AFRL-IF-RS-TR-2004-275
Final Technical Report
October 2004

DYNAMIC ASSEMBLY, ASSESSMENT, ASSURANCE, AND ADAPTATION VIA HETEROGENEOUS SOFTWARE CONNECTORS

University of Southern California at Los Angeles

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K514

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

APPROVED: /s/
RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: /s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** OCTOBER 2004	**3. REPORT TYPE AND DATES COVERED** Final Jun 00 – Jun 03

4. TITLE AND SUBTITLE DYNAMIC ASSEMBLY, ASSESSMENT, ASSURANCE, AND ADAPTATION VIA HETEROGENEOUS SOFTWARE CONNECTORS	**5. FUNDING NUMBERS** C - F30602-00-2-0615 PE - 62301E PR - DASA TA - 00 WU - 13
6. AUTHOR(S) Barry Boehm and Nenad Medvidovic	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Southern California at Los Angeles 837 West Downey Way Los Angeles California 90089-1147	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714; AFRL/IFTB 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2004-275

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Raymond A. Liuzzi/IFTB/(315) 330-3577/ Raymond.Liuzzi@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

This effort provided innovative capabilities for two key stages of software development. During specification and design time, component mismatch detection gauges are provided, indicating the particular type, dimension, and value of the mismatch. This mapped into the taxonomy of software architectural connectors used for resolving the mismatch. Examples of mappings from the mismatches into the effective classes of connectors included procedure calls, events, arbitrators, adaptors, and distributors. This effort also developed techniques for specifying and analyzing properties of product line architectures (PLAs) and extended existing architecture analysis techniques and tools for dynamic composition and assessment/verification to ensure that the selected components and connectors were appropriately configured and dynamically integrated into the operational system. For the deployment and run time stage, this effort focused on application architectures and gauges tailored for distributed, mobile, heterogeneous, and possibly resource constrained platforms. Several different gauges were provided: 1) gauges for assessing new component versions when performing component upgrades, 2) gauges for assessing properties of heterogeneous connectors, and finally, 3) gauges to support awareness and quality of service (QoS) for distributed applications. The research also extended these gauges based on a problem-driven set of priorities.

14. SUBJECT TERMS Knowledge Base, Data Bases, Artificial Intelligence, Software, Information Systems	**15. NUMBER OF PAGES** 20
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1 Objectives 1
2 Approach... 2
  2.1 Introduction and Motivation 2
  2.2 Capabilities and Limitations in Current Software Architecture Technology 2
  2.3 Key elements of USC-CSE’s approach 2
3 Discussion of Tasks 4
  3.1 Incremental Modeling and Analysis of Architectures for Product Families 4
  3.2 Architectural Refinement.. 4
  3.3 Assessing the Structural Quality of Product Line Architectures 4
  3.4 Prism architectural style and Prism middleware.. 5
  3.5 Support for Dynamic Assembly, Assessment, and Adaptation of Heterogeneous Connectors 6
4 Accomplishments.. 7
  4.1 Multi-Versioning Connectors (MVC)... 7
  4.2 Architectural modeling and analysis... 7
  4.3 Prism middleware 8
  4.4 Managing architectural evolution 9
  4.5 Alfa 10
  4.6 Assessing the Quality of Product Line Architectures 10
5 Technology Transitions 12
  5.1 Collaboration with other DARPA DASADA contractors 12
  5.2 Technology Transition to other non DARPA DASADA efforts 12
6 Homepages.. 13
7 Publications... 13

List of Figures

Figure 1: USC-CSE's DASADA approach 3
Figure 2. Class design view of the Prism middleware core components........................... 6

1 Objectives

University of Southern California-Center for Software Engineering (USC-CSE's) dynamic, architecture-based assembly technology provided innovative capabilities during two key stages of software development. During *specification and design time*, component mismatch detection gauges are provided, indicating the particular type, dimension, and value of the mismatch. This mapped into USC-CSE's taxonomy of software architectural connectors [1] used for resolving the mismatch. Examples of mappings from the mismatches into the effective classes of connectors included procedure calls, events, arbitrators, adaptors, and distributors. USC-CSE developed techniques for specifying and analyzing properties of product line architectures (PLAs) [11] and extended existing architecture analysis techniques and tools for dynamic composition and assessment/verification to ensure that the selected components and connectors were appropriately configured and dynamically integrated into the operational system. At the *deployment and run time* stage, USC-CSE’s support focused on application architectures and gauges tailored for distributed, mobile, heterogeneous, and possibly resource constrained platforms [6]. Several different gauges were provided: 1) gauges for assessing new component versions when performing component upgrades, 2) gauges for assessing properties of heterogeneous connectors, and finally, 3) gauges to support awareness and quality of service (QoS) for distributed applications. USC-CSE’s Dynamic Assembly for Systems Adaptability, Dependability, and Assurance (DASADA) research extended these gauges based on a problem-driven set of priorities, determined in concert with a subcontractor, Lockheed Martin.

2 Approach

2.1 Introduction and Motivation

The DARPA DASADA vision involved the orchestrated use of (1) composability *gauges* enabling assessment of software component composability needs, (2) component and connector adaptation mechanisms to enable component composition, (3) verification mechanisms to ensure trouble-free composition, and (4) dynamic composition mechanisms for "on-the-fly" system reconstitution. The DASADA research challenge was to achieve these capabilities in ways that scaled up to large and complex systems, as well as dynamic and complex situations. That was also the reason why the most attractive emerging technology area for achieving the DASADA vision was software architecture and why USC-CSE's efforts were centered on architecture-based system modeling, analysis, implementation, deployment, and evolution.

2.2 Capabilities and Limitations in Current Software Architecture Technology

Software architectures provide a simple set of abstractions: *components* (computational and data storage elements), *connectors* (component interaction facilities), and *configurations* (interconnections of components and connectors in a system). Of special interest to the software architecture community are *software connectors*. Given that the size, sophistication, and complexity of software components are steadily growing, it is reasonable to expect that their interactions will become more complex as well. Connectors facilitate communication, coordination, arbitration, and adaptation of components [1] and have been shown to directly enable architectural dynamism. In principle, architectural connectors provide an attractive set of capabilities for achieving the DASADA vision. USC-CSE's DASADA work indeed focused on connectors as the centerpiece of its approach.

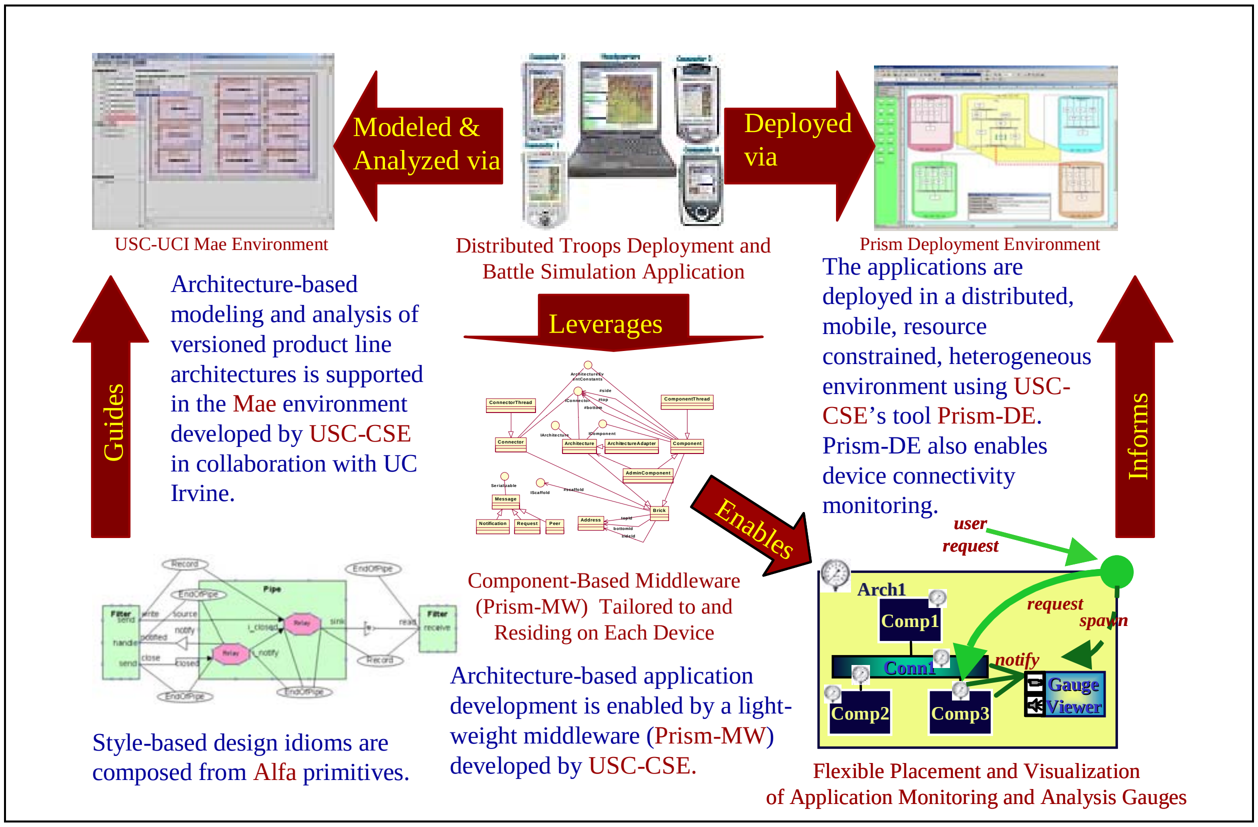
2.3 Key elements of USC-CSE's approach

- USC-CSE's emphasis on architecture-based software development enabled architects to detect mismatches, inconsistencies, and inadequacies early in the development process, thus reducing the overall development costs.
- Support for modeling and analyzing architectures of large and complex software systems provided developers with the ability to detect design problems early. USC-CSE focused on this problem by developing notations and their accompanying toolsets for describing different characteristics of software product families. Managing evolution of architectural artifacts had an impact on developing representation and analysis capabilities for product line architectures (PLA).
- Reusable frameworks enabled architects and developers to build their tools such that they could reuse the communication capabilities of the underlying framework. USC-CSE created a family of such frameworks to be used for architecture-based design and development of software systems on distributed, heterogeneous, mobile, and possibly resource-constrained devices. Communication across the network can be unpredictable and unreliable. Support for disconnected operations, security, and reliability at the architectural level was among the aspects supported by USC-CSE's

frameworks. The frameworks comprised reusable middleware platforms that substantially aided system development.

- Software deployment techniques enabled managing software systems on the running platforms. It is important to develop and apply such techniques in an architecture-centric fashion such that the mapping between the architecture, design, and the running system can be traced and controlled. USC-CSE focused on developing a deployment environment that was architecture-aware and provided capabilities for managing network related problems and their manifestations in the system.
- Measuring system properties at the architectural level is challenging. Defining metrics for comparing and measuring properties of a system's architecture was another focal point of our research.
- Finally, USC-CSE's research focused on identifying and codifying the fundamental principles and "atomic" constructs that underlie all software architectures and architectural styles.

USC-CSE's efforts in the above areas were all concentrated around the same architectural concepts and were aimed in providing support for development from modeling architectures, to analysis, to system implementation, and, finally, evolution. The figure below shows the high level view that connects all the key elements in our approach.



Modeled & Analyzed via
Deployed via
USC-UCI Mae Environment
Distributed Troops Deployment and Battle Simulation Application
Prism Deployment Environment
Leverages
Guides
Informs
Enables
Architecture-based modeling and analysis of versioned product line architectures is supported in the Mae environment developed by USC-CSE in collaboration with UC Irvine.
The applications are deployed in a distributed, mobile, resource constrained, heterogeneous environment using USC-CSE's tool Prism-DE. Prism-DE also enables device connectivity monitoring.
Component-Based Middleware (Prism-MW) Tailored to and Residing on Each Device
Architecture-based application development is enabled by a light-weight middleware (Prism-MW) developed by USC-CSE.
Style-based design idioms are composed from Alfa primitives.
Flexible Placement and Visualization of Application Monitoring and Analysis Gauges
user request
Arch1
Comp1
Conn1
Comp2
Comp3
request
spawn
notify
Gauge Viewer
Pipe
Filter
Relay
Record
EndOfPipe

Figure 1: USC-CSE's DASADA approach

3 Discussion of Tasks

3.1 Incremental Modeling and Analysis of Architectures for Product Families

USC-CSE developed a technique and accompanying tool support for specifying architectures of product families in terms of the key properties of the system (its structure, behavior, non-functional properties, etc.) [11]. The technique also supported the partial specification of services of an architectural component. This was representative of situations in which an OTS component was introduced into a system with a known interface (API), but unspecified behavior. Providing gauges that would quickly determine that such a component was not a good fit for the system would eliminate the costs of (ultimately unsuccessful) integration. A related gauge enabled automated component discovery and retrieval for insertion into an architecture, based on the requirements of the surrounding components/connectors and the specified match precision threshold. Finally, the use of invariants and pre- and postconditions resulted in a static view of component semantics. This static view was augmented with a dynamic behavior technique, namely StateCharts [24]. USC-CSE provided a gauge for ensuring the internal consistency of a given component's static and dynamic models (i.e., ensuring that a component's pre- and postconditions matched its hierarchical state machine).

3.2 Architectural Refinement

Refining an architecture into its implementation in a property-preserving manner was a challenging task for several reasons. Maintaining traceability of decisions and ensuring consistency between two software models at different levels of detail is inherently difficult. Furthermore, different refinement steps may require different modeling notations, motivating the need to augment (high-level) architectural notations with (lower-level) design notations [1]. USC-CSE employed the Unified Modeling Language (UML) for that purpose.

USC-CSE pioneered an approach to relating Architecture Description Language (ADLs) and UML, resulting in the SAAGE environment. SAAGE provides a set of gauges that enable automatic transformation of an architectural model (described in USC-CSE's C2SADEL language) into a corresponding UML model. Another, emerging aspect of this work was a framework for ensuring the consistency of multiple views in a software model. A preliminary prototype implementation of the framework, called UML/Analyzer, was based on UML as the modeling notation and was used in concert with SAAGE.

3.3 Assessing the Structural Quality of Product Line Architectures

USC-CSE developed several novel metrics to assess the structural quality of product line architectures. Throughout the evolution of a product line, these metrics would guide the architect in making more informed architectural decisions. The metrics were based on the concept of service utilization and were designed to take into account the context in which individual architectural elements were placed.

3.4 Prism architectural style and Prism middleware

USC-CSE proposed a new architectural style intended for use in architecting complex, highly distributed, mobile, and resource constrained systems. The software development in this new setting was referred to as *programming-in-the-small-and-many (Prism)*. The style has inherent support for architectural monitoring and analysis, distribution, dynamism, mobility, and disconnected operation. USC-CSE chose to use the existing C2 style as the basis of the Prism style, with three major enhancements to account for a set of new problems that arose in this novel setting:

- Peer-to-peer interactions

  While it is still allowed to have the C2-style vertical topology in Prism architectures and communication via requests and notifications, a third component port (called *side*) and message category (called *peer*) were introduced. Side ports allowed to address the relative topological rigidity of C2. They proved particularly effective in component interactions across devices on a network. In order to maintain component decoupling, the side ports exchange peer messages through “peer” connectors. Basic peer connectors have simple message broadcast semantics: a peer message incoming on any port is forwarded as an outgoing message through all of the connector’s remaining ports.

- Architectural self-awareness

  Prism supports architectures at two levels: application-level and meta-level. The role of components at the Prism meta-level is to gauge and/or facilitate different aspects of the execution, dynamic evolution, mobility, and disconnected operation of application-level components. Application-level and meta-level components execute side-by-side in Prism. Meta-level components are aware of application-level components and may initiate interactions with them, but not vice versa. The Prism style rules apply to both component categories: meta-level components also engage in connector-mediated, message-based interactions with each other (and with application-level components).

  In support of this two-level architecture, Prism distinguishes among four types of messages. Similarly to C2, *ApplicationData* messages are used by application-level components to communicate during execution. The other three message types, *ComponentContent*, *ArchitecturalModel*, and *SystemMonitoring*, are used by Prism meta-level components. *ComponentContent* messages contain mobile code and accompanying information (e.g., the location of a migrant component in the destination configuration); *ArchitecturalModel* messages carry information needed to perform architecture-level analyses of prospective Prism configurations; finally, *SystemMonitoring* messages supply runtime data to Prism gauges.

- Border connectors

  The third significant departure from C2 in formulating the Prism style is the key role of connectors that span device boundaries. Such connectors, called *border connectors*, enable the interactions of components residing on one device with components on other devices. The high degrees of distribution and mobility, as well as the high probability of disconnected operation in Prism architectures created a need to place special importance upon border connectors. A single border connector may service network links to multiple devices. A border connector marshals and unmarshals data, code, and architectural models; dispatches and receives messages across the network; and monitors the network links for disconnection. It may also perform data compression for efficiency and encryption for security.

Prism's middleware, *Prism-MW*, comprises an extensible framework of implementation-level classes representing the key elements of the Prism style (e.g., components, connectors, messages) and their characteristics (e.g., a message has a name and a set of parameters). An application architecture is then constructed by instantiating and/or extending the appropriate classes in Prism-MW with application-specific detail.

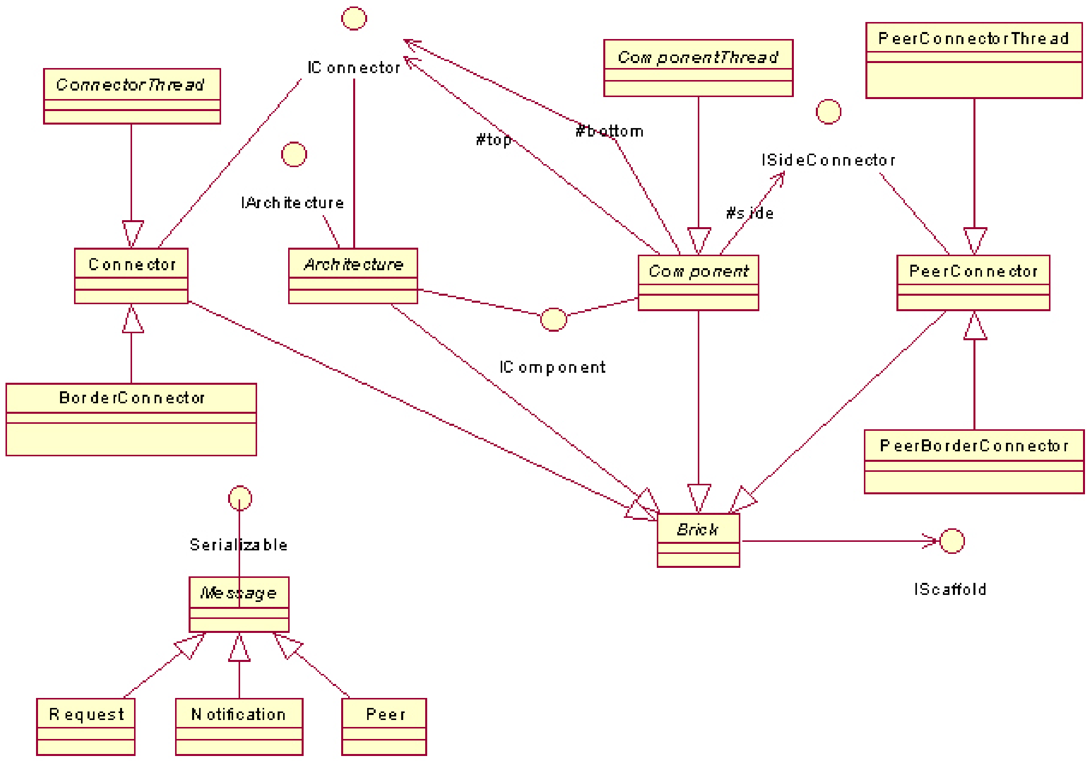


Figure 2. Class design view of the Prism middleware core components

3.5 *Support for Dynamic Assembly, Assessment, and Adaptation of Heterogeneous Connectors*

USC-CSE identified primitives for representing various architectural styles at an "architectural assembly" level. These primitives form an _a_ssembly _l_anguage _f_or _a_rchitectures, Alfa, which can be used to construct various architectural styles. There are four architectural styles based on Alfa that were successfully modeled and implemented:

Client-Server, pipe-and-filter, C2, and push-based. The models indicated that it was possible to construct various architectural styles, as well as various connectors used within different styles, out of the same primitives, and that there were repeating patterns across styles.

4 Accomplishments

On the runtime side of USC-CSE's research, software connectors were upgraded to enable runtime adaptation of an application by modifying its architectural model. USC-CSE provided a light-weight infrastructure for prototyping and implementing architectures, in which connectors remain explicit entities and the relationship between the architectural model and the implementation is maintained. It also provided runtime gauges that identify circumstances under which dynamic manipulation of connectors may and may not be safely performed.

The rest of this section describes the major accomplishments. The complete list of publications is given in Section 7.

4.1 Multi-Versioning Connectors (MVC)

Representative of runtime monitoring gauges are *multiversioning* gauges, which monitor and analyze different versions of the same component that co-exist in a system and execute in parallel [10]. USC-CSE developed connectors that allowed flexible insertion of new component versions and multicasting of invocations and data to both the old and new versions; furthermore, the invocations and data originating from the multiple versions of the same component must be merged by the connector before they are forwarded to their target components. The multiversioning connectors are equipped with gauges enabling them to gather and evaluate each component's runtime behavior to determine properties such as correctness, performance, and reliability.

USC-CSE designed and implemented a preliminary version of multi-versioning software connectors (MVCs), used to gauge and ensure reliable upgrades of components at system runtime. These gauges directly aid the large, complex, long-lived systems undergoing continuous upgrades. Lockheed Martin expressed their interest in employing MVCs in their TBMCS system. National Reconnaissance Office also expressed initial interest in this capability.

4.2 Architectural modeling and analysis

USC-CSE made several enhancements to the existing USC-CSE's DRADEL environment for architecture-based modeling, analysis, and implementation of software systems. A type checking mechanism was developed to gauge the interface and behavior match between a given, possibly only partially modeled component and a target collection of components within an architecture. USC-CSE also developed a gauge to measure the consistency of a static model of a system's architecture described in the C2SADEL architecture description language (ADL), which utilized invariants and pre- and post-conditions, with the architecture's StateCharts dynamic model, which utilized states, transitions, events, and actions. USC-CSE also collaborated with Jet Propulsion

Laboratory (JPL) in applying this technology to their Mission Data Systems (MDS) project.

USC-CSE also focused on the study of the techniques for specifying components' behavior. Component modeling was categorized in the following way:
1. structural (interfaces),
2. static behavior (pre- and post- conditions, invariants),
3. dynamic behavior (extended FSM notation that describes internal functionality of the component), and
4. interaction protocols.

4.3 Prism middleware

USC-CSE extended the Prism-MW middleware with several properties that were specifically intended to support development of highly-distributed, highly-mobile and resource constrained applications. These include efficiency, mobility, dynamic reconfigurability, awareness, and graceful degradation. The middleware was implemented in Java KVM, Java JVM, and Embedded Visual C++ and had been tested both on desktop (PC) and hand-held (Palm Pilot and Windows-CE compatible) platforms. USC-CSE enhanced this middleware with special-purpose, inter-device software connectors, including XML-based and infrared connectors. Several optional features for inter-device connectors were also implemented, such as data compression, security, and support for real-time message delivery. These connectors are equipped with gauges to measure their own throughput and load. These new versions of the middleware complemented the existing USC-CSE middleware versions implemented for the Windows and Unix platforms to allow architecture-based implementation, deployment, monitoring, and dynamic manipulation of applications in a distributed, mobile, and heterogeneous setting. Such a setting is commonly present in military applications, such as the USC-CSE prototype distributed troops deployment and battle simulation application. Both Lockheed Martin and the US Army TACOM group expressed a strong interest in this technology. Prism-MW was extensively and successfully evaluated by Lockheed Martin for possible use in their AWACS system.

- *Support for disconnected operation*

USC-CSE’s approach to the problem of disconnected operation proposed migrating components from neighboring hosts to a local host *before* the disconnection occurred. The set of components to be migrated is chosen such that it maximizes the autonomy of the local subsystem during disconnection, stays within the memory constraints posed by the device, and can be migrated within the time remaining before disconnection occurs.

USC-CSE implemented an extension of a *Border Connector* to support disconnected operation. The *Border Connector* utilizes a degraded mode indicator for each operation exported by a component. The indicator is intended to reflect an operation’s dependence on component state: some operations do not depend on component state and are fully accessible during disconnection (*allowed*); other operations are *delayed* until the connection is restored; finally, access to yet other operations is *disallowed*.

- *Deployment Support*

USC-CSE designed and implemented a prototype system deployment environment, *Prism-DE.* The environment integrates Microsoft's Visio tool as a graphical front-end for specifying the deployment configurations. A deployment configuration is specified as a set of target hardware hosts, a set of processes that will run on these hosts, and a software (architectural) configuration that needs to be deployed onto each one of the processes. The environment ensures the validity of specified configurations before the automated deployment is performed, and provides monitoring of connectivity between specified hardware hosts.

- *Delivery Guarantees*

USC-CSE's architectural middleware technology also provides support for connectors that handle messages with different delivery policies: at least once, at most once, best effort, and exactly once. By introducing priority-based scheduling of messages it was possible to provide support for handling messages with both soft and hard real-time delivery constraints. Different scheduling algorithms are used for periodic and aperiodic messages. Moreover, various gauges are used with USC-CSE connectors: they demultplex and dispatch incoming messages using request and notification filtering.

- *Security*

Secure communication in USC-CSE's architectural middleware technology was achieved by using composite connectors that encapsulated various security services. USC-CSE developed authentication, authorization, encryption, and message integrity modules that might be added to an arbitrary connector. These services were gauged using various cryptographic algorithms.

- *Distributed Computing*

USC-CSE developed a simulation of distributed environments to be hosted on Prism-MW. The environment provides simulation of distributed network comprising of a given number of hardware hosts, with varying properties (e.g., memory capacity of each host, network bandwidth and reliability of connectivity between hosts). Additionally, each host is capable of running a given set of software components, whose memory requirements and frequency of interaction can be varied.

4.4 Managing architectural evolution

USC-CSE developed a novel approach for managing architectural evolution. The existing DRADEL environment was integrated with UC Irvine's Ménage xADL 2.0 technologies. The result was an architectural evolution environment, called Mae, that enabled architects to specify, model, and analyze architectures of product families. Mae brings together architecture based software development and configuration management (CM). It also provides the architect with capabilities to manage the evolution of architectural artifacts. This is done by designating components, connectors, and their interfaces as optional or variant, and bringing versioning schemas to the level of architectural artifacts. Mae utilizes behavioral invariants and pre- and post-conditions as well as specification of a system's configuration to analyze the system's architectural model for possible inconsistencies and inadequacies. This reduces the development costs by detecting faults early in the development process.

4.5 Alfa

USC-CSE implemented a framework for the Alfa architectural assembly language using Java. The framework supports a dynamic architectural model and allows dynamic adaptation of software connectors corresponding to a variety of architecture styles. The framework implementation allows users to verify the suitability of such an assembly language, and construct an experimental model for the use of Alfa. Further, the models of the client-server, pipe-and-filter, C2, and push-based styles expressed in Alfa were implemented in Java using the Alfa framework. These implementations verified USC-CSE's models, and provided useful assessment of the approach in terms of framework properties.

USC-CSE also implemented an experimental database management system using these frameworks. The database was implemented using a combination of different kinds of connectors including object-oriented method calls, Alfa-level messages and calls, as well as Client-Server protocols. USC-CSE successfully completed adaptation of the system to different connectors using the Alfa framework and the implementation of the Alfa-Client-Server framework.

The Alfa framework was also applied to network-based architectural styles. The basis of this research was the observation that architectural styles shared many underlying concepts. These shared concepts lead to "architectural primitives" that can be systematically and constructively composed to obtain elements of architectural styles. Total of eight forms and nine functions were identified as architectural primitives since they reflected the syntactic and semantic characteristics of a large number of styles. While proving such a hypothesis was difficult in the general case, USC-CSE demonstrated it within the domain of network-based styles. Partial formal models of style elements composed from these architectural primitives were also constructed using Alloy and shown to be analyzable.

USC-CSE also created an extensible notation, xAlfa, for precisely composing architectural styles in the Alfa framework. This notation was also used to construct architectures using architectural styles, as well as implement them using programming languages and style-compliant middleware. This notation was integrated with the xADL architecture description language in order to allow rapid development of associated tools.

4.6 Assessing the Quality of Product Line Architectures

USC-CSE developed several novel metrics to assess the structural quality of product line architectures. Throughout the evolution of the product lines, these metrics would guide the architect in making more informed architectural decisions. The metrics are based on the concept of service utilization and are designed to take into account the context in which individual architectural elements are placed [11].

USC-CSE evaluated these metrics in the context of different case studies. The first case study was a two-semester project course for computer science graduate students at the University of Southern California (USC). Assigned teams developed a variety of digital library applications for a real client, the Library Information Services Division (ISD) at USC. About sixty such applications were developed, comprising several distinct product lines [16]. Compared to the existing evaluations of the involved architectures by the project customers and course instructors, USC-CSE's service utilization metrics' values of this case study indicated that low values were a sign of low quality. This was also confirmed with the project-based rankings assigned to these projects. Additionally, alternative architectural solutions were tested based on the values generated by the proposed metrics. For some of these solutions, the utilization values of certain components were increased, hence increasing the quality of the product line overall.

In addition to the structural quality, USC-CSE also investigated other relevant measurable quality attributes. Architectural vulnerability was selected as a potential quality attribute. Architectural vulnerability assessment is the architectural examination of a system to identify the critical architectural elements (machines, components, methods, etc.) that may be at risk (a particular threat that will exploit a weakness in the architectural elements based on the resource access-levels these elements are granted) from an attack. As the software systems are becoming more decentralized and distributed, assessing quality attributes relevant to the system security is even more meaningful. Furthermore, the basic principles of USC-CSE's approach relied on two concepts: permissions granted to the service elements (e.g. public interface methods), and the deployment of these service elements (e.g. which components reside on which machines).

5 Technology Transitions

5.1 Collaboration with other DARPA DASADA contractors

- USC-CSE collaborated with University of California at Irvine (UCI), Carnegie Mellon University (CMU), and Teknowledge on formalizing and assessing the strengths and shortcomings of the Unified Modeling Language (UML) in supporting architecture-based software development. This would enable software developers, in industry and academia, to specify, architect, and design software systems more accurately, and detect problems earlier during the development process. All this would result in the reduction of the development cost for a software system.
- USC-CSE also collaborated with UCI, the University of Colorado at Boulder, and the University of Oregon, the goal of which was to assess the ability of combined capabilities to address problems faced by Lockheed Martin's AWACS system.
- Finally, the development of the Mae system for architectural evolution discussed above was a major collaboration effort between UCI and USC-CSE primarily, with CMU's contribution housed in the development of the underlying xADL infrastructure used to integrate USC's DRADEL and UCI's Ménage tools.

5.2 Technology Transition to other non DARPA DASADA efforts

- In late 2000 USC-CSE completed the integration of the UML/Analyzer tool with Rational Rose, a commercial UML modeling tool, for the purpose of using them to create and modify modeling diagrams. Rational Rose models are converted through an automated process into a system model called UML-A where they are analyzed via UML/Analyzer. Transformed modeling information as well as identified model inconsistencies can be fed back into Rational Rose for visualization. The concepts behind UML/Analyzer were developed in collaboration with the Rational Software Corporation. Additionally, Rational implemented a version of USC-CSE's UML/Analyzer tool under the name Rose/Architect. For more information on UML/Analyzer please visit http://www.if.afrl.af.mil/tech/programs/dasada/tools/umlanalyzer.html.
- Lockheed Martin evaluated the Prism-MW in the context of their AWACS project. This evaluation indicated that Prism-MW was efficient and suitable for use in the AWACS project.
- USC-CSE's architectural model-to-StateChart consistency rules were evaluated by the Jet Propulsion Laboratory (JPL) as a possible aid for their envisioned architectural testing framework in the MDS project. Furthermore, the Mae environment for managing architectural evolution was evaluated by JPL's MDS group. The collaboration with JPL resulted in a prototype of integrated architectural analysis and testing gauges.
- Our collaboration with UC Davis focused on their expertise in software security applied to USC-CSE's architectural connectors.

- USC-CSE established a collaborative relationship with US Army TACOM, who acted as early evaluators of USC-CSE's Prism technology for use in their ground vehicle systems.
- Finally, USC-CSE started collaborating with Boeing Anaheim, who have expressed particular interest for USC's DASADA technology for use in their on-going Future Combat Systems (FCS) project.

6 Homepages

Center for Software Engineering, University of Southern California

- http://sunset.usc.edu

DASADA project homepage

- http://sunset.usc.edu/research/DASADA

Software Architecture Research Group

- http://sunset.usc.edu/~softarch

7 Publications

1. Nikunj R. Mehta, Nenad Medvidovic, and Sandeep Phadke. "Towards a Taxonomy of Software Connectors." In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 178-187, Limerick, Ireland, June 4-11, 2000.
2. Nenad Medvidovic, Rose F. Gamble, and David S. Rosenblum. "Towards Software Multioperability: Bridging Heterogeneous Software Interoperability Platforms." In *Proceedings of the Fourth International Software Architecture Workshop (ISAW-4)*, pages 77-83, Limerick, Ireland, June 4-5, 2000.
3. Paul Gruenbacher, Alexander Egyed, and Nenad Medvidovic. "Dimensions of Concerns in Requirements Negotiation and Architecture Modeling." In *Proceedings of the Workshop on Multi-Dimensional Separation of Concerns in Software Engineering*, Limerick, Ireland, June 6, 2000.
4. Alexander Egyed, Nenad Medvidovic, and Cristina Gacek. "A Component-Based Perspective on Software Mismatch Detection and Resolution." *IEE Proceedings – Software Engineering*, vol. 147, no. 6, pages 225-236 (December 2000).
5. Rohit Khare, Michael Guntersdorfer, Peyman Oreizy, Nenad Medvidovic, Richard N. Taylor. "xADL: Enabling Architecture-Centric Tool Integration With XML." In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, January 3-6, 2001.
6. Nicolas Rouquette, Nenad Medvidovic, and David Garlan. "Dependable Autonomous Systems = knowing well what to do + knowing how to do it well." In *Proceedings of the NASA High Dependability Computing Consortium Workshop*, NASA AMES, Moffet Field, CA, January 10-12, 2001.

7. Nenad Medvidovic and Marija Rakic. “Exploiting Software Architecture Implementation Infrastructure in Facilitating Component Mobility.” In *Proceedings of the Workshop on Software Engineering and Mobility*, Toronto, Canada, May 13-14, 2001.
8. Alexander Egyed, Paul Gruenbacher, and Nenad Medvidovic. “Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach.” In *Proceedings of the From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, May 14, 2001.
9. Marija Rakic and Nenad Medvidovic. “Run-time Support for Architecture-Level Configuration Management.” In *Proceedings of the Tenth International Workshop on Software Configuration Management (SCM-10)*, Toronto, Canada, May 14-15, 2001.
10. Alexander Egyed and Nenad Medvidovic. “Consistent Architectural Refinement and Evolution Using the Unified Modeling Language.” In *Proceedings of the Workshop on Describing Software Architecture with UML*, Toronto, Canada, May 15, 2001.
11. Marija Rakic and Nenad Medvidovic. “Increasing the Confidence in Off-the-Shelf Components: A Software Connector-Based Approach.” In *Proceedings of the 2001 Symposium on Software Reusability (SSR 2001)*, pages 11-18, Toronto, Canada, May 17-19, 2001.
12. Nenad Medvidovic, Paul Gruenbacher, Alexander Egyed, and Barry W. Boehm. “Software Model Connectors: Bridging Models across the Software Lifecycle.” In *Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering (SEKE 2001)*, pages 387-396, Buenos Aires, Argentina, June 13-15, 2001.
13. Paul Gruenbacher, Alexander Egyed, and Nenad Medvidovic. “Reconciling Software Requirements and Architectures: The CBSP Approach.” In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE’01)*, Toronto, Canada, August 27-31, 2001.
14. Lei Ding and Nenad Medvidovic. “Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution.” In *Proceedings of the 2001 Working IEEE/IFIP Conference on Software Architectures (WICSA 2001),* Amsterdam, the Netherlands, August 27-29, 2001.
15. Andre van der Hoek, Marija Mikic-Rakic, Roshanak Roshandel, and Nenad Medvidovic. “Taming Architectural Evolution.” In *Proceedings on the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2001)*, Vienna, Austria, September 10-14, 2001.
16. Ebru Dincel, Nenad Medvidovic, and Andre van der Hoek. Measuring Product Line Architectures. In *Proceedings of the 4th International Workshop on Product Family Engineering (PFE-4)*, Bilbao, Spain, October 3-5, 2001.
17. Nenad Medvidovic and Marija Mikic-Rakic Programming-in-the-Many: A Software Engineering Paradigm for the 21st Century. Workshop on New Visions

for Software Design and Productivity: Research and Applications, Nashville, Tennessee, December 2001.

18. Roshanak Roshandel and Nenad Medvidovic. Coupling Static and Dynamic Semantics in an Architecture Description Language. Working Conference on Complex and Dynamic System's Architecture (CDSA 2001), Brisbane, Australia, December 2001.
19. Nenad Medvidovic, David S. Rosenblum, Jason E. Robbins, and David F. Redmiles. Modeling Software Architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, January 2002.
20. Roshanak Roshandel and Nenad Medvidovic, Static and Dynamic Modeling of Architecture, in GSAW Sixth Ground System Architectures Workshop, The Software Aerospace Corporation El Segundo, CA, February, 2002.
21. Marija Mikic-Rakic and Nenad Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments. In Proceedings of CD 2002, Berlin, Germany, June 2002.
22. Nenad Medvidovic. On the Role of Middleware in Architecture-Based Software Development. In *Proceedings of the The Fourteenth International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, July 15-19, 2002.
23. Nenad Medvidovic, Nikunj R. Mehta and Marija Mikic-Rakic. A Family of Software Architecture Implementation Frameworks. *In Proceedings of the 3rd IFIP Working International Conference on Software Architectures*, Montreal, Canada, August 2002.
24. Ebru Dincel, Nenad Medvidovic, Andre van der Hoek. An example product line architecture: Digital Library Projects, USC Center for Software Engineering Technical Report, USC-CSE-2002-505.
25. Ebru Dincel, Nenad Medvidovic, Andre van der Hoek. An example product line architecture: Troops Deployment System, USC Center for Software Engineering Technical Report, USC-CSE-2002-506.
26. Ebru Dincel, Nenad Medvidovic, Andre van der Hoek. An example product line architecture: The Library System, USC Center for Software Engineering Technical Report, USC-CSE-2002-507.
27. Marija Mikic-Rakic and Nenad Medvidovic. Middleware for Software Architecture-Based Development in Distributed, Mobile, and Resource-Constrained Environments. TR USC-CSE-2002-508.
28. M. Mikic-Rakic, N. Mehta, and N. Medvidovic. Architectural Style Requirements for Self- Healing Systems. *Proceedings of the 1st International Workshop on Self-Healing Systems (WOSS'02)*, Charleston, SC, November 2002.
29. N. R. Mehta and N. Medvidovic. Understanding Software Connector Compatibilities Using a Connector Taxonomy. Appeared in *Proceedings of First Workshop on Software Design and Architecture (SoDA'02)*, December 2002, Bangalore, India

30. Nenad Medvidovic and Vladimir Jakobac. “A Focused Approach to Software Architectural Recovery,” *Ground Systems Architectures Workshops GSAW 2003,* Manhattan Beach, CA, March 2003.
31. Marija Mikic-Rakic and Nenad Medvidovic. A Connector-Aware Middleware for Distributed Deployment and Mobility. *Proceedings of ICDCS Workshop on Mobile Computing Middleware (MCM'03)*, Providence, Rhode Island, May 2003.
32. Marija Mikic-Rakic and Nenad Medvidovic. Towards a Framework for Classifying Disconnected Operation Techniques. *Proceedings of ICSE Workshop on Software Architectures for Dependable Systems*, Portland, Oregon, May 2003.
33. Marija Mikic-Rakic and Nenad Medvidovic. Adaptable Architectural Middleware for Programming-in-the-Small-and-Many. *Proceedings of ACM/IFIP/USENIX International Middleware Conference*, Brazil, June 2003.
34. Nenad Medvidovic, Marija Mikic-Rakic, Nikunj Mehta, and Sam Malek. Software Architectural Support for Handheld Computing. Cover feature in *IEEE Computer*, September 2003.
35. Nenad Medvidovic, Marija Mikic-Rakic, and Nikunj Mehta. Improving Dependability of Component-Based Systems via Multi-Versioning Connectors. *Architecting Dependable Systems*, Springer-Verlag. Lecture Notes in Computer Science (LCNS 2677). R. de Lemos, C. Gacek, and A. Romanovsky (Eds.), 2003.
36. Nenad Medvidovic, Marija Mikic-Rakic and Sam Malek. Software Architectures for Embedded Systems. In Proceedings of Monterey Workshop Series, Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation. Chicago, Illinois, September 2003.
37. Nikunj Mehta and Nenad Medvidovic. Composing architectural styles from architectural primitives. In *Proceedings of ESEC FSE 2003*, Finland, Helsinki, September 2003.
38. Nikunj Mehta. Composing network-based architectural styles from architectural primitives. In *Proceedings of ESEC FSE 2003 Doctoral Symposium*, Finland, Helsinki, September 2003.
39. Roshanak Roshandel and Nenad Medvidovic, Modeling Multiple Aspects of Software Components, in Proceeding of workshop on Specification and Verification of Component-Based System, ESEC-FSE03, Helsinki, Finland, September 2003.
40. Roshanak Roshandel, Mae: An Architectural Evolution Environment, Research Demonstration, in Proceeding of workshop on Specification and Verification of Component-Based System, ESEC-FSE03, Helsinki, Finland, September 2003.