Position Paper:
Challenges in Visualizing and Reconstructing Architectural Views

Juergen Rilling
Concordia University,
QC, Canada, H3G1M8
rilling@cs.concordia.ca

Michel Lizotte
Defence R&D Canada
QC, Canada, G3J 1X5
Michel.Lizotte@drdc-rddc.gc.ca

Abstract

A common approach to cope with software architecture comprehension is to provide higher levels of abstraction of lower level system information. Architectural recovery tools provide such high-level views by extracting and abstracting a subset of the software entities. In this research we are focusing on challenges in visualizing and reconstructing architectural views. In particular we are looking into issues related to the applicability of current visualization representations generated by architectural recovery tools to support views and products specified by the C4ISR architecture framework.

1. Introduction

One aid to improve the understanding of large programs is to reduce the amount of detail a programmer sees by using a higher level of abstraction to represent a program. Over the last decade, programs became larger and more complex, causing new challenges to the programmer in visualizing these complex and large source code structures. Different techniques and approaches have been developed and validated with users. However, providing different levels of abstraction might not be sufficient since users might be still dealing with a large amount of information and data. Not every visualization technique is equally usable in displaying a particular dataset. The visualization technique might lack an appropriate navigation support or may not allow the effective reduction of the amount of information displayed through a choice of distinct views.

Software visualization can be described as analyzing a subject system (a) to identify the system's components and their interrelationships, (b) to create representations of a system in another form at a higher level of abstraction and (c) to understand the program execution and the sequence in which it occurred. It would be ideal to be able to simultaneously view and understand detailed information about a specific activity in a global context at all times for any size of program. As Ben Shneiderman explains in [12], the main goal of every visualization technique is: "Overview first, zoom and filter, then details on demand". This means that visualization should first provide an overview of the whole data set then let the user restrict the set of data on which the visualization is applied, and finally provide more details on the part the user is interested in. Software visualization of source code can be further categorized in static views and dynamic views. The static views are based on a static analysis of the source code and its associated information and provide a more generic high-level view of the system and its source code. The dynamic view is based on information from the analysis of recorded or monitored program execution. Based on their available run-time information, dynamic views can provide a more detailed and insightful view of the system with respect to a particular program execution. As Mayhauser [9] illustrated, dynamic and static views should be regarded as complementary views rather than being mutually exclusive. Users tend to apply an opportunistic approach, using both static and dynamic views to achieve a specific task. The software visualization techniques used by recovery tools are in most cases a carry over from the more traditional reverse engineering tool domain. With the majority of tools providing support for UML visualization based techniques or procedural orientated visuals, like call-graphs, tree structures. Ideally, the high-level views provided by these tools should be organized in a hierarchical/layered fashion, allowing users to navigate through different layers of abstraction.

Software Architecture

Software architecture has been defined as a structure composed of components and rules characterizing the interaction of these components [13]. In [11] it has been defined as elements, form, and rationale. Another definition is presented in [6] where it was defined as components, connectors, and configurations [6]. C4ISR AF is using a definition, not limited to software, based on the IEEE STD 610.12 and established by the DoD

Integrated Architecture Panel in 1995 [7]. They define *"architecture"* as *"the structure of components, their relationships, and the principles and guidelines governing their design and evolution over tim*e." One of the earliest definitions of software architectures, by Perry and Wolf [6], has remained one of the most insightful.

Architecture Recovery

Architecture recovery can be seen as a discipline within the reverse engineering domain that is aimed at recovering the software architecture of a system [2]. It can be described as the process of recovering up-to-date architectural information from existing software artefacts [2, 16]. The rational of system architectural recovery and comprehension is to provide reasoning behind the software architecture or high-level system organization of a system. There may be little or no documentation available and the documentation that does exist probably does not resemble the current system due to drift and erosion [3]. The application of system understanding tools goes beyond mere object identification - it includes a generation of (interactive) documentation, quality assessment, and introducing novice programmers to a legacy application. Architectural recovery is motivated by (re)generate coherent abstractions of existing systems to guide analysts during the comprehension of large existing systems and to provide some reasoning about the system architecture.

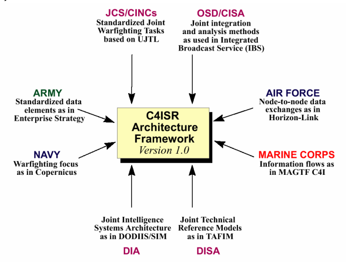
Motivation

The presented research is conducted under a project of the Defense Research and Development Canada (DRDC) at Valcartier. The focus of this project is the visualization support for the various products described in the US Department of Defense (DoD) Architectural Framework (AF), better known as the Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework (AF) [10]. As part of this research, we extended a previously performed survey of current reverse and architectural recovery tools, with a focus on visualization support for C4ISR AF, its views and products. Tools should provide adequate visualization support, by providing on the one hand users with views and information abstraction that are beneficial for the recovery process, as well as visualization techniques that are required by architectural frameworks to document the architecture.

The remainder of this article is organized as follows. Section 2 introduces provides a brief overview and background C4ISR architectural framework. Section 3 maps and discusses the applicability of the surveyed tools to the C4ISR AF. Section 4 provides a discussion about challenges and pitfalls of current visualization techniques in supporting architectural views.

2 The DoD Architecture Framework

The purpose of the DoD AF is to improve capabilities by enabling the synthesis of requirements with sound investments leading to the rapid employment of improved operational capabilities, and enabling the efficient engineering of warrior systems. This framework formerly called the Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) *Architecture Framework* [10] is intended to ensure that the architecture descriptions developed by the Commands, Services, and Agencies are inter-relatable between and among each organization's operational, systems, and technical architecture views, and are comparable and able to integrate across Joint and combined organizational boundaries. It provides the rules, guidance, and product descriptions for developing and presenting architecture descriptions that ensure a common denominator for understanding, comparing, and integrating architectures. This section is based on the C4ISR Architecture Framework (Version 2.0 as published by the AWG)



JCS/CINCs
Standardized Joint Warfighting Tasks based on UJTL
OSD/CISA
Joint integration and analysis methods as used in Integrated Broadcast Service (IBS)
ARMY
Standardized data elements as in Enterprise Strategy
AIR FORCE
Node-to-node data exchanges as in Horizon-Link
C4ISR Architecture Framework Version 1.0
NAVY
Warfighting focus as in Copernicus
MARINE CORPS
Information flows as in MAGTF C4I
Joint Intelligence Systems Architecture as in DODIIS/SIM
Joint Technical Reference Models as in TAFIM
DIA
DISA

Figure 1: C4ISR Architecture Framework

The *operational architecture view* is a description of the tasks and activities, operational elements, and information flows required to accomplish or support a military operation. It contains descriptions (often graphical) of the operational elements, assigned tasks and activities, and information flows

The *systems architecture view* is a description, including graphics, of systems and interconnections. For a domain, the systems architecture view shows how multiple systems link and interoperate, and may describe the internal construction and operations of particular systems within the architecture.

For an individual system, the systems architecture view includes the physical connection, location, and identification of key nodes (including materiel item nodes), circuits, networks, warfighting platforms, etc., and specifies system and component performance parameters (e.g., mean time between failure, maintainability, availability). The systems architecture view associates physical resources and their performance attributes to the operational view and its requirements per standards defined in the technical architecture.

The *technical architecture view* is the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements.

In what follows, we present a case study based on a survey of 23 architectural recovery and reverse engineering tools (see appendix) that was performed as part of this project and map their capabilities in supporting the visualization products described in the C4ISR system view. The other two views described in the C4ISR, the operational and technical view were not considered in this survey, since these views are mostly based on domain knowledge, rather than information that can be recovered by analyzing program artifacts.

3. Case study – C4ISR Capability matrix

The motivation for the presented case study and the resulting C4ISR visualization support capability matrix are two-fold. The first objective was to analyze the current state of the art support of architectural views and visualization techniques provided by recovery tools and their applicability in support for the different visualization products described in the system view of the C4ISR architectural framework. Secondly, the resulting capability matrix can serve as guidance for directing future research, by addressing shortcomings of current tools.

Visualization techniques supporting system view products

The system view products described within the C4ISR architecture framework suggest certain visualization and diagrammatic techniques that should be provided to document an existing architecture. One intend of the C4ISR AF was to guide tool developers by providing templates for suitable/expected visualization and representation techniques, to support the various system view products. The suggested templates are not compulsory and can be replaced by other visualization techniques. There is a currently a tendency in applying the standard UML notations to document software architectures within the C4ISR framework. This approach has both advantages and disadvantages.

Advantages can be found in using a well-known standard notation, in reducing the learning overhead that might be caused by introducing new visualization techniques and their notations. Furthermore, over the last several years, UML established itself as a viable approach for documenting various aspects of the requirement, specification and design phase

One of the major disadvantages of the UML standard notation is its limited expressiveness with respect to architectural aspects. Firstly, its notation does not provide enough expressive power to describe the specific requirements of architectural artifacts. Secondly, the levels of abstraction provided by UML might not be sufficient to provide some of the required views.

The open framework approach of the C4ISR AF with respect to visual representations encourages tool developers to explore new avenues and derive new visualization techniques that might lead to more intuitive and architectural specific representations. In particular tool developers are facing during architectural recovery additional challenges having no or only limited domain knowledge available to derive the visual abstractions.

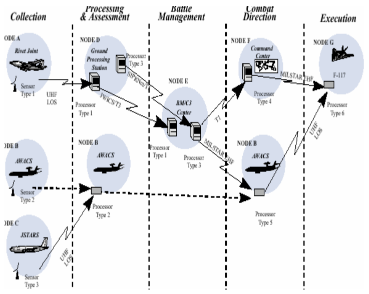


Figure 2: System interface description

Figure 2 and 3 illustrate this situation, with figure 2 abstracting the system interfaces in a high-level view (using a non UML notation), which can easily be understood by both novice and experts. Comparing this with the UML view of the system interface description (Figure 3), the differences in both the capabilities, abstractions and applicability of the visualization becomes evident.

The following are some of the visualization techniques templates described in the C4ISR standard document that should be created to document system view specific products.

With current recovery tools focusing on the structural analysis of existing system artifacts, one of the challenges can be found in the reconstruction of visual abstractions is their lack of domain knowledge. Figure 2 is an example for domain knowledge based visualization. The graphic requires not only specific annotations, but also domain specific representations of the objects (e.g. different types of airplanes) involved in the system and their intercommunication.

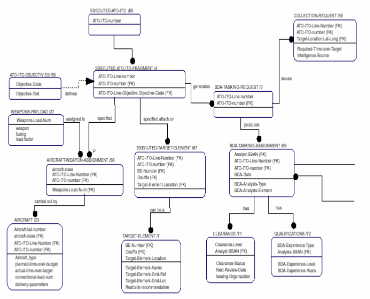


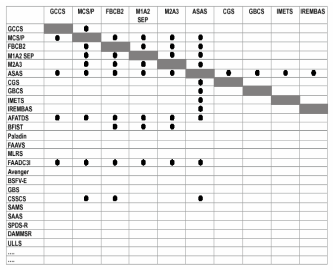
Figure 3: System interface description (UML based)

Figure 3 on the other hand is based solely on structural analysis through lexical and semantic parsing of existing system source code. This information can almost completely automatically be extracted, without any prior domain knowledge.

Some other visualization challenges include the support for building traceability matrixes. These traceability matrixes are an essential part of architectural documentation and re-documentation not only within the C4ISR architectural framework but also within other frameworks (e.g. Zachmann). Matrixes are used widely by the following products within the system view (C4ISR):

System Performance Matrix: Depict current performance of each system, and the expected or required performance characteristics at specified times in the future (soft and hardware).

Operational Activity to System Function Traceability Matrix: Maps operational activities to system functions in the form of a matrix (Figure 4)



	GCCS	MCS/P	FBCB2	M1A2 SEP	M2A3	ASAS	CGS	GBCS	IMETS	IREMBAS
GCCS	▨	●								
MCS/P	●	▨	●	●	●	●				
FBCB2		●	▨	●	●	●				
M1A2 SEP		●	●	▨	●	●				
M2A3		●	●	●	▨	●				
ASAS	●	●	●	●	●	▨	●	●	●	●
CGS						●	▨			
GBCS						●		▨		
IMETS						●			▨	
IREMBAS						●				▨
AFATDS	●	●	●	●	●	●				
BFIST			●	●	●					
Paladin										
FAAVS										
MLRS										
FAADC3I	●	●	●	●	●	●				
Avenger										
BSFV-E										
GBS										
CSSCS		●	●			●				
SAMS										
SAAS										
SPDS-R										
DAMMSR										
ULLS										
....										
....										

Figure 4: *Operational Activity to System Function Traceability Matrix*

System Information Exchange Matrix: Shows the data exchange among nodes in different systems in the form of a matrix.

Systems Matrix: The product focuses on the flow of data among system functions, and on the relationships between systems or system functions and activities at nodes.

Behavioral modeling

Within the C4ISR architectural framework the importance of documenting and being able to trace the dynamic and behavioral system aspects is reflected by the following system behavior modeling products.

? *Systems Rules Model:* A rule base for actions occurring as part of the trace. The rule base applies for the different visualization techniques within the system activity product.

? *Systems State Transition Description:* State transition descriptions describe system responses to sequences of events. Events may also be referred to as inputs, transactions, or triggers (Figure 5).

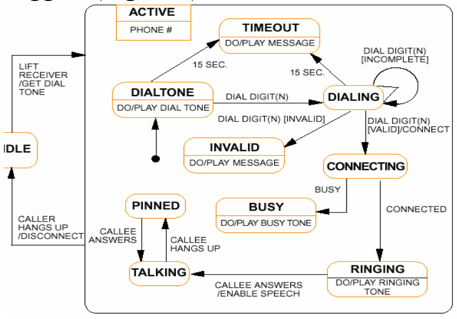


Figure 5*: State transition diagram*

? *Systems Event/Trace Description:* The system event trace describes the timing and behavior (based on the rule model) between nodes, as well as the interaction among these nodes. The standard UML sequence diagram notation can be applied to capture the behavior (Figure 6).

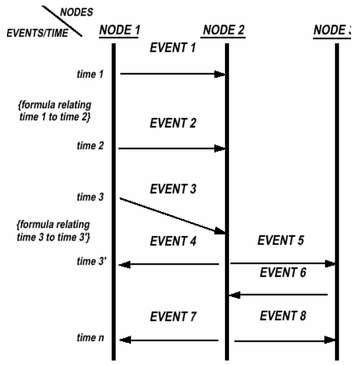


Figure 6. Modeling dynamic behavior within the C4ISR AF

Physical Data Model: Describes the physical implementation of the logical data model from an operational view point. The product is supported in the form of standard E/R diagrams, etc.

Capability matrix

Table 1 shows a capability matrix (based on the survey of 23 architectural recovery and reverse engineering tools) and maps their overall capability to the products and their visual representations as described by the C4ISR architecture framework. The matrix provides a general summary of the overall tool capabilities rather than focusing on the specifics of a particular tool.

System view product	*Visualization Support*
System Performance Parameters Matrix	Partially
Systems Functionality Description	Partially
Operational Activity to System Function Traceability Matrix	No
System Information Exchange Matrix	No
System Interface Description	Partially
Systems Communications Description	Partially
Systems Matrix	Partially
System Evolution Description	No
System Technology Forecasts	No
Systems Rules Model	No
Systems State Transition Description	Partially
Systems Event/Trace Description	No
Physical Data Model	Fully

Table 1: Visualization Capability

Partial visualization support is achieved if at least one or more tools provide capabilities required by the particular system view product. The capabilities are often limited and do not exist, because of a lack of domain knowledge, that is necessary to re-create these views and products.

4. *Discussion: Challenges and Pitfalls*

Larger software systems place an enormous cognitive load on users and humans are limited in the density of information they can resolve and comprehend [5,8]. Visualization facilitates the discovery of new science by revealing hidden structures and behaviours in model output. It is in the areas of insight and understanding that visualization plays a central role [8]. Many reverse engineering tools have been built to help the comprehension of large software systems. Software visualizations are one approach being investigated worldwide to provide some assistance in program understanding. It should be recognized that visualization is a complementary technique and is to be used in conjunction with other program understanding techniques such as software inspection, metrics, static and dynamic source code analysis, etc.

Throughout a software product's life cycle, many different people are responsible for understanding the design details of the software code. Learning the structure of code developed by others is especially time consuming and effort intensive during the software maintenance phase. From an architectural recovery perspective the challenges becomes even more aggravating, because the maintainer has to create a mental model of a larger system that might include several subsystem and the interaction among these subsystems.

One of the shortcomings of current architectural recovery tools is their lack of supporting architectural views and abstractions. Architectural views require often notations other than the ones provided by current reverse engineering tools In particular traditional visualization techniques are limited by their available notations and their ability to map between visualization elements and architecture components (e.g. throughput, dynamic linked information, etc.). Other factors are the lacking support for architectural views that match the more traditional architectural views (e.g. 4+1 or C4ISR AF). The creation of architectural views requires often additional user domain knowledge, architectural design decisions and analysis support in form of grouping/clustering have also to be considered.

The majority of the surveyed tools focus on the visualization of static system structures rather than dynamic interaction aspects. System architectures are often based on distributed and dynamic systems that take run-time behaviour into consideration. In particular the mapping of these dynamic architectural aspects to the static visualization techniques is often difficult, because these techniques do not support natively graphical notations for representing these dynamic aspects. Examples are their lack of support for e.g. remote connectors, throughput, performance, resource requirements, etc. Furthermore, in visualizing architectures there exists an explicit need for views and visualization techniques that are based on dynamic tracing and profiling aspects. This aspect are addressed and acknowledged for example by the System Activity Sequence and Timing product in the C4ISR Architecture Framework. For architectural recovery tools to be able to manage and display dynamic behaviour, often a large amount of data has to be processed. Additionally, the tools have to facilitate notations that support the visualization of these dynamic aspects.

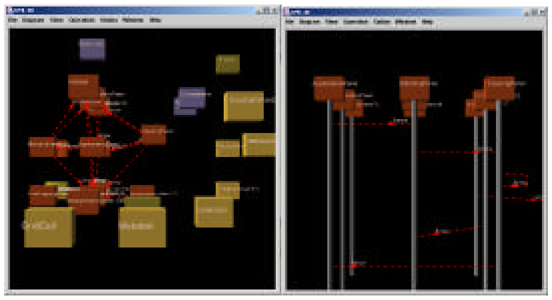


Figure 7 Moving to 3D worlds

Furthermore visualization techniques should take advantage of 3D [5], virtual reality [8], multimedia to provide intuitive and meaningful representations of the underlying architectural structure, its behaviour and relationships. In the context of the C4ISR framework there are further needs to provide views that combine system views with operational views, as well as the technical with the system view. Feature extraction and concept analysis techniques have to be integrated to facilitate this. Clustering and grouping requires application–specific data and domain knowledge, as well as source code analysis techniques. It is important to note that clustering can be used for functions such as filtering and search. Scripting support is also essential to create abstract views on the underlying repository

Different levels of granularity are required, often not facilitate in the current tools, e.g. UML does not provide enough meaningful levels of abstraction. Navigation and context switching has to be further improved to help the architects and maintainers to navigate through the recovered information.

6. References

1. Ball T., Eick Stephen G., "Software Visualization in the Large". *IEEE Computer* 29(4): 33-43 (1996).
2. O'Brien, L., Stoermer, C., Verhoef, C. 2002. Software Architecture Reconstruction: Practice Needs and Current Approaches ; CMU/SEI-2002-TR-024 ADA407795
3. Clements, P., Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. 2002. Documenting Software Architectures: Views and Beyond. Boston, MA: Addison-Wesley.
4. Deursen van A. 2002.Software Architecture Recovery and Modelling. ACM Applied Computing Review 10(1):4-7.
5. Feijs, L.M.G. & de Jong, R.P. 1998. 3D Visualization of Software Architectures. Communications of the ACM 41, 12 (December 1998): 73-78.
6. Garlan, D. and M. Shaw, An introduction to software architecture, in: V. Ambriola and G. Tortora, 1993, Advances in Software Engineering and Knowledge Engineering, World Scientific Publishing Company, 1993 pp. 1--39.
7. Institute of Electrical and Electronics Engineers. IEEE Std 1471-2000. Piscataway, NJ: IEEE Computer Press.
8. Knight C., Munro M., 2001. Visualising the non-existing", IASTED International Conference: Computer Graphics and Imaging, Hawaii, USA..
9. Mayrhauser A., A. M. Vans, "Program Understanding Behavior During Adaptation of Large Scale Software", *Proceedings of the 6th Intl. Workshop on Program Comprehension*., IWPC '98, pp. 164-172, Italy, June 1998.
10. Office of the Secretary of Defense Working Group. 1997 C4ISR Architecture Framework, Version 2.0. Washington, DC.
11. Perry D. E. and Wolf A. L. 1992, Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, 17:40--52, October 1992.
12. Shneiderman, Ben, "Tree Visualization with Tree-Maps: A 2-D Space-Filling Approach". In *ACM Trans. of Computer-Human Interaction*, vol. 11, no. 1, 1992, pp. 92-99.
13. Shaw M. and Garlan D. 1996. Software architecture: Perspectives on an emerging discipline, Prentice-Hall.
14. Sneed, H. M. 1998.Architecture and Functions of a Commercial Software Reengineering Workbench. 2-10. Proceedings of the Second Euromicro Conference on Maintenance and Reengineering. Florence, Italy, March 8-11. Los Alamitos, CA: IEEE Computer.
15. Storey M.-A., Fracchia F. and Müller H.., "Cognitive Design Elements to support the Construction of a Mental Model During Software Exploration, *Journal of Software Systems*, special issue on Program Comprehension, v 44, pp.171-185, 1999
16. Trevors A. and Godfrey M.W., 2002. Architectural Reconstruction in the Dark, Position paper, Workshop on Software Architecture Reconstruction collocated with WCRE '02, Richmond, VA, October 2002

Appendix A: Tool survey

1. Argo/UML : http://argouml.tigris.org/servlets/ProjectSource
2. Bauhaus: http://www.informatik.uni-stuttgart.de/ifi/ps/bauhaus/
3. CIAO http://www.research.att.com/~ciao/
4. CodeCrawler: http://www.iam.unibe.ch/~lanza/CodeCrawler/codecrawler
5. CodeSurfer:http://www.grammatech.com/home/index.htm
6. Columbus/CAN : http://www.frontendart.com/
7. CONCEPTwww.cs.concordia.ca/CONCEPT
8. The Dali Architecture Reconstruction Workbench. http://www.sei.cmu.edu/ata/products_services/dali
9. Fujaba: http://www.uni-paderborn.de/cs/fujaba/
10. G^{SEE}http://www-adele.imag.fr/~jmfavre/GSEE/
11. Headway: http://www.headwaysoft.com/index.htm
12. Imagix4Dhttp://www.imagix.com/index.html
13. KLOCworkinSight. www.klocwork.com/products/inSight.
14. ManSARThttp://www.mitre.org/pubs/edge/january_98/first
15. Rational http://www.rational.com/index.jsp
16. Red Hat Source-Navigatorhttp://sourcenav.sourceforge.net/
17. Refine/C Illuma: http://www.frontendart.com/
18. SniFF++: http://www.takefive.com/
19. SoftArch: http://www.cs.auckland.ac.nz/~john-g/projects.html#softarch
20. Soloway E. and Ehrlich K.,1994. Empirical studies of programming knowledge, IEEE Transactions on Software Engineering, SE-10, 595--609 (1984).
21. SWAG tool kit: http://swag.uwaterloo.ca/pbs/
22. Understand for C++: http://www.scitools.com/ucpp.html
23. Visual Paradigm :http://www.visual-paradigm.com/index.php