and resources, building up much of the same social dynamics that one finds in everyday life.

In summary I argue for 4 elements in an agent-based approach to reuse:
(1) identification based on formal features (e.g. syntactic compatibility), (2) integrated testing and validation behavior, (3) adaptation and learning and (4) deferred commitment. The first one is classical but over-emphasized. The second one leads to what I call a courting behavior of software agents. Software agents can, by virtue of being complete executable entities, try eachother out, so to speak. The fourth element, deferred commitment, refers to the fact that the decision to use is always revocable. An agent's usefulness is constantly questioned and its service subject to competition from other agents. A whole new type of software economy can be based on this.

Of course we can only start to explore the implications for reuse, and this is the main theme of the presentation. What is possible and useful can only be guessed right now but that there will be a hugue impact on the practice in research, education, business and everyday life is beyond doubt. If knowledge engineering fails to track these evolutions then it will become increasingly irrelevant to practical demands.

Dagstuhl-Seminar 9519:

List of Participants

Agnar **Aamodt**
University of Trondheim
Department of Informatics
College of Arts and Science
N-7055 Dragvoll
Norway
agnar@ifi.unit.no
tel.: +47-73 59-1838 /1840

Manfred **Aben**
Unilever Research Laboratories
Technology Application Unit
Olivier van Noortlaan 120
NL-3133 AT Vlaardingen
The Netherlands
manfred.aben@2488taux.urlnl.sprint.com
tel.: +31-10-460-5716

Stuart **Aitken**
University of Glasgow
Department of Computing Science
17 Lilybank Gardens
Glasgow G12 8QQ
Great Britain
stuart@dcs.gla.ac.uk
tel.: +44-41-3398855 x 2049

Hans **Akkermans**
Universiteit Twente
Information Systems DepartmentINF/IS
Postbus 217
NL-7500 AE Enschede
The Netherlands
akkerman@cs.utwente.nl or
akkermans@ecn.nl
tel.: +31-53-893690

Richard **Benjamins**
Université Paris Sud
Lab. de Recherche en Informatique
Bâtiment 490
F-91405 Orsay Cedex
France
richard@lri.lri.fr

Frances **Brazier**
Vrye Universiteit Amsterdam
Dept. of Mathematics and
Computer Science
De Boelelaan 1081 a
NL-1081 HV Amsterdam
The Netherlands
frances@cs.vu.nl
tel.: +31-20-444-7737

Joost **Breuker**
Universiteit van Amsterdam
Department Social Science Informatics
Roetersstraat 15
NL-1018 WB Amsterdam
The Netherlands
breuker@swi.psy.uva.nl
tel.: +31-20-525-3494 /6789

Jose **Cuena**
University of Madrid
Dep. de Intelligencia Artificial
Facultad de Informatica
Campus de Montegancedo s/n
E-28660 Boadilla del Monte
Spain
jcuena@dia.fi.upm.es
tel.: +34-352-48-03

Henrik **Eriksson**
Linköping University
Dept. of Computer and
Information Science
S-58183 Linköping
Sweden
her@ida.liu.se
tel.: +46-13 28-26 73

Dieter **Fensel**
University of Amsterdam
Dept. of Social Science Informatics
Roetersstraat 15
1018 WB Amsterdam
The Netherlands
dieter@swi.psy.uva.nl
tel.: +31 20-525.6791/525.6789

Ute **Gappa**
Universität Karlsruhe
Institut fur Logik / Komplexität
und Deduktionssysteme
Am Fasanengarten 5
D-76128 Karlsruhe
Germany
gappa@ira.uka.de
tel.: +49-721-608-4209

Nicola **Guarino**
National Research Council
LADSEB-CNR
Corso Stati Uniti 4
I-35020 Padova
Italy
guarino@ladseb.pd.cnr.it
tel.: +39-49-82 95-751

Rune **Gustavsson**
University College of Karlskrona/Ronneby
Dept of Computer Science & Economy
S-372 25 Ronneby
Sweden
rune.gustavsson@ide.hk-r.se
tel.: +46-457 717 00

Frank **van Harmelen**
Universiteit van Amsterdam
Department Social Science Informatics
Roetersstraat 15
NL-1018 WB Amsterdam
The Netherlands
frankh@swi.psy.uva.nl
tel.: +31-20-525-67 91

Masahiro **Hori**
Tokyo Research Laboratory
IBM Japan Ltd.
1623-14 Shimo-tsuruma
Kanagawa-ken 242 Tokyo
Japan
hori@trl.ibm.co.jp
tel.: +81-462-73-46 67

Todd R. **Johnson**
Ohio State University
Division of Medical Informatics
Rm 395
2051 Neil Ave.
Columbus OH 43210
USA
tjohnson@magnus.acs.ohio-state.edu

Dieter **Landes**
Universität Karlsruhe
Institut fur Angewandte Informatik
und Formale Beschreibungsverfahren
Englerstr. 11
D-76128 Karlsruhe
Germany
landes@aifb.uni-karlsruhe.de
tel.: +49-721-608-3998

Frank **Maurer**
Universität Kaiserslautern
FB Informatik
Postfach 3049
D-67653 Kaiserslautern
Germany
maurer@informatik.uni-kl.de
tel.: +49-631-205-3356

Pedro **Meseguer**
Universidad Politécnica de Cataluña
Dept. LSI
Pau Gargallo 5
E-08028 Barcelona
Spain
meseguer@lsi.upc.es
tel.: +34-3401-73 26

Enrico **Motta**
Open University
Walton Hall
Milton Keynes MK76AA
Great Britain
e.motta@open.ac.uk
tel.: +44-908-65-3506

Mark **Musen**
Stanford University
School of Medicine
Knowledge Systems Laboratory
Section on Medical Informatics
Stanford CA 94305-5479
USA
musen@camis.stanford.edu
tel.: +1-415-723-6979

Rainer **Perkuhn**
Universität Karlsruhe
Institut AIFB
Englerstr. 11
D-76128 Karlsruhe
Germany
perkuhn@aifb.uni-karlsruhe.de
tel.: +49-721-608-4754

Christine **Pierret-Golbreich**
Université Paris Sud
Laboratoire de Recherche en Informatique
Bât 490 CNRS URA 410
F-91405 Orsay Cedex
France
pierret@lri.fr

Karsten **Poeck**
Universität Würzburg
Institut für Informatik VI
Allesgrundweg 12
D-97218 Gerbrunn
Germany
poeck@informatik.uni-wuerzburg.de
tel.: +49-931-70 56-118

Frank **Puppe**
Universität Würzburg
Lehrstuhl für KI und
Angewandte Informatik
Allesgrundweg 12
D-97218 Gerbrunn
Germany
puppe@informatik.uni-wuerzburg.de
tel.: +49-931-70561-10

Thomas **Rothenfluh**
University of Zurich
Zürichbergstr. 43
CH-8044 Zürich
Switzerland
rothen@ifi.unizh.ch
tel.: +41-1-257-21 06

Wilhelm **Schäfer**
Universität Paderborn
FB 17 - Informatik
Warburger Str. 100
33095 Paderborn
Germany
wilhelm@uni-paderborn.de
tel.: +49-5251/60-24 28

Franz **Schmalhofer**
Deutsches Forschungszentrum
für Künstliche Intelligenz
Erwin-Schrödinger-Straße
D-67663 Kaiserslautern
Germany
schmalho@dfki.uni-kl.de
tel.: +49-631-205-3465

Guus **Schreiber**
Universiteit van Amsterdam
Department Social Science Informatics
(SWI)
Roetersstraat 15
NL-1018 WB Amsterdam
The Netherlands
schreiber@swi.psy.uva.nl
tel.: +31-20-525-6792/6789

Nigel **Shadbolt**
University of Nottingham
Dept. of Psychology
University Park
Nottingham NG7 2RD
Great Britain
nrs@psyc.nott.ac.uk
tel.: +44-115-951-5317

Yurval **Shahar**
Stanford University
School of Medicine
Dept. of Medicine
Medical School Office Building X215
Stanford CA 94305-5479
USA
shahar@camis.stanford.edu
tel.: +1-415-723-3393

Mario **Stefanelli**
Laboratory for medical Informatics
Dept. of Computer & Systems Sciences
Via Abbiategrasso 209
I-27100 Pavia
Italy
mstefa@ipvstefa.unipv.it

Remco **Straatman**
University of Amsterdam
Department SWI
Roeterstraat 15
NL-1018 WB Amsterdam
The Netherlands
remco@swi.psy.uva.nl
tel.: +31-20-525-6787

Rudi **Studer**
Universität Karlsruhe
Institut AIFB
D-76128 Karlsruhe
Germany
studer@aifb.uni-karlsruhe.de
tel.: +49-721-608-3923

William R. **Swartout**
USC/ISI
Intelligent Systems Division
4676 Admiralty Way
Marina del Rey CA 90292
USA
swartout@isi.edu
tel.: +1-310-822-15 11

Jan **Treur**
Vrye Universiteit Amsterdam
Dept. of Mathematics and
Computer Science
De Boelelaan 1081 a
NL-1081 HV Amsterdam
The Netherlands
treur@cs.vu.nl
tel.: +31-20-444-7763/ 7730

Samson **Tu**
Stanford Univ. School of Medicine
Medical School Office Building
Room X-215
Stanford CA 94305-5479
USA
tu@camis.stanford.edu
tel.: +1-415-725-3391

Johan **Vanwelkenhuysen**
INRIA
ACACIA Project
2004 route des Lucioles
F-06902 Sophia Antipolis Cedex
France
jvanwelk@sophia.inria.fr
tel.: +33 93 65 77 88

Walter **van de Velde**
Free University of Brussels
Dept. of Computer Science
Pleinlaan 2
B-1050 Brussels
Belgium
walter@arti.vub.ac.be
tel.: +32-2641-3700

Thomas **Wetter**
IBM Wiss. Zentrum
Institut für Logik und Linguistik
Vangerowstr. 18
D-69121 Heidelberg
Germany
twetter@vnet.ibm.com
tel.: +49-6221-59-4212

Zuletzt erschienene und geplante Titel:

R. Giegerich, J. Hughes (editors):
Functional Programming in the Real World, Dagstuhl-Seminar-Report; 89; 16.05.-20.05.94 (9420)

H. Hagen, H. Müller, G.M. Nielson (editors):
Scientific Visualization , Dagstuhl-Seminar-Report; 90; 23.05.-27.05.94 (9421)

T. Dietterich, W. Maass, H.U. Simon, M. Warmuth (editors):
Theory and Praxis of Machine Learning, Dagstuhl-Seminar-Report; 91; 27.06.-01.07.94 (9426)

J. Encarnação, J. Foley, R.G. Herrtwich (editors):
Fundamentals and Perspectives of Multimedia Systems, Dagstuhl-Seminar-Report; 92; 04.07.-08.07.94 (9427)

W. Hoeppner, H. Horacek, J. Moore (editors):
Principles of Natural Language Generation, Dagstuhl-Seminar-Report; 93; 25.07.-29.07.94 (9430)

A. Lesgold, F. Schmalhofer (editors):
Expert- and Tutoring-Systems as Media for Embodying and Sharing Knowledge, Dagstuhl-Seminar-Report; 94; 01.08.-05.08.94 (9431)

H.-D. Ehrich, G. Engels, J. Paredaens, P. Wegner (editors):
Fundamentals of Object-Oriented Languages, Systems, and Methods, Dagstuhl-Seminar-Report; 95; 22.08.-26.08.94 (9434)

K. Birman, F. Cristian, F. Mattern, A. Schiper (editors):
Unifying Theory and Practice in Distributed Systems, Dagstuhl-Seminar-Report; 96; 05.09.-09.09.94 (9436)

L. Bannon, R. Keil-Slawik, I. Wagner (editors):
Interdisciplinary Foundations of Systems Design and Evaluation, Dagstuhl-Seminar-Report; 97; 19.09.-23.09.94 (9438)

M. Broy, L. Lamport (editors):
Specification and Refinement of Reactive Systems - A Case Study, Dagstuhl-Seminar-Report; 98; 26.09.-30.09.94 (9439)

M. Jarke, P. Loucopoulos, J. Mylopoulos, A. Sutcliffe (editors):
System Requirements: Analysis, Management, and Exploitation, Dagstuhl-Seminar-Report; 99; 04.10.-07.10.94 (9440)

J. Buchmann, H. Niederreiter, A.M. Odlyzko, H.G. Zimmer (editors):
Algorithms and Number Theory, Dagstuhl-Seminar-Report; 100; 10.10.-14.10.94 (9441)

S. Heinrich, J. Traub, H. Wozniakowski (editors):
Algorithms and Complexity for Continuous Problems, Dagstuhl-Seminar-Report; 101; 17.10.-21.10.94 (9442)

H. Bunke, T. Kanade, H. Noltemeier (editors):
Environment Modelling and Motion Planning for Autonomous Robots, Dagstuhl-Seminar-Report; 102; 24.10.-28.10.94 (9443)

W. Maass, Ch. v.d. Malsburg, E. Sontag, I. Wegener (editors):
Neural Computing, Dagstuhl-Seminar-Report; 103; 07.11.-11.11.94 (9445)

G. Berry, W.P. de Roever, A. Poigné, A. Pnueli (editors):
Syncronous Languages, Dagstuhl-Seminar-Report; 104; 28.11.-02.12.94 (9448)

B. Becker, R. Bryant, O. Coudert, Ch. Meinel (editors):
Computer Aided Design and Test, Dagstuhl-Seminar-Report; 105; 13.02.-17.02.95 (9507)

D. Garlan, F. Paulisch, W. Tichy (editors):
Software Architectures, Dagstuhl-Seminar-Report; 106; 20.02.-24.02.95 (9508)

W.J. Cullyer, W.A. Halang, B. Krämer (editors):
High Integrity Programmable Electronic Systems, Dagstuhl-Seminar-Report; 107; 27.02.-03.03.95 (9509)

J. Gruska, H. Umeo, R. Vollmar (editors):
Cellular Automata, Dagstuhl-Seminar-Report; 108; 06.03.-10.03.95 (9510)

H. Alt, B. Chazelle, R. Seidel (editors):
Computational Geometry, Dagstuhl-Seminar-Report; 109; 13.03.-17.03.95 (9511)

W. Bibel, K. Furukawa, M. Stickel (editors):
Deduction, Dagstuhl-Seminar-Report; 110; 20.03.-24.03.95 (9512)

B. Freitag, C.B. Jones, Ch. Lengauer, H.-J. Scheck (editors):
Object-Orientation with Parallelism and Persistence, Dagstuhl-Seminar-Report; 111; 03.04.-07.04.95 (9514)

J. Doran, N. Gilbert, U. Mueller, K. Troitzsch (editors):
Social Science Microsimulation: A Challenge for Computer Science, Dagstuhl-Seminar-Report; 112; 01.05.-05.05.95 (9518)

R. Studer, M. Musen (editors):
Shareable and Reusable Problem Solving Methods, Dagstuhl-Seminar-Report; 113; 08.05.-12.05.95 (9519)

J. Blazewicz, K. Ecker, L. Welch (editors):
Scheduling in Computer & Manufacturing Systems, Dagstuhl-Seminar-Report; 114; 15.05.-19.05.95 (9520)

H. Beilner, G. Ciardo, C. Lindemann, K. Trivedi (editors):
Performance and Dependability Modeling with Stochastic Petri Nets, Dagstuhl-Seminar-Report; 115; 22.05.-26.05.95 (9521)

M. Aigner, J. Spencer, E. Triesch (editors):
Computing with Faulty Inputs, Dagstuhl-Seminar-Report; 116; 29.05.-02.06.95 (9522)

J.-R. Abrial, E. Börger, H. Langmaack (editors):
Methods for Semantics and Specification, Dagstuhl-Seminar-Report; 117; 05.06.-09.06.95 (9523)

W. Effelsberg, D. Ferrari, O. Spaniol, A. Danthine (editors):
Architecture and Protocols for High Performance Networks, Dagstuhl-Seminar-Report; 118: 19.06.-23.06.95 (9525)

Ph. Flajolet, R. Kemp, H. Prodinger, R. Sedgewick (editors):
Average-Case'-Analysis of Algorithms, Dagstuhl-Seminar-Report; 119; 03.07.-07.07.95 (9527)

D. Gustfield, T. Lengauer, C. Sander (editors):
Molecular Bioinformatics, Dagstuhl-Seminar-Report; 120; 10.07.-14.07.95 (9528)

J. Chomicki, G. Saake, C. Sernadas (editors):
Role of Logics in Information Systems, Dagstuhl-Seminar-Report; 121; 17.07.-21.07.95 (9529)

R.S. Boyer,A. Bundy, D.Kapur, Ch. Walther (editors):
Automation of Proof by Mathematical Induction, Dagstuhl-Seminar-Report; 122; 24.07.-28.07.95 (9530)

P. Cousot, R. Cousot, A. Mycroft (editors):
Abstract Interpretation, Dagstuhl-Seminar-Report; 123; 28.08.-01.09.95 (9535)

P. Brunet, D. Roller, J. Rossignac (editors):
CAD Tools for Products, Dagstuhl-Seminar-Report; 124; 04.09.-08.09.95 (9536)

C. Dwork, E.W. Mayr, F. Meyer a.d. Heide (editors):
Parallel and Distributed Algorithms, Dagstuhl-Seminar-Report; 125; 11.09.-15.09.95 (9537)

C. Hankin, H. R. Nielson (editors):
New Trends In the Integration of Paradigms, Dagstuhl-Seminar-Report; 126; 18.09.-22.09.95 (9538)