bmc

Technical White Paper

The Power behind BMC Atrium Discovery and Dependency Mapping

By **Dr. Duncan Grisby**
R&D Solutions Architect, Atrium Discovery and Dependency Mapping

Table of Contents

Executive Summary

BMC Atrium Discovery and Dependency Mapping (ADDM) is a data center discovery solution that automatically discovers IT assets and the relationships between them. It enables a wide range of use cases for IT service management, including asset management, software lifecycle analysis, and application modeling. The solution is delivered as a Linux®-based virtual appliance, which makes it exceptionally fast and easy to deploy.

ADDM has scalability, flexibility, and ease of use as its core tenets. From the beginning, the solution has been developed to enable users to quickly understand the state of their infrastructure and to identify the important relationships that exist within it—regardless of the size of their IT environment.

This document details some of the technology that is foundational to ensuring ADDM can meet its core tenets.

It is intended to help customers understand how and why technology choices were made, which ultimately led to a market-leading solution and a user base of raving fans. It discusses:

- **ADDM NoSQL Graph Database:** The custom-built, high-performance, extendable, graph-based database ADDM uses to store discovery data.
- **The Reasoning Engine:** ADDM's "secret sauce" for connecting IT infrastructure elements through relationship mapping.
- **Big Discovery:** ADDM's easy-to-use appliance clustering technology to achieve exceptional performance and near-limitless scale.

ADDM NOSQL GRAPH DATABASE

ADDM collects information with some unusual and challenging characteristics:

- The information ADDM collects from each element varies widely on a case-by-case basis, making it hard to define a schema for the data. For example, for many discovered software products, ADDM can retrieve product-specific configuration information such as important file paths, authentication options, and licensing configuration. The database must be able to store this diverse information, so other areas of the system can use it and users can manipulate it in reports.
- Connections between elements are at least as important as the elements themselves. In mathematical terms, the data is a "graph," and many data manipulations involve traversing across the graph structure. For example, it is common to wish to identify a dependency between two host computers by traversing from one host to an application server running on it, then to a related database that the application server is using, and then to the host on which the database is running.
- With data spread across a graph, and varying types of data stored for different elements, it is critical to be able to easily find required information. Users expect to be able to find pertinent information as easily as a web search, using just a few key words or phrases.

In recent years, the NoSQL movement has been advocating that a one-size-fits-all approach to data storage—where everything is stored in SQL-based relational databases—is limiting and damaging. ADDM recognized this issue early on. As a result, the first version, released in 2003, contained a first-generation graph database specifically tailored to the requirements of a discovery system, including:

- Natively stores a graph, consisting of 'nodes' connected by 'relationships', and has built-in support for efficiently traversing across the graph structure.
- Nodes and relationships store arbitrary data attributes, with no schema to restrict them.
- Database maintains a full-text index of all data, enabling simple and quick searches of everything within the graph.
- A search language inspired by SQL which allows the natural expression of queries and reports that make use of data within the graph.

It would, of course, be possible to provide these facilities on top of a traditional SQL database, or these days use one of the widely available NoSQL databases such as Cassandra, MongoDB, or Neo4j. However, by using its own database with features that exactly match its data storage requirements, ADDM obtains unique capabilities in terms of performance, scalability, and flexibility that could not be achieved with other technologies. The database implementation is now in its third generation and forms the robust basis for many of ADDM's most powerful features.

THE REASONING ENGINE

The "secret sauce" of ADDM is its Reasoning Engine. The Reasoning Engine is the intelligent part of ADDM, responsible for choosing and orchestrating the discovery actions and constructing the model of the environment within the graph database.

Early versions of the Reasoning Engine used a pipelined architecture, where processing moved through a fixed collection of stages like a flow chart. Each stage in the pipeline performed a particular task, and when processing completed, control moved on to the next stage in the pipeline. While flexible, that design had some significant limitations. For example, one pipeline stage was used to retrieve files from a discovery target, but because processing always flowed from one pipeline stage to the next, it was not possible to use the information found in one file to choose another file to retrieve.

The need for flexibility within the Reasoning Engine's actions led to the development of an Event-Condition-Action (ECA) engine. In an ECA engine, *events* representing discovery results or data changes are matched by *conditions*, leading to *actions* that request more discovery activities or manipulate data.

The whole Reasoning system is built on this principle of events that occur, conditions that match the events, and actions that need to be taken. For example, when the system performs a discovery request to obtain a list of processes from a server, the results come back as an event. A condition matches that event and leads to the action of creating nodes in the database to represent each discovered process. Creation of those nodes in turn leads to data events. A condition could exist that matches a process node being created with a particular command line, which causes an action to create a node to represent that running software, thus enabling ADDM to identify software running on a server. The way the ECA engine makes decisions is extremely flexible and enables the system to be extended to serve virtually any need. The flow of control simply becomes an emergent flow based on the events and data that flow through it.

While this type of rule-based system is not a novel approach, the implementation within ADDM does have some unique characteristics. Most systems of this kind support conditions that match combinations of events, meaning that two or more matching events must occur before the condition as a whole is triggered. ADDM's ECA engine only supports conditions that match single events. This limitation is intentional and very important. It allows the system to scale, both vertically within a single engine and horizontally across multiple engines. In a system that can match multiple combined events, the system must maintain the state of partially matched conditions for a potentially unbounded time. This causes resource constraints and limits the number of events that can be handled. By avoiding stateful event handling, ADDM's ECA engine is able to scale to process billions of events with minimal resource usage. Of course, there are many situations where previously handled state is essential for correct event processing; this is where the close coupling between the ECA engine and the graph database comes into play. As events are processed, the actions have extremely rapid access to all the information stored in the graph, and as a result, decisions involving multiple data items are managed efficiently despite the ECA engine itself being essentially stateless.

Although the ECA engine provides a flexible foundation for ADDM processing, it is generally hard to decompose problems into suitable collections of events, conditions, and actions. For this, ADDM employs The Pattern Language (TPL), a high-level, domain-specific programming language. TPL programmers write logic in a natural form, without regard for the semantics of the underlying ECA engine. Then the system automatically translates it into suitable collections of rules within the ECA engine. Since it is a programming language designed specifically for discovery purposes, TPL allows the author to focus on discovery, data manipulation, and data modeling tasks without the overhead and complexity of using an API from a general-purpose programming language, or the challenge of casting the tasks in the form of rules for the ECA engine.

TPL is used by ADDM's end users to perform customized discovery activity relating to their environment. It is also used in the monthly Technology Knowledge Updates (TKUs) that are provided by the ADDM engineering team. The value of TKUs is that every month, ADDM users enjoy extensions to ADDM that enable it to discover new hardware and software.

In summary, the Reasoning Engine is a scalable, general-purpose engine that allows ADDM to perform all the discovery tasks required by its users. Users can easily define what the system should do during discovery, how to manipulate the data that comes back from discovery, and how to represent the results within the graph database.

BIG DISCOVERY

Although there are plenty of situations in which the performance of an ADDM appliance is not a concern, scale and performance challenges can occur. Aside from the disk space required to store the database, there are no real limits to how large an environment a single ADDM appliance can scan. There are limits on how rapidly it can scan however. This becomes a challenge in four main scenarios:

1. In large environments with tens or hundreds of thousands of server computers, it can be difficult to scan the entire estate on a daily basis.
2. ADDM is sometimes given narrow time windows in which it is permitted to scan, so even in quite small environments it can be a challenge for an appliance to scan as rapidly as required.
3. In today's dynamic data centers, it is often important to be able to scan some parts of the environment several times per day.
4. To achieve maximum scanning performance, ADDM can saturate the appliance's resources, meaning that interactive query performance can suffer due to competition with scanning activities.

Prior to ADDM's Version 10 release, there were three main ways to handle these sorts of scale and performance challenges.

1. The first way was to give more powerful hardware to the ADDM appliance, which is, of course, limited in scope and quickly becomes wildly expensive for only modest performance gains.
2. The second method was to make use of ADDM's consolidation feature, whereby multiple scanning appliances feed data to a consolidation appliance for processing. This does not particularly improve the data processing speed on the consolidation appliance, but it does avoid bottlenecks due to limited scanning windows, since the consolidation appliance can continue to process data outside the permitted scanning times.
3. The third commonly used approach to scale was to simply segment the environment into a number of regions and scan them from separate, independent ADDM instances. That is, in a simplistic way, a form of horizontal scalability. However this approach greatly reduces ADDM's utility, because it cannot identify dependencies between the regions. It also creates management overhead to maintain multiple instances of ADDM separately.

Combinations of these three techniques have been quite successful, but with recent trends like Cloud and consumerization of IT, data centers are growing at a rapid pace and the systems deployed within them are becoming ever-more dynamic. It was clear that to keep up with modern IT requirements, a step change in ADDM's scale and performance capabilities was needed.

Big Discovery was introduced in ADDM Version 10 to provide a complete solution to these scale and performance challenges. Now, the user can take two or more ADDM appliances and simply join them together into a Big Discovery cluster. The system automatically spreads processing and data storage load across all the members of the cluster, allowing it to scale horizontally. As the cluster size increases, scanning and processing performance grows nearly linearly with the number of cluster members. To address the needs of interactive users of the system, scanning performance can in fact be turned down, making more of the cluster's resources available for real-time use.

In keeping with our core tenet that ease of use is paramount, building a Big Discovery cluster does not mean an increase in management overhead. Constructing the cluster involves nothing more than telling one ADDM appliance the addresses of the others that should join the cluster. Once the system is clustered it is operated and managed just like a single appliance. System configuration, knowledge updates, and even product upgrades are automatically synchronized across the cluster.

One risk with any system involving multiple computers is that the probability of hardware failure increases. As the number of machines in a cluster grows, so does the chance that any one of those machines will fail. Therefore, Big Discovery has an optional fault tolerance feature, allowing data to be automatically replicated across multiple machines in the cluster. If a machine in the cluster fails, all of the data is still available and the system can continue to operate without interruption. If the failed machine rejoins the cluster, the system automatically brings it up to date with activity that it missed. If the machine has a catastrophic failure, it can be replaced or permanently removed from the cluster, while the rest of the cluster continues to operate as normal.

A more subtle benefit to the Big Discovery approach is the potential to reduce ADDM hardware costs. Rather than investing in a single, top-of-the-range machine to run ADDM, it can instead run on a cluster of smaller, much cheaper machines. Typically, the most powerful servers are wildly more expensive than more modest ones. Users can avoid that inflection point where the price points jump and reap substantial savings on infrastructure.

The benefit of Big Discovery is the ability to scale an ADDM system to handle any scale and performance requirements, while achieving real cost savings. Setup is simple too—just a matter of grouping multiple machines into a cluster, which then acts as a single ADDM appliance.

CONCLUSION

In summary, Atrium Discovery and Dependency Mapping has been designed from its inception to be highly scalable, flexible, and easy to use. Developing foundational technologies like the ADDM NoSQL Graph Database, the Reasoning Engine, and Big Discovery came from real customer needs and use cases. These technologies enabled ADDM to establish its market leadership and continue to grow its thought leadership position today.

FOR MORE INFORMATION

To learn more about BMC Atrium Discovery, please visit **www.bmc.com/discovery**

ABOUT THE AUTHOR

Dr. Grisby was a co-founder of Tideway Systems in 2002. As Chief Software Architect, he was responsible for the design and implementation of Tideway's world-leading IT infrastructure discovery product. Tideway was acquired by BMC Software in 2009, where Duncan is now an R&D Solutions Architect with responsibility for discovery and other related products within the BMC portfolio.

451897