# Software Connector Classification and Selection for Data-Intensive Systems

Chris A. Mattmann[1,2], David Woollard[1,2], Nenad Medvidovic[2], Reza Mahjourian[3]

[1] Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109, USA
{mattmann,woollard}@jpl.nasa.gov

[2]Computer Science Department
Univ. of Southern California
Los Angeles, CA 90089, USA
neno@usc.edu

[3]CISE Department
Univ. of Florida
Gainesville, FL 32611, USA
rezam@ufl.edu

## Abstract

*Data-intensive systems and applications transfer large volumes of data and metadata to highly distributed users separated by geographic distance and organizational boundaries. An influential element in these large volume data transfers is the selection of the appropriate software connector that satisfies user constraints on the required data distribution scenarios. Currently, this task is typically accomplished by consulting "gurus", who rely on their intuitions, at best backed by anecdotal evidence. In this paper we present a systematic approach for selecting software connectors based on eight key dimensions of data distribution that we use to represent the data distribution scenarios. Our approach, dubbed DISCO, has been implemented as a Java-based framework. The early experience with DISCO indicates good accuracy and scalability.*

## 1 Introduction

Data volumes in modern software systems are growing by orders of magnitude. Companies such as Google are indexing 10s of *billions* of records; electronic DVD distribution software is providing 1000s of movies which range in size from traditional DVD containing gigabytes of data to higher resolution (dual-sided) DVDs to users across the internet; scientific data systems are collecting large amounts (*terabytes*) of data from high resolution scientific instruments and distributing that data to users around the world [6, 9].

Though the management of large amounts of data has been dealt with practically for some time (e.g., credit card companies such as VISA routinely handle hundreds of billions of transactions every day), and has already garnered a lot of attention from the database research community, one important aspect of these data-intensive systems is that they are not *closed-loop* systems. In the past, because the data systems were closed loop, there were never stringent requirements on the distribution of data from them.[1] Today, however, with the increase in network bandwidth, and with the lower costs of computer hardware (particularly storage hardware), it is regularly expected in such environments that if the system collects data, it must also distribute it.

The preceding statement begets the question: *how do we distribute such voluminous data sets in a manner that is performant, and that adheres to a user's requirements as well as the overall data system architecture*? There are a number of available off-the-shelf (OTS) technologies that claim to handle large scale data movement, including commercial (Bittorrent, SOAP/Web Services) and open source (UFTP, GridFTP, bbFTP) technologies.

If we ask ourselves "which technology is the most appropriate?" we must qualify the *scenario*s in which a data movement technology is appropriate. As an example, we must understand if a data movement technology is appropriate in *larger volume* scenarios, across the *WAN*; in *medium volume* scenarios, adhering to many *users*, *user types*, and *data types*; in *larger volume* scenarios, but over a single interval, across the *LAN*; and so on. These types of questions imply the need for a *language* for describing data distribution scenarios. Additionally, there must be a way of understanding how the available OTS data movement technologies map to the distribution scenarios described by the designers of data distribution systems.

The study of software architecture [13, 15] can aid us in handling the latter problem. A *software architecture* consists of *components* (the units of computation in a software system), *connectors* (facilities that model the interactions between software components), and *configurations* (assemblages of components and connectors, and the rules that guide their composition) [11]. Software connectors provide us with appropriate modeling facilities for large scale data

[1]For exposition purposes, we use the terms *data distribution*, *data movement* and *data dissemination* interchangeably throughout the paper

distribution technologies (which we refer to as "data distribution connectors", or "distribution connectors" for short).

The consequences for improperly selecting a connector are substantial. Imagine a situation in which a software architect assembling the architecture of a movie distribution service decides it appropriate to use HTTP/REST (a client/server connector) as the primary distribution connector to the outside world. This bodes well for customers who periodically log onto the movie distribution site and download one or two movies. However, the primary advertised benefit of the movie distribution service is a feature that affords a user the ability to *subscribe* to genres of movies she is interested in, and have the movie delivered to her desktop incrementally whether she is physically present or not. These type of requirements imply the need for an event-based or peer-to-peer distribution connector (as oppossed to client/server), supporting asynchronous stream delivery and aperiodic distribution. In order to augment the HTTP/REST connector with these types of capabilities, a *substantial* engineering effort would be required.

However, in most organizations, the architect for the movie distribution service would not have made such a subtle error when selecting the appropriate software connector. This is due to the fact that most organizations that distribute large amounts of data have one or two data distribution"gurus", individuals who have built many types of these distribution systems, and know what the right connector to select is, from experience. Such gurus would have easily spotted the fact that HTTP/REST bears no native support for aperiodic, subscription-based data delivery (or would they have?). Unfortunately, there are two major problems with these gurus, namely: (1) they are few and far between, and when they leave an organization, so does their knowledge; and (2) they cannot explain in detail *why* or *how* they made the connector selection, choosing instead to point out past experience and similar systems built.

In our recent work we have leveraged one of the widely accepted models of software connectors, proposed by Mehta et al. [12] to *classify* distribution connectors using standard metadata. Using these classifications, we have developed complementary selection algorithms that map the connectors (and their metadata) to data distribution scenarios, and that map system performance requirements to both the connectors and scenario dimensions. Our framework, dubbed DISCO (for *D*ata-*I*ntensive *S*oftware *CO*nnectors), has been implemented as a Java-based API and tool suite, and has proved valuable in providing a means for formally understanding the relationship between different data distribution scenarios and the connector technologies that exist. We view this to be an important initial step in alleviating the need to rely on organizational gurus to make such decisions. To date, we have used DISCO as a means for selecting connectors for 30 real world distribution scenarios that have emerged from projects in which we have participated at NASA's Jet Propulsion Laboratory (JPL). The early results indicate that DISCO is accurate and scalable.

The rest of this paper is organized as follows. Section 2 describes background and related work in understanding software connectors, and the issues that exist between selecting, and using them to satisfy user requirements. Section 3 describes the DISCO framework for software connector classification and selection using a small illustrative example. Section 4 presents our preliminary evaluation results obtained using DISCO. Section 5 discusses those results and concludes the paper.

## 2 Background and Related Work

Our work is influenced by several areas of existing research, including classifying and selecting software components and connectors, and studies of middleware technologies. We survey representative examples of related work in each of these areas below.

Our principal source for the classification of software connectors to date has been the taxonomy of software connectors by Mehta et al. [12]. Although other works provide a theoretical foundation for software connectors (e.g., [1, 14]) our work within DISCO on classifying distribution connectors has drawn heavily from this taxonomy.

Several approaches for the selection of COTS components have been developed over the past: we consider three of them that are most closely related to our own approach below.

The approach by Mancebo et al. [7] focuses on selecting COTS components during the architecture design phase of software development. The authors focus on three key trade dimensions that are important in selecting COTS components in an architecture: (1) *requirements satisfaction*, (2) *architectural assumptions*, and (3) *provided and required COTS component functionality*. They construct matrices that evaluate COTS components for an exact fit along these three dimensions, and use that data to guide the selection. The approach relies on the knowledge of the "key architectural assumptions" that a *component* makes. Our approach considers similar trade dimensions; however, in contrast to Mancebo et al., our approach uses the connector information as a means of ranking connectors, allowing comparison of "how well" a connector fits a particular scenario.

Alves et al. [2] developed a goal-based quality model approach for assessing COTS components, and their applicability to the architecture of a software system. Similar to our score-based algorithm discussed below, their approach requires the definition of "quality functions" that map a user's satisfaction of a particular COTS component to a given goal or requirement for the system. In contrast to our approach, this approach is entirely *black box*.

We have recently completed a prototype study that integrated DISCO with a framework for COTS component selection for interoperability assessment [3]. In our study, we developed a COTS component and connector selection approach that utilized COTS definition models and integration assessment rules to determine COTS components that could be applied to implement architectures, increasing the chances of interoperability with other components in the system.

Middleware and software bus [10] technologies have often been thought of as implementation-level artifacts of architectural connection evidenced by the fact that many data distribution connectors (e.g., OODT [9], GridFTP) are implemented by leveraging capabilities from existing middleware technologies. Although there have been several recent attempts to classify and compare middleware technologies (see Emmerich [4], Zarras [17]), that work has been very broad. Our work on DISCO provides a systematic approach for classifying how different connectors affect the data distribution application family, and for selecting connectors to satisfy data distribution scenarios along their specific dimensions.

# 3 DISCO: A Framework for Connector Classification and Selection

In this section, we will first describe the DISCO framework. We will then present a small illustrative example highlighting the need for DISCO's automated connector selection. The section concludes by describing two connector selection algorithms that we have devised. The first is a *black box* approach focusing on a connector's relationship to four performance dimensions: consistency, efficiency, scalability, and dependability. The second is a *white box* approach that relates the DCP information about a connector to the scenario dimensions, including the performance requirements.

## 3.1 Framework Overview

To perform connetor selection in DISCO, a user specifies a *data distribution scenario* as input into the framework. Each distribution scenario is a set of constraints on one or more of eight key architectural dimensions of data distribution. The eight dimensions were identified via a thorough literature study (see [8] for a full survey), and in the context of our own experience on the OODT project constructing data distribution systems for planetary science and cancer research at NASA's Jet Propulsion Laboratory (JPL) [9]. The eight dimensions are:

1. *Total Volume* - the total amount of data that needs to be transferred from providers to consumers of data.
2. *Delivery Intervals* - the number, size and frequency (timing) of intervals within which the data should be delivered.
3. *Performance Requirements* - any constraints and requirements on the consistency, efficiency, scalability and dependability of the distribution scenario.
4. *Number of Users* - the number of unique users to whom data volume needs to be delivered.
5. *Number of User Types* - the number of unique user types (e.g., scientists, students) to whom the data volume needs to be delivered.
6. *Data Types* - The number of different data types (e.g., stream, header, metadata) that are part of the total volume to be delivered.
7. *Geographic Distribution* - The geographic location of the data providers and consumers.
8. *Access Policies* - The number and types of access policies in place at each producer and consumer site.

A data distribution scenario in DISCO is expressed as a constraint query against the above dimensions. Examples of constraint queries (i.e., distribution scenarios) include:

$$TotalVolume = 100GB \wedge NumUsers = 10000 \quad (1)$$

$$TotalVolume \geq 50GB \wedge NumUsers = 100 \wedge DeliverySchedule.NumIntervals = 2 \quad (2)$$

Distribution scenarios can be expressed using ranged and discrete values, depending upon whether the dimension is a number (e.g., *TotalVolume*), or a string (e.g., *GeographicDistribution*). We have found this representation to be sufficiently expressive. It was derived from direct contact with scientists in different domains at JPL, and from a review of the available literature on data distribution.

Users input a distribution scenario into DISCO, and then DISCO is tasked with deciding what available connectors will suit the given scenario. This is accomplished by maintaining a knowledge base of *distribution connector profiles* (or DCPs). The DCPs contain metadata described by Mehta et al. [12], including: (1) *data access metadata* - information including locality, number of receivers/senders, and transient availability; (2) *stream metadata* - information including bounds, buffering, and throughput; and (3) *distributor metadata* - information including naming, delivery semantics, and routing. New DCPs are added by architects as new connectors become available.

DISCO uses the DCP information coupled with the distribution scenario as input to a *Selector* component. We have chosen to make the selector an interface, so that developers can create their own selection algorithms, allowing them to devise new selection approaches that relate the scenario dimensions to the available connector information.

### 3.2 Motivating Example

To motivate DISCO's automated connector selection capability, we will revisit the construction of a movie distribution system from Section 1. The architecture of such a system is shown in Fig. 1. The digital movie archive consists of a *DigitalMovieRepository* that stores the physical movie files in AVI or MPEG format, a *DigitalMovieCatalog* that stores metadata information about a movie, including information about its genre, director, and cast. The repository and catalog are fronted by APIs that allow access to the both the movie data and its metadata. A *BackupSite* periodically connects to the *DigitalMovieRepository* to backup its movie data and metadata. *InterestedUsers* are the observable customers of the system: they expect the ability to both download movies at their own request, and have movies delivered to them by subscription mechanisms using the movie metadata as criteria for inclusion.

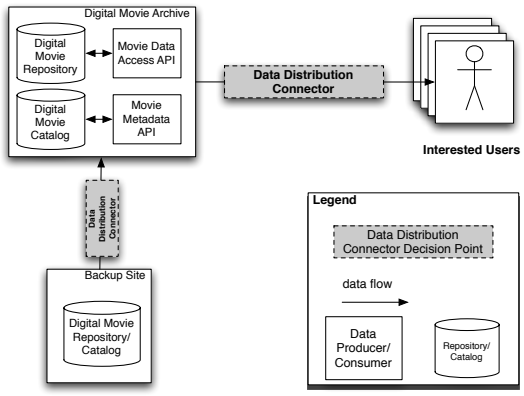


**Figure 1. High level architecture of a movie distribution system.**

Envisage that the architect building the system has decided by examining various trade articles, documentation, and mailing lists, that four data distribution connectors are appropriate to consider given the requirements of the system: GridFTP, Bittorrent, HTTP/REST, and FTP. There are at least three distinct distribution scenarios that can be extrapolated from the architecture of the system. The first scenario (representing the movie backup activity) can be expressed in human readable fashion as *periodic delivery of terabytes of data across the WAN to 1 user*. In turn, this can be translated into DISCO using the constraint query, $DeliverySchedule.NumIntervals = 1 \wedge TotalVolume > 1\text{ TB} \wedge GeographicDistribution = \text{WAN} \wedge NumUsers = 1$. The other two scenarios represent the aperiodic and periodic movie deliveries to *InterestedUsers*.

Even with just three scenarios, manual connector selection becomes unweildy. A reasonable idea would be for the architect to choose the connector that allows her to handle all three distribution scenarios: a connector supporting (a)periodic delivery, terabytes of data, and thousands of users. In that case, documentation suggests that GridFTP supports all three scenarios, except for aperiodic deliveries. Aperiodic deliveries are the cornerstone of the business model for the movie distribution company though, hence they *must* be supported. A quick examination of the actual scenario constraints leads the architect to realize that, without a clear way to rule out the other three connectors besides GridFTP, for each connector she must, examine between 4 to 6 scenario constraints (up to eight depending upon the level of specificity), and determine how *well* (if at all) each connector handles the scenario as a whole.

In the worst case, the architect for the movie distribution system will have to evaluate 32 scenario dimensions for the four connectors (eight scenario dimensions times four connectors), a seemingly small number, there are several subtle complexities: (1) constructing a connector evaluation matrix comprising 32 requirements to check is a large enough decision matrix in its own right, and the addition of each connector into the decision problem introduces up to eight more requirements to check; (2) the satisfaction of the scenario requirements denoted by the columns in the matrix may be difficult to discern without extensive empirical evaluation; and (3) even if the architect is able to choose the right connector(s), the critical information of *why* and *how* the connector was deemed to satisfy the requirement remains inside the architect's head, yielding yet another guru.

In the remaining section, we summarize two connector selection algorithms that are part of DISCO that deal with these stated complexities accurately and efficiently.

### 3.3 Connector Selection Algorithms

The different subsets of our team have independently developed two connector selection algorithms to date: score-based and Bayesian.The *Score-based* connector selection algorithm takes a black-box approach to selecting connectors. To use the algorithm, architects develop *score functions* comprised of discrete, known points relating a particular distribution scenario dimension to the performance requirements on the scenario (recall dimension 3 in Section 3.1). As an example, an architect may define a score function relating the NumUsers dimension with the efficiency performance requirement using a few empirically determined performance points for a given connector. Functions derived from these points for a subset of the connectors in our knowledge base are shown in Fig. 2.

In all, there can be 28 score functions for each connector (seven dimensions excluding dimension 3 times the four performance requirements). To compute a rank for each

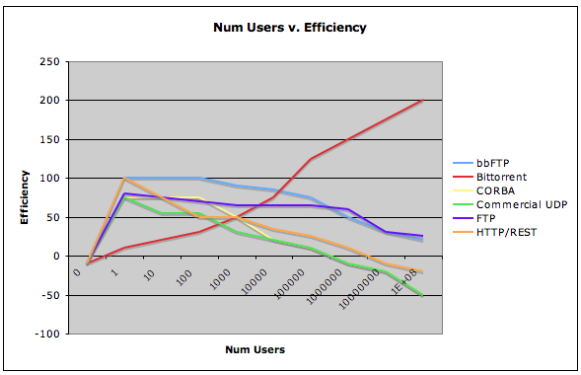


**Figure 2. Sample score function relating NumUsers and efficiency.**

connector, the algorithm iterates over all the connectors. For each connector, sums are computed corresponding to the performance requirements of interest. For each sum, the algorithm iterates over the seven scenario dimensions. If there are score functions defined for the connector for the given performance requirement and scenario dimension, then the function is evaluated (using the distribution scenario value for the function's dimension) and the resultant value added to the corresponding sum. Finally, for each of the connectors, a weighting function (provided as input to the algorithm) is used to influence each of the sums, allowing a user to define her interest in the performance requirements. As an example, the user may choose an objective function that maximizes consistency while neglecting efficiency.

The *Bayesian* [16] selection algorithm takes a white-box approach to connector selection. Its goal is to use a connector's architectural metadata (provided by its DCP) as a means of relating the scenario dimensions to a connector. The Bayesian algorithm takes as input a *domain profile* (the equivalent of a *conditional probability table* [16]) that captures an architect's knowledge of the relationship between the DCP metadata and the values for scenario dimensions.

As an example, consider a distribution scenario in which the $TotalVolume \geq 100TB$. Additionally, assume that we are given DCPs for two connectors, A and B, and that we are interested in the *data access transient availability* metadata attribute, specifying whether a connector has *Session-based*, *Cache-based*, or *Peer-based* access to data. The Bayesian algorithm allows an architect to favor *Cache-based* data access, which is faster and more scalable than the other two alternatives. An architect may encode this knowledge by assigning the probability value $0.85$ to any connector that has *Cache-based* data access, $0.30$ to connectors that exhibit *Peer-based* access, and finally $0.10$ to connectors that have *Session-based*. This information would be recorded as shown in Table 1.

**Table 1. Example domain profile knowledge for the Bayesian algorithm.**

| **Data access transient availability** | $TotalVolume \geq$ 100 TB | ... |
|---|---|---|
| *Cache* | 0.80 | ... |
| *Peer* | 0.30 | ... |
| *Session* | 0.10 | ... |

For each attribute of each DCP, the algorithm computes the value range for the attribute, and applies the discrete version of Bayes theorem [16] to arrive at a probability distribution for each value, for each attribute. The probabilities corresponding to the attribute value for each DCP are aggregated through multiplication to formulate a set of final probabilities for each DCP. These are, in turn, used to rank and compare the connectors of interest.

## 4 Preliminary Evaluation

In order to evaluate both our capacity to capture connector profile knowledge and the ability of each selection algorithm to accurately classify connectors appropriate to given distribution scenarios, we performed a precision-recall analysis against an answer key agreed upon by a number of experts. Our experiment conducted involved the use of 13 DCPs, including canonical profiles we built for the major distribution connectors GridFTP, bbFTP, Bittorrent, and HTTP/REST. The DCPs were constructed by examining documentation for each of the connectors including expert reviews, research literature, and from our own experience. For the score-based algorithm, 24 score functions were defined for each of the 13 connectors: we omitted the definition of score functions for *Access Policy* as we are still solidifying its value representations. For the Bayesian algorithm, a Bayesian domain profile consisting of 100 conditional probabilities was used.

### 4.1 Precision-Recall Analysis

Each of DISCO's algorithms was run against 30 real world distribution scenarios, including 10 high volume scenarios, 11 medium volume, and 9 low volume scenarios, with appropriate variations in each of the eight scenario dimensions (number of users and users types, WAN vs. LAN, etc.). Using our collective domain knowledge, we devised an "answer key" ahead of time in which we classified each connector analyzed into one of two classes: *appropriate* for the scenario or *inappropriate*.

In evaluating DISCO's algorithms for connector selection, we preformed precision-recall analysis in order to

measure error rate (an estimate of the probability of correctly labeling a connector as appropriate for a scenario), recall (the probability of detecting a connector as appropriate for a scenario), and precision (the fraction of selected connectors which are actually appropriate for a scenario).

**Table 2. Precision-Recall Confusion Matrix**

| | Bayesian | score-based |
|---|---|---|
| True Positives (*TP*) | 101 | 63 |
| False Positives (*FP*) | 25 | 200 |
| True Negatives (*TN*) | 245 | 67 |
| False Negatives (*FN*) | 19 | 60 |

Results from both algorithms were further classified into appropriate and inappropriate connectors via an exhaustive implementation of the k-means clustering algorithm (k=2). Comparing these results to our answer key, we produced the confusion matrix presented as Table 2. Error rate, precision, and recall were calculated from this confusion matrix and are presented in Table 3.

**Table 3. Precision-Recall Experiment Results**

| | Bayesian | score-based |
|---|---|---|
| Error Rate | 11.28% | 32.56% |
| Precision | 80.16% | 48.46% |
| Recall | 25.90% | 16.15% |

## 5 Discussion and Future Work

We found that the Bayesian algorithm had a higher precision rate (80%) than the score-based algorithm (48%). While the exact reasons for this trend are still being investigated, we suspect that the answer lies in the fact that the Bayesian algorithm is a white-box approach, requiring the architect to understand how the internal workings of a connector suit a given distribution scenario. The score-based approach, while more efficient than the Bayesian, shows lower precision due to the fact that it requires the user to have an understanding of how performance requirements and distribution scenarios influence each other depending on the connector, ignoring its internal architecture. This leaves a disconnect, where the architect must maintain an implicit mental model of the connector, as opposed to the Bayesian algorithm, where this model is made explicit.

While the early experience with DISCO has proved valuable, a number of pertinent research issues remain unexplored. In the immediate future, we plan on performing extensive evaluation of DISCO's precision and recall against that of empirical survey data gathered from data distribution experts in the fields of software engineering, data management, science data systems, and information retrieval. Additionally, we plan on investigating architectural mismatch within connectors and data system architectures [5].

## References


[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[2] C. Alves, X. Franch, J. P. Carvallo, and A. Finkelstein. Using goals and quality models to support the matching analysis during cots selection. In *ICCBSS*, pages 146–156, 2005.

[3] J. Bhuta, C. A. Mattmann, N. Medvidovic, and B. Boehm. A framework for the assessment and selection of software components and connectors in cots-based architectures. In *WICSA*, 2007.

[4] W. Emmerich. Software engineering and middleware: a roadmap. In *ICSE - Future of SE Track*, pages 117–129, 2000.

[5] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. In *ICSE*, pages 179–185, 1995.

[6] H. Kincaid, S. Kelly, D. J. Crichton, D. Johnsey, M. Winget, and S. Srivastava. A national virtual specimen database for early cancer detection. In *CBMS*, pages 117–123, 2003.

[7] E. Mancebo and A. A. Andrews. A strategy for selecting multiple components. In *SAC*, pages 1505–1510, 2005.

[8] C. Mattmann. Software connectors for highly distributed and voluminous data-intensive systems. Qualifying Exam Report USC-CSE-2006-602, Univ. of Southern California, 2006.

[9] C. Mattmann, D. J. Crichton, N. Medvidovic, and S. Hughes. A software architecture-based framework for highly distributed and data intensive scientific applications. In *ICSE*, pages 721–730, 2006.

[10] N. Medvidovic, E. M. Dashofy, and R. N. Taylor. The role of middleware in architecture-based software development. *International Journal of Software Engineering and Knowledge Engineering*, 13(4):367–393, 2003.

[11] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Software Eng.*, 26(1):70–93, 2000.

[12] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *ICSE*, pages 178–187, 2000.

[13] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[14] M. Shaw. Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status. In *ICSE Workshop on Studies of Software Design*, pages 17–32, 1993.

[15] M. Shaw and D. Garlan. *Software Architecture - Perspectives on an emerging discipline*. Prentice Hall, 1996.

[16] R. L. Winkler. *An Introduction to Bayesian Inference and Decision*. Probabilistic Press, 2003.

[17] A. Zarras. A comparison framework for middleware infrastructures. *Journal of Object Technology*, 3(5):103–123, 2004.