Composing Reusable Synthesis Methods through Graph-based Viewpoints

Rationale

Jutta Eusterbrock
Computing Science Department, University of Aberdeen
Aberdeen AB24 3UE, Scotland, UK
email: jeusterb@csd.abdn.ac.uk

Abstract

To make sense of research on constructive knowledge-based methods for synthesis tasks, it has to be shown how to reuse and adopt these results in realistic settings. Cooperative reuse-oriented heterogeneous systems have been proposed as a solution for achieving high-quality user-friendly systems within changing and distributed environments. This causes the question, how could heterogeneous knowledge sources, application systems and various kinds of synthesis knowledge, built upon different paradigms, be accessed and used by logic-based synthesis applications. *Viewpoints* introduced in [12] are intended to provide intermediate user-definable independent components between reasoning components and the domain resources of interest. The starting point is given by a logical theory, which provides a generic description of a class of problems by defining generic actions, goals and relationships. The definitions serve as initial elements of a task specific library of synthesis methods which are derived as meta-level theorems. A viewpoint includes a set of lifting rules and is a commitment to represent domain specific knowledge fragments and problem-solving knowledge in terms of the vocabulary of an associated generic theory, such that correctness axioms are preserved. As a major point, it is proposed to share abstract structure through graphs rather than to use translations for knowledge integration. It will be claimed that an extended viewpoint concept is an adequate means for the additive integration of various multi-paradigm resources, making them reusable for component-based constructive synthesis and synthesis methods reusable within a multi-paradigm distributed environment.

1 The Need of Abstraction for Cooperative Synthesis

Addressing the field of systems for automated synthesis, like program synthesis, design, configuration or scheduling, the overall goal is to construct plans or programs that conform with user-needs and adapt to changing environments. Formal methods for synthesis should be like mathematical theorems in the sense that they state the assumptions which yield correctness, completeness or efficiency, and once methods have been deployed, they are reusable, can be seamlessly combined and never have to be recoded manually. Recent progress towards this purpose stems from several directions. It realizes that it needs multiple knowledge representation and reasoning mechanism, abstraction, meta-level and schema-guided approaches, rather than uniform general purpose deductive methods (cf. [20]) and that general purpose strategies like the LOPS strategy (cf. [6]) need to be formalised in order to yield an implementational framework for synthesis. D. Smith derived formalisations of various general schemas for algorithm design, like the well-known “Divide-and-Conquer” (cf. [28]) or “Global Search” programs. [22] emphasises that it needs a formalisation of the meta-theory of programming and formally proven synthesis methods. Research in knowledge-based systems has led to multi-level architectures and libraries of reusable generic problem solving methods, most often on a pragmatic basis, aiming at efficient and effective reasoning for generic tasks and taking into account several kinds of strategic knowledge (cf. [31]). Experimental studies have produced evidence that automated synthesis of complex efficient algorithms, based on formal methods is possible, if substantial domain-specific problem solving knowledge is provided (cf. [10]). Our research effort on knowledge-based synthesis has resulted in the comprehensive SEAMLESS multi-layer architecture (cf. [12]) which is intended to provide a sound framework for structuring types, synthesis knowledge and reasoning methods for synthesis and knowledge acquisition tasks as reusable components on three independent abstraction layers, combining approaches from knowledge-based systems and logic-based synthesis. "Logical viewpoints" are independent intermediate components introduced for the loose coupling of knowledge on different abstraction layers.

In this paper, the focus of attention is on the interplay between meta-level synthesis at the one and heterogeneous data sets or development tools from various sources and heuristic expert knowledge at the other side. Taking a theoretical basis for granted, having some experimental success, it has long been claimed that formal methods are bound to be increasingly applied for synthesising realistic application systems. There is still a considerable dichotomy between research in automated synthesis, and the practical wide-spread use of commercial environments for system development, like CASE tools (cf. [16]) and databases. User requirements and domain models tend to be hard to capture as what can be expressed by a logical formalism doesn't match with common representation means, eg. entity relationship diagrams or examples. Also, there is now a shift of the classical programming paradigm – transforming specifications into programs – that further complicates the process of building synthesis systems. Reuse by combining, merging or adapting components has been proposed as a solution for achieving high-quality user-friendly software within changing and distributed environments.

Hence, putting logic-based synthesis into practice involves providing construction knowledge and synthesis objects by precise logical specifications. Most important, to deal with intractable complexities, human synthesis knowledge like design rules or solved cases has to be operationalised. Finally, deductive results have to be communicated back in a user-friendly way, for example by graphical support. Traditionally, synthesis systems are devised as stand-alone approaches. They rely on the user to pass information to the system as and when it needs it to perform certain deductions, though this is a tedious and difficult task and guidelines are missing. It has become clear that it is increasingly necessary to integrate logic-based synthesis systems, external knowledge resources and development tools. Moreover, it is desirable to have open interfaces for the provision of comprehensive domain-specific problem-solving knowledge.

Current Internet technologies make reuse oriented cooperative problem solving a viable option as they allow the loose coupling of heterogeneous distributed systems in such a way that they keep their autonomy (cf. [27]). There are several efforts on semantics-based information or knowledge sharing and interoperable systems. Agent-based communication mandates a universal communication language in which inconsistencies and notational variations are being eliminated. Establishing communication between heterogeneous systems which range from conventional databases to high-level logic-based synthesis systems and user-provided expert knowledge on a syntactical variant of any formalism is complicated, if at all possible. It seems to be counterintuitive to mix rigid and abstract formalism with semi-structured and heuristic expert knowledge. Communication between a synthesis agent, who wants to decompose a problem into subproblems, thereby reducing a well-founded ordering, a broker who delivers schedules for a specific airline and a traveller who wants to provide personal travel experience needs some more subtle interaction mechanism rather than mere translation into a common format and polishing up notational variants. In the remainder of this paper, we propose

- to use a Blackboard architecture for the loose cooperation of heterogeneous resources;
- to use graphs as a natural means for the sharing of abstract structure between multi-paradigm knowledge resources and to consider component-based synthesis as an automated two-step meta-level graph construction and graph unfolding process;
- to adopt the SEAMLESS viewpoint concept as an independent intermediate theory to make domain-specific knowledge reusable for logic-based synthesis;

2 SEAMLESS for Reuse-based Synthesis

Most often, it is difficult to reuse synthesis and knowledge-based systems outside of their original context. The SEAMLESS framework addresses these problems in two ways:

- A synthesis process is seen as an evolutionary knowledge construction process, divided into the development steps "**S**pecification, **E**xperimentation, **A**bstraction, **M**aintenance, **L**earning and **E**xtraction for **S**ystem **S**ynthesis". A knowledge-based synthesis system is conceived to provide generic methods, abstract and concrete types which assist automation of the above development steps.
- A multi-layer architecture provides a coherent concept for (1) partitioning various kinds of knowledge fragments on different layers of abstraction into an easy extensible collection of reusable abstract and concrete components and (2) building a domain-specific synthesis system by the seamless composition of knowledge components into a network of resources.

The SEAMLESS architecture (cf. [12] and Figure 1) to model synthesis knowledge is based upon the definition of three hierarchically organised abstraction layers. The layers refer to the organisation of

- Domain Knowledge;
- Generic Domain Abstractions;
- Distributed Knowledge Sources.

The layers are abstractions from lower layers and may be regarded as separated knowledge bases associated with distinguished formal specification languages and interpreters for expressions. Two views are possible: a layer may be seen as meta-layer, controlling the behaviour of the lower (object) layers, as well as an object layer which is available to manipulation by higher layers and provides services to them. The communication among layers is realized by *viewpoints*. Each layer is subdivided into the levels data types, axiomatic theories and derived knowledge. *Tasks* to be solved on different levels and layers are submitted as *goals* in a Prolog-like style. Solutions are generated by interpretation.
Using SEAMLESS, at least three user-groups are involved: (1) method developers who want to evolve the collection of reasoning components to incorporate new capabilities; (2) domain-experts who actually want to reuse synthesis methods and add their own components specific to a particular application to get a domain-specific synthesis system; (3) end-users who want to use synthesised systems for their purposes. Because the different user groups view the system from different perspectives, some may need a more detailed insight into a specific layer than

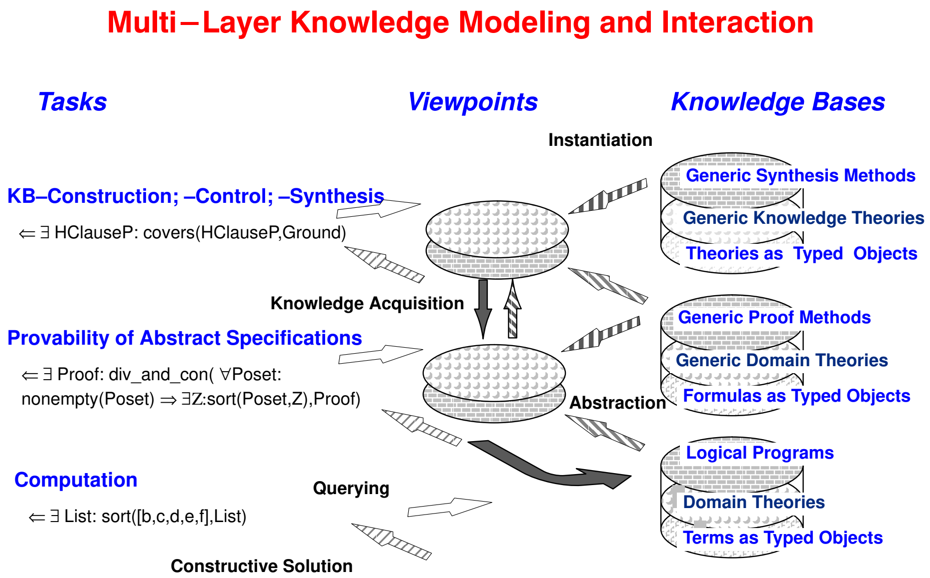

Figure 1: SEAMLESS Layers

others, for instance, developers of logic-based methods and data types will need to have every of the three layers at their fingertips, while domain-experts are only interested in augmenting high-level abstract applications with domain-specific knowledge. Before discussing what kind of abstractions and relationships might be employed to couple object resources with knowledge-based synthesis systems, we elaborate the SEAMLESS architecture a bit further. Coupling domain-knowledge mainly concerns concepts introduced on the second SEAMLESS layer, which are analysed in more detail:

Abstract Data Types have to be introduced in order to be able to treat object level expressions as terms. Especially, this refers to executable objects like Horn-clause programs, which should be subject to modifying operations during the synthesis process. Within SEAMLESS the structure of object level programs is represented by terms, implementing directed graphs.

Generic Theories used for Domain Clustering Starting point for method developers as well as domain experts are generic theories which are considered to be abstract components. Generic theories are intended to provide generic descriptions over a class of similar problems in terms of recurrent concepts, actions and relationships. Generic theories are logical theories, ie. ddefined by a signature which introduces a vocabulary and axioms which constrain the well-formed use of expressions. We distinguish generic meta-predicates into "hard" and "soft". Definitions for hard predicates have to be provided in order that a synthesis method has some desirable properties, ie. constructs a solution in finite time. Definitions for soft predicates can be provided additionally and are intended to capture additional strategic problem solving knowledge, which might result in reduced search spaces or generally in more effective processes. In [11], we used the relationships "necessary condition", "sufficient condition", "behavioural equivalence", "known subcase", "probably there exist a solution", "there exist no solution" and "upper" and "lower bounds" to capture algorithm design knowledge and some more generic predicates to capture control knowledge. The relations "well-founded ordering", "split", "decompose" and "compose" describe the core of a generic "divide-and-conquer" theory.

The Constructive Generic Synthesis Approach Taking a logic-based approach, generally a synthesis process like configuration, design, program generation is considered to be a reasoning process, starting from a given domain theory $DomainTheory$. A solution in the form of a logic program LP is derived which fulfills the specified goal $Goal$, eg. functionality in the form of pre- and postconditions, and further constraints $Constraints$ on the program itself like efficiency conditions

$$\begin{array}{l} DomainTheory \vdash LP \\ LP \vdash Goal \\ Constraints(LP) \end{array}$$

Rather than to use general-purpose proof procedures – which leads to serious theoretical and practical problems, because $Constraints$ is already a meta-level property, we favour a generic approach. In principle, generic synthesis is realized as a meta-level constructive proof of a $Task$ having the form

$$Task \doteq\leftarrow \exists LP : generic\_synthesis((Goal \leftarrow LP) \wedge Constraints(LP))$$

whereby the meta-level variable LP is being substituted by a term which refers to an object-level program. The question arises how to synthesise generic synthesis methods, ie. meta-interpreters, which can be reused across applications and are correct? The formal approach is to consider the definitions for a generic synthesis method as an operational theory, which is derived from the axioms stated by the associated generic theory.

$$\begin{array}{l} GenericTheory \vdash GenericMethod \\ GenericMethod \vdash Task \end{array}$$

That means, generic theories provide the set of generic building blocks, which can be used to construct task-specific synthesis method. As before, most important are methods, which satisfy some further constraints, like efficiency. But discussing this formally requires another meta-level.

Viewpoints The result is a task-, and class specific, abstract and domain-independent, highly generic synthesis approach. The concrete domain-specific problem solving knowledge can be provided through a viewpoint (VP) which is an intermediate component between generic methods and domain-specific knowledge. A viewpoint is a user-defined component which provides a meta-level representation of object-level resources and implements additional problem solving knowledge by meta-level formulas with respect to the vocabulary introduced by a pre-existing generic theory such that correctness axioms are preserved. Or shortly, a viewpoint instantiates an abstract generic theory. Bidirectional relationships between object- and meta-level are modelled by sets of rewrite rules, called *lift definitions* (cf. [31]). Once implemented,

- Viewpoints provide the context in which generic, formally verified, methods are used by identifying the assertions pertaining to the local environment:
- Object-level values, expressions and synthesis knowledge may be queried - also in a Prolog-like style - by high-level applications using the vocabulary introduced by the associated generic theory.

This means a domain-specific solution LP is synthesised by constructively proving

$$\begin{array}{l} (GenMethod >> VP) \vdash \\ \qquad (\leftarrow generic\_synthesis(Goal \leftarrow LP) \wedge Constraints(LP)) \end{array}$$

where $>>$ denotes the context (cf. [8]) which is dynamically added.

3 Viewpoints for Loose Cooperation of Multi-Paradigm Resources

The term viewpoint has been borrowed from the Software Engineering Community, where it is widely acknowledged that different users and different applications have different views on object resource and that viewpoints assist effective development (cf. [26]): "A viewpoint is a loosely coupled, locally managed, distributable object, encapsulating partial representation knowledge, development process knowledge, application architectures and specification knowledge, eg. functionality or behaviour, about a (software) component and its domain relevant for reuse."

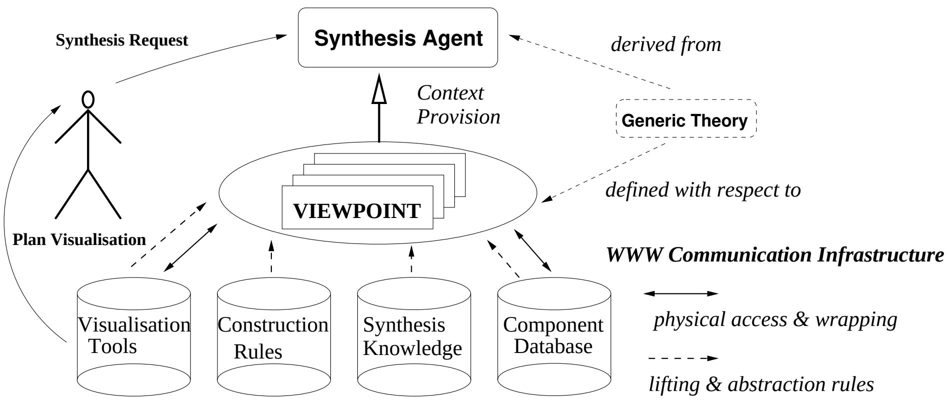

Figure 2: The Cooperation Architecture

Generally, viewpoints in the SEAMLESS architecture are user-definable, intermediate and distributable components between knowledge resources on different abstraction layers, and intended to make object-level resources and knowledge accessible by high-level constructive knowledge-based reasoning. Practically, a viewpoint is an agreement to model object level expressions by terms and to make assertions about the object-level with respect to the vocabulary introduced by associated generic theories, such that correctness axioms are preserved. Figure 2 illustrates the interplay between the multi-paradigm resources. Now, aiming at accessing external semi-structured knowledge resources, a viewpoint is roughly intended to capture the following "real-world" data and knowledge fragments on different levels:

1. **Computational Level.** A VP is intended to ensure physical access to pre-existing knowledge resources, like databases. Having linked a database or a program to a VP, entries may be selected or function values may be computed by querying Prolog predicates. There are various ways to integrate a knowledge-based synthesis systems with external knowledge sources. A tight coupling between Prolog and a relational database is described in [3]. Integration could also be based on interface functions in the usual way or simple message exchanging, a common protocol and an intermediate interchange format. For example, the HTTP protocol enables network connectivity across heterogeneous networks and hardware platforms, URLs provide an universal transparent addressing scheme and blackboard systems allow to exchange information physically (cf. [9]) and provide the basis for sophisticated cooperation models when building complex distributed systems (cf. [27]). One of the proposed interchange formats is called KIF (Knowledge Interchange Format) is a prefix version of first-order predicate calculus with various extensions.

2. **Meta-Representation of Object-Level Structure by Types.** Knowledge about the semantical structure and organisation of the underlying resources, like software architectures or database schemas, can be represented explicitly, using type predicates (cf. [21]).

For example the atom $relation(r_1(a1, \ldots, a_n))$ introduces a relation r_1, having relation attributes $a_1, \ldots, a_n$ as term of type *relation* on the meta-level (cf. [3]). Term rewrite rules provide a means for the abstract simulation of object-level operations, eg. appending two program components.

3. **Problem-Solving Knowledge.** The most important feature of this viewpoint approach – which distinguishes it significantly from transformational and interface-based approaches for knowledge reuse – is that it allows to store and access additional problem solving knowledge explicitly and in a flexible way. It is supposed that a broad range of problem solving knowledge, can be categorised by few generic building blocks as they have been assembled in section 2. For example, technical construction knowledge could be marked with annotations of the following form: A sequence of steps gives a reusable solution to a subproblem; a given problem is equivalent to another one whose solution is known; plugging in a specific component is necessary in order that a system satisfies a desired functionality.

   Having these annotations, a human problem solver as well as a generic synthesis method, can find a solution in a very effective way, even without having any domain-specific knowledge, just by applying generic problem solving knowledge. Technically, problem-solving knowledge is provided through *lift definitions*. Lift definitions are in a Prolog-like style of the form (i) $Head$ or (ii) $Head \leftarrow Body$, where $Head$ is an atom, build up from the vocabulary of the associated generic theory, and $Body$ is a logical expression formed from the vocabulary introduced by the generic theory and the declarative interfaces from underlying resources. Correctness axioms have to be preserved.

As before, viewpoints provide the context in which an instantiated generic task is being solved.

4 Capturing Structure through Graph-based Abstraction

Representing the structure of object resources and the computational operations on the meta-level by terms and term rewrite rules in a nearly one-to-one way is costly. It might result in the well-known exponential complexities if the meta-models form the assumptions for constructive reasoning. Most important, for any class of object resources which is being coupled to the knowledge-based synthesis system, a specific abstract data type has to be devised and implemented. We claim that it needs abstraction mechanism in order to get effective synthesis procedures. Graphs are perhaps the most common natural means for capturing abstractions of semi-structured data, documents, processes, or complex programs, hardware- and software systems. Generally, nodes represent data or values and edges the relationships between them. The well-known configuration system XCON [24] used directed graphs for the representation of connected hardware components. A software system may be modularised as a graph or configuration of "components" and "connectors" (cf. [17]). Even commercial software transformation systems use graphs as means to represent and manipulate abstract software structure. A complex site in the World-Wide Web has a natural abstraction as graph where node parameters may be bound to page locations (URL's) and links describe structural and semantical relationships between parts of the hyper-document. Several query languages use graphs to model and use regular path-expressions to navigate through graphs. For example, UnQL (cf. [15]) is a language for querying semi-structured data that can be modelled as labelled graphs.
As graphs have been proven very useful for solving various practical engineering tasks, we employ them in the SEAMLESS framework as a means to share structure between "real-world" semi-

structured knowledge resources and constructive methods for knowledge-based synthesis. In the SEAMLESS framework, directed hypergraphs are used as basic means to represent the general structure of connected components on different levels of abstraction, whereas nodes refer to values or components and links refer to relationships. A representation graph (nested graph) maybe an elementary graph, where nodes are associated with names or concrete values, or an advanced graph, composed of linked feature terms (cf. [5]) or graphs itself. In [13] we devised term representatives and term rewriting rules for implementing directed graphs and their operations. Moreover, some desirable properties could be proven. A well-founded ordering was constructed which yields a procedure for constructing canonical forms for equivalence classes of terms representing classes of isomorphic directed acyclic graphs. The subsumption relation on graph terms implements entailment of the associated graph classes. Terms provide a simple, but versatile, mechanism for representing arbitrary data. The particular way in which a directed graph is associated with a given program, algorithm, database schema, collection of WWW addresses or a system model, is not important as long as there exist homomorphic mappings from domain structures into canonical graph terms such that graph theoretical composition and inclusion properties are being preserved. For example, the following simplified rules illustrate graph construction primitives, supposing databases on notebooks, communication software and modems.

$$\begin{aligned}&connector(notebook(index\_nb, -, -, , ...), communication\_software(index\_cs, -, -, ...))\\ &\qquad \leftarrow notebook.operating\_system = communication\_software.operating\_system.\end{aligned}$$

$$\begin{aligned}&connector(notebook(index\_nb, -, -, ...), modem(index\_m, -, -, ...))\\ &\qquad \leftarrow notebook.pc\_card\_slot \neq \emptyset.\end{aligned}$$

A synthesis process is considered to be a graph construction process – guided by lifted domain-specific rules – where a functional or service-oriented requirement specification is broken down into manageable pieces and a solution graph is constructed bottom-up. Automation is based on domain-independent general purpose logic-based inference mechanism that exploit graph-theoretical properties for retrieval and constructive reasoning. The existence of a well-founded ordering on graph terms allows to apply synthesis methods which are based on decomposition properties.

5 Multiple Views by Unfolding Canonical Terms

Canonical graph terms serve as index on the abstractions of interest, like software or document structures and provide the means for effective constructive reasoning. Using canonical forms to store the abstract structure of constructed systems avoids maintaining a combinatorial number of transformations into all the different formats required for multiple purposes. However, across a range of end-users and applications, there is typically no best way to view the derived terms:

- Term representations might be sufficient for processing by further logic-based applications;
- For easy understanding, visualisations could be most appropriate;
- For the application of synthesised results, it is necessary to have executable source code in the desired target language.

Within the SEAMLESS system, in general and as in [23], multiple views are obtained by unfolding the canonical expressions. Generated views may be executable or serve as input for further applications. So far the following different views on constructed graph terms can be derived:

(1) More readable representations of complex terms, using simple pretty-print programs.

(2) Executable programs. Program extraction is based on composing software components by instantiating a given generic program schema with domain-specific functions and data types, using the viewpoint rules downwards from the meta-level to the object-level, providing function values and parameters by traversing graph terms and using programming knowledge about the target environment.

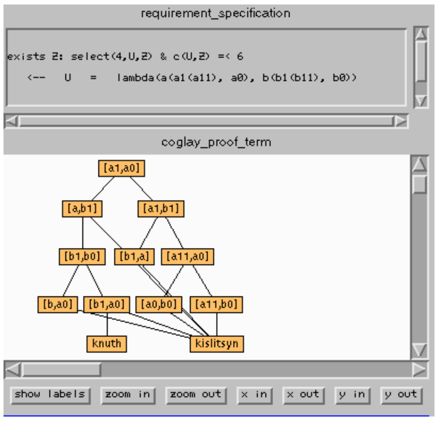

Figure 3: Visualisation of a synthesised decision graph

(3) Visual representations of nested graph terms are synthesised invoking the CoGLay graph layout tool kit through communication mechanism. CoGLay takes string representations of nested graphs, obtained through rewriting terms, and some layout constraints as input in order to compute a layout which satisfies aesthetic criteria and visualises derived results through a World-Wide Web interface as it is shown in Figure 3 (cf. [14]).

6 Conclusions

In this paper, we proposed to adopt the SEAMLESS framework aiming at reuse, based on *viewpoints*, to achieve loose composition of logic-based meta-level synthesis methods and distributed knowledge resources built upon different paradigms in an additive way. A viewpoint provides the assumptions pertaining to the local environment represented in terms of an explicit generic theory, based on the principles of sharing abstract structure through graphs and modelling synthesis knowledge by formulas, containing graph terms and graph term variables. Sharing structure through graphs provides a natural means to achieve cooperating problem solving systems which integrate heterogeneous components like visualisation systems or database schemas even if they have been built upon different paradigms. Viewpoints allow for a significant amount of reuse by providing the dynamic context with a formally derived synthesis method. The major achievement of this paper is, it needs few new theoretical results to incorporate logic-based synthesis methods within a distributed multi-paradigm environment when adopting the SEAMLESS architecture. The challenging task remains to provide the associations which relate domain objects and composition operators with graph concepts and to instantiate generic problem solving concepts with concrete domain-specific problem solving knowledge.
So far, viewpoint concepts have not been discussed within Logic Programming. Using viewpoints seem to accord with many other current developments on "middle layers" in computing for

reuse, knowledge integration, distributed computing and software engineering. The usage of the concept viewpoint stems from software engineering (cf. [4, 26]). The need for viewpoints when building open distributed systems is also recognised by the efforts on standardisation (cf. [1]). The proposed viewpoint concept comprises the widely use of schematic mapping functions and theory morphism, which translate expressions from one set of vocabulary to the vocabulary of an underlying theory such that natural language semantics or correctness axioms are preserved, as a basis for reuse in different areas (cf. [25, 30, 29]). The mediator which is an independent layer between applications and information resources was introduced as a very general concept by Wiederhold (cf. [32]), resulting in various implementational realizations. Information sources are linked to the mediator using wrapper components which translate queries from the mediator format to the source and vice-versa. Knowledge unification by translation into a common format and intersection in the sense that inconsistencies and notational variations are being resolved are the most crucial operations. KOMET (cf. [7]) is an example for a mediator-based system and uses annotated order-sorted logic to integrate heterogeneous information sources. Lifting axioms in our case are used for describing abstract properties of knowledge fragments rather than rewriting statements into a common format such that they can be queried in an uniform way. The SEAMLESS viewpoint approach stresses the need for multiple perspectives, user-defined abstractions on domain data and the incorporation of various kinds of synthesis knowledge in order to get a flexible representation which enables effective constructive meta-level reasoning. The viewpoint architecture targeting at reusable synthesis methods is being applied in the distributed KRAFT (Knowledge Reuse and Transformation) architecture (cf. [19]) which aims at transforming knowledge to make it usable by distributed problem-solvers.

Acknowledgements

The SEAMLESS architecture is a reflection on the comprehensive research on synthesis methods and knowledge representation techniques, having started in the DFG project "Program Synthesis from Examples" at the Technical University of Darmstadt, taking into account experience from more practical work at the GMD on knowledge-based system configuration, the Internet and reuse in software engineering. Recent work in the KRAFT project at the University of Aberdeen has brought a database perspective into my mind. Especially, I want to thank Jon Atle Gulla for fruitful discussions on using viewpoints as a means to integrate CASE tools and logic-based synthesis methods and Michalis Nicolaides for intensive work on the integration of the visualisation component.

References

[1] ISO/IEC 10746-4. Information Technology - Open Distributed Processing - Reference Model - Open Distributed Processing, 1995. Part 4: Architectural Semantics.

[2] S. Contiero A. Brogi and F. Turini. The use of renaming in composing general programs, 1998.

[3] F. Abbruzzese, F. Mele, and F.M. Sacerdoti. A knowledge-based approach to integration between prolog and relational databases. In *Computational Intelligence I*, pages 135–142. North Holland, 1989.

[4] ACM. Proc. International Workshop on Multiple Perspectives in Software Development (Viewpoints'96), San Francisco, ACM Symposium on Foundations of Software Engineering, 1996.

[5] H. Ait-Kaci and R. Nasr. Login, a logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[6] W. Bibel. Syntax-directed, semantics supported program synthesis. *Artificial Intelligence*, 14:243–261, 1980.

[7] J. Calmet, S. Jekutsch, P. Kullmann, and J. Schü. Komet - a system for the integration of heterogenous information sources. In *Proc. ISMIS'97*, pages 318–327. Springer Verlag, 1997.

[8] I. Cicekli. Design and implementation of an abstract metaprolog engine for metaprolog. In H. Abramson and M. Rogers, editors, *Meta-Programming in Logic Programming*, pages 417–434. MIT Press, 1989.

[9] K. de Bosschere and P. Tarau. Blackboard-based extensions in Prolog. *Software - Practice and Experience*, 26(1):49–69, 1996.

[10] J. Eusterbrock. Errata to "Selecting the top three elements" by M. Aigner: A Result of a computer assisted proof search. *Discrete Applied Mathematics*, 41:131–137, 1992.

[11] J. Eusterbrock. *Wissensbasierte Verfahren zur Synthese mathematischer Beweise: Eine kombinatorische Anwendung*, volume 10 of *DISKI*. INFIX, 1992.

[12] J. Eusterbrock. A multi-layer architecture for knowledge-based system synthesis. In *Proc. ISMIS*, pages 582–592. Springer Verlag, 1996.

[13] J. Eusterbrock. Canonical term representations of isomorphic transitive DAGs for efficient knowledge-based reasoning. In *Proceedings of the International KRUSE Symposium, Knowledge Retrieval, Use and Storage for Efficiency*, pages 235–249, 1997.

[14] J. Eusterbrock and M. Nicolaides. The visualization of constructive proofs by compositional graph layout: A world-wide web interface. *Proc. CADE Visual Reasoning Workshop, Rutgers University*, 1996.

[15] M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. Int'l Conference on Data Engineering 1998*, page to appear, 1998.

[16] A. Fugetta. A classification of case technology. *COMPUTER*, pages 25–38, 1993.

[17] D. Garlan. Research directions in software architecture. *ACM Computing Surveys*, 27(2):257–260, 1995.

[18] C. H. Goh, S. E. Madnick, and M. D. Siegel. Ontologies, contexts, and mediation: Representing and reasoning about semantic conflicts in heterogeneous and autonomous systems. Technical report, Sloan School of Management Working Paper 3848, 1995.

[19] P. Gray and et al. KRAFT: Knowledge fusion from distributed databases and knowledge bases. In R. Wagner, editor, *Proc. Int'l Workshop on Databases and Expert Systems (DEXA'97)*, pages 682–691. IEEE Press, 1997.

[20] C. Green. Application of theorem proving to problem solving. In B. L. Webber, editor, *Readings in Artificial Intelligence*, pages 202–222. Tioga Publishing Company, 1981.

[21] P. Hill and J.W. Lloyd. Analysis of meta-programs. In *Meta-Programming in Logic Programming*, pages 23–51. MIT, 1989.

[22] Chr. Kreitz. Metasynthesis - deriving programs that develop programs, 1993. Habilitationsschrift, Technische Hochschule Darmstadt, 1992.

[23] C.D. Marlin. Multiple views based on unparsing canonical representations - the multiview architecture. In *Proc. International Workshop on Multiple Perspectives in Software Development (Viewpoints'96)*, pages 217–221. Association for Computing Machinery, 1996.

[24] J. McDermott. R1: A rule based configurer of computer systems. *Artificial Intelligence*, 19:29–88, 1982.

[25] G.S.Jr. Novak. Composing reusable software components through views. In *Knowledge-Based Software Engineering Conference*, pages 39–47, 1994.

[26] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–772, 1994.

[27] D.G. Schwartz. *Cooperating Heterogenous Systems*. Kluwer Academic Publishers, 1995.

[28] D.R. Smith. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence*, 27:43–96, 1985.

[29] D.R. Smith. Constructing specification morphisms. *Journal Symbolic Computation*, 15:571–606, 1993.

[30] Y.V. Srinivas and R. Jüllig. SPECWARE (TM): Formal support for composing software. In *Proceedings of the International Conference on the Mathematics of Program Construction*. Pitman Publ., 1995.

[31] F. van Harmelen and J. Balder. $(ML)^2$: A formal language for KADS models. In *Proc. of the 10th European Conference on Artificial Intelligence*, pages 582–586. John Wiley & Sons, 1992.

[32] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, 1992.

[33] G. Wiederhold, Rathmann, T. P. Barsalou, B. Lee, and D. Quass. Partitioning and composing knowledge. *Information Systems*, 15(1):61–72, 1990.