

Exploiting Connector Knowledge To Efficiently Disseminate Highly Voluminous Data Sets

Chris A. Mattmann^{1,2}

David Woollard^{1,2}

Nenad Medvidovic²

¹Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109, USA
mattmann@jpl.nasa.gov

²Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA
{veno,woollard}@usc.edu

ABSTRACT

Ever-growing amounts of data that must be distributed from data providers to consumers across the world necessitate a greater understanding of the software architectural implications of choosing data movement technologies. Currently, this understanding is mired in the minds of software architects who have “been there before,” and who rely on past intuition and choices, failing to properly document their rationale and context. In this paper we describe a software architecture-based decision making framework called DISCO for selecting data movement technologies, or software connectors. DISCO effectively captures (traditionally undocumented) insight, observation and ultimately architectural knowledge about the connectors, demonstrating the effectiveness of using such information to accurately encode the connector selection decision making process.

1. INTRODUCTION

Large virtual communities of scientists are cropping up around the world enabled by the increasing power of modern hardware, networking infrastructure, and software, and their combined ability to collect, process, and share *data*. Scientists rely on data to record observations about a scientific event, e.g., a collection of lung cancer patient bronchoscopy images identifying locations of cancerous lesions, or spectral band trace results measuring carbon dioxide in the atmosphere as seen from space. Growing requirements on scientific accuracy, as well as technological advances in measurement, have ushered in an era in which the amount of data collected during a scientific event is orders of magnitude more voluminous than the same events just ten years past. Data collected today is approaching the *terabyte to petabyte* range in many scientific disciplines (high-energy physics [1], for example) and is poised to grow even more voluminous in the next ten years.

Data dissemination is achieved at the software level through *data movement technologies* that rely upon many layers of underlying software and hardware, including the operating system (software), the networking protocol (software/hardware), networking infrastructure (hardware) and dissemination strategies (software).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SHARK '08, May 10–18, 2008, Leipzig, Germany.
Copyright 2008 ACM 1-58113-000-0/00/0004...\$5.00.

A data movement technology is an advanced and first-class element in data dissemination systems and is often thought of at the software architectural level as a *software connector* [2]. Connectors embody the interactions between software components, including behaviors, data transfer (and representations) and any communication protocols employed.

A significant problem in today’s data dissemination systems¹ is the selection of the appropriate software connector during the design phase of system development. There are many connectors that claim to handle varying levels of service for large-scale data movement, e.g., connectors such as bbFTP, GridFTP, Bittorrent, FTP, SCP, JXTA, RMI, XML-RPC, and so on [3]. Though many of these technologies purport to offer similar capabilities, the danger in simply selecting one of them and integrating it into a dissemination system is that the implications of this decision are not well understood. In our recent work [4] we have shown that connectors integrated in dissemination systems required to support highly heterogeneous *distribution scenarios* (variations in the dimensions of a data distribution, e.g., its *total volume*, its *number of users*, the *types of data* that is distributed, and so on) have critical implications on its performance (efficiency, scalability, consistency, dependability). In other words, *it does matter* what connector is selected and clearly not at all are equally applicable in every dissemination system, let alone in every distribution scenario.

Unfortunately the state of the art in selecting software connectors for data distribution is less of a principled process and more of an art form. An expert in an organization often chooses so-called “pet technologies” — connectors that have proven successful in the past or simply those with which the expert is familiar — regardless of the current distribution scenario requirements or dissemination system architecture. Clearly, this is an error-prone process in which the decisions employed by the expert architect are not properly codified in any fashion. These decisions, if properly codified, could be examined later and tailored as necessary to improve desired properties (e.g., performance, scalability, etc.).

To address the aforementioned lack of an audit trail for the data-intensive connector selection process, we have developed a software decision-making framework called DISCO (Data-Intensive Software Connectors). DISCO provides XML-based facilities for encoding two distinct types of architectural information that, to our knowledge, are currently unrecorded. First, we provide an explicit connector model, borne from select

¹ Also referred to as dissemination systems, data distribution systems, and distribution systems throughout the paper

properties present in a widely cited and accepted connector taxonomy [2] that capture the essential architectural variation points for a connector technology. Second, we provide an explicit model of a distribution scenario, allowing an architect to express tangibly the required scenarios that any selected connector must support. We then leverage these two models to provide *insight*, mapping properties from our connector model to discrete facts that an expert architect would have stored in her head (e.g., “if the total volume of the transfer is greater than 100 megabytes, then a connector with *cache*-based data access is better than another with *process*-based access”). Finally, we provide *observations*, codifying the relationship between external connector characteristics (e.g., its performance) and the connector’s utility in a data distribution scenario. We assert that the marriage of our connector and scenario models coupled with observations and insight about them is in fact what comprises *architectural knowledge*. Using architectural knowledge, DISCO is highly accurate and reliable in selecting connectors that both satisfy data distribution scenarios and that fit the needs of the data dissemination system architecture.

The rest of this paper is organized as follows. Section 2 points the reader to related work in the areas of capturing and reusing architectural knowledge, with an explicit focus on software connectors. Section 3 describes in detail the types of architectural knowledge captured in DISCO, using an illustrative example. Section 4 describes the results of recoding connector architectural knowledge in DISCO. Conclusions are presented in Section 5.

2. RELATED WORK

Our work on DISCO is similar to work in the areas of software connector models, architectural design decisions and knowledge management, and software architectural ontologies. In this section, we will discuss some representative projects in each of these five areas.

There are several proposed models that represent *software connectors* [5-8], the architectural element that embodies the interactions between *software components* [9]. In DISCO, we have chosen to leverage a widely accepted connector taxonomy model [2] that summarizes many of the important connector properties described by the other proposed models [5-8]. The connector taxonomy identifies eight primitive types of software connectors and their salient characteristics (e.g., *data access locality*, *stream bounds*, *event synchronicity*, etc.). While a full treatment of the taxonomy is beyond the scope of this paper, we note that DISCO employs six of the primitive connector types described in the taxonomy, namely: an invocation connector (either *procedure calls*, *arbitration*, or *events*), a *data access* connector, *stream* connectors and *distributor* connectors. DISCO’s model of software connectors leverages these primitive connectors to suggest that a data dissemination connector is an aggregate of four of the six primitive connector types, as shown in Figure 1.

Capilla et al. describe a web-based tool (called ADDSS for architecture design decision support system) that manages architectural design decisions. [10]. ADDSS grew out of the realization that architectural design decisions are frequently tied un-captured, and even when captured, are not explicitly tied to rationale, or the original system requirements. Capilla et al.’s tool explicitly handles the author’s proposed “decision” view that extends Krutchen’s “4+1” architectural model [11] and provides explicit means for tying rationale to system requirements, and

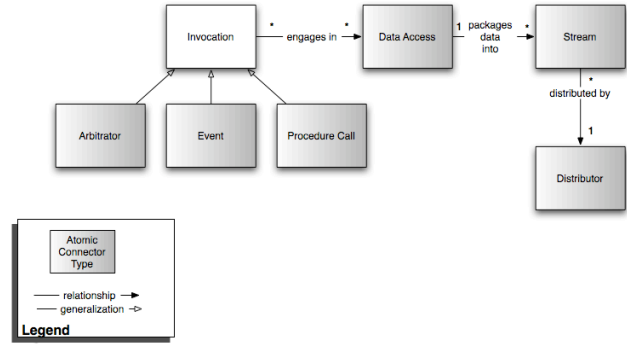


Figure 1. DISCO's model of software connectors for data distribution

ultimately to the system design context. The ADDSS tool stores architectural design decisions as free text, compared to DISCO where we leverage structured, design decisions and capture them in the form of XML files. DISCO can provide structured design decisions because its domain is constrained to data-intensive connectors, as opposed to focusing on entire architectures, variation points, and styles as Capilla et al. do. In our experience, restricting the focus of design decisions to a particular element in an architecture (e.g., connectors) helps to compactly and accurately record meaningful design decisions that can be fed into a decision making process.

Ali Babar et al. [12] survey the state of the research literature and practice in architectural knowledge management. The authors note that current approaches to architectural knowledge management are of two forms: *codification*, where architectural knowledge is captured and stored in a structured knowledge repository and made available for reuse and analysis at a later date, and *personalization*, where architectural knowledge is left un-codified (and in minds of the experts) and Yellow Page-type directories are constructed identifying architects with expertise in specific areas of architectural knowledge. According to the authors, while codification is widely employed in literature, the research community would benefit from leveraging personalization, an approach often used in practice. Personalization is inherently more “light-weight”, and does not require the use of formalisms (as codification typically does).

In our work within the context of DISCO, we have found that codification has its benefits, however. Based on a recent survey of data distribution experts [3], personalization approaches fall prey to the problem of organizational change – when domain experts leave an organization, they take their knowledge with them. In addition, because codification approaches (like DISCO) produce knowledge artifacts, such artifacts can be used as a shared means of recording key design decisions similar to the work of Capilla et al. [10].

Lenin et al. [13] describe the need for an ontology of software architecture, and a manual and (semi-)automated process for constructing such an ontology using widely available sources of architectural information. The authors’ manual process consists of examining the end matter for several authoritative software architecture books, and determining a set of *key terms* that are then further reduced (e.g., removal of synonyms, etc.), and finally categorized and grouped hierarchically by a domain expert. The authors’ (semi-)automated process consists of developing a tool called HyperOnto that queries Wikipedia’s software architecture

page for outgoing links, and then requires a domain expert to manually categorize the topics of those outgoing links that are relevant to her software domain. The selected terms are then classified into nine categories, forming an ontology of architectural categories and relevant terms. DISCO’s architectural knowledge provides connector information from five of the nine categories identified by Lenin et al., including *non-functional (quality) requirements, scenarios, patterns, styles and architectural frameworks*.

3. CONNECTOR KNOWLEDGE

DISCO codifies two different types of knowledge artifacts: *distribution scenarios* and *connector profiles*. Distribution scenarios capture the features of data distribution salient to the selection process, such as type of network used (*GeographicDistribution*), expected number of delivery intervals (*DeliverySchedule.NumIntervals*), and relative volumes of data (*TotalVolume*) to be delivered. An example of a scenario is shown in the top portion of Table 1. DISCO relates distribution scenarios to connector profiles that describe the intrinsic attributes of each connector. As an example, partial connector profiles showing attribute values for four key properties (*procedure call data transfer method, data access transient availability, stream formats and distributor delivery mechanisms*) of two postulated connectors c_1 and c_2 are given in Table 1. DISCO allows architects to leverage distribution scenarios and connector profiles to record their *insights* about a connector’s ability to satisfy a distribution scenario. In addition, DISCO allows an architect to record *observations* about the performance of particular connectors in a given distribution scenario. We have currently developed three connector selection algorithms (the description of which is beyond the scope of this paper – see [3] for a complete description) that leverage our connector knowledge to pair connectors to distribution scenarios: Bayesian, Scored-based, and Optimizing [4]. Below we discuss how observation and insight are captured and used in greater detail.

3.1 Observation

In DISCO, architects can form observations about connector technologies by making pair-wise comparisons between distribution scenario constraints and desired performance properties (e.g., the example in Table 1 shows that connector c_1 can send 1 terabyte of data more *efficiently* than connector c_2 because the architect has recorded normalized efficiency scores of 90 and 15, respectively) called *score functions*. Observations about architectural knowledge can be captured in DISCO using normalized scales and linear programming models recorded by architects with experience evaluating the performance of a set of connectors (as illustrated above). Additionally, observations can be recorded using conditional probabilities to make similar determinations. In this case, rather than assigning a numerical score, the architect would assign her “belief” that a connector c is appropriate for a particular scenario (e.g., a requirement of high efficiency and low dependability). This belief is captured in the form of a table of conditional probabilities, as will be explained in more detail below.

3.2 Insight

In addition to observation, DISCO captures a second distinct source of information about distribution technologies that assists in making connector selections. Using conditional probabilities, an architect can codify her *architectural insight* regarding the

Distribution Scenario								
Dimension		Constraint						
<i>Total Volume</i>		> 1 TB						
<i>Geographic Distribution</i>		WAN						
<i>DeliverySchedule.NumIntervals</i>		1						
<i>NumUsers</i>		1						
Connector Profiles								
	<i>Procedure Call</i>	<i>Data Access</i>	<i>Stream</i>	<i>Distributor</i>				
	Data transfer method	Transient Availability	Formats	Delivery Mechanisms				
c_1	Value	Cache	Raw	Broadcast, Unicast				
c_2	Reference	Session	Structured	Unicast, Multicast				
Insight								
Data access transient availability	Total Volume > 1 TB	Geographic Distribution = WAN	Delivery Schedule Num Intervals = 1	Num Users = 1				
<i>Cache</i>	0.80	0.80	0.60	0.20				
<i>Peer</i>	0.30	0.95	0.95	0.80				
<i>Session</i>	0.10	0.10	0.10	0.80				
Observation								
	<i>e</i>		<i>s</i>		<i>c</i>		<i>d</i>	
	c_1	c_2	c_1	c_2	c_1	c_2	c_1	c_2
Total Volume								
100 GB	20	70	30	100	70	70	95	95
1 TB	15	90	20	120	70	70	80	80
Num Users								
1	50	80	40	90	70	70	95	95
10	50	80	40	90	70	70	9	95

Table 1. Example knowledge captured by DISCO. e = efficiency, s = scalability, c = consistency, d = dependability.

applicability of a connector to a given data distribution scenario. Insight can capture information such as architectural mismatch [14] and intrinsic connector properties. Each insight is recorded as an explicit probabilistic statement (a conditional probability) relating a particular connector technology to a distribution scenario constraint (e.g., *TotalVolume* > 1 TB). Each connector technology may have many statements recorded by an architect. As an example, consider the statement from the Insight section of Table 1: connector c_2 is well suited for single user, point-to-point data distributions because of its *Session*-based data access transient availability. An architect may record this statement by using the probability 0.80 for c_2 — the highest value recorded for the possible values of the data access transient availability connector property.

As previously stated, it is our contention that both observation and architectural insights must be recorded in order to truly capture architectural *knowledge* of connectors. In the next section, we will explore this contention via experimentation.

4. THE ROLE OF INSIGHT

In order to test the role of insight and observation in DISCO, we decided to compare the results of DISCO's connector selection against an expert answer key identifying the "appropriate" and "inappropriate" connectors for 30 different distribution scenarios. These scenarios represent a cross-section of data dissemination systems in the realms of earth science, planetary science and cancer research at NASA's Jet Propulsion Laboratory. To examine the role of architectural knowledge in connector selection, we varied the amount of insight and observations used in each selection.

For inputs to the Bayesian and Score-based algorithms, we constructed low, medium and high knowledge bases. The low knowledge bases consisted of 10 randomly chosen conditional probabilities and 8 score functions used as input to the Bayesian and Score-based selection algorithms, respectively. The medium quality domain knowledge bases provided 50 randomly chosen conditional probabilities and 16 score functions. Finally, high quality knowledge bases included all as-developed domain knowledge that we have constructed so far, including 100 conditional probabilities and 24 score functions.²

Comparing the precision/recall of the Score-based algorithm against the Bayesian algorithm illustrates the role of insight versus observation. As shown in Figure 2, for low-medium quality knowledge profiles, the Bayesian and Score-based selection algorithms are relatively close in precision rate (respectively, 72.3-73.3% compared to 61.2%-62.3); however, the Bayesian algorithm with a high quality knowledge profile is the more precise algorithm, with 80.6% precision. More interestingly, while the recall abilities of the Score-based approach remain relatively constant, the recall on the Bayesian algorithm is helped significantly by more complete knowledge.

Because conditional probabilities (the input to our Bayesian algorithm) capture both observations and insight while Score-based selection considers only observation, this experiment shows that observation alone is insufficient when codifying the software architect's knowledge of distribution connectors.

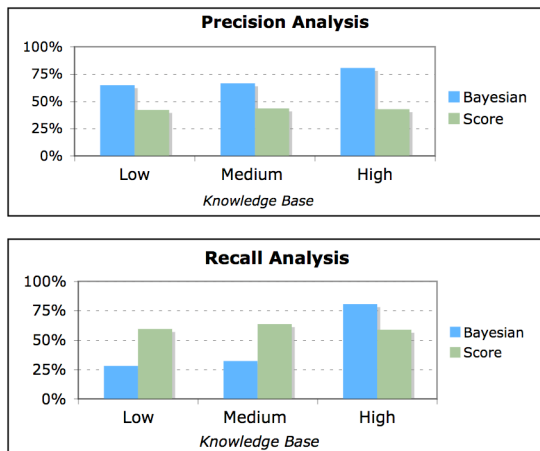


Figure 2. Precision and recall analysis for Bayesian and Score-based selection given varying knowledge bases.

5. CONCLUSIONS

In this work, we have shown the positive impact of both architectural insight into the makeup and application of connector technologies as well as observations of a connector's performance in data distribution scenarios. Codifying the knowledge of the software architect in the form of connector profiles, DISCO assists software architects in recording key design decisions that went into the choice of a particular data distribution technology.

In addition to continuing to develop connector profiles as new data distribution connectors are identified, our future work will be to provide means for evaluating the quality of knowledge cataloged in our connector knowledge repository.

6. REFERENCES

- [1] B. Alcock, J. Bester, et al., "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, pp. 749-771, 2002.
- [2] N. Mehta, N. Medvidovic, et al., "Towards a Taxonomy of Software Connectors," In Proc. *International Conference on Software Engineering (ICSE)*, Limerick, Ireland, 2000.
- [3] C. Mattmann, "Software Connectors for Highly Voluminous and Distributed Data-Intensive Systems", Ph.D. Dissertation, USC, 2007.
- [4] C. Mattmann, D. Woollard, et al., "Software Connector Classification and Selection for Data-intensive Systems," In Proc. *ICSE 2007 Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS)*, Minneapolis, MN, 2007.
- [5] R. J. Allen and D. Garlan, "A Formal Basis for Architectural Connection," *ACM Transactions on Software Engineering and Methodology*, 1997.
- [6] F. Arbab, "A Channel-based Coordination Model for Component Composition," CWI Technical Report SEN-R0203, 2002.
- [7] A. Lopes, M. Wermelinger, et al., "Higher-order architectural connectors," *ACM Transactions on Software Engineering and Methodology*, vol. 12, pp. 64-104, 2003.
- [8] M. Shaw, "Procedure Calls are the Assembly Language of Software Interconnections: Connectors Deserve First-Class Status," In Proc. *Workshop on Studies of Software Design*, 1993.
- [9] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, vol. 26, pp. 70-93, 2000.
- [10] R. Capilla, F. Nava, et al., "A Web-based Tool for Managing Architectural Design Decisions," In Proc. *SHARK 06'*, Torino, Italy, 2006.
- [11] P. Krutchen, "Architectural Blueprints — The "4+1" View Model of Software Architecture," *IEEE Software*, vol. 12, pp. 42-50, 1995.
- [12] M. A. Babar, R. C. d. Boer, et al., "Architectural Knowledge Management Strategies: Approaches in Research and Industry," In Proc. *SHARK-ADI'07*, Minneapolis, MN, 2007.
- [13] B. T. Lenin, R. M. Seetha, et al., "ArchVoc -- Towards an Ontology for Software Architecture," In Proc. *SHARK-ADI'07*, Minneapolis, MN, 2007.
- [14] D. Garlan, R. Allen, et al., "Architectural Mismatch or Why it's hard to build systems out of existing parts," In Proc. *ICSE*, 1995.

² Available at <http://sunset.usc.edu/~mattmann/disco/evaldata/>