# Enhancing Domain-Specific Software Architecture Recovery

**Igor Ivkovic**
School of Computer Science
University of Waterloo
Waterloo, ON N2L3X1 Canada
iivkovic@uwaterloo.ca

**Michael Godfrey**
School of Computer Science
University of Waterloo
Waterloo, ON N2L3X1 Canada
migod@uwaterloo.ca

## Abstract

*Performing software architecture analysis and recovery on a large software system is expensive and time consuming; when it is done at all, it is often performed within a narrow context, focused on a few areas of particular concern. However, for a long-lived system within a well understood application domain, the costs for performing detailed architecture recovery may be amortized over several generations of the system; the resulting models can also be broadened and put into context by incorporating information about the history and anticipated future evolution of both the application and its underlying domain.*

*This paper proposes a systematic approach for organizing application domain knowledge into a unified structure called the Architectural Domain Assets Set (ADAS). The ADAS structure builds on previous research, as well as our experience in performing an architecture recovery of IBM's DB2. Our initial experiences in using ADAS suggest that it brings needed focus to the recovery process and provides assistance to domain-specific architecture recovery.*

## 1. Introduction

The maintenance phase in the development lifecycle of large systems is inevitably the costliest part [14]. Performing maintenance — that is, correcting outstanding problems or adding new functionality — is hard without a clear understanding of the structure and functionality underlying the system at hand. However, the explicit modelling of a system's software architecture — its structure, its major components and their interactions, and the rationale for the chosen design —- is rarely performed in practice. Instead, in industrial settings, it is most common to target a particular subsystem, a set of features, or a limited set of properties that may involve many pieces of the system (*e.g.*, components that interface with the operating system). That is, architecture recovery is performed strategically, with particular goals in mind and making use of detailed knowledge of the application domain and the application itself.

For such focused *domain-specific architecture recovery*, one would most benefit by using a recovery process that best suits the particular type of the system (*e.g.*, distributed database management systems for Linux) and utilizes a well-specified knowledge base for the corresponding application domain and its related domain architecture. Such a domain-specific architectural knowledge base could be *a collection of application domain-specific architectural artifacts meant to aid architecture recovery*. Without the clear linkage between the selected approach for architecture recovery and the application's characteristics, the usefulness of the overall re-engineering efforts to system's stakeholders will be significantly decreased [1, 15].

To further advance the research area of architecture recovery, we propose an organization of application domain knowledge for purposes of domain-specific architecture recovery. In addition, we offer brief guidelines on how to use this organization on practical case studies, which are needed for validation and further refinement. Derivation of this organization and related guidelines are based on our experience in architecture recovery of IBM's DB2 [4].

The rest of this paper is organized in the following manner: Section 2 discusses the creation of our architectural knowledge base. Section 3 defines the idea of the Architectural Domain Assets Set (ADAS) and its key components. Section 4 describes guidelines for how to use ADAS in domain-specific architecture recovery. Section 5 compares ADAS to other related approaches. And, Section 6 summarizes the results of this paper and proposes areas of future research.

## 2. Basis for ADAS

The roots of ADAS can be traced back to a case study in software architecture recovery of IBM's DB2 relational database management system (RDBMS) [4]. Based on this experience and encountered problems, we have derived a corresponding architectural knowledge base that can be used to facilitate more efficient domain-specific software architecture recovery.

## 2.1. Software Architecture Recovery of IBM's DB2

IBM's DB2 RDBMS belongs to a class of very large applications with its size of more than five million lines of code (5 MLOC). The particular code base that we had access to was the core of the DB2 software system, and it represented a previously released distribution of this application for the Linux platform. The complexity of this system stemmed from its size and its high interaction with other related technologies that were not part of this code base but were quite frequently referenced. [1]

Our initial goals in this architecture recovery process were:

- to help the DB2 developers understand the architecture of the system more clearly for purposes of program comprehension and software evolution, and
- to verify our current tools and capabilities in architecture recovery against an application as complex and as large as DB2.

Our main obstacles in the process were identified as follows:

**Application Related**

- Inappropriate architectural design documents.
- Inaccessible or inappropriate lower level design documents.
- Inability to contact the DB2 developers regularly due to their high involvement in the release of a new version of DB2.
- High internal complexity of the application due to its size and underlying functionality.

**Domain Related**

- Lack of understanding or inaccessible documentation of specialized, for the RDBMS domain, theories or technologies employed by the system.
- Lack of evolutionary history for DB2 or inaccessible evolutionary history for the RDBMS domain.
- Lack of knowledge of underlying DB2 functionality or lack of an appropriate architectural domain model.
- Lack of relevant reference architecture that could be used as a starting point for our recovery efforts.
- Lack of understanding of DB2's future evolution or inaccessible evolutionary directions for the RDBMS domain.

**Technology Related**

- Nonexistent guidelines for systematic architecture recovery of an application as large and as complex as DB2.
- Inability of our tools such as the PBS toolkit [9] to deal with detailed architecture recovery of a system of this size and complexity.

These obstacles resulted in our overall recovery efforts being quite inefficient and in industrial settings quite infeasible, especially given that our team consisted of only two software engineering graduate students.

## 2.2. Perceived ADAS Benefits

The domain-related obstacles could have been resolved if a corresponding architectural knowledge base had been available to us before we started with the recovery. For example, our lack of domain expertise could have been resolved by having access to systematically organized knowledge base of specialized theories or technologies employed by the system. In addition, we would have been able to more clearly specify our goals for architecture recovery if we understood or had access to appropriate evolutionary directions.

Based on this experience, we have derived the following organization of application domain knowledge, used for domain-specific software architecture recovery, into a unified structure called the Architectural Domain Assets Set (ADAS).

## 3. Introducing ADAS

ADAS is structured in accordance with the overall desire to support domain-specific architecture recovery by systematically organizing and making available application domain knowledge. ADAS is meant to be a relatively simple architectural knowledge base that can aid architecture recovery and evolution through domain understanding (Figure 2).

In addition, prior to devising the resulting ADAS structure, we analyzed the term *domain expertise*, in regards to the knowledge of the application domain, that is usually mentioned as one of the key sources of information for initial architecture recovery steps [6, 8, 9]. Based on this analysis, we aimed through ADAS to provide organization for

[1] Sensitive details of the actual application are left out of this discussion in compliance with a non-disclosure agreement.

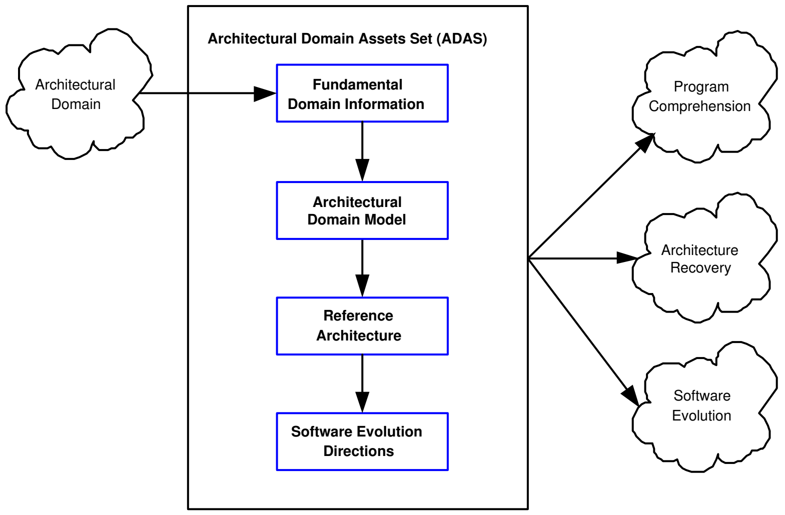
Architectural Domain
Architectural Domain Assets Set (ADAS)
Fundamental Domain Information
Architectural Domain Model
Reference Architecture
Software Evolution Directions
Program Comprehension
Architecture Recovery
Software Evolution

**Figure 1. Architectural Domain Assets Set (ADAS)**

this implied knowledge base, and potentially allow those who are not experts in the field of architecture recovery to carry out this process successfully.

More formally, we define ADAS as follows.

> *ADAS is a collection of architectural information that consists of fundamental domain information, architectural domain model, reference architecture for the domain, and directions for future evolution.*

Figure 1 illustrates this model and its process of creation. More details of ADAS can be found in [11].

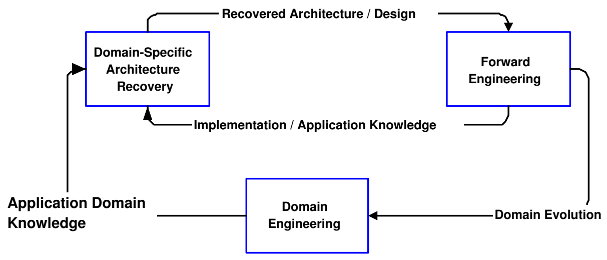
Recovered Architecture / Design
Domain-Specific Architecture Recovery
Forward Engineering
Implementation / Application Knowledge
Application Domain Knowledge
Domain Engineering
Domain Evolution

**Figure 2. Domain-Specific Re-Engineering**

## 3.1. ADAS: Fundamental Domain Information

Performing architecture recovery without a clear understanding of the application domain is risky. The software engineers performing the recovery will either have to rederive a conceptual model of the domain each time a recovery is undertaken or risk making ill-informed decisions. For this reason, the ADAS structure includes an explicit model of *Fundamental Domain Information* that can be of aid to the software engineers in their recovery efforts. Having such a repository of information about the application domain can improve the quality of the resulting architecture models and reduce the risk of poor decisions being made subsequently. Additionally, it can serve as a teaching tool for engineers that are new to the project or the application domain.

In ADAS, fundamental domain information consists of elements relevant for the understanding of the analyzed domain and its evolution. Such elements are:

**Domain Origins** To understand an application domain, one must understand why the domain exists in the first place and how it was derived. This information represents a rationale for major design decisions and is also relevant when deciding the future of an application in one such domain. For example, in the context of ADAS for RDBMS, to understand the current status of DBMS and related functionality and provide justification for existing conditions, one should realize that DBMSs were created in response to the inadequacies of the file-based system that preceded them. Documenting this information and making it available allows those working on a DBMS to quickly derive the rationale behind the current structure, and make more informed decisions about the future of the application.

**Domain Evolution** Another important part of the application domain understanding is the comprehension of the domain's evolutionary history. Understanding the key environmental forces that influenced the domain evolution allows software engineers to proactively deal with the complexity of change in an application that belong to the domain of interest. This evolutionary information should at a minimum contain a lightweight summary of the domain history noting the key milestones in the domain evolution.

**Domain Fundamentals** Each application domain is supported by specific theory and technology that represents the basis of each such domain. Understanding this basis and its key parts is crucial to comprehending the domain itself. Therefore, the domain fundamentals part of the fundamental domain information includes a comprehensive summary of the relevant theoretical and technological pieces of the application domain. For example, for the RDBMS domain, domain fundamentals include: data definition languages (DDLs), data manipulation languages (DMLs), data dictionary, indexes, transactions, and stakeholders.

**Domain Taxonomy** Application domains are not homogenous; they typically exhibit a certain degree of variability. Understanding this variability and its factors is necessary for comprehending the details of the domain itself. As a result, domain taxonomy includes the description of the driving forces that define and separate subfamilies of applications inside the domain. For example, for the RDBMS domain, factors that influence this variability include: data model type, number of supported users, number of database sites, type of the DBMS software, cost, and functionality.

## 3.2. ADAS: Architectural Domain Model (ADM)

We define an architectural domain model (ADM) as a collection of the following artifacts [2]:

- A definition of the program family and its scope; that is, a defined boundary between system and environment for each member of the family: required to understand the components of a typical system in the domain that are encapsulating the interactions between the system and the outside world.
- A definition and semantics of the information that flows across the defined system boundaries: required for a more in-depth understanding of the boundary components and their interactions with the environment.
- A set of features that are common for each member of the family: required for understanding the purpose and structure of the non-boundary components of a typical system in the domain.

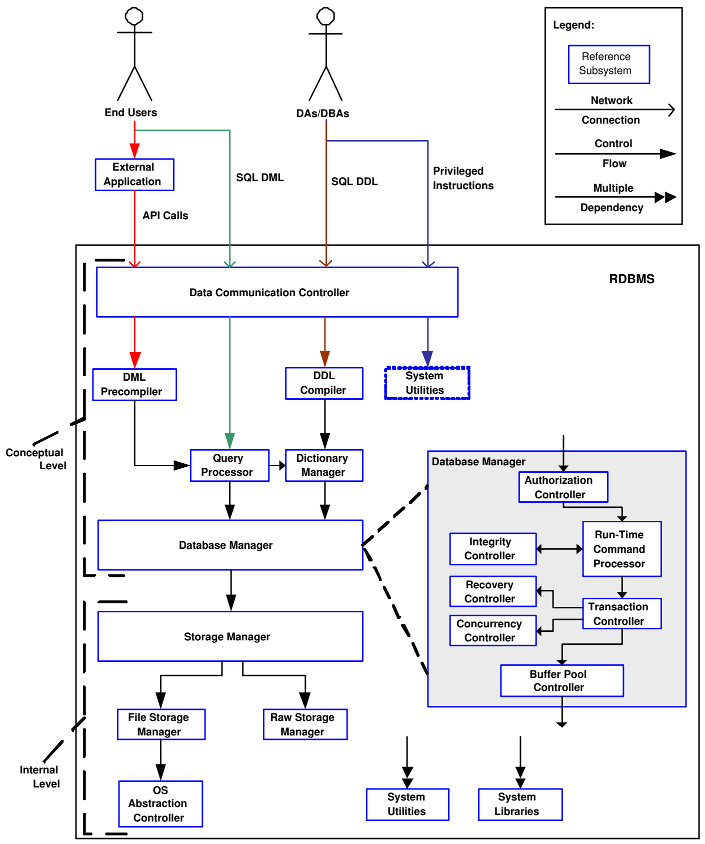

**Figure 3. RDBMS Reference Architecture**

- Qualitative requirements to be met by each member of the family: required for understanding the intricacies of the communication among components.

Described artifacts are usually extracted using a typical domain analysis method such as the Feature-Oriented Domain Analysis (FODA) [12].

## 3.3. ADAS: Reference Architecture for the Domain

The ADM is used as a basis for extraction of a reference architecture for the domain, and *reference architecture is a result of mapping the ADM onto software components and data flows among those components*. Such structure is extracted using corresponding commonality analysis [7] and functionality-to-component mapping, and an architectural style that is valid for the product domain. More specifically, extraction of this structure includes the following steps.

**Step 1. Architectural Style Extraction:** Based on the analysis of the underlying ADM, related literature, and related fundamental domain information, a corresponding architectural style can be derived and can serve as a starting point in the extraction of a related reference architecture. Properties of existing architectural styles (*e.g.*, separation of concerns in tiered architectures) and familiarity of domain stakeholders with a particular style can also be considered and taken into account in this process. For example, in the extraction of a reference architecture for the RDBMS domain, it was recognized that the most RDBMSs reflect the underlying ANSI-SPARC three-level data architecture that emphasizes three-level separation of concerns [3].

**Step 2. Commonality Analysis:** Using previously extracted architectural style and Eixelsberger's commonality analysis [7], the corresponding component-level details of the reference architecture can be extracted. This process emphasizes representation of the core functionality found in the related ADM through as-minimal-as-possible set of components and as-simple-as-possible structure.

**Step 3. Functionality-to-Component Mapping:** Represents a verification that all of the core functionality contained in the related ADM is satisfied by the extracted reference architecture. Most suitably represented in a tabular format where each of the ADM functionalities is described in the context of a particular reference architecture component. Adjustments are made for the missing or incorrectly represented functionality.

**Step 4. Evaluation:** The extracted reference architecture will not satisfy all aspects of interests in regards to a particular program family. Therefore, in this step, the scope of applicability of the extracted reference model is explicitly stated and justified.

Figure 3 depicts a reference architecture for RDBMSs that was derived using the afore-mentioned approach.

## 3.4. ADAS: Directions for Future Evolution

After extracting the model of the domain through the previous three artifact subsets, in the context of understanding a particular domain, it is also important to consider directions for future evolution of applications in the domain. Having a clear insight of the related future trends allows interested stakeholders to proactively deal with the evolution of software systems in the domain.

The evolution of a particular domain is characterized by the following [5, 13]:

- Number of iterations
- Total time of the evolution process
- Scale of iteration
- Type of iteration (*i.e.*, extension, correction, adaptation, perfection)
- Granularity of increments in terms, size, or components
- Staff (*e.g.*, how many person involved)

Identification and classification of potential evolutionary directions for the application domain is necessary to proactively deal with change of any particular member of the program family. Such information is collected in this ADAS component, and it includes data of the form:

- Problem / Cause — description of the environmental or system factor that requires change;
- Iteration Type — indicator whether a change is an extension, correction, adaptation, or perfection of the existing functionality; and
- Direction — description of the future evolutionary direction.

In an industrial setting, this evolutionary information could be extended with the application or product line specific information. Such an extension will describe factors and resulting directions of change not already included in the directions for future evolution of the domain itself.

## 3.5. Creating and Maintaining ADAS

For a well-established domain such as RDBMS, successfully creating an ADAS requires:

- in-depth review of the available literature for the domain, and available documentation or code bases for the applications in the domain;
- experience in architecture recovery of at least one application that is an authentic representative of the selected program family; and
- consultation with the domain experts for validating the extracted domain knowledge.

After creating the ADAS repository, one then needs to make it available to other software engineers working in the area or in the domain to facilitate reusability and extensibility. However, systematic creation and maintenance of ADAS is an area of future research.

Extracting ADAS for an emerging application domain (*e.g.*, web servers) would not be as straightforward given that the information about such a domain is not as easily available and not as clearly specified in literature. In such situations, one would have to focus on extracting the required domain information (*e.g.*, domain fundamentals) first and then work on documenting such information in the context of ADAS structure. More specifically, for the creation of ADAS for various other domains, the following aspects would be of importance:

**Application-Domain Maturity** — maturity indicator for a particular application domain. Most clearly shown by the accumulated amount of recognized information (*i.e.*, books, papers, tech reports) about applications in one such domain. For a more mature domain such as RDBMS, extraction of the required ADAS components would focus on unifying the information from various sources instead of creating it from scratch. For a less mature domain, the required ADAS components would first have to be created and then specified in the ADAS context, thereby leading to greater complexity and time requirements.

**Application-Domain Complexity** — complexity indicator for a particular application domain. Some domains are relatively easy to understand and quite clearly represent a particular functionality set (*e.g.*, compilers). At the same time, some domains are relatively hard to understand, and represent very complex functionality sets (*e.g.*, large GUI-intensive applications). For these relatively complex application domains, creation of ADAS would first have to entail clear definition of scope of analysis based on the identification of particular functionality set, and then would have to proceed with the extraction of other ADAS components.

**Application-Domain Size** — size indicator for a particular application domain. Domains represented by a relatively large number of applications (*e.g.*, image and graphics editing applications) would be more complex to analyze with precision as the coverage of all of these applications would entail significant amount of time and effort.

**Application-Domain Variability** — change indicator for a particular application domain. Rate of change for a particular domain would be a direct influence on the usefulness of the extracted ADAS. For less mature domains that are still experiencing rapid change (*e.g.*, web applications), creating ADAS would not be as beneficial given that the extracted information would, after the occurrence of the next major shift in the application domain, lose its significance and would need to be updated. Therefore, the amount of time spent in creating and updating of the corresponding ADAS could outweigh the related benefits.

## 4. Enhancing Domain-Specific Software Architecture Recovery: Guidelines

Detailed extraction of software architecture artifacts in practice is not always feasible [16]. For the domain with the corresponding architectural artifacts and the application from the program family that represents that domain, this recovery should not aim for overall program comprehension, due to related costs and time requirements, but could instead aim to benefit a particular evolutionary or program comprehension aspect of the system. Every one of these recovery efforts is then meant to satisfy a particular set of goals, and such goals are not trivial and have to be expressed in detail.

Derivation of these goals is made easier by having the ADAS for the domain in question available before starting the extraction endeavor, and using ADAS to derive the semantic information (*e.g.*, rationale, meaning) about different elements in the actual architecture of the system. Furthermore, measuring the success of reverse engineering efforts in general depends on the initial context of extraction that is clearly specified. Based on the context, various extraction strategies are then to be identified and evaluated, and the best-matching one is selected. Derivation of such strategies is supported by the ADAS knowledge base.

After establishing the context and selecting the corresponding strategy, the actual execution can begin and is performed iteratively with the evaluation stage and if necessary recovery context and strategy adjustments at the end of each cycle. Initiation of the next cycle will depend on whether the current extraction results satisfy the initial context and if they do, to what extent, and the human judgement reflecting the overall re-engineering strategy will decide if another iteration is indeed required. Figure 4 depicts the proposed domain-specific architecture recovery method, provided only as guidelines on how to use ADAS in domain-specific software architecture recovery.

## 5. Usefulness of ADAS

We now relate ADAS to previously published research to validate its usefulness and applicability, and to determine what improvements are made through ADAS over the existing methods.

### 5.1. ADAS vs Application Domain Model

Having an application domain model itself is not sufficient for successful architectural recovery efforts, and extraction of architectural views for a particular application requires more than just an understanding of the underlying functionality. In addition, it requires in-depth understanding

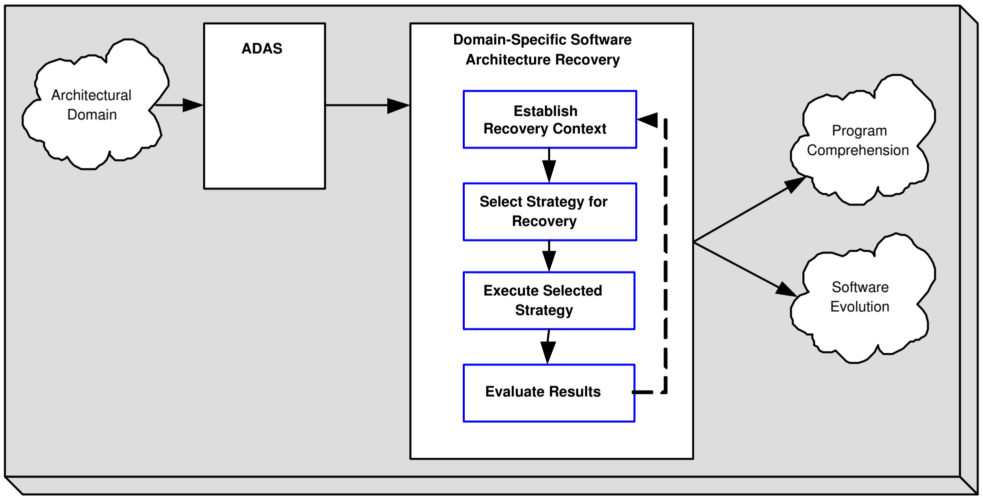
Architectural Domain
ADAS
Domain-Specific Software Architecture Recovery
Establish Recovery Context
Select Strategy for Recovery
Execute Selected Strategy
Evaluate Results
Program Comprehension
Software Evolution

**Figure 4. Enhancing Domain-Specific Architecture Recovery with ADAS**

of the domain itself and the related theories and technologies.

ADAS includes a version of the application domain model, which we named architectural domain model and specified using previously published research by Clements [2], and therefore provides equivalent benefits to it in terms of architecture recovery. However, ADAS also includes related fundamental domain information, which helps facilitate more efficient understanding of the domain and related theory and technology.

### 5.2. ADAS vs Reference Architecture

A reference architecture represents a mapping of the domain model onto software components and data flows among components, which is extracted by analyzing relevant case studies in the domain. Benefits of reference architecture include reduction of software development and maintenance complexity, and streamlining the documentation production.

ADAS includes a reference architecture for the domain, but unlike previously published research, it provides detailed subdomain information and a systematic approach for its extraction (*e.g.*, Eixelsberger's commonality analysis). It therefore provides equivalent benefits of reference architecture along with systematic guidelines for its extraction and use. Moreover, it provides directions for future evolution which is useful for understanding the evolutionary context of architecture recovery.

### 5.3. ADAS as a Teaching Tool

Recently, the ADAS structure was used in a third year software engineering course at Wilfrid Laurier University as a structure to facilitate understanding of a particular domain [10]. Students in the course were asked to analyze an application domain of interest, and create ADAS-like knowledge bases about the applications in the domain. After the project was completed, student projects were read and marked, and were mostly quite successful. In general, students were able to use this structure to understand a complex application domain (*e.g.*, operating systems, word processors, cryptography software, enterprise management systems, graphics software), extract needed information from the domain, and produce precise documentation that was well structured and easily comprehended by a non-expert.

Based on this experience of successfully applying ADAS to application domains of varied complexity, size, and maturity, we conclude that this structure has potential as a teaching tool in software engineering. ADAS provides means to more efficient learning about applications in a particular domain and becoming a domain expert.

## 6. Conclusions

In this paper, we have proposed a structure called the Architectural Domain Assets Set (ADAS), an architectural knowledge base that can aid architecture recovery and evolution through domain understanding. We have also provided guidelines on how to use ADAS through an iterative approach to domain-specific architecture recovery.

Based on our findings, we claim that our proposed ADAS structure provides benefits including:

- Expanding previously published application domain models with the corresponding fundamental domain information for more efficient understanding of the domain, and related theory and technology.

- Expanding previously published reference architecture information by including
  - a systematic approach for its extraction,
  - specific area of its applicability (*i.e.*, exact application type based on domain taxonomy),
  - rationale used in the extraction, and
  - directions for its future evolution.
- Allowing reusability of architectural domain knowledge through its lightweight character that is not architectural style or recovery process specific.
- Becoming a teaching tool by showing potential for facilitating more efficient learning about applications in a particular domain.

In general, based on our experience on architecture recovery of IBM's DB2, it is apparent that extracting this structure and having it available before initiating the recovery process would have:

- helped bring focus into the entire effort,
- assisted with particular steps in the process, and
- provided more value to the system developers who claim interest in the endeavor's final results.

### 6.1. Future Research

Future research in this area includes:

- Refinement of the proposed ADAS structure through additional case studies in RDBMS domain and other well-established domains.
- Refinement and more detailed specification of an iterative process for domain-specific architecture recovery that is ADAS-based as we have only provided guidelines for how to use ADAS.
- Development of an infrastructure for creating, accessing and expanding the ADAS structure for a particular domain.

## References

[1] J. Bergey, D. Smith, N. Weiderman, and S. Woods. Options analysis for re-engineering (oar): Issues and conceptual approach. Technical Report CMU/SEI-99-TN-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1999.

[2] P. Clements. From domain models to architectures. In *Proceedings of the Workshop on Software Architecture 1994*, Los Angeles, CA, 1994. USC Center for Software Engineering.

[3] T. Connolly and C. Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, Essex, England, 2002.

[4] I. B. M. Corporation. Db2 product family. Online, 2002. http://www.ibm.com/software/data/db2/.

[5] S. Demeyer, T. Mens, and M. Wermelinger. Towards a software evolution benchmark. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE2001)*, Vienna, Austria, September 2001.

[6] L. Ding and N. Medvidovic. Focus: A light-weight, incremental approach to software architecture recovery and evolution. In *Proceedings of the 2001 Working IEEE/IFIP Conference on Software Architecture (WICSA 2001)*, Amsterdam, the Netherlands, August 2001.

[7] W. Eixelsberger. Recovery of a reference architecture: A case study. In *Proceedings of the 3rd International Workshop on Software Architecture*, Orlando, FL, November 1998.

[8] R. R. Group. Rigi: A visual tool for understanding legacy systems. Online, 2000. http://www.rigi.csc.uvic.ca/.

[9] R. C. Holt. Pbs: The portable bookshelf. Online, 2002. http://www.swag.uwaterloo.ca/pbs.

[10] I. Ivkovic. Cp 317 - software engineering. Wilfrid Laurier University, Online, 2002. http://sauron.wlu.ca/physcomp/cp317/.

[11] I. Ivkovic. Enhancing domain-specific software architecture recovery. Master's thesis, University of Waterloo, 2002.

[12] L. Kean. Feature-oriented domain analysis. Software technology review, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997. http://www.sei.cmu.edu/str/descriptions/foda.html.

[13] T. Mens. Proceedings of ecoop2002 workshop on benchmarks for empirical studies in object-oriented software evolution. Online, 2002. http://www.cs.kuleuven.ac.be/ dirk/ada-belgium/events/02/020610-ecoop-besoose.html.

[14] S. R. Schach. *Classical and Object-Oriented Software Engineering with UML and C++*. WCB/McGraw-Hill, Reading, MA, 1998.

[15] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Upper Saddle River, NJ, 1996.

[16] M. Shaw. The coming-of-age of software architecture research. ICSE 2001 Keynote Presentation, 2001.