Hewlett Packard Enterprise

Data sheet

HPE DRAGON

A way to federate and expose heterogeneous data repositories

Providing single, uniform access to a set of disparate data repositories is a hard task, which becomes even more challenging if access to the data is restricted depending on privileges. HPE DRAGON provides a perfect solution for this task, while adding a flexible federation engine that opens up new ways to integrate data.

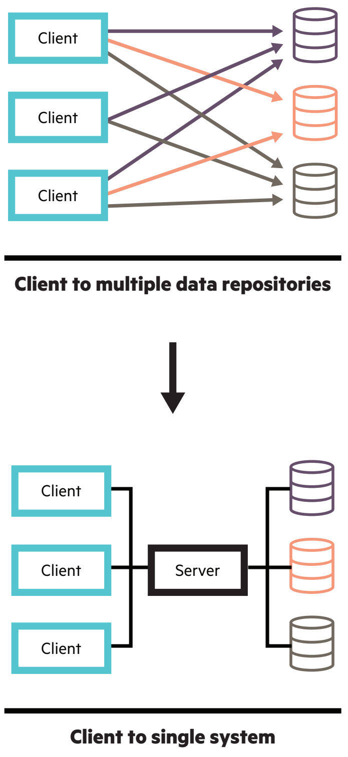

Figure 1. Going from unorganized data access on the top to organized and structured access on the bottom

Data access simplified

In these days, communications service providers (CSPs)—or Telco operators for short—are large companies that are deeply organized into departments: marketing, accounting, networking, and security, just to mention some of them. Due to several reasons, each one has its own data repositories, spanning from traditional databases to legacy and ad-hoc systems. Different repositories and communication protocols make it hard to exchange data: traditional databases are accessible through the SQL language, other systems through web services and legacy, and ad-hoc systems use proprietary protocols. Even within SQL, different dialects complicate any integration.

Data exchange between departments is mandatory. It is usually fulfilled by a large amount of dedicated scripts and software components, which are difficult to implement due to the complex data retrieval and integration logic, and data is scattered all over. Furthermore, with each script acting as a client, each has to implement all of the communication protocols required by all the different systems each connects to. This clearly leads to a huge amount of code duplication.

Clearly there is a need to simplify data access by all these clients. In the current situation, depicted above the arrow in Figure 1, every client connects to almost every repository. The ideal solution—depicted below the arrow in Figure 1—is a system in between the clients and the repositories that provides:

- A single and simple interface in which all clients can connect. This means one interface and one communication protocol for all data repositories.
- A uniform data abstraction layer to access all repositories in the same way.
- A standard query language that is widely known and applicable even to legacy and ad-hoc systems.
- A way to federate heterogeneous data repositories.
- A way to limit data access to only portions that a user can see.

Such a system addresses all of these issues and solves them in an elegant way.

Anatomy of a solution

In the Big Data ecosystem, Hewlett Packard Enterprise (HPE) DRAGON is a data retention solution with API management functionalities for data exposure. It fits well in the previous scenario, offering flexible ways to integrate and federate Big Data archives. In the context of data federation and exposure, its architecture can be depicted as in Figure 2. The different components include:

REST exposure interface

This layer provides a REST interface to clients that want to access data archives. A set of built-in services provides search capabilities and system maintenance functionalities; custom services can be added on demand to fulfill special requirements. A command layer takes care of exposing these services to clients, enforcing the proper authentication and authorization policies. Every service access is tracked by the audit component, which can write the log to a disk or different system, and is analyzed by a statistics component that updates some usage counters. Clients can easily connect to DRAGON using provided SDK libraries. This simplifies access to built-in services and new custom ones.

Real-time processing

Most service calls go through a real-time pipeline, which implements the full stack for querying data repositories in real time. Here, a streaming layer takes care of executing the search and returning data in a streaming mode to cope with Big Data needs. The execution can go through two different paths, depending on the configuration. SQL queries or queries written in a special query language—like Apache PIG—go through the standard engine, which is the most used case. Sometimes, when deeply structured or nested data has to be returned, an advanced XQuery engine can be used. Queries are written in the XQuery language, which provides the possibility to return nested structures, and can be combined together to provide different levels of views on the same data—virtual views. Results from a query are sent to clients using an output generator that takes care of formatting the data into the format required by the client, such as XML, JSON, or others.

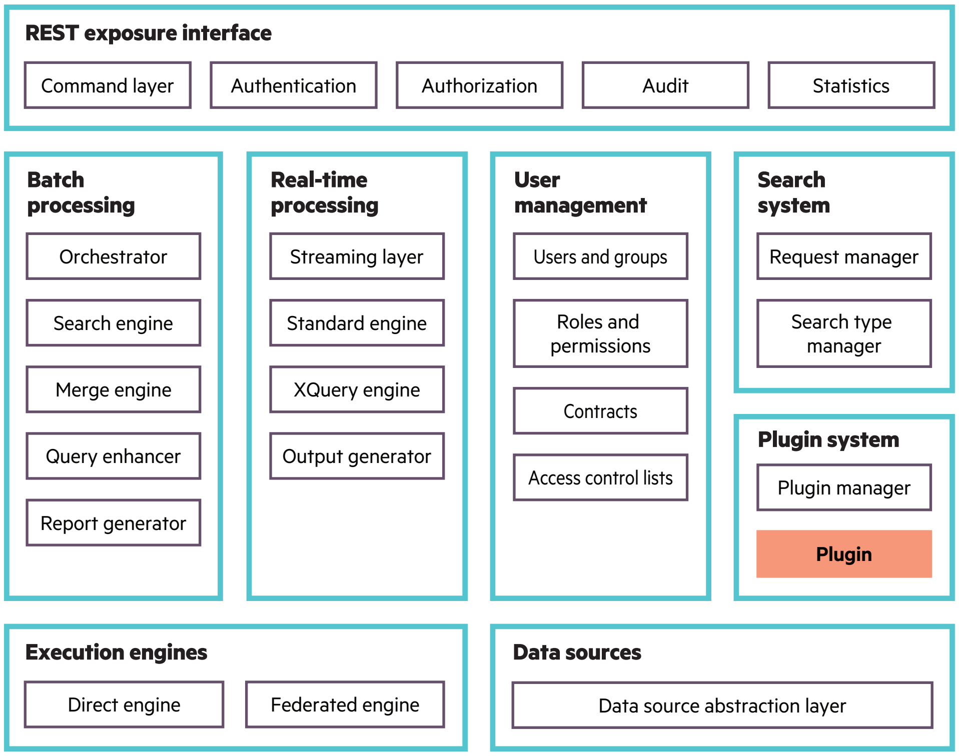

Figure 2. HPE DRAGON architecture diagram—server component

Batch processing

Batch processing of a search offers advanced features that are not available in real-time processing and is typically used when reports have to be generated from the resulting data. Here, queries are scheduled and executed depending on their priority. More queries can be executed in parallel, and results with different schemas can be returned and enhanced with extra information taken from other systems. Multiple ad-hoc reports can be generated using many different formats, such as PDF, XLS and XLSX, or CSV.

User management

Access to data is restricted, so each client must authenticate itself before requesting a service. During authentication, the proper privileges are checked to grant access to the requested service. On top of this, returned data can be restricted depending on its visibility with respect to the client. In DRAGON, user management is flexible: Users and groups can be combined in a hierarchical way, and the hierarchy obeys a fixed set of visibility rules. This means a restriction or enlargement on data—at a group level—is inherited by all groups and users below it. This hierarchy structure reaches the highest flexibility with the XQuery engine, where access control lists can be applied to prune sub-trees resulting from XQueries. Figure 3 shows a complex user hierarchy where, besides normal users, some administrators are at different levels. Dotted gray lines indicate which administrator created a specific object—group or user.

Search system

A search in DRAGON is characterized by a code, one or more queries to execute, and a set of parameters to use in the queries. This is called a search-type, and clients typically submit the code together with the parameters' values. This model provides a simple way to specify a search, hides the complexity of the queries themselves, and avoids any kind of misuse of the search service. This model also prevents all "SQL injection"-based attacks. Regarding user management, a client can access a search-type if it has the proper privilege together with a valid contract. A contract defines the validity of the permission, which can be based on time—temporal validity—or another metric, like size of the returned data, number of service calls, among others. This feature enables shaping the service calls to implement the required accounting policies for the enterprise's departments.

Execution engines

Once the system has to execute some queries, there are two types of engines that can be used. The Direct engine, which is a sort of pass-through engine, allows the use of any kind of query language. This way, it is possible to exploit any kind of native functionality of the underlying archive. On the other side, the federated engine works only with a subset of the ANSI SQL-92 language, but allows it to spread a query over any kind of archive. Tables in the FROM clause of a query can belong to different archives. The engine breaks each query into fragments and sends

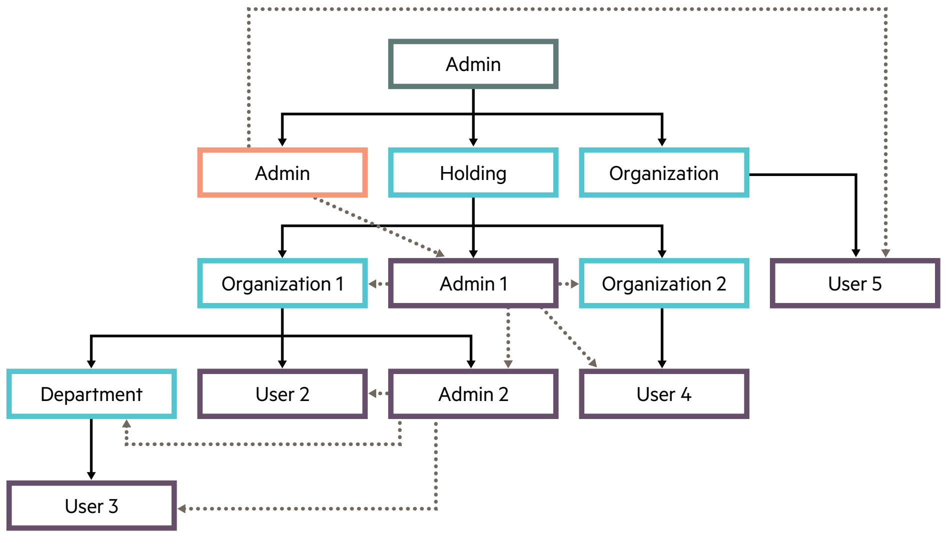

Figure 3. Example of a complex hierarchy of users and groups handled by the system

as many fragments as possible to each archive to exploit a local query engine. After this, the federated engine takes care of any operation that must be executed on top of these fragments, like joins, grouping, aggregation functions, and more.

Data sources

In DRAGON, each archive is accessed through a data source connector, which takes care of the low-level details of the archive and executes native and federated queries. Due to the disparate nature and architecture of the involved systems, the connector also has the duty of adapting the resulting data to match the internal DRAGON representation. This is mandatory in order to use a common processing pipeline. A data source abstraction layer is needed to provide a common framework for connectors and a single interface to query every type of archive. Currently, DRAGON ships with the connector for Oracle, MySQL, Vertica, and PostgreSQL, and on the Hadoop ecosystem for Hive, PIG, Drill, and Impala.

Plugin system

The DRAGON system is highly modular and built around a big set of plugins. Almost all aspects of the system are handled by a plugin: services, authentication and authorization mechanisms, connectors, and all real-time and batch pipeline components. Plugins can be changed at will, and ad-hoc ones can be developed for custom needs.

Table 1: List of archive systems supported by DRAGON

Archive	Type	Environment
Oracle	Commercial	
MySQL	Commercial / open source	
Vertica	Commercial	
PostgreSQL	Open source	
Apache PIG	Open source	Hadoop
Apache Hive	Open source	Hadoop
Apache Drill	Open source	Hadoop
Cloudera Impala	Open source	Hadoop

Security

This aspect is not shown in the diagram of Figure 2 because security is a pervasive aspect on all product components. On the frontend, the HTTPS protocol is used during the communication between clients and DRAGON. Internally, the software is robust against different types of attacks—like SQL injection. On the backend, encryption can be used to hide sensitive data on the database side to make it inaccessible, even to database administrators.

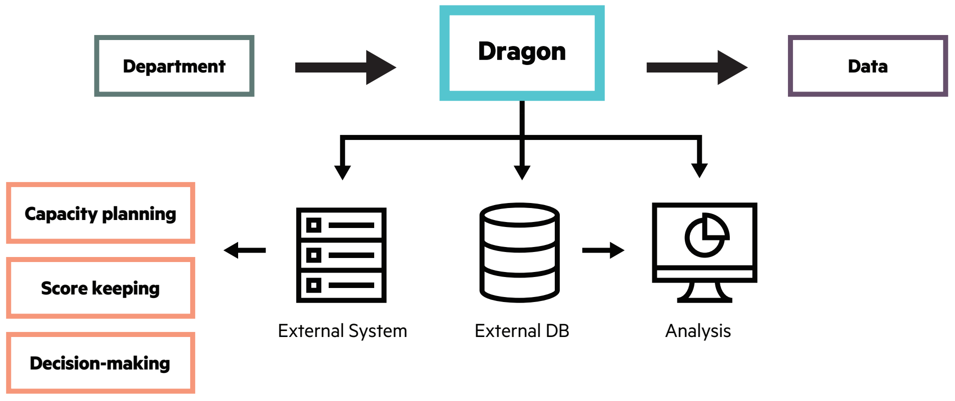

Figure 4. Accounting and capacity planning scenario

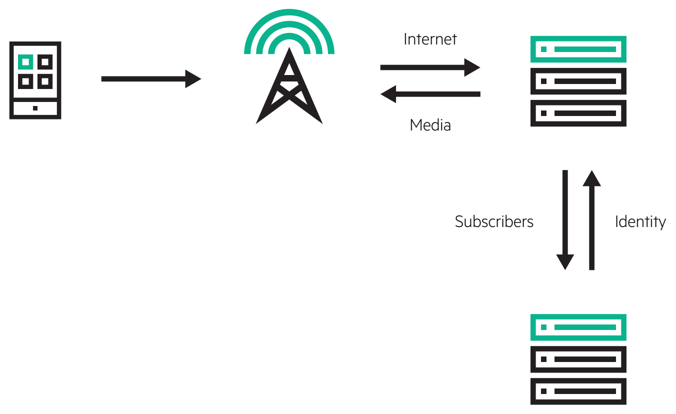

Figure 5. Data monetization use case—personalized advertising

Typical use cases

Use cases can be classified into two categories: ones that belong to API management on the frontend and ones that belong to data integration on the backend. Due to system flexibility, each one in the first group can be combined with each one in the second.

API management use cases

- **Accounting and capacity planning**—In this scenario, shown in Figure 4, DRAGON is used to provide connectivity to repositories and accounting facilities. Each CSP department uses a specific user—or set of users—to access services, and a proper access policy can be put in place to restrict access to data. During normal activities, each service call, made by a department's script, is audited to keep track of resource usage: All client and service processing information is available to enable proper analysis. The accounting information can be written to a log file, DRAGON's internal database, an external database, or any external system with the proper plugin.
- **Data monetization**—CSPs are becoming aware they can extract valuable customer information from their archives using different kinds of analytic processing. The problem is monetizing this information while delivering it to third-party applications. As an example, consider a user's preferences on sports, news, movies, and so on. They could be identified looking at the user's traffic data and sold to organizations to provide, for example, personalized advertising. Different customers could subscribe to different contracts with the CSP, with each contract giving a specific set of rights on the data and system. So, the more a customer pays for a contract, the more rights it gets.

This use case is shown in Figure 5. A subscriber, using a mobile device, opens a web page on a generic customer site. The customer delivers the content to the subscriber, but also wants to deliver some advertising, too. After signing a contract with the CSP, using the subscriber's IP address, it connects to the CSP's environment and asks for the subscriber's preferences. Examining the resulting preferences, the customer can choose the best advertising to show.

Backend use cases

Direct queries

This use case, shown in Figure 6, goes in the direction of flexibility: Any kind of query written in any language can be specified. The query goes straight to the archive, using the archive's native functionalities. Case (A) shows the most common case: an SQL query that goes to a relational database system. Here, the connector exploits the JDBC driver associated to the database management system DBMS to execute the query. In case (B), a PIG query is issued into a Hadoop cluster. The connector uses the Apache PIG libraries to transform the query into map-reduce tasks that will be executed into the cluster. In case (C), there is no query at all: The data repository is a legacy system with a fixed query system that is exposed as a set of web services. The query does not make sense here, just the values of the parameters are important.

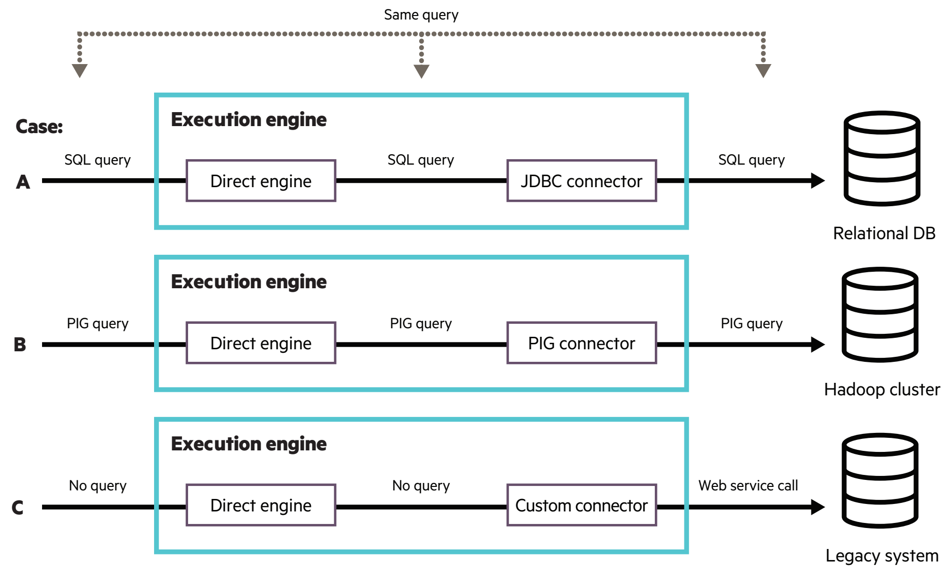

Figure 6. Direct query use cases—relational, nonrelational, generic

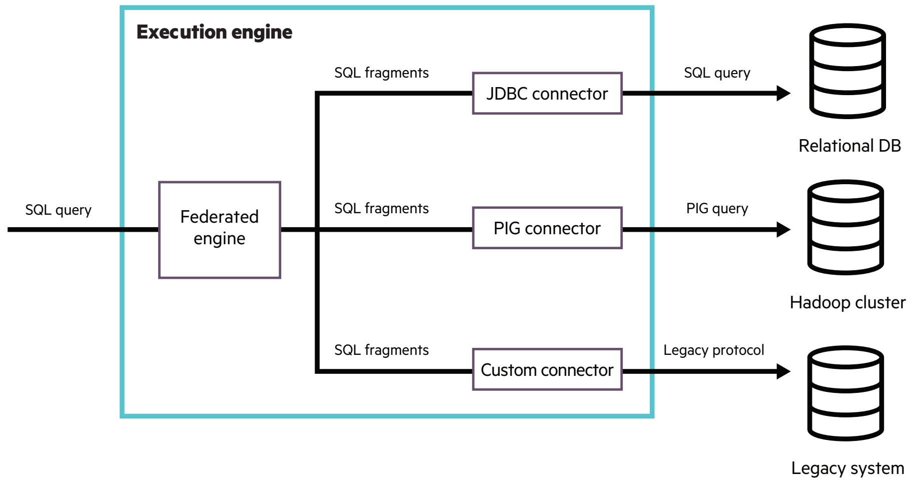

Figure 7. Federated query use case—query is spread over several archives implementing cross-archive join operations

Federated queries
Figure 7 depicts a typical federated query use case. Here, the query must be written using SQL language to be processed by the federated engine. The tables specified in the FROM clause must have a schema prefix that identifies the proper archive the table resides in. During processing, the engine converts the query into an abstract tree representation of SQL fragments, and during execution, it tries to send to each archive as many fragments as possible. The remaining fragments refer to cross archive operations—like joins, aggregation functions, and so on—and the engine takes care of implementing them.

Another less common use case is to provide an SQL interface to a system that is not SQL, because it uses a different query language or does not support a query language at all. This is depicted in the middle and at the bottom of Figure 7. In the first case, the query in SQL is converted to a query in another language, as is the case when querying Hadoop via PIG. In the second case, there is a legacy system with a legacy protocol, and there is no query at all. Here, the connector must translate each SQL fragment into a set of calls to the system using its protocol to implement the original query. In both cases, DRAGON connects to only one system, so there is no federation.

An emerging scenario, suited for a federated query, is depicted in Figure 8. A large Hadoop cluster is used to store many years of traffic data, like voice, GPRS, SMS, among others. Here, the most used DBMS to answer queries is Hive, because it scales well with many parallel queries, and it is not memory demanding compared to others. Another cluster, much smaller than the previous one, is used to store subscribers' information. A fast SQL engine, like Drill and Impala or, in some cases, HBase is used to access data while the HDFS replication factor is exploited to avoid periodic data backup. These two clusters must be combined in some way, because it is important to associate the subscriber's information to a specific traffic CDR. This is a perfect fit for a federated query; an example of this is shown in Figure 9.

In the query, traffic and subs represent the two clusters, and voice and subscriber are the tables that belong to the first and second cluster. The join condition is stated in the WHERE clause—v.subscriberID = s.id—using the table aliases. There are other conditions that belong to the first cluster (v.date) or the second (s.age). The $START_DATE, $END_DATE, and $AGE are the query's parameters that are specified when submitting the request. They contain the parameter's value (value) and the parameter's operator (operator) with that value (typically =, <, <=,

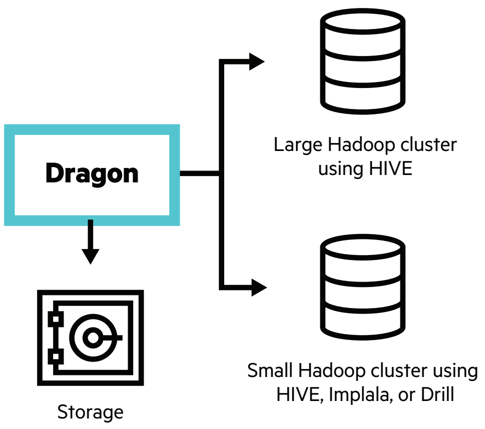
Dragon
Large Hadoop cluster using HIVE
Small Hadoop cluster using HIVE, Implala, or Drill
Storage

Figure 8. Typical example of a multicluster environment, where local storage is used to spool temporary data

```
SELECT v.date, v.details, s.id, s.name, s.age
FROM traffic.voice AS v, subs.subscriber AS s
WHERE
  v.subscriberID = s.id
  AND v.date $START_DATE.operator $START_DATE.value
  AND v.date $END_DATE.operator $END_DATE.value
#if ($AGE)
  AND s.age $AGE.operator $AGE.value
#end
```

Figure 9. Example of federated query based on the scenario in Figure 8

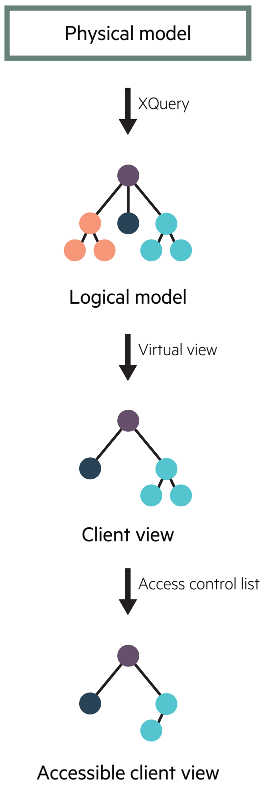

Figure 10. Physical to logical view transformation using XQueries

and so on). The #if and #end commands are used during the query preprocessing stage, to shape the query—in this case, to handle an optional parameter. This way, it is possible to completely change the query at runtime, adapting it to specific needs.

XQueries

This kind of query comes into play when the need is for real-time delivery of information that can be hierarchical, providing a different view for various users/clients—restricted, expanded, or just different, and can be driven by a contract, such as temporal validity, bandwidth throttling, and so on. The use of an XQuery decouples the physical model from the logical one, providing an abstract hierarchical representation of the data. A typical example is when returning a subscriber's information, which is better represented by a nested structure. This process is depicted in Figure 10, where a set of relational tables is transformed into a logical model, which is then shaped to fix a client view and pruned to remove restricted data.

The XQuery engine lies on top of the execution engines, so it is possible to create a standard XQuery that uses the direct engine, or a federated one that federates data from different archives.

DRAGON—driving your business

HPE DRAGON proposes a new solution to the integration and exposure of data repositories. Its flexibility addresses a wide range of API management and data retrieval use cases while its easy configurability contributes to a low total cost of ownership.

With more than 10 years of experience on the market, DRAGON provides a mature platform to drive your business. As a Hewlett Packard Enterprise product, you can rely on high-quality support and free rights to the new version.

About the author

Andrea Carboni is the chief architect and research and development manager of the HPE DRAGON product line. He is responsible for DRAGON product development and its evolution in the telecommunication industry. Carboni has a bachelor's degree in computer engineering, 20+ years of experience in software development, and 8+ years of experience in the telecommunication industry. He can be reached at **andrea.carboni@hpe.com**.

Learn more at
hpe.com/CSP/dragon

Sign up for updates

Hewlett Packard Enterprise

4AA6-5780ENW, June 2016