Galapagos: Model-driven discovery of end-to-end application–storage relationships in distributed systems

K. Magoutis
M. Devarakonda
N. Joukov
N. G. Vogl

Modern business information systems are typically multi-tiered distributed systems comprising Web services, application services, databases, enterprise information systems, file systems, storage controllers, and other storage systems. In such environments, data is stored in different forms at multiple tiers, with each tier associated with some level of data abstraction. An information entity owned by an application generally maps to several data entities, logically associated across tiers and related to the application. Discovery of such relationships in a distributed system is a challenging problem, complicated by the widespread adoption of virtualization technologies and by the traditional tendency to manage each tier as an independent domain. In this paper, we present a system and methodology for model-driven discovery of end-to-end application–data relationships spanning multiple tiers, from the applications to the lowest levels of the storage hierarchy. The key to our methodology involves modeling how data is used and transformed by distributed software components. An important benefit of our system, which we call Galapagos, is the ability to reflect business decisions expressed at the application level to the level of storage.

Introduction

Modern applications follow a layered architecture using application, middleware, and storage tiers [1–5]. The layering reflects various levels of application abstraction such as the user interface, business logic, application services, data services, and group communication services, as well as infrastructure (servers, network, and storage) virtualization. In such layered systems, IT (information technology) architects and administrators typically pose questions such as, "Which storage resources are used by a particular application?" and "Which applications and middleware systems depend on a particular file?" Answers to these questions, in the form of end-to-end application–data relationships, have a number of important uses, such as deriving business-driven information life-cycle management (ILM) policies involving data, improving accuracy of root-cause analysis in case of failures, accounting for storage usage on a per-application basis, and reasoning about data and application availability and reliability.

The task of discovering application–data relationships in distributed systems is complicated by two trends in system design. First, virtualization [6–9] results in systems consisting of multiple tiers separated by narrowly defined interfaces, leading to limited visibility of the underlying resources [10]. Second, the traditional trend of viewing and managing each tier as an independent domain (e.g., application [11], database [12], or storage [13]) contributes to disconnections in the flow of information linking applications to data at each tier. In this paper, we present a system and methodology for discovering end-to-end relationships between specific instances of data and application components, that is, from high-level applications to the low-level data and storage tiers.

An example of the complexity of the problem is shown in **Figure 1** in which a hosting environment is shared by

0018-8646/08/$5.00 © 2008 IBM

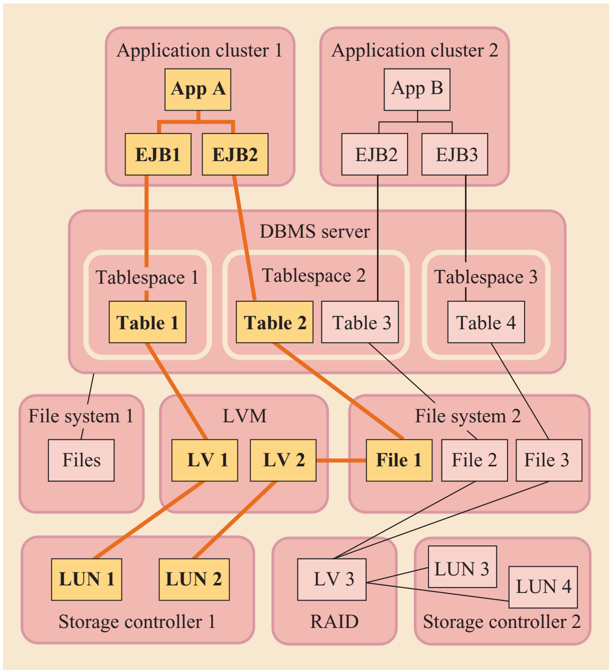


Figure 1

Relationships between applications and data objects through a distributed system. Relationships specific to application A are shown in bold. (EJB: Enterprise JavaBeans**; App: application; DBMS: database management system; LVM: logical volume manager; LUN: logical unit number; RAID: Redundant Array of Independent Disks.)

two applications, A and B. Each middleware tier supporting A and B implements several instances of data abstractions. In addition, each tier is typically partitioned over several physical resources (e.g., servers and storage controllers). End-to-end relationships between applications, data-access objects (e.g., Java** Version 2 Enterprise JavaBeans [EJB]), relational tables, files, logical volumes, and storage volumes can be fairly complex, as shown in this example. The complexity arises from the number and types of middleware technologies involved and the large number of tiers present in virtualized distributed systems.

Previous research [1–3, 5] focused on methods to discover cross-domain relationships in distributed systems, either by statistically analyzing system behavior [1, 5], on the basis of observation of system activity, or by using system support (e.g., passing tokens or other metadata over communication between layers [2, 3]). In addition, several commercial tools focus on discovery of infrastructure assets by scanning a range of IP (Internet Protocol) addresses and querying the systems that respond [14–16]. Network communication relationships among applications are discoverable by capturing network packets and analyzing their headers [17].

Additional refinement of asset discovery has been achieved through template-driven discovery of applications [18]. Our system and methodology, called Galapagos, follows a model-driven approach [19, 20] to enrich basic infrastructure discovery with more comprehensive information about dependency between applications and data (e.g., business objects, tables, files, and other information entities).

Galapagos uses reusable data-location and data-mapping models of individual software components, which contain introspection code (mostly scripts) to extract data-specific information from software components. **Figure 2** depicts a representation of the key system aspects captured in any Galapagos model. The modeling in Galapagos is characteristic of the gray-box approach [21] to systems analysis, which is based on understanding of general behavior rather than internal details of systems. It is important to note that Galapagos models are unrelated to the queuing-network models that are typically used for performance analysis [22].

Galapagos combines models of software components with a distributed crawling (graph traversal) algorithm to discover end-to-end, multi-tier dependencies between applications and data in a tiered distributed system. This is accomplished in an easily extensible fashion. The act of adding a new middleware tier simply “plugs” the new tier into the overall end-to-end relationship representation model.

Because Galapagos does not rely on monitoring runtime activity, it is applicable even in cases where system activity cannot be captured, because of privacy, performance, or intrusion constraints. Therefore, Galapagos is complementary to systems such as Project5 [1], Causeway [2], and Pinpoint [3].

The key elements of Galapagos include the following:

1. Use of a composable model of data use and mapping by a middleware system or an application, as well as associated introspection methods.
2. A scalable crawling algorithm that composes information from several model instances in a distributed system, creating a complete set of end-to-end application–data relationships.
3. Analysis of collected information to solve a broad range of storage management problems.

The remainder of this paper is structured as follows: First we describe the application awareness that Galapagos brings to storage management. Next, we provide an overview of the Galapagos system and methodology. We then describe the Galapagos models and the discovery process. The implementation and evaluation of Galapagos are described next, and the last

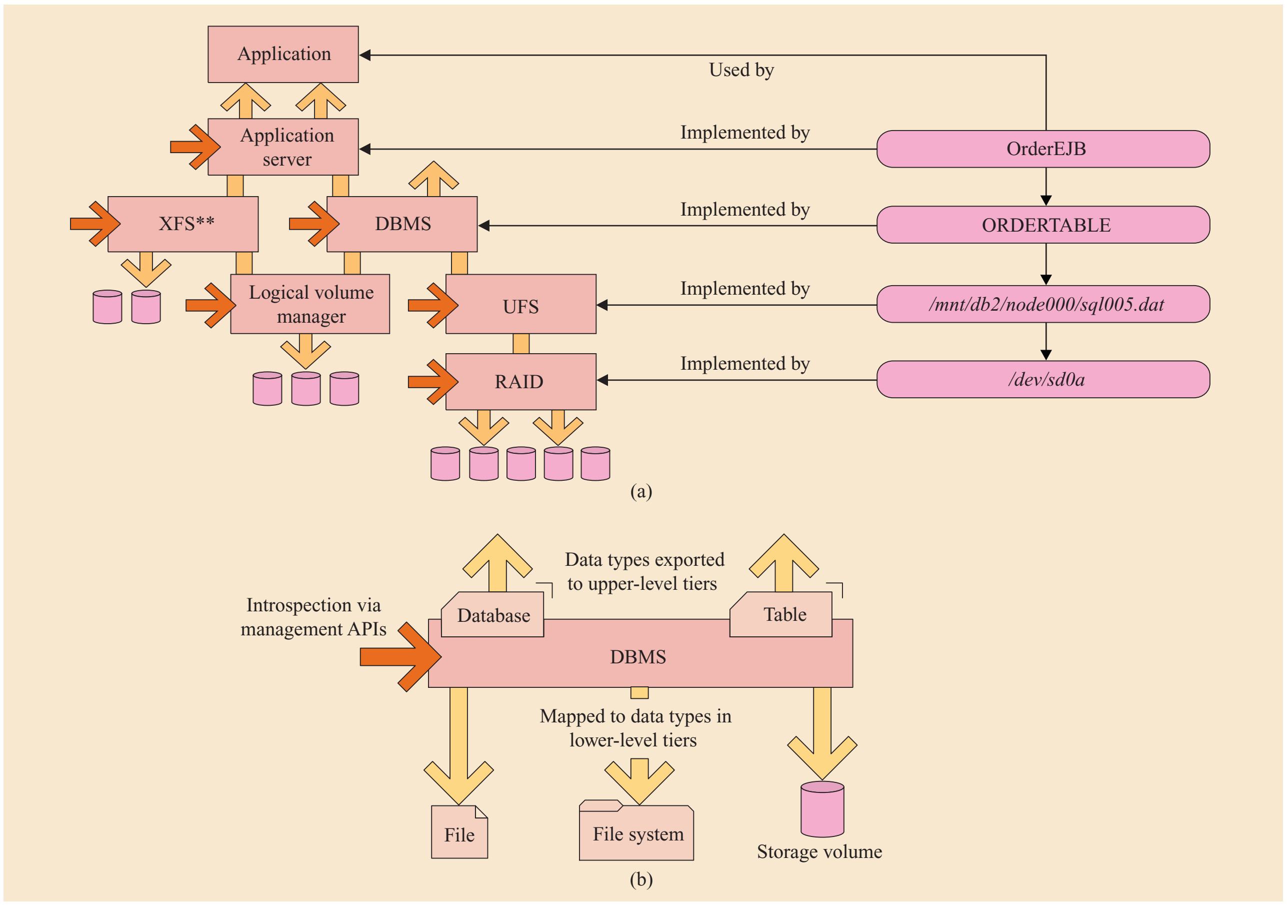


Figure 2

Composition of DLT models creating a graph for the distributed system (a) and key aspects of a DBMS captured in its DLT model (b). [XFS: extended file system; DBMS: database management system; UFS: UNIX** file system; DLT: data-locations template; OrderEJB: an Enterprise JavaBeans (EJB) data-access object; sd0a: storage device 0a; ORDERTABLE: database table.]

two sections present related work and concluding remarks.

Application-aware storage management

Manual discovery of the storage dependencies and usage of an application is difficult and impractical for large distributed systems. For example, consider the file */mnt/db2/node000/sql005.dat* on a server file system, as shown in Figure 2(a). It is evident that manual identification of that file (e.g., which relational table is it part of or which application does it map to) is difficult and error prone, even for experts in system administration. The ability to identify the data and storage resources used by an application enables a wide range of important uses, discussed in the following paragraphs.

Information life-cycle management (ILM)—Galapagos-discovered applications–data relationships facilitate the alignment of enterprise information with the most appropriate and cost-effective storage infrastructure using these relationships. A number of important ILM tasks such as provisioning, back up, and migration can be performed in an application-aware manner, reflecting the business value of applications to the data.

Documentation and reporting of storage usage—Current storage reporting tools lack knowledge of end-to-end application–data relationships and thus present storage usage only on a per-host or per-controller basis. Documentation and reporting of storage usage, a key deliverable in storage service outsourcing deals, can indicate storage inefficiencies or improve the accuracy of charging for storage use.

Server consolidation—Technology advances, cost reduction, or data-center maintenance tasks often require migration of data between storage systems. To ensure minimal impact on online operations, there is a need to plan migration actions in an application-aware manner, that is, related volumes or files should be migrated together as a group or only when the application is unavailable due to maintenance. This ability restricts the impact of consolidation to specific, known applications and enables phased planning of migration actions.

End-to-end reliability assessments—The reliability of enterprise applications depends on the reliability of the many different middleware components and the underlying storage infrastructure on which they are deployed. To understand and reason about application availability, one needs to understand 1) various reliability characteristics of each middleware tier, 2) specific ways that middleware is configured to serve a particular application, and 3) the reliability of the underlying infrastructure as it relates to that application. Galapagos combines its discovered end-to-end application–data relationships with information about the storage infrastructure (e.g., replication, striping, parity schemes, and other reliability characteristics) to construct an application-specific, end-to-end reliability assessment that can be used to indicate potential weaknesses in enterprise infrastructure.

Recovery planning—Large enterprise systems comprise several geographically distributed components. Within these systems, sequences of communicating applications that participate in a business process must be grouped together for capacity, availability, and performance planning. Recovery actions must ensure that the data of all applications in the process has been recovered and is available before resuming operation of the entire process.

Storage consolidation and isolation—Hosting multiple applications on a shared infrastructure is an effective way to improve efficiencies and reduce operational costs in consolidated data centers. However, applications that belong to different customers are often required to be isolated in order to ensure that they do not interfere. Through the information it collects, Galapagos can verify that no two customers share storage systems, data objects, or software components. It can also help where consolidation can be achieved effectively.

System and methodology

The basic Galapagos system and methodology is depicted in **Figure 3**. In step 1, Galapagos uses the basic infrastructure information about the target IT infrastructure (i.e., installed software and hardware components) as a starting point in the discovery methodology. This information is provided in the form of a system configuration (SC) model using industry-standard representations [23, 24]. An instance of the SC model can be populated by existing IT infrastructure discovery systems [14–16] or it can be manually populated. The SC instance helps identify data- and storage-related middleware systems that are present in a particular system configuration; these systems include application servers, database management system (DBMS), file systems, logical volume managers, storage virtualization engines, and RAID (Redundant Array of Independent Disks) systems.

In step 2, each identified middleware system is abstracted in a data-locations template (DLT) model, which describes the data locations and data-mapping characteristics of the middleware software components, as shown in Figure 2(b) for the case of a typical DBMS. Specifically, a DLT describes 1) the data types implemented and exported by the component, 2) the mappings of these data types to underlying data types, and 3) reference to introspection support (scripts) to extract data-specific information from software components. DLTs are "point descriptions," that is, each DLT is used to model only a single software component. Only one DLT model is required for each software component type (e.g., one DLT for IBM DB2* version 9.1).

In step 3, DLTs are, by design, *composable* through their data mappings. Each data type exported by a DLT is mapped to data type(s) exported by other DLTs creating a composition of the corresponding DLTs. Composition of DLTs from successively stacked middleware in a given configuration produces a *model graph*, an example of which is shown in Figure 2(a). In this example, a top-level application is deployed in a Java Version 2 Enterprise Edition (J2EE) application server, implementing and providing the data-access objects (e.g., OrderEJB, an EJB) used by the application. The application server in turn connects to lower-level data services such as a DB2 DBMS instance and a file system (XFS) implementing and providing relational tables (e.g., OrderEJB) and files (e.g., the DB2 software installation files), respectively. XFS is supported by a logical volume manager (LVM), whereas the DBMS instance is supported by the LVM and a file system (UNIX file system [UFS]), which implement logical disks and files (e.g., */mnt/db2/node000/sql005.dat*), respectively. These data and storage tiers are supported by lower-level storage abstractions such as RAID (implementing disk device */dev/sd0a*) and storage virtualization engines.

In step 4, the model graph produced in step 3 is traversed by a distributed crawling algorithm using DLT information along the way to discover end-to-end application–data relationships across the distributed system. Starting at the root elements in the model graph (i.e., the applications and the information entities they use), Galapagos uses DLT information to map higher-level data to the lower-level data objects and records the discovered relationships. The process is repeated recursively along the model graph, eventually producing a transitive closure that includes all end-to-end relationships.

Creation of a DLT requires some degree of manual effort (for a detailed discussion, see the section "Complexity of creating DLT models"). This effort, however, is expended only once, ideally at the time of development of a major version of the middleware system, and is amortized over repeated uses of the technology. The key is that a DLT expresses the "how" (i.e., how to discover data use in any installation of this middleware), whereas in each installation we discover the "what" (i.e., what end-to-end application–data relationships exist in this particular installation).

The crawling algorithm often requires invocation of remote management application programming interfaces (APIs). To avoid the need to install and maintain software ("agents") on the managed environment, Galapagos uses ephemeral remote processes ("sensors"), transferring scripts over the network on demand and executing them on remote management servers with appropriate credentials. This approach is often called "agent-free" discovery.

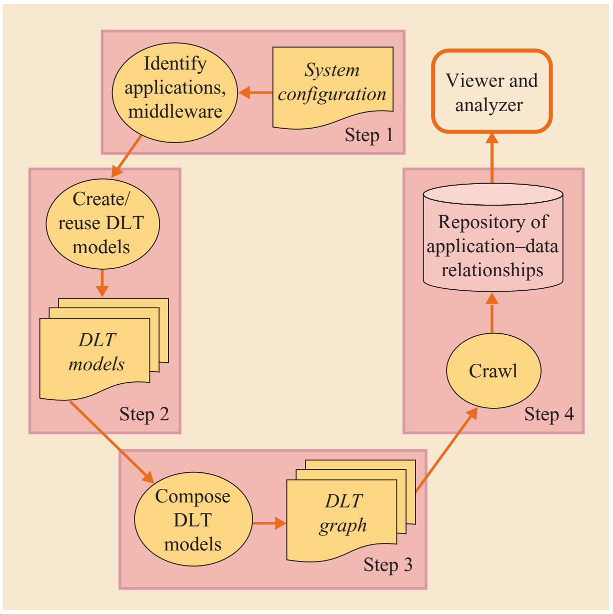


Figure 3

Galapagos methodology consists of a four-step process leading to discovery of application–data relationships stored in a repository. A viewer and analyzer can leverage the collected information to solve a broad range of storage management problems.

Galapagos models

Galapagos is designed to discover and represent all end-to-end, multi-tier relationships between applications and data in an m-tiered distributed system, in a manner that is technology independent and easily extensible: The act of adding a new $(m+1)$th middleware tier in the m-tiered distributed system easily "plugs" the new tier into the existing Galapagos discovery framework. To achieve interoperability across different distributed systems technologies, Galapagos adopts key principles of the Object Management Group Model Driven Architecture** (MDA**) [19, 20]. MDA is a method for creating system specifications separating high-level functions from implementation details. With MDA, functionality and behavior are modeled only once. An MDA specification consists of a base platform-independent model (PIM) plus one or more platform-specific models (PSMs) and sets of interface definitions, each describing how the base model is implemented on a different middleware platform.

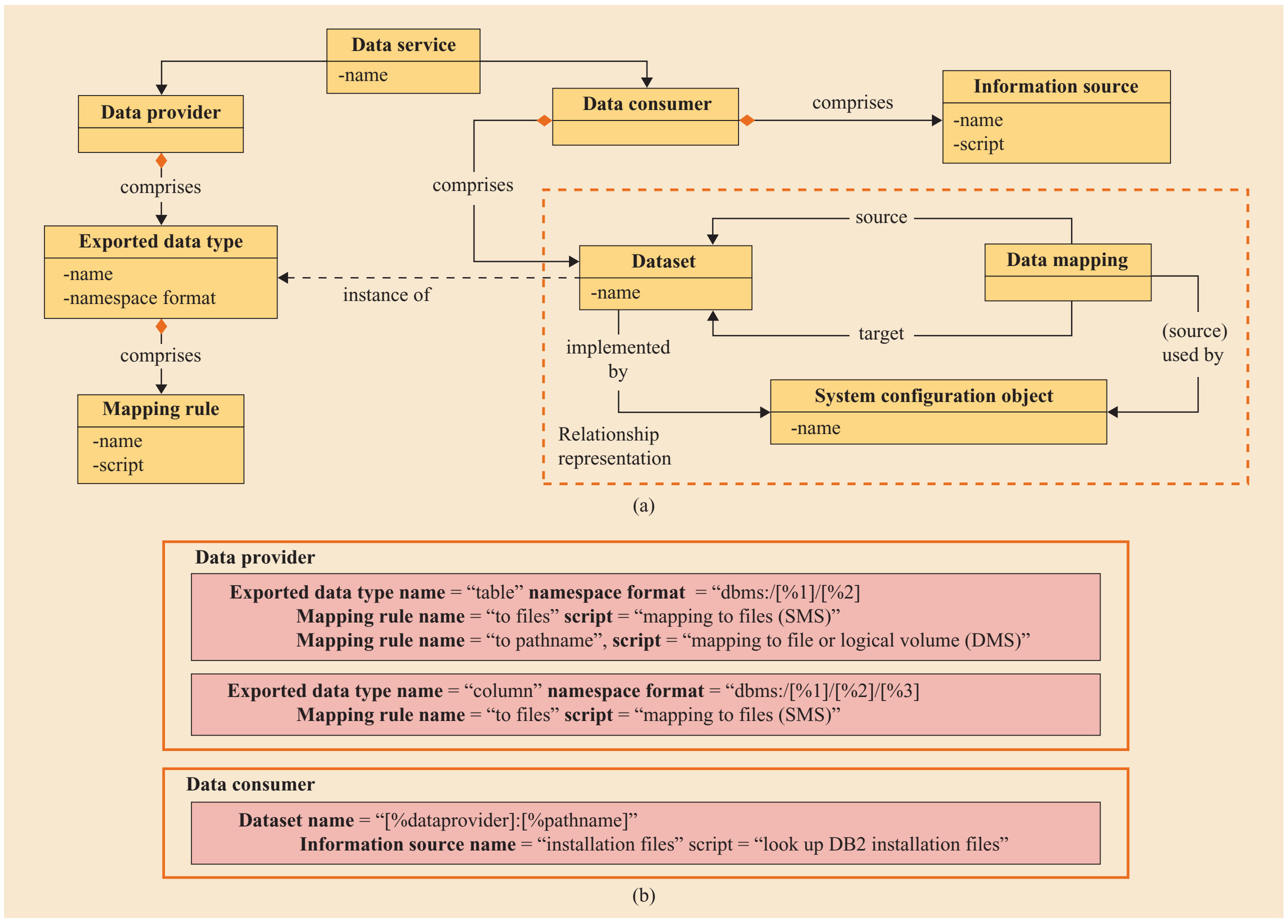
Data-locations template

As introduced in the section "System and methodology," the DLT is defined by a platform-independent meta-model [**Figure 4(a)**] encapsulating technology-independent data specification and mapping concepts. The root of a DLT is a data service, which is a technology-independent reference to a middleware system, and it consists of two parts: a data provider, focusing on provision and mapping of data, and a data consumer, focusing on consumption of data.

A key notion in Galapagos is that of an exported data type, which is an abstraction for data entities, implemented, stored, and accessed anywhere in a distributed middleware system. Exported data types have associated namespaces, which are uniform resource indicator (URI)-like descriptions of data with the following general syntax: PROVIDER:/NAME$_1$/NAME$_2$/.../NAME$_i$. In this scheme, PROVIDER refers to the software component that exports the data type. Instances of exported data types are called *datasets*. A dataset in the above format may also contain variables that are bound at a later time to the output of scripts, as well as wildcards (e.g., the equivalents of *, % in UNIX).

Another key notion in Galapagos is data mappings between exported data types of a middleware system and underlying data types provided by back-end data services. (Here, the term *back-end data service* refers to any data service that supports another data service.) Such mappings are expressed as mapping rules in the data provider section of the DLT model. Mapping rules represent introspection methods (e.g., scripts) that extract data-specific information from software components. In general, a mapping rule between an exported data type α and a lower-level exported data type β maps a dataset of type α to one or more datasets of type β. Given a source dataset, the target datasets are determined by invoking the script specified in the mapping rule.

Aside from datasets produced and exported by middleware on behalf of their clients, applications or middleware systems consume data directly for their own



Data provider

Exported data type name = “table” **namespace format** = “dbms:/[%1]/[%2]
Mapping rule name = “to files” **script** = “mapping to files (SMS)”
Mapping rule name = “to pathname”, **script** = “mapping to file or logical volume (DMS)”

Exported data type name = “column” **namespace format** = “dbms:/[%1]/[%2]/[%3]
Mapping rule name = “to files” **script** = “mapping to files (SMS)”

Data consumer

Dataset name = “[%dataprovider]:[%pathname]”
Information source name = “installation files” script = “look up DB2 installation files”

Figure 4

(a) Data-locations template (DLT) and relationship-representation meta-models (in Unified Modeling Language** format). (b) Also shown is an excerpt of a DLT model for DB2 version 9.1. (SMS: systems-managed space; DMS: database-managed space.)

purposes. Examples are the files that make up a software installation or the business objects used by an application. (Business objects are data objects accessed by applications in the course of executing a business process.) If their names are unchanged across installations, these datasets can be enumerated in the data-consumption section of the DLT model. Otherwise, they are dynamically discoverable by querying information sources listed in the DLT. Information sources represent introspection methods that extract data-specific information from software components.

Technology-specific customization

The platform-independent DLT meta-model is customized for specific software components, producing platform-specific DLT models. This process requires mapping DLT meta-model concepts to underlying software components. The starting point in this process is the identification of all data representations, implemented and exported by a specific software component, which are modeled as exported data types. **Figure 4(b)** shows an excerpt of the DLT model for the DB2 version 9.1 DBMS describing two exported data types, *table* and *column* with namespace syntax *dbms:/db-name/table-name/column-name*.

The data mappings of each exported data type are determined on the basis of the underlying data types to which the exported data type maps. For example, tables in a DBMS may map to files (within a file system directory) or blocks (within a file or logical volume), depending on the configuration of the database. In the example in Figure 4(b), the underlying data types are files and pathname, corresponding to the system-managed and database-managed storage management modes of DB2, respectively. The two mapping rules, “to files” and “to pathname,” point to the introspection methods (scripts) “mapping to files (SMS)” and “mapping to file or logical volume (DMS),” respectively.

Discovery of data consumed by (i.e., used by) an application or middleware (i.e., a software component) for its own purposes requires separate introspection methods. In the example in Figure 4(b), discovery of the files making up a software installation (e.g., DB2 version 9.1) is encapsulated in the information source referred to as "look up DB2 installation files."

Technology-specific customization of the DLT model requires some extensions to the system configuration (SC) model [23, 24]. These extensions describe the mapping between specific DLT entities and domain-specific SC entities where applicable. For example, in a DLT instance for the IBM WebSphere* Application Server, the data service entity corresponds to a WebSphere cell.

Installation-specific customization

A DLT model for a software component is, by design, independent of any specific installation of that software component. However, to be used for end-to-end relationship discovery, DLT models must encapsulate installation-specific information, such as absolute paths to data and references to SC model entities (such as machine names and installed software). In most cases, creating installation-specific DLT instances can be automatically performed by extracting information from various sources such as operating system registries, middleware management APIs, or configuration management databases [14–16].

Representation of end-to-end relationships

The Galapagos platform-independent model for the representation of the discovered relationships consists of dataset entities, data-mapping entities, and SC object entities (see Figure 4). The data-mapping entity represents a discovered relationship between a source and a target dataset (s, t). Each dataset can participate in any number of data mappings. A set of data mappings $\{(d_1, d_2), (d_2, d_3), \ldots, (d_{n-1}, d_n)\}$ represents a data relationship between d_1 and d_n. The "implemented by" association (also known as *data scope*) between dataset and the SC object (which is a reference to an SC model entity) points to a logical or administrative subdivision of a middleware system that implements the specific dataset. A data scope can be a volume group in a RAID controller, a partition group in a database system, or a cluster of application servers in a J2EE environment. The "used by" association (also known as *data client*) between data mapping and SC object points to an SC model entity that uses the source dataset of the data mapping. For example, a data client of a database table may be a J2EE application server.

Complexity of creating DLT models

DLT model instances and associated runtime support are currently created by experts, developers, or practitioners with intimate knowledge of the data export mechanics of the particular middleware system. We believe that a first-order approximation of the DLT can be produced automatically by software modeling tools and subsequently validated and enhanced by a human expert. We outline the complexity of several real-life DLT models in the section on implementation. DLT models are easier to create in the case of execution environments with well-defined installation and data-access interfaces, such as J2EE and SAP (Systems, Applications, and Products in Data Processing). Discovery of data use is more difficult in less-structured application containers. Particularly challenging cases include those involving unstructured applications running on standard operating systems and placing data in shared directories (e.g., /*tmp*). However, such data use can be discovered by examining the relevant installation logs or by analyzing program sources.

The correctness and accuracy of a DLT is a critical issue. A DLT model that does not fully describe all uses and transformations of data by a software component will cause some application–data relationships to be missed by the discovery process. When a model is produced by an expert and considered mature (e.g., after a reasonable amount of experimentation, debugging, and production use), it is expected to be accurate when applied to its intended domain, which is the correct version of the middleware system it models. Any DLT, however, can be continuously validated during production use, using runtime monitoring techniques.

Discovery process

The Galapagos discovery process starts with identification of data services (applications and middleware systems that can be modeled with DLTs) in the SC model of the distributed system. Next, DLT models for each data service are created or reused as well as customized to the specific installation. Data dependencies between DLT models form a graph structure such as that shown in Figure 2(a). In this graph, two DLT models (e.g., DBMS and UFS) are connected by a rectangle representing data dependence over a data type β, when the first DLT features a mapping rule M_1 between data types α and β, and the second DLT features a mapping rule M_2 between β and γ. An invocation of M_1 may be followed by an invocation of M_2, resulting in a data mapping between α and γ. This composition of M_1 and M_2 extends to any number of DLTs and mapping rules.

Distributed crawling algorithm

The crawling algorithm performs a depth-first search of the multirooted graph shown in **Figure 5**. In the general case, we consider n applications A_1–A_n deployed over m

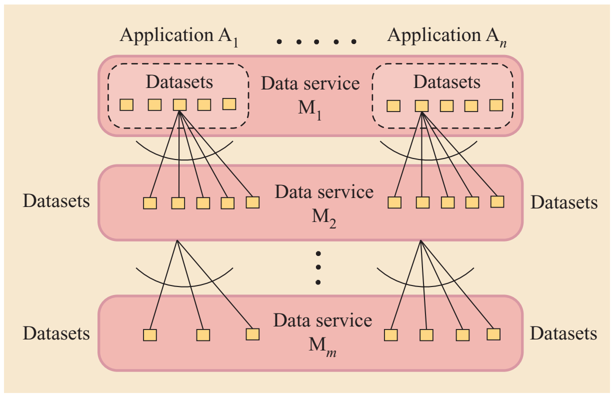


Figure 5

A distributed system consisting of m data services (which are middleware tiers) supporting n applications. Solid squares at each data service represent datasets. Lines between datasets represent data mappings.

successive data services (which are middleware tiers) M_1 to M_m. Data types at the leaves of the graph that are not associated with any data mappings are referred to as "final."

The inputs for the algorithm include 1) an SC model annotated with identified applications and data services, and 2) DLT model instances for each identified application and data service The algorithm is as follows.

Algorithm

1. For each root application A_r in the SC model, discover the datasets $\{D_i\}$ used by A_r by querying the data services to which A_r connects.
2. For each D_i,
   a. If D_i is a final data type, record data relationship and backtrack.
   b. If D_i is not a final data type and D_i has been encountered before, record the discovered data relationship and backtrack.
   c. If D_i is not a final data type and D_i has not been encountered before, visit the data service exporting D_i and access the DLT model of the data service.
      i. Use applicable mapping rules in DLT to map D_i to a list of datasets $\{D'_j\}$. The rules may require the injection of a sensor into a remote administrative point.
      ii. For each D'_j, record the discovered dependency of D'_j, then go to step 2(a) and repeat for D'_j.

The output contains application–data relationships stored in a repository.

Complexity

Each application uses a number of datasets (up to b such datasets), which are provided by data service M_1. Each such dataset maps to a number of dependent datasets (up to f_1 such datasets) provided by data service M_2. In a recursive manner, each dataset maps to a number of underlying datasets [f_i such datasets in this $(i+1)^{\text{th}}$ tier] up to the last data service M_m. Note that each dataset may be mapped multiple times under different data mappings. In practice, the number of mappings per data type rarely exceeds four. The complexity of the algorithm is proportional to the number of nodes visited, of which a conservative bound is

$$O\left(n \times b \times \prod_{i=1}^{m-1} f_i\right). \tag{1}$$

Here, n is the number of root applications. The exponential trend in the product of Equation (1) suggests that scaling challenges may arise as the number of data services (m) or the data-mapping fanout (controlled by f) increases. However, in typical commercial installations, the parameters f and m are small, so the exponential factor is not expected to be a problem in practice. Another factor that affects scalability is the degree of data sharing between applications. For datasets shared by more than one application or middleware component, Galapagos need only crawl downstream from them once, the first time they are visited.

The application–data relationships discovered by Galapagos indicate datapaths that can potentially be followed by transactions through the distributed system. There is no guarantee, however, that these paths are followed on a production system. As such, the set of reported application–data relationships can be a superset of the set of realized relationships.

Implementation

The current Galapagos prototype is implemented in Java and includes a centralized discovery engine, a set of sensors, an Eclipse** platform-based user interface, and two repositories: a repository of models and scripts, and a repository of application–data relationships. Eclipse is an open-source software framework written primarily in Java. The SC model representation that we currently use is a manually pre-populated XML (Extensible Markup Language) file. Although our prototype includes some support for basic IT infrastructure discovery, the general problem has been addressed elsewhere [14] and is outside the scope of our work.

Our current prototype includes DLT models for 1) J2EE application servers such as distributed

Table 1 Representative complexity of creating DLTs models.

DLT model	*Exported data types*	*Runtime support for model (lines of code)*
WebSphere version 5.1 or 6.0	Application, module, message queue, Enterprise JavaBeans	5,940
DB2 version 8 or 9	Database, table, column	924
Spring/iBATIS	Application, data-access object	1,045
Redundant Array of Independent Filesystems (RAIF)	File	287

WebSphere and Spring (a layered Java/J2EE application framework), 2) a persistence framework that enables mapping SQL queries to Java objects (such as iBATIS), 3) the DB2 database, and 4) Redundant Array of Independent Filesystems (RAIF) [25], a stackable (extensible) file system that maps files to other files using striping, replication, or parity schemes. The implementation details of these models are outlined in **Table 1**. We are currently creating DLT models for IBM HTTP Server, IBM WebSphere MQ network communication technology, SAP, and IBM TotalStorage* Productivity Center (TPC). Our experience indicates that the Galapagos modeling framework fits a variety of enterprise middleware systems.

The typical effort required for creating a DLT model, including introspection support, ranges from 2 days to 2 weeks, with additional time required for testing and documentation. Accessing management APIs during the discovery process requires credentials that can be acquired from interviews with IT infrastructure managers and application owners. For security reasons, these credentials offer limited privileges and may require periodic revalidation. In our experience so far, the process of acquiring credentials can benefit from streamlining the interview process and analysis of access requirements and their security implications.

Applications–data relationships in the distributed system may continuously change as a result of installation of new applications or middleware, creation of new data by existing applications, or dynamic modification to system configuration for workload management. The speed with which Galapagos adapts to such changes depends on whether its discovery process is restarted periodically or it is performed in response to notification of a change in the system (such as system configuration modification or new data created). The appropriate choice of method depends on the frequency of change.

We evaluated Galapagos on two system configurations: a cluster of four IBM xSeries* blades running WebSphere Network Deployment version 5.1 and DB2 version 8.2 (Enterprise Edition), and a cluster of two IBM pSeries* servers hosting two applications developed for the Spring J2EE framework using iBATIS to map data-access objects to tables in a DB2 database. Our scalability experiments confirmed that the running time of Galapagos is consistent with the behavior suggested by Equation (1); that is, the running time is proportional to the number of relations. We observed that the absolute running times are in the range of minutes, even for relatively large configurations. For example, in a configuration of 2,874 relations, the running time was 245 seconds. We also measured average CPU utilization on otherwise idle test systems to be less than 5% during crawling. We are currently scanning configurations with hundreds of applications and thousands of hosts. The feedback from administrators of these systems suggests that the overheads are well within tolerable limits.

Related work

Past studies on discovering dependencies between distributed systems tiers using online system monitoring of network traffic and statistical heuristics [1, 5, 17] are also applicable to discovering applications–data relationships. However, such systems generally have drawbacks. First, because they are based purely on heuristic rules, they cannot eliminate the possibility of missing some application–data relationships. Second, they are not easily generalizable to multi-tiered distributed systems. We believe, however, that a heuristic approach is useful (particularly when modeling information is not available) and complementary to the approach described in this paper.

Various systems have investigated building distributed system-dependency graphs using passive methods such as trace collection and offline analysis [1–5, 17] or active methods such as fault injection [26]. Some of the uses of a dependency graph include problem determination, performance analysis, and visualization. Galapagos differs from these approaches in that it specifically discovers the use of data by applications. Thus, it provides a finer-grain scope than dependency between software components. Systems that trace the provenance of data [27, 28] are also related to our work in that they establish a history of changes to data, and the history may include the applications that made the changes. However, distributed multi-tiered systems are beyond the scope of present provenance prototypes. Galapagos is a step toward that direction.

Conclusions and future work

In this paper, we presented a novel approach to model-driven discovery of application–data relationships in multi-tiered distributed systems. We showed that our models are sufficiently general to encompass a wide range of middleware systems and applications. At the same time, our models are reusable assets that are relatively easy to create for a variety of systems.

We described a distributed crawling algorithm that uses the model information to automatically construct applications–data relationships. For most real-life multi-tiered distributed systems, Galapagos has reasonable running times because of the limited number of middleware tiers and degrees of data-mapping fanout in such systems. On the other hand, multi-tiered systems may involve a large number of applications with many business objects. Fortunately, the performance of Galapagos depends linearly on these parameters.

Galapagos has important practical applications. Information collected by Galapagos allows analysis for information life-cycle management, accounting for storage use on a per-application basis, detailed reliability analysis of complex multi-tiered systems, storage consolidation, recovery planning, customer isolation in shared hosting environments, and improved accuracy of root-cause analysis in case of failure.

We are currently deploying Galapagos in an enterprise environment with hundreds of applications and thousands of servers. Our experience with this deployment will be used to create a reusable service offering with standardized methodology and delivery process. We are also working on automating the model creation process to a greater extent. We envision automated creation of models that are based on program source or binary code. We are experimenting with leveraging online activity monitoring tools for validation of discovered relationships. Eventually, we hope to see software developers producing DLT-like models as a side effect of code development to enable Galapagos-style automated dependency discovery and analysis.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Sun Microsystems, Inc., Silicon Graphics, Inc., The Open Group, Object Management Group, or Eclipse Foundation, Inc., in the United States, other countries, or both.

References

1. M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes," *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003, pp. 74–89.
2. A. Chanda, K. Elmeleegy, A. L. Cox, and W. Zwaenepoel, "Causeway: Support for Controlling and Analyzing the Execution of Multi-tier Applications," *Proceedings of Middleware 2005*, Vol. 3790, Springer, Berlin, 2005, pp. 42–59.
3. M. Y. Chen, E. Kıcıman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," *Proceedings of the International Conference on Dependable Systems and Networks*, June 2002, pp. 595–604.
4. K. R. Joshi, M. A. Hiltunen, W. H. Sanders, and R. D. Schlichting, "Automatic Model-Driven Recovery in Distributed Systems," *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, Orlando, FL, October 26–28, 2005, pp. 25–26.
5. H. Kashima, T. Tsumura, T. Ide, T. Nogayama, R. Hirade, H. Etoh, and T. Fukuda, "Network-Based Problem Detection for Distributed Systems," *Proceedings of the 21st International Conference on Data Engineering*, April 5–8, 2005, pp. 987–989.
6. R. J. Adair, R. U. Bayles, L. W. Comeau, and R. J. Creasy, "A Virtual Machine for the 360/40," IBM Corporation, Cambridge Scientific Center, Report 320-2007, May 1966.
7. Enterprise Volume Management System; see *http://evms.sourceforge.net/*.
8. W. de Jonge, M. F. Kaashoek, and W. C. Hsieh, "The Logical Disk: A New Approach to Improving File Systems," *ACM Operating Syst. Rev.* **27**, No. 5, 15–28 (1993).
9. J. S. Glider, C. F. Fuente, and W. J. Scales, "The Software Architecture of a SAN Storage Control System, *IBM Syst. J.* **42**, No. 2, 232–249 (2003).
10. M. Welsh and D. Culler, "Virtualization Considered Harmful: OS Design Directions for Well-Conditioned Services," *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, May 20–22, 2001, pp. 139–146.
11. Sun Microsystems, Inc., Java™ 2 Platform Enterprise Edition Specification, v1.4, 2003; see *http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf*.
12. ODBC Overview; see *http://msdn.microsoft.com/en-us/library/ms710220.aspx*.
13. Storage Networking Industry Association, Storage Management Initiative Specification (SMI-S); *http://www.snia.org/tech_activities/standards/curr_standards/smi/SMI-S_Technical_Position_v1.2.0r6.zip*.
14. Configuration Management Database (CMDB), Office of Government Commerce (OGC), ed.: Service Support. IT Infrastructure Library (ITIL). The Stationery Office, Norwich, UK (2000).
15. IBM Corporation, Tivoli Application Dependency Discovery Manager; see *http://www-306.ibm.com/software/tivoli/products/taddm*.
16. Hewlett-Packard Development Company, HP Discovery and Dependency Mapping Software; see *https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-15-25%5E767_4000_100__*.
17. A. Kind, D. Gantenbein, and H. Etoh, "Relationship Discovery with NetFlow to Enable Business-Driven IT Management," *The First IEEE/IFIP International Workshop on Business-Driven IT Management*, April 7, 2006, pp. 63–70.
18. V. Machiraju, M. Dekhil, K. Wurster, P. Garg, M. Griss, and J. Holland, "Towards Generic Application Auto-Discovery," *Network Operations and Management Symposium*, April 2000, pp. 75–87.
19. Object Management Group, OMG Model Driven Architecture; see *http://www.omg.org/mda/*.
20. B. Hailpern and P. Tarr, "Model-Driven Development: The Good, the Bad, and the Ugly," *IBM Syst. J.* **45**, No. 3, 451–561 (2006).

21. A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Information and Control in Gray-Box Systems," *Proceedings of Symposium on Operating Systems Principles*, 2001, pp. 43–56.
22. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*, Prentice-Hall, Upper Saddle River, NJ, 1984.
23. Distributed Management Task Force, Inc., Common Information Model (CIM) Standards; see *http://www.dmtf.org/standards/cim*.
24. W3C, Service Modeling Language; see *http://www.serviceml.org/*.
25. N. Joukov, A. M. Krishnakumar, C. Patti, A. Rai, S. Satnur, A. Traeger, and E. Zadok, "RAIF: Redundant Array of Independent Filesystems," *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, San Diego, CA, September 24–27, 2007, pp. 199–214.
26. A. Brown, G. Kar, and A. Keller, "An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Environment," *Proceedings of the Seventh IEEE/IFIP International Symposium on Integrated Network Management*, Seattle, WA, May 2001, pp. 377–390.
27. K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-Aware Storage Systems," *Proceedings of the 2006 USENIX Annual Technical Conference*, Boston, MA, June 2006, pp. 43–56.
28. P. Groth, M. Luck, and L. Moreau, "A Protocol for Recording Provenance in Service-Oriented Grids," *Lecture Notes in Computer Science*, Vol. 3544, Springer, Berlin, 2005, pp. 124–139.

Received September 15, 2007; accepted for publication October 18, 2007; Internet publication June 6, 2008

Kostas Magoutis *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (magoutis@us.ibm.com)*. Dr. Magoutis is a Research Staff Member in the Services Research Department at the IBM T. J. Watson Research Center. His research interests are in distributed systems, storage systems, and IT services. Recently, he has been working on modeling and management of distributed middleware systems, self-regulating, high-speed access to network storage systems, and improving the design and delivery of IT services. He holds a Ph.D. degree in computer science from Harvard University.

Murthy Devarakonda *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598*. Dr. Devarakonda is a Senior Manager and Research Staff Member in the Services Research department at the IBM T. J. Watson Research Center. He received his Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign in 1988. Presently, his research is focused on distributed file systems, Web technologies, storage and systems management, and now services computing. He received three IBM Research Division Awards for his work on distributed file systems and Global Technology Outlook development. Dr. Devarakonda is a Senior Member of the IEEE and the ACM.

Nikolai Joukov *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (njoukov@us.ibm.com)*. Dr. Joukov is a Research Staff Member in the Services Research department at the IBM T. J. Watson Research Center. He received an M.S. degree in physics from Moscow State University, Russia, in 2000, and M.S. and Ph.D. degrees in computer science from Stony Brook University in 2003 and 2006, respectively. He subsequently joined IBM at the Thomas J. Watson Research Center, where he is working on storage-related services problems.

Norbert G. Vogl *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (vogl@us.ibm.com)*. Mr. Vogl holds degrees in mathematics and computer science from Clarkson and Penn State Universities. He develops service and application prototypes, and his experience includes decision support for storage allocation, IT in the small and medium business sector, bulk file delivery via satellite communication systems, video and data transmission over residential broadband, and workflows of intraenterprise electronic commerce.