

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
9 March 2006 (09.03.2006)

PCT

(10) International Publication Number
WO 2006/026659 A2

(51) International Patent Classification:
G06F 15/16 (2006.01)

(21) International Application Number:
PCT/US2005/030953

(22) International Filing Date: 31 August 2005 (31.08.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/606,370 31 August 2004 (31.08.2004) US

(71) Applicant (for all designated States except US): **ASCENTIAL SOFTWARE CORPORATION** [US/US]; 50 Washington Street, Westborough, MA 01581 (US).

(72) Inventors; and

(75) Inventors/Applicants (for US only): **ARJUN, Vinodh** [—/US] (US). **BORDA, Hernando** [—/US] (US). **CHEREL, Thomas** [—/US] (US). **KADAYAM, Rajiv** [—/US] (US). **LE, Truong** [—/US] (US). **MAMOU, Jean-Claude** [FR/US]; 2 Bill Graham Lane, Millbury, MA 01527 (US). **SCHEFFLER, Lee J.** [US/US]; 77 Fairway Drive, West Newton, MA 02465 (US). **STILLER, Rick** [—/US] (US). **TAWIL, Christian** [—/US] (US).

TINNEL, Brian [—/US] (US). **ZHENG, Henry** [—/US] (US).

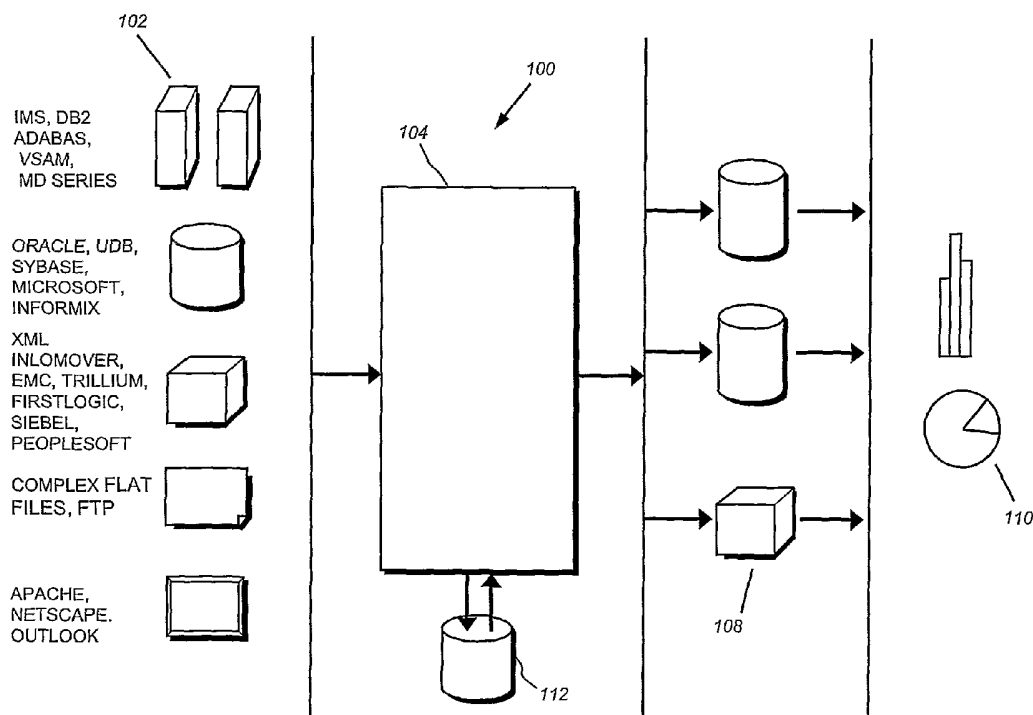
(74) Agent: **WANG, Elissa, Y.**; INTERNATIONAL BUSINESS MACHINES CORPORATION, INTELLECTUAL PROPERTY LAW, SVL, 555 Bailey Avenue, G4, San Jose, CA 95141 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: SERVICES ORIENTED ARCHITECTURE FOR DATA INTEGRATION SERVICES



(57) Abstract: Methods and systems are provided for deploying services, such as product services, real-time services and common services, in a services oriented architecture, including for enterprise data integration functions.



Published:

— without international search report and to be republished
upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SERVICES ORIENTED ARCHITECTURE FOR DATA INTEGRATION SERVICES

RELATED APPLICATIONS

This application claims priority to U.S. Provisional Application Number 60/606,370, filed on August 31, 2004 and entitled "Services Oriented Architecture for Data Integration Services."

BACKGROUND

1. Field.

This invention relates to the field of information technology, and more particularly to the field of data integration systems.

2. Description of the Related Art.

The advent of computer applications made many business processes much faster and more efficient; however, the proliferation of different computer applications that use different data structures, communication protocols, languages and platforms has led to great complexity in the information technology infrastructure of the typical business enterprise. Different business processes within the typical enterprise may use completely different computer applications, each computer application being developed and optimized for the particular business process, rather than for the enterprise as a whole. For example, a business may have a particular computer application for tracking accounts payable and a completely different one for keeping track of customer contacts. In fact, even the same business process may use more than one computer application, such as when an enterprise keeps a centralized customer contact database, but employees keep their own contact information, such as in a personal information manager.

While specialized computer applications offer the advantages of custom-tailored solutions, the proliferation leads to inefficiencies, such as repetitive entry and handling of the same data many times throughout the enterprise, or the failure of the enterprise to capitalize on data that is associated with one process when the enterprise executes another process that could benefit from that data. For example, if the accounts payable process is separated from the supply chain and ordering process, the enterprise may accept and fill orders from a customer whose credit history would have caused the enterprise to decline the order. Many other examples can be provided where an enterprise would benefit from consistent access to all of its data across varied computer applications.

A number of companies have recognized and addressed the need for sharing of data across different applications in the business enterprise. Thus, enterprise application integration, or EAI, has emerged as a message-based strategy for addressing data from disparate sources. As computer applications increase in complexity and number, EAI efforts encounter many challenges, ranging from the need to handle different protocols, the need to address ever-increasing volumes of data and numbers of transactions, and an ever-increasing appetite for faster integration of data. Various approaches to EAI have been taken, including least-common-denominator approaches, atomic approaches, and bridge-type approaches. However, EAI is based upon communication between individual applications. As a significant disadvantage, the complexity of EAI solutions grows geometrically in response to linear additions of platforms and applications.

While data integration systems provided useful tools for addressing the needs of an enterprise, such systems are typically deployed as custom solutions. They have a lengthy development cycle, and may require sophisticated technical training to accommodate changes in business structure and information requirements. There

remains a need for data integration methods and systems that permit use, reuse, and modification of functionality in a changing business environment. To facilitate such methods and systems, a need also exists for improved methods and systems for deploying data integration functions.

5 SUMMARY

Provided herein are methods and systems for data integration. The methods and systems may include providing one or more modules, tools, facilities, functions, services, processes or the like that perform a data integration function. The methods and systems may also include providing a registry of services that can be accessed by users, such as users identifying, designing, developing, deploying and using data integration jobs or
10 platforms. One or more of the modules, tools, facilities, functions, services, processes or the like can be provided with an input stage, and output stage, or both, such as a binding, that permits the data integration module, tool, function, service, or process to be accessed through the registry, such as for real-time or batch execution of the data integration functionality that is supported by the module, tool, facility, function, service or process. Thus, the module, tool, facility, function, service or process can be identified and used as a service in a services oriented
15 architecture. Among other things, by deploying the data integration platform modules, tools, facilities, functions, services and processes in a services oriented architecture, any one of them can be modified without impacting the performance of other related items.

A wide range of data integration modules, tools, facilities, functions, services and processes can be deployed as services in a services oriented architecture for a data integration job, method, platform or system. For
20 example, the foregoing may include an extraction function, a data transformation, a loading function, a metadata management function, a data profiling function, a mapping function, a data auditing function, a data quality function, a data cleansing function, a matching function, a probabilistic matching function, a metabroker function, a data migration function, an atomic data repository function, a semantic identification function, a filtering function, a refinement and selection function, a design interface function, or many others.

Thus, the methods and systems described herein include providing a module for a data extraction function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a data transformation function, providing a registry of services, providing an interface for the module, and identifying the module in the
30 registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a data loading function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a metadata management
35 function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a data profiling function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a data auditing function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

5 The methods and systems described herein also include providing a module for a data cleansing function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a data quality function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

10 The methods and systems described herein also include providing a module for a data matching function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the data matching function may be a probabilistic matching function.

15 The methods and systems described herein also include providing a module for a metabroker function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the metabroker function maintains the semantics of a data integration function across multiple data integration platforms.

20 The methods and systems described herein also include providing a module for a data migration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for an atomic data repository, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

25 The methods and systems described herein also include providing a module for a semantic identification function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

30 The methods and systems described herein also include providing a module for a filtering function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the filtering is based on a level of abstraction. In embodiments the level of abstraction can be at least one of a physical level of abstraction and a logical level of abstraction.

35 The methods and systems described herein also include providing a module for a refinement and selection facility, providing a registry of services, providing an interface for the facility, and identifying the facility in the registry, wherein the facility can be accessed as a service in a services oriented architecture. In embodiments the refinement and selection facility allows the system to distinguish between a logical level of abstraction and a physical level of abstraction.

The methods and systems described herein also include providing a module for analyzing the contents of a database, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for analyzing a table of a database, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

5 The methods and systems described herein also include providing a module for analyzing a row of a database, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for analyzing a data structure, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

10 The methods and systems described herein also include providing a module for recommending a target data facility, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for providing a primary key for a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

15 The methods and systems described herein also include providing a module for providing a foreign key for a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

20 The methods and systems described herein also include providing a module for providing a table normalization for a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

25 The methods and systems described herein also include providing source-to-target mapping for a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for automatically generating a data integration job from a profile for a data integration job, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

30 The methods and systems described herein also include providing a module for defect detection, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

35 The methods and systems described herein also include providing a module for measuring the performance of a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

40 The methods and systems described herein also include providing a module for data de-duplication, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the de-

duplication module matches data items based on a probability. In embodiments the de-duplication module discards duplicate items.

5 The methods and systems described herein also include providing a module for statistical analysis of a plurality of data items, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for reconciling data from a plurality of data facilities, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

10 The methods and systems described herein also include providing a module for accessing library of transformation functions, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for managing versions of a data integration job, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

15 The methods and systems described herein also include providing a module for managing versions of a data integration job, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the module allows a user to share a version with another user. In embodiments the module allows a user to check in and check out a version of a data integration job in order to use the data integration job.

20 The methods and systems described herein also include providing a module for parallel execution of a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for partitioning data, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

25 The methods and systems described herein also include providing a module for partitioning and repartitioning data, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

30 The methods and systems described herein also include providing a database interface module, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the interface module facilitates an interface to databases of a plurality of database vendors.

35 The methods and systems described herein also include providing a module for a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

40 The methods and systems described herein also include providing a module for synchronizing data, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the module facilitates synchronization of data across a plurality of hierarchical data formats. In embodiments the module facilitates synchronization of data across a plurality of transactional formats. In embodiments the module facilitates synchronization of data across a plurality of operating environments. In embodiments the module facilitates

synchronization of Electronic Data Interchange format data. In embodiments the module facilitates synchronization of HIPAA data. In embodiments the module facilitates synchronization of SWIFT format data.

The methods and systems described herein also include providing a module for supplying a metadata directory, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for graphical depiction of the impact of a change to a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for creating a metabroker, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a hub repository of metadata, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the hub stores semantic models for a plurality of data integration platforms.

The methods and systems described herein also include providing a packaged application connectivity kit (PACK), providing a registry of services, providing an interface for the PACK, and identifying the PACK in the registry, wherein the PACK can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for storing an industry-specific data model, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. The model may be a manufacturing industry model, a retail industry model, a telecommunications industry model, a healthcare industry model, a financial services industry model or a model from any other industry.

The methods and systems described herein also include providing template for building a data integration function, providing a registry of services, providing an interface for the template, and identifying the template in the registry, wherein the template can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for creating a business rule, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for creating a validation table, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for a data integration function, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for creating a business metric, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for defining a target database, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

5 The methods and systems described herein also include providing a module for profiling mainframe data, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for batch processing a batch of data, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

10 The methods and systems described herein also include providing a module for cross-table analysis, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for relationship analysis, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

15 The methods and systems described herein also include providing a module for generating data definition language (DDL) code, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the methods and systems may further include using the module to create a mapping between source and target data facilities.

20 The methods and systems described herein also include providing a design interface module for designing a data integration job, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

25 The methods and systems described herein also include providing a module for developing a data integration job, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

The methods and systems described herein also include providing a module for deploying a data integration job, providing a registry of services, providing an interface for the module, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

30 In embodiments the services in a services oriented architecture for a data integration platform or process may be services that are useful for a wide range of integration and computing tasks, including modules that perform functions that are required or beneficial for many common tasks.

A monitoring service may be deployed as a service with an input stage and/or output stage in a services oriented architecture. For example, the monitoring service may be invoked by a user to monitor some aspect of the performance of a data integration job or task, or to monitor an event or process. For example, a data integration system may have a service called a job execution service, the purpose of which is to run a job, such as a batch job. Using a monitoring service, a user can monitor how many times the job execution service has been run, how long it took to run, the minimum execution time, maximum execution time, average execution time and other statistics. The user can accomplish all of those functions without seeing the code of the underlying job execution service. The fact that all monitoring services are deployed as services means that inside the execution of the job a user can ask, for example, how many databases have been touched or other monitoring items that are specific to the semantics of

the job execution service. Thus, the job execution service can itself be a client of the monitoring service. Thus, through a monitoring service, the system can tell what is happening inside the implementation of another service. In embodiments, each common service, such as the monitoring service and the other services, various areas can be established for each service, such as what to monitor, the runtime of the service, and an administration part. To
5 invoke the monitoring service, the user may be queried as to what to monitor. Thus, the monitoring service can be used by services in a services oriented architecture to monitor what the services do or may be used to conduct domain-specific monitoring for other events and conditions.

A security module or service may be deployed as a service in a services oriented architecture for providing a security capability, such as in connection with a data integration job or task. When a user requires a security
10 facility, such as password protection, encryption, tracking access, restricting access, or the like, the user can invoke a security module as a service in a services oriented architecture, so that the user does not have to create a separate security facility for each data integration job or task.

A licensing module may be deployed in a services oriented architecture, for enabling licensing functions when invoked by a user. For example, a job designer may cause a data integration job to invoke the licensing
15 service to determine whether a particular task to be executed at runtime does or does not comply with license restrictions, such as license restrictions related to the number of machines, number of users, or the like. The user avoids the need to prepare separate licensing code for each data integration job or task the user creates.

An event management module may be deployed in a services oriented architecture for tracking and managing events when invoked by a user through a services registry. The user may access the event management
20 module for any event management required for a data integration job or task, such as tracking events in order to determine when to execute a process or function. The user avoids the need to create separate event management code for each different data integration task or job.

In embodiments a provisioning capability may be provided as a common service in a services oriented architecture.

A transaction module may be deployed in a services oriented architecture with an input stage and/or output stage that allows a user to access the transaction module through a services registry, avoiding the need to create
25 separate transaction management code for each application created by the user, such as for a data integration job or task.

An auditing module can be deployed in a services oriented architecture with an input stage and/or output stage that allows a user to access the auditing module through a services registry, avoiding the need to create
30 separate auditing code for each application created by the user, such as for a data integration job or task. Thus, by accessing the auditing module by invoking the service, the user can audit events, such as auditing what users have accessed a particular database or process, what events have taken place, and the like. An auditing module can allow a user to conveniently audit past events without having to generate separate code.

Thus, a wide variety of common tasks that are necessary or beneficial for data integration jobs or platforms
35 can be created as modules and deployed as services in a services oriented architecture.

In the various embodiments of modules and services that are described herein, techniques of aspect-oriented programming (AOP) can be used to implement services in a services oriented architecture. For example, various metadata functions and modules can be implemented as services with AOP. In embodiments, bindings for
40 services, such as EJBs (such as EJB 3.0) may use AOP. A policy may be associated with any of the foregoing services and/or modules using aspect oriented programming.

In services oriented architectures one attaches bindings that allow the service to be accessed, such as through ports. As described herein, various bindings, such as EJB, JMS, web services and JCA bindings can be used to invoke services in the various embodiments of services oriented architectures described herein. In embodiments, an application programming interface (API) may be provided for assisting access to a service. The API may be provide various functions, such as selecting a particular binding for a service, where the selection is based on a condition or event, such as selecting a binding that is appropriate for a particular application. For example, bindings may vary in their flexibility, and an API may apply a tight or loose binding based on the conditions of the application or device that accesses the service. In embodiments the API may be a Java API or similar facility. In embodiments the same Java API may be used for different kinds of bindings. In embodiments, a smart client may be supplied for a service. The smart client may be another layer on top of the API. The smart client may be stored and access through a registry associated with a service. For example, an application may download the appropriate smart client based on the device using the application, the context of the application, or the like. For example, a smart client may be used to buffer certain information that is used by a service and send the information to the service in a package, rather than having an application access the service constantly. For example, when accessing a logging service, a user may wish to log only errors, rather than all events. By holding events until predetermined time periods, the user can reduce the number of calls to the server while still capturing all of the necessary events. The smart client can thus execute various rules that optimize the use of a service by a device or application. In embodiments the smart client can select a binding, either alone or by interaction with an API, that optimizes the binding of the client-side device or application to the service based on the conditions of access, the capabilities of the device, the context of the access, or the like. The smart client or API can be used to store various access rules. For example, the rules might indicate that if a device or application is inside a firewall, then it can access a service using EJB bindings, while if the device or application is outside the firewall then it will access a service using a web service binding. Any such rules can be embodied in the API or may be included in a smart client, which may optionally be listed in a registry with the service and downloaded by a client device or application that will access the service.

One of the benefits of a services oriented architecture is that it facilitates loose coupling between a client device or application that accesses a service and the code for the service itself; that is, a client device or application can invoke and use the service without knowing very much about the code for the service, needing to satisfy only certain predetermined inputs, such as what to input to the service (e.g., a file, an answer to a query, or the like). However, the absence of a tight coupling can result in performance problems, as context-dependent optimizing routines are omitted from the service description in order to make it more generically useful. An API and/or smart client can make up for diminished performance by ensuring that a service is accessed optimally, such as by selecting a correct binding, caching data into batches, to avoid constantly invoking services for small jobs, or the like. Thus, a smart client provides effective performance in a loose coupling environment. The smart client thus bridges the gap between a tight coupling environment and a loose coupling environment and allows the user, application or device that accesses a service to choose a type of binding along the spectrum between loose coupling and tight coupling (such as EJB) according to the performance expectation or requirements. For example, EJB coupling may perform better than web services, because EJB couplings are by nature more tightly coupled between client applications and the server side. The smart client improves performance of both EJBs and web services by caching or buffering and sending things in appropriate batches. In situations where it is impossible or not desirable to cache or buffer items, a system can use a tight EJB binding to achieve good performance. In embodiments the

API may hide the binding that the client device or application is using. With a smart client, a user can tune the performance of the system by tuning the level of coupling between the client and the server.

In embodiments the runtime of a service in a services oriented architecture may be a client itself of another service, such one or more of the common services described above. In embodiments the foregoing can be accomplished using AOP. In AOP, entities known as interceptors can associate a policy to a service. Inside the policy of the service, interceptors can be plugged into the policies, and the interceptors can be clients of the common services. For example, a policy in a service can include an interceptor that invokes a monitoring service. Thus, AOP techniques can be used to insert code of interceptors into the code of various services described herein. In AOP, a user can create a piece of code and associate an "aspect" – a list of things to insert at runtime to the code as it is being executed. At that point in the code, the runtime program calls another piece of code, such as invoking a service, rather than doing what the code would normally do. At that point, the code calls another function that is compiled independently. Thus, when programmer looks at the source code for a runtime program, the programmer doesn't see the source code for the piece that is invoked by the interceptor. For example, in Java, the program can compile the source code to create the byte code, which is the runtime of Java, and a Java virtual machine reads the byte code. The program has the Java code and the aspect. The AOP compiler does byte code manipulation and calls other types of code, such as the services in the services oriented architecture.

The methods and systems described herein also include providing a framework for data integration including a module of code developed for accomplishing a data integration task and the deployment of code as a service that is organized in a services oriented architecture. The module of code may be a J2EE artifact.

The methods and systems described herein also include providing a framework for data integration including establishing a services oriented architecture and within the services oriented architecture, deploying common services that can be accessed by any other service, including the services oriented architecture itself. The common service may be a monitoring service, a transaction service, a provisioning service, an event management service, a security service, an auditing service and/or a logging service. The service may be invoked by an interceptor inserted in the code for at least one of an application and another service. The service may further comprise associating an extension with a common service, wherein the extension identifies the common service within the architecture for monitoring by at least one of a module, an application and another service within the architecture.

The methods and systems described herein also include providing a method of providing data integration services including establishing a services oriented architecture for deploying a service, wherein the service has more than one binding and automatically generating a binding for the service based on an access characteristic. The access characteristic may be based on at least one of a device, an application and/or a service that invokes the service.

Thus, the methods and systems described herein include using common services in various ways, such as explicitly from an application or from another service, or from an interceptor inserted in a service policy. That allows the same common service to be used by any service implementer and by the services oriented architecture framework transparently, such as through an AOP sub-system.

In various embodiments the modules, facilities, tools, jobs, services, processes and functions described herein may be accessed through various input and output facilities, including bindings and similar facilities, such as EJBs, JMS, web services, SOAP and other bindings.

In embodiments the methods and systems described herein may include a client-side facility for optimizing access of a module, facility, job, service, process, function or the like by a client device. In embodiments the methods and systems described herein may include a server-side facility for optimizing access of a module, facility, job, service, process, function or the like by a client device.

5 In another aspect, a method disclosed herein may include providing a services oriented architecture for deploying services; deploying a service within the services oriented architecture, the service having a number of bindings available in the services oriented architecture; and automatically selecting one of the number of bindings for the service. A computer program product embodied in a computer readable medium may include: computer executable code for providing a services oriented architecture for deploying services; computer executable code for
10 deploying a service within the services oriented architecture, the service having a number of bindings available in the services oriented architecture; and computer executable code for automatically selecting one of the number of bindings for the service. A system may include a services oriented architecture for deploying services; a service within the services oriented architecture, the service having a number of bindings available in the services oriented architecture; and a software module for automatically selecting one of the number of bindings for the service.

15 In the methods, systems, and computer program products above, automatically selecting may include making a rules-based selection of one of the number of bindings for the service, and/or may include making a selection based upon an access characteristic of an entity invoking the service. The entity may include one or more of a device, an application, and a service.

The methods, systems, and computer program products above may include providing a client invocation
20 framework for invoking the service. The client invocation framework may include an interface for dynamically calling any one of the number of bindings for the service. The methods, systems, and computer program products may include generating a plurality of proxies for the interface, which may include, for example at least one C++ proxy and at least one C# proxy. The client invocation framework may be language independent. The client invocation framework may be proxy-based. The service may include at least one data integration function.

25 In another aspect, the methods, systems, and computer program products described herein may include an Aspect Oriented Programming system for implementing policy management for services in a services oriented architecture wherein a policy manager manages services and binding policies. Aspect Oriented Programming interceptors may associate a policy to a service, and each interceptor can be a client of a common services service that provides policy management functions at runtime.

30 In other aspects, a computer program product may include a computer useable medium including computer readable program code, wherein the computer readable program code when executed on one or more computers causes the one or more computers to perform any one or more of the methods above.

As used herein, "data source" or "data target" are intended to have the broadest possible meaning consistent with these terms, and shall include a database, a plurality of databases, a repository information manager,
35 a queue, a message service, a repository, a data facility, a data storage facility, a data provider, a website, a server, a computer, a computer storage facility, a CD, a DVD, a mobile storage facility, a central storage facility, a hard disk, a multiple coordinating data storage facilities, RAM, ROM, flash memory, a memory card, a temporary memory facility, a permanent memory facility, magnetic tape, a locally connected computing facility, a remotely connected computing facility, a wireless facility, a wired facility, a mobile facility, a central facility, a web browser, a client,
40 laptop, a personal digital assistant ("PDA"), a telephone, a cellular phone, a mobile phone, an information platform, an analysis facility, a processing facility, a business enterprise system or other facility where data is handled or

other facility provided to store data or other information, as well as any files or file types for maintaining structured or unstructured data used in any of the above systems, or any streaming, messaged, event driven, or otherwise sourced data, and any combinations of the foregoing, unless a specific meaning is otherwise indicated or the context of the phrase requires otherwise. A storage mechanism is any logical or physical device, resource, or facility capable of acting as a data source or data target.

“Enterprise Java Bean (EJB)” shall include the server-side component architecture for the J2EE platform. EJBs support rapid and simplified development of distributed, transactional, secure and portable Java applications. EJBs support a container architecture that allows concurrent consumption of messages and provide support for distributed transactions, so that database updates, message processing, and connections to enterprise systems using the J2EE architecture can participate in the same transaction context.

“International Business Machines” or “IBM” as used herein shall refer to International Business Machines Corporation of Armonk, New York.

“JMS” shall mean the Java Message Service, which is an enterprise message service for the Java-based J2EE enterprise architecture. “JCA” shall mean the J2EE Connector Architecture of the J2EE platform described more particularly below. It should be appreciated that, while EJB, JMS, and JCA are commonly used software tools in contemporary distributed transaction environments, any platform, system, or architecture providing similar functionality may be employed with the data integration systems described herein.

“Real time” as used herein, shall include periods of time that approximate the duration of a business transaction or business and shall include processes or services that occur during a business operation or business process, as opposed to occurring off-line, such as in a nightly batch processing operation. Depending on the duration of the business process, real time might include seconds, fractions of seconds, minutes, hours, or even days.

“Business process,” “business logic” and “business transaction” as used herein, shall include any methods, service, operations, processes or transactions that can be performed by a business, including, without limitation, sales, marketing, fulfillment, inventory management, pricing, product design, professional services, financial services, administration, finance, underwriting, analysis, contracting, information technology services, data storage, data mining, delivery of information, routing of goods, scheduling, communications, investments, transactions, offerings, promotions, advertisements, offers, engineering, manufacturing, supply chain management, human resources management, data processing, data integration, work flow administration, software production, hardware production, development of new products, research, development, strategy functions, quality control and assurance, packaging, logistics, customer relationship management, handling rebates and returns, customer support, product maintenance, telemarketing, corporate communications, investor relations, and many others.

“Service oriented architecture (SOA)”, as used herein, shall include services that form part of the infrastructure of a business enterprise. In the SOA, services can become building blocks for application development and deployment, allowing rapid application development and avoiding redundant code. Each service may embody a set of business logic or business rules that can be bound to the surrounding environment, such as the source of the data inputs for the service or the targets for the data outputs of the service. Various instances of SOA are provided in the following description.

“Metadata,” as used herein, shall include data that brings context to the data being processed, data about the data, information pertaining to the context of related information, information pertaining to the origin of data, information pertaining to the location of data, information pertaining to the meaning of data, information pertaining

to the age of data, information pertaining to the heading of data, information pertaining to the units of data, information pertaining to the field of data and/or information pertaining to any other information relating to the context of the data.

“WSDL” or “Web Services Description Language” as used herein, includes an XML format for describing network services (often web services) as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

“Metabroker” as used herein, shall include systems or methods that may involve a translation engine or other means for performing translation operations or other operations on data or metadata. The translation operations or other operations may involve the translation of data or metadata from one or more formats, languages and/or data models to one or more formats, languages and/or data models.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a schematic diagram of a business enterprise with a plurality of business processes, each of which may include a plurality of different computer applications and data sources.

Fig. 2 is a schematic diagram showing data integration across a plurality of business processes of a business enterprise.

Fig. 3 is a schematic diagram showing an architecture for providing data integration for a plurality of data sources for a business enterprise.

Fig. 4 shows an item in relation to other items.

Fig. 5 shows an item in relation to other items.

Fig 6A shows an item in a certain context.

Fig 6B shows an item in a certain context.

Fig. 7 shows certain strings.

Fig. 8 shows an item and a corresponding string.

Fig. 9 shows a string and certain of its variations.

Fig. 10 shows a translation engine acting on certain strings.

Fig. 11 shows an item that may exist in multiple forms or instances.

Fig. 12 shows an item that may exist in multiple forms or instances in a hub or database.

Fig. 13 shows an item in a hub at various levels of abstraction.

Fig. 14 shows a translation process in which all items are grabbed at the database or hub.

Fig. 15A shows a translation process in which items are filtered at the database or hub.

Fig. 15B shows a translation process in which the query is translated.

Fig. 16A shows an overview of an architecture for a data integration system that includes a services oriented architecture facility.

Fig. 16B shows a high level schematic view of another similar architecture for a data integration system that includes a services oriented architecture.

Fig. 16C shows modules for enabling services in a services oriented architecture.

Fig. 16D shows additional modules for enabling services in a services oriented architecture.

Fig. 16E shows a services oriented architecture with a smart client.

Fig. 16F shows a particular embodiment of a services oriented architecture.

Fig. 16G shows the development and deployment of a module, service and/or facility as services in a services oriented architecture.

5 Fig. 17 shows the deployment of a module as a service in a services oriented architecture.

Fig. 18 shows the development and deployment of a data transformation module as a service in a services oriented architecture.

Fig. 19 shows the development and deployment of a data loading module as a service in a services oriented architecture.

10 Fig. 20 shows the development and deployment of a metadata management module as a service in a services oriented architecture.

Fig. 21 shows the development and deployment of a data profiling module as a service in a services oriented architecture.

15 Fig. 22 shows the development and deployment of a data auditing module as a service in a services oriented architecture.

Fig. 23 shows the development and deployment of a data cleansing module as a service in a services oriented architecture.

Fig. 24 shows the development and deployment of a data quality module as a service in a services oriented architecture.

20 Fig. 25 shows the development and deployment of a data matching module as a service in a services oriented architecture.

Fig. 26 shows the development and deployment of a metabroker module as a service in a services oriented architecture.

25 Fig. 27 shows the development and deployment of a data migration module as a service in a services oriented architecture.

Fig. 28 shows the development and deployment of an atomic data repository module as a service in a services oriented architecture.

Fig. 29 shows the development and deployment of a semantic identification module as a service in a services oriented architecture.

30 Fig. 30 shows the development and deployment of a filtering module as a service in a services oriented architecture.

Fig. 31 shows the development and deployment of a refinement and selection module as a service in a services oriented architecture.

35 Fig. 32 shows the development and deployment of a database content analysis module as a service in a services oriented architecture.

Fig. 33 shows the development and deployment of a database table analysis module as a service in a services oriented architecture.

Fig. 34 shows the development and deployment of a database row analysis module as a service in a services oriented architecture.

40 Fig. 35 shows the development and deployment of a database structure analysis module as a service in a services oriented architecture.

Fig. 36 shows the development and deployment of a recommendation module as a service in a services oriented architecture.

Fig. 37 shows the development and deployment of a primary key module as a service in a services oriented architecture.

5 Fig. 38 shows the development and deployment of a foreign key module as a service in a services oriented architecture.

Fig. 39 shows the development and deployment of a table normalization module as a service in a services oriented architecture.

10 Fig. 40 shows the development and deployment of a source-to-target mapping module as a service in a services oriented architecture.

Fig. 41 shows the development and deployment of an automatic data integration job generation module as a service in a services oriented architecture.

Fig. 42 shows the development and deployment of a defect detection module as a service in a services oriented architecture.

15 Fig. 43 shows the development and deployment of a performance measurement module as a service in a services oriented architecture.

Fig. 44 shows the development and deployment of a data de-duplication module as a service in a services oriented architecture.

20 Fig. 45 shows the development and deployment of a statistical analysis module as a service in a services oriented architecture.

Fig. 46 shows the development and deployment of a data reconciliation module as a service in a services oriented architecture.

Fig. 47 shows the development and deployment of a transformation function library module as a service in a services oriented architecture.

25 Fig. 48 shows the development and deployment of a version management module as a service in a services oriented architecture.

Fig. 49 shows the development and deployment of a version management module as a service in a services oriented architecture.

30 Fig. 50 shows the development and deployment of a parallel execution module as a service in a services oriented architecture.

Fig. 51 shows the development and deployment of a data partitioning module as a service in a services oriented architecture.

Fig. 52 shows the development and deployment of a partitioning and repartitioning module as a service in a services oriented architecture.

35 Fig. 53 shows the development and deployment of a database interface module as a service in a services oriented architecture.

Fig. 54 shows the development and deployment of a data integration module as a service in a services oriented architecture.

40 Fig. 55 shows the development and deployment of a synchronization module as a service in a services oriented architecture.

Fig. 56 shows the development and deployment of a metadata directory supply module as a service in a services oriented architecture.

Fig. 57 shows the development and deployment of a graphical depiction module as a service in a services oriented architecture.

5 Fig. 58 shows the development and deployment of a metabroker module as a service in a services oriented architecture.

Fig. 59 shows the development and deployment of a metadata hub repository module as a service in a services oriented architecture.

10 Fig. 60 shows the development and deployment of a packaged application connectivity kit module as a service in a services oriented architecture.

Fig. 61 shows the development and deployment of an industry-specific data model storage module as a service in a services oriented architecture.

Fig. 62 shows the development and deployment of a template module as a service in a services oriented architecture.

15 Fig. 63 shows the development and deployment of a business rule creation module as a service in a services oriented architecture.

Fig. 64 shows the development and deployment of a validation table creation module as a service in a services oriented architecture.

20 Fig. 65 shows the development and deployment of a data integration module as a service in a services oriented architecture.

Fig. 66 shows the development and deployment of a business metric creation module as a service in a services oriented architecture.

Fig. 67 shows the development and deployment of a target database definition module as a service in a services oriented architecture.

25 Fig. 68 shows the development and deployment of a mainframe data profiling module as a service in a services oriented architecture.

Fig. 69 shows the development and deployment of a batch processing module as a service in a services oriented architecture.

30 Fig. 70 shows the development and deployment of a cross-table analysis module as a service in a services oriented architecture.

Fig. 71 shows the development and deployment of a relationship analysis module as a service in a services oriented architecture.

Fig. 72 shows the development and deployment of a data definition language code generation module as a service in a services oriented architecture.

35 Fig. 73 shows the development and deployment of a design interface module as a service in a services oriented architecture.

Fig. 74 shows the development and deployment of a data integration job development module as a service in a services oriented architecture.

40 Fig. 75 shows the development and deployment of a data integration job deployment module as a service in a services oriented architecture.

Fig. 76 shows the development and deployment of a logging service module as a service in a services oriented architecture.

Fig. 77 shows the development and deployment of a monitoring service module as a service in a services oriented architecture.

5 Fig. 78 shows the development and deployment of a security module as a service in a services oriented architecture.

Fig. 79 shows the development and deployment of a licensing module as a service in a services oriented architecture.

10 Fig. 80 shows the development and deployment of an event management module as a service in a services oriented architecture.

Fig. 81 shows the development and deployment of a provisioning module as a service in a services oriented architecture.

Fig. 82 shows the development and deployment of a transaction module as a service in a services oriented architecture.

15 Fig. 83 shows the development and deployment of an auditing module as a service in a services oriented architecture.

Fig. 84 shows a service, API and smart client.

DETAILED DESCRIPTION

20 Throughout the following discussion, like element numerals are intended to refer to like elements, unless specifically indicated otherwise.

The invention(s) disclosed herein can take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment containing both hardware and software elements. In a preferred embodiment, the invention is implemented in software, which includes but is not limited to firmware, resident software, microcode, etc.

25 Furthermore, the invention(s) can take the form of a computer program product accessible from a computer-usable or computer-readable medium providing program code for use by or in connection with a computer or any instruction execution system. For the purposes of this description, a computer-usable or computer readable medium can be any apparatus that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device.

30 The medium can be an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system (or apparatus or device) or a propagation medium. Examples of a computer-readable medium include a semiconductor or solid state memory, magnetic tape, a removable computer diskette, a random access memory (RAM), a read-only memory (ROM), a rigid magnetic disk and an optical disk. Current examples of optical disks include compact disk – read only memory (CD-ROM), compact disk – read/write (CD-R/W) and DVD.

35 A data processing system suitable for storing and/or executing program code will include at least one processor coupled directly or indirectly to memory elements through a system bus. The memory elements can include local memory employed during actual execution of the program code, bulk storage, and cache memories which provide temporary storage of at least some program code in order to reduce the number of times code must be retrieved from bulk storage during execution.

40

Input/output or I/O devices (including but not limited to keyboards, displays, pointing devices, etc.) can be coupled to the system either directly or through intervening I/O controllers.

Network adapters may also be coupled to the system to enable the data processing system to become coupled to other data processing systems or remote printers or storage devices through intervening private or public networks. Modems, cable modem and Ethernet cards are just a few of the currently available types of network adapters.

Fig. 1 represents a platform 100 for facilitating integration of various data of a business enterprise. The platform includes a plurality of business processes, each of which may include a plurality of different computer applications and data sources. The platform may include several data sources 102, which may be data sources such as those described above. These data sources may include a wide variety of data types from a wide variety of physical locations. For example, the data source may include systems from providers such as Sybase, Microsoft, Informix, Oracle, Inlomover, EMC, Trillium, First Logic, Siebel, PeopleSoft, IBM, Apache, or Netscape. The data sources 102 may include systems using database products or standards such as IMS, DB2, ADABAS, VSAM, MD Series, UDB, XML, complex flat files, or FTP files. The data sources 102 may include files created or used by applications such as Microsoft Outlook, Microsoft Word, Microsoft Excel, Microsoft Access, as well as files in standard formats such as ASCII, CSV, GIF, TIF, PNG, PDF, and so forth. The data sources 102 may come from various locations or they may be centrally located. The data supplied from the data sources 102 may come in various forms and have different formats that may or may not be compatible with one another.

Data targets are discussed later in this description. In general, these data targets may be any of the data sources 102 noted above. This difference in nomenclature typically denotes whether a data system provides data or receives data in a data integration process. However, it should be appreciated that this distinction is not intended to convey any difference in capability between data sources and data targets (unless specifically stated otherwise), since in a conventional data integration system, data sources may receive data and data targets may provide data.

The platform illustrated in Fig.1 also includes a data integration system 104. The data integration system may, for example, facilitate the collection of data from the data sources 102 as the result of a query or retrieval command the data integration system 104 receives. The data integration system 104 may send commands to one or more of the data sources 102 such that the data source(s) provides data to the data integration system 104. Since the data received may be in multiple formats including varying metadata, the data integration system may reconfigure the received data such that it can be later combined for integrated processing. The functions that may be performed by the data integration system 104 are described in more detail below.

The platform 100 also includes several retrieval systems 108. The retrieval systems 108 may include databases or processing platforms used to further manipulate the data communicated from the data integration system 104. For example, the data integration system 104 may cleanse, combine, transform or otherwise manipulate the data it receives from the data sources 102 such that a retrieval system 108 can use the processed data to produce reports 110 useful to the business. The reports 110 may be used to report data associations, answer complex queries, answer simple queries, or form other reports useful to the business or user, and may include raw data, tables, charts, graphs, and any other representations of data from the retrieval systems 108.

The platform 100 may also include a database or data base management system 112. The database 112 may be used to store information temporally, temporarily, or for permanent or long-term storage. For example, the data integration system 104 may collect data from one or more data sources 102 and transform the data into forms that are compatible with one another or compatible to be combined with one another. Once the data is transformed,

the data integration system 104 may store the data in the database 112 in a decomposed form, combined form or other form for later retrieval.

Fig. 2 is a schematic diagram showing data integration across a plurality of entities and business processes of a business enterprise. In the illustrated embodiment, the data integration system 104 facilitates the information
5 flowing between user interface systems 202 and data sources 102. The data integration system 104 may receive queries from the interface systems 202, where the queries necessitate the extraction and possibly transformation of data residing in one or more of the data sources 102. The interface systems 202 may include any device or program for communicating with the data integration system 104, such as a web browser operating on a laptop or desktop computer, a cell phone, a personal digital assistant ("PDA"), a networked platform and devices attached thereto, or
10 any other device or system that might interface with the data integration system 104.

For example, a user may be operating a PDA and make a request for information to the data integration system 104 over a WiFi or Wireless Access Protocol/Wireless Markup Language ("WAP/WML") interface. The data integration system 104 may receive the request and generate any required queries to access information from a website or other data source 102 such as an FTP file site. The data from the data sources 102 may be extracted and
15 transformed into a format compatible with the requesting interface system 202 (a PDA in this example) and then communicated to the interface system 202 for user viewing and manipulation. In another embodiment, the data may have previously been extracted from the data sources and stored in a separate database 112, which may be a data warehouse or other data facility used by the data integration system 104. The data may have been stored in the database 112 in a transformed condition or in its original state. For example, the data may be stored in a
20 transformed condition such that the data from a number of data sources 102 can be combined in another transformation process. For example, a query from the PDA may be transmitted to the data integration system 104 and the data integration system 104 may extract the information from the database 112. Following the extraction, the data integration system 104 may transform the data into a combined format compatible with the PDA before transmission to the PDA.

Fig. 3 is a schematic diagram showing an architecture for providing data integration for a plurality of data sources 102 for a business enterprise. An embodiment of a data integration system 104 may include a discover data stage 302 to perform, possibly among other processes, extraction of data from a data source and analysis of column values and table structures for source data. A discover data stage 302 may also generate recommendations about table structure, relationships, and keys for a data target. More sophisticated profiling and auditing functions may
30 include date range validation, accuracy of computations, accuracy of if-then evaluations, and so forth. The discover data stage 302 may normalize data, such as by eliminating redundant dependencies and other anomalies in the source data. The discover data stage 302 may provide additional functions, such as drill down to exceptions within a data source 102 for further analysis, or enabling direct profiling of mainframe data. A non-limiting example of a commercial embodiment of a discover data stage 302 may be found in IBM's WebSphere ProfileStage product.

The data integration system 104 may also include a data preparation stage 304 where the data is prepared, standardized, matched, or otherwise manipulated to produce quality data to be later transformed. The data preparation stage 304 may perform generic data quality functions, such as reconciling inconsistencies or checking for correct matches (including one-to-one matches, one-to-many matches, and deduplication) within data. The data preparation stage 304 may also provide specific data enhancement functions. For example, the data preparation
35 stage 304 may ensure that addresses conform to multinational postal references for improved international communication. The data preparation stage 304 may conform location data to multinational geocoding standards

for spatial information management. The data preparation stage may modify or add to addresses to ensure that address information qualifies for U.S. Postal Service mail rate discounts under Government Certified U.S. Address Correction. Similar analysis and data revision may be provided for Canadian and Australian postal systems, which provide discount rates for properly addressed mail. A non-limiting example of a commercial embodiment of a data preparation stage 304 may be found in IBM's WebSphere QualityStage product.

The data integration system may also include a data transformation stage 308 to transform, enrich and deliver transformed data. The data transformation stage 308 may perform transitional services such as reorganization and reformatting of data, and perform calculations based on business rules and algorithms of the system user. The data transformation stage 308 may also organize target data into subsets known as datamarts or cubes for more highly tuned processing of data in certain analytical contexts. The data transformation stage 308 may employ bridges, translators, or other interfaces (as discussed generally below) to span various software and hardware architectures of various data sources and data targets used by the data integration system 104. The data transformation stage 308 may include a graphical user interface, a command line interface, or some combination of these, to design data integration jobs across the platform 100. A non-limiting example of a commercial embodiment of a data transformation stage 308 may be found in IBM's WebSphere DataStage product.

The stages 302, 304, 308 of the data integration system 104 may be executed using a parallel execution system 310 or in a serial or combination manner to optimize the performance of the system 104.

The data integration system 104 may also include a metadata management system 312 for managing metadata associated with data sources 102. In general, the metadata management system 312 may provide for interchange, integration, management, and analysis of metadata across all of the tools in a data integration environment. For example, a metadata management system 312 may provide common, universally accessible views of data in disparate sources, such as IBM WebSphere's ODBC MetaBroker, CA ERwin, IBM WebSphere ProfileStage, IBM WebSphere DataStage, IBM WebSphere QualityStage, IBM DB2 Cube Views, and Cognos Impromptu. The metadata management system 312 may also provide analysis tools for data lineage and impact analysis for changes to data structures. The metadata management system 312 may further be used to prepare a business data glossary of data definitions, algorithms, and business contexts for data within the data integration system 104, which glossary may be published for use throughout an enterprise. A non-limiting example of a commercial embodiment of a metadata management system 312 may be found in IBM's WebSphere MetaStage product.

Referring to Fig. 4, items that are relevant to an enterprise can be described in terms of various contexts or hierarchies, such as to capture the semantic context of the items. Thus, Fig. 4 depicts a semantic identifier for an item. The item may be an object, class, attribute, data item, data model, metadata model, model, definition, identity, structure, language, mapping, relationship, instance or other item or concept, including another semantic identifier. The semantic identifier may identify the item based on the item's attributes, the item's physical location, the relationship of the item with one or more other items, such as in a hierarchy, or the like. In some cases a relationship may be defined as the absence of some particular relationship. A relationship may be based on semantics. A relationship may involve the position of the item in a relational hierarchy. For example, in Fig. 4 item 1 5202 may be identified based on its relationship with the other items to which it is related. Item 1 5202 may be identified as being directly related to item 2 5204, item 3 5208 and item 4 5210, indirectly related to item 5 5212 and indirectly related to item 6 5214 through item 5 5212 and item 4 5210. Item 1 may also be identified as being directly related to item 2 5204, item 3 5208 and item 4 5210. In embodiments, the indirect relationships between

item 1 5202 and item 5 5212 and item 6 5214 may be captured in the relationship of item 5202 1 to item 4 5210. This concatenation or recursive type of identification may permit dynamic, in addition to static, identifiers. For example, if the relationship between item 4 5210 and item 6 5214 changes, the semantic identifier for item 1 5202 which incorporates item 2 5204, item 3 5208 and item 4 5210 would incorporate this change through incorporation of item 4 4210 and would not need to be updated to account for the changes in item 6 5214 as it would if item 6 5214 was directly included in the semantic identifier.

Figure 5 presents a more concrete example of a semantic identifier. Jim may be identified as Jim, residing at 111 Anyroad, Anytown, Anystate USA, with phone number 555-555-5555 and social security number 013-65-8067. Alternatively, Jim may be identified in terms of his relationships with others. As depicted in Figure 5, Jim may be identified as the son of Betty, brother of Larry and Jeff, father of Jessica and nephew of Frank.

The semantic identifier may be a unique identifier for an item. In the example of Figure 5, if there were only one Jim in the world who was the son of Betty, brother of Larry and Jeff, father of Jessica and nephew of Frank, this semantic identifier would be a unique identifier for Jim. It is possible that a unique semantic identifier to an item takes into account fewer than all of the relationships of that item with other items. In the example of Figure 5, if there were only one Jim in the world who was the son of Betty, brother of Larry and father of Jessica, the existence of these relationships alone would be enough to create a unique semantic identifier. Jim's relationships with Jeff and Frank would not need to be considered. It may be advantageous to create a semantic identifier that is based on the minimum number of relationships that ensure uniqueness. For example, if the semantic identifier was to be stored in a database 112 or processed by a data integration system 104, a less complex semantic identifier would require less space and would allow for faster processing.

The number of relationships required to create a unique semantic identifier for an item may vary based on context. Figure 6A depicts two items of interest: item 1 5402 and item 7 5404. In context A 5408, item 1 5402 may be distinguished from item 7 5404 by item 1's 5402 relationship with item 5 5410 and item 6 5412. That is, in context A, the unique semantic identifier for item 1 5402 may be that it is directly related to items 2, 3 and 4, indirectly related to item 5 5410 through item 4 and indirectly related to item 6 5412 through item 5 5410 and item 4. In context A, the unique semantic identifier for item 7 5404 may be that it is directly related to only items 2 and 3. Figure 6B presents item 1 5402 in a different context, context B 5414. To uniquely identify item 1 5402 in context B 5414 any one or more of item 1's 5402 direct relationships with item 4, absence of a direct relationship with item 6 or indirect relationship with item 5 may be taken into account. In context B 5414 item 1 5402 may be uniquely semantically identified as directly related to items 2 and 3, but not directly related to item 6. Thus, the unique identifier for item 1 differs between context A 5408 and context B 5414. Thus, in embodiments of the data integration methods and systems described herein, a semantic identifier for an item, such as an item related to a data integration job or a data integration platform, may be provided with a context-dependent identifier for the item. In embodiments such a context-dependent identifier may be stored in an atomic format, such as in a data repository.

In other embodiments, contexts A 5408 and B 5414 may be two different imports, mappings, run versions, models, metabroker models, instances, tools, views, objects, classes, items, relationships, attributes, or any combination of any of the foregoing. A matching or comparison facility may compare the syntax of the identity of an item in different imports, run versions, models, metabroker models, instances, tools and/or items and determine or assist with the determination of what action to take or refrain from taking based on the comparison. For example, a matching engine may compare the model used by import instance A to the model used by metabroker B. Based on this comparison it may be decided that metabroker B can access the data and metadata of import instance

A without transformation or modification, and the comparison facility may direct the metabroker B to proceed. In another example, tool A 5408 may be compared to tool B 544, and it may be determined to perform a cross-tool object merge, wherein each tool can access and use the objects of the other tool. In embodiments the comparison facility may trigger a translation facility to assist the cross-tool object merge, such as establishing a bridge, metabroker, hub or the like for translating any objects that require translation, such as translation that is based on the different syntax for the handling of the identity of particular items in each respective tool, or based on other differences between the tools as determined by the comparison.

In embodiments a semantic identifier may be stored, maintained, recorded, processed and/or interpreted in a syntax that may be stored, maintained, recorded, processed and/or interpreted in a string structure or format.

Figure 7 depicts an example of a syntax and a corresponding string composed in that syntax. The syntax 5502 may be column name::table name::database name. This syntax may be related, for example, to a semantic identifier that identifies a column of a table in a database. A string composed in this syntax 5504 may be age::employee::employee database. This string may be related, for example, to a semantic identifier that identifies the age of an employee in a particular employee database. In the example of Figure 6B, the string corresponding to the semantic identifier for item 1 5402 in context B 5414 may be: direct relation to item 2::direct relation to item 3::direction relationship to item 4. The semantic identifier and corresponding string may also incorporate the lack of a direct relationship between items 1 5402 and item 6.

In Figure 8 the semantic identifier in string format for item 9 5602 may be: direct to item 2::direct to item 3::direct to item 4::indirect to item 5 5604. A string may be capable of being parsed. A syntax and/or string may be truncated, modified and/or the elements of a syntax and/or string may be re-ordered. In Figure 9 string 5702 is a truncation of string 5604, string 5704 is a truncation and modification and/or re-ordering of string 5604 and string 5708 is a modification and/or re-ordering of string 5606. The truncation, modification and/or re-ordering may be performed by a translation engine. It may be useful to truncate a syntax and/or string when all of the relationships included in the syntax and/or string are not required for the uniqueness of the semantic identifier. Suppose that in a given context for string 5604 all items were directly related to item 3; for example, item 3 was a database in which all the items were stored. String 5604 could be truncated, such as to create string 5702, omitting the relationship-involving item 3, and still remain a unique semantic identifier. Truncating a syntax and/or string may reduce storage requirements and increase processing efficiency. It may also be useful to change the order of the relationships in a syntax and/or string, for example, to reduce processing time for data integration processes. If the less common relationships are processed first, a system will likely need to access and process fewer relationships associated with an item in order to identify the item. For example, if very few items were related to item 3, even fewer related to item 4 and many items related to item 2, depending on the context, string 5708 may allow for the identification of item 9 in a shorter time than string 5604. It could be that only the first two elements of string 5708 are needed to uniquely identify item 9 in the context, while the first three elements of string 5604 are needed.

A translation engine may perform translation operations with respect to one or more semantic identifiers, databases 112, databases 112 including semantic identifiers, systems of information, systems of information including semantic identifiers or other items. Figure 10 depicts a translation engine 5802 acting on a semantic identifier embodied as a string 5804 and on a semantic identifier embodied as a string located in a database 5808. The translation operation may translate or otherwise modify the format, language and/or data model of a semantic identifier. A translation operation may involve a translation or mapping to or from one or more data tools, languages, formats and/or data models to or from at least one other data tool, language, format and/or data model.

For example, a translation operation may involve a translation or mapping to, from or between known data integration tools, such as WebSphere DataStage 7 from IBM, WebSphere QualityStage from IBM, Business Objects tools, IBM - DB2 Cube Views, UML 1.1, UML 1.3, ERStudio, IBM 's WebSphere ProfileStage, PowerDesigner (with added support for Packages and Extended Attributes) and/or MicroStrategy tools. A translation engine and/or translation operation may optionally be embodied in a metabroker. A translation operation may be performed, executed and/or conducted in batch, real-time and/or on a continuous basis. A translation operation may be provided or made available as a service, for example, as part of a service oriented architecture. The SOA can be part of the infrastructure of an enterprise computing system of a business enterprise. In the SOA, services become building blocks for application development and deployment, allowing rapid application development and avoiding redundant code. Each service embodies a set of business logic or business rules that can be blind to the surrounding environment, such as the source of the data inputs for the service or the targets for the data outputs of the service. As a result, services can be reused in connection with a variety of applications, provided that appropriate inputs and outputs are established between the service and the applications. The service-oriented architecture allows the service to be protected against environmental changes, so that the architecture functions even if the surrounding computer environment is changed. As a result, services may not need to be recoded as a result of infrastructure changes, which may result in savings of time and effort. An SOA may be for a web service and may involve three entities, a service provider, a service requester and a service registry. The registry may be public or private. The service requester may search a registry for an appropriate service. Once an appropriate service is discovered, the service requester may receive code, such as Web Services Description Language ("WSDL") code, that is necessary to invoke the service. WSDL is a programming language conventionally used to describe web services. The service requester may then interface with the service provider, such as through messages in appropriate formats (such as the Simple Object Access Protocol ("SOAP") format for web service messages), to invoke the service. The SOAP protocol is a preferred protocol for transferring data in web services. The SOAP protocol defines the exchange format for messages between a web services client and a web services server. The SOAP protocol uses an eXtensible Markup Language ("XML") schema, XML being a generic language specification commonly used in web services for tagging data, although other markup languages may be used.

Once a translation operation exists for a semantic identifier, database 112, database 112 including one or more semantic identifiers, system of information, system of information including one or more semantic identifiers or other item it can be translated to or from, mapped to, linked to, used with or associated with any other semantic identifier, database 112, database 112 including one or more semantic identifiers, system of information, system of information including one or more semantic identifiers or other item sharing at least one translation operation. In embodiments, such as using an atomic data repository as a hub for a translation operation, the mapping of a translation operation can, among other things, trace data that is translated in the execution of the operation backward and forward between an original semantic context and a translated semantic context. Depending on the context, the appropriate identifier for the data item may vary, such as by varying or truncating a syntax and/or string to enable more efficient storage or faster processing, or by varying the relationships used to form a unique identifier where the semantic context varies. Thus, a dynamic identifier may combine the benefits of retraceable translation with the benefits of rapid processing, efficient data processing and effective operation in various contexts in which a data item is used.

A given item, such as an item that has an identity in a model, may exist in multiple forms or instances, such as a physical instance and a logical modeling instance. Figure 11 depicts an item, namely, a table of employee information 5902. However the concept or entity "employees" can exist in a number of different forms within an enterprise. For example, the employee table 5902 may exist as a physical table that stores values related to employees in a physical data storage facility. On the other hand, the entity employee may also be represented as a logical entity, such as an icon or text that represents employees in a logical modeling activity 5908, or in various other forms or instances. That is, the same item, including any associated data or metadata, may exist in multiple forms or instances across views, models, structures or a data integration environment, such as in databases, data repositories, models, hubs, or the like. Figure 12 depicts the employee table 5902 in one form or a single instance in a database 6002 and/or more than one form or instance in a database 6004 or hub 6008.

In order to distinguish between the various forms or instances of an item, any differentiating characteristic may be used, such as a level of abstraction, a physical property of an item, a location of the item within a hierarchy, a location of an item in a database, a context in which an item is found, a syntax of an item, a relationship of an item to other items, an attribute of an item, the class of an item, or other characteristic. For example, referring back to Figure 5, the items, or individuals in this case, may be distinguished based on age, gender, hair color, IQ, political affiliation and/or number of trips to the doctor in the past three months. For example, if age was selected as the product differentiator, it may be the case that Jessica is the only individual under ten years old, Betty is the only individual between fifty-seven and sixty-seven years old and Jim is the only individual who is thirty-seven years old. In another example, different forms or instances of the item may exist at different levels of abstraction or in different contexts. For example, the employee table may exist in multiple forms or instances in the hub 6102, such as a physical employee table 5904, such as used to store values in a database that relate to data that pertains to employees, and a logical employee model 5908, such as to be used in a view of process that relates to employees..

Distinguishing between the different instances of a particular identified item can enable a variety of other methods and processes. For example, in one embodiment, an item, such as a table named "employee," may be brought into a hub. A hub collector may have two forms or instances of "employee" in the hub; one corresponding to the physical database instance and another corresponding to the logical modeling activity. A differentiating characteristic, such as a property of the item attributed to the item in the hub allows for the differentiation between the physical instances and the logical model instances or forms. In embodiments that differentiating characteristic can be called a level of abstraction, such as to distinguish between logical and physical levels of abstraction. In other cases the hub may associate other characteristics with items, such as different forms of identifiers, relationships, classes, attributes, physical locations, logical positions, models and the like.

As depicted in Figure 14, when performing an operation, such as selecting data to be loaded into a database, translating data, generating a query, or the like, a system, such as a translation engine 6204, may grab, load or obtain all of the items from a hub 6208 or database 6210. It may select or filter 6204 the items based on any differentiating characteristic. For example, it may select or filter out those instances or forms that have a physical level of abstraction, that have a particular relationship to other items, that have a logical level of abstraction, that are created prior to a specified date and time, or that have any other distinguishing characteristics.. Thus, the methods and systems described herein provide for selective handling of instances of the same item or entity based on any differentiating characteristic..

As depicted in Figure 15A, when performing a data integration operation, such as a translation operation, which may be in response to a query 6202, a translation engine 6204 may filter or select items, including any data

and/or metadata, at the hub 6208 or database 6210 and grab, load or obtain only those items of the relevant level of abstraction. For example, it may filter or select out those instances or forms with a logical level of abstraction, keeping only those with a physical level of abstraction. The filtering or selection may be performed at runtime or design time and may be conducted in batch, real-time or on a continuous basis. In embodiments such a method of
 5 filtering or selection may be provided as an RTI service in a services oriented architecture.

The filtering or selection may be based on information, such as a mapping of a data model, a mapping of a metadata model, a differentiating characteristic, a relationship of an item to another item, an attribute of an item, or the syntax of an identifier, that is obtained by the translation engine and/or system at development-time, design-time or run-time. In embodiments the information may be updated in a dynamic fashion in real-time.

10 The closer in the overall process the filtering or selection is to the hub or database the more efficient and faster the operation. As depicted in Figure 15B, the translation engine 6204 may perform a translation operation on the query 6202 itself, resulting in a revised query 6302, which may be sent for further processing, such as directly to the hub 6208 or database 6210. For example, the revised query 6302 may be rendered in a format that is directly compatible with the native format of the hub 6208 or database 6210. For example, by rendering the query in the
 15 native format of the database 6210, the system may increase processing efficiency for the query. Similarly, the query 6302 may be filtered or a command such as a select command may be generated to keep a logical modeling entity rather than a physical entity, in which case the query 6302 may be rendered in a format suitable for a logical modeling activity (such as a graphical user interface), rather than for the database. Of course, not only queries but other messages and operations may be filtered according to level of abstraction, enabling the same entity to be
 20 tracked across the data integration platform and handled according to the suitable operating environment of a particular data integration activity.

The methods and systems described herein can be used to capture semantic contexts and to handle data integration tasks with respect to a wide range of items related to an enterprise, such as an object, data item, datum, column, row, table, database, instance, attribute, metadata, concept, topic, subject, semantic identifier, other
 25 identifier, RFID tag, vendor, supplier, customer, person, team, organization, user, network, system, device, family, store, product, product line, product feature, product specification, product attribute, price, cost, bill of materials, shipping data, tax data, course, educational program, location, map, division, organization, organism, process, rule, law, rating system, good, service and/or service offering.

The methods and systems described herein can be used in a variety of semantic contexts, such as a step in
 30 an enterprise method, a datum in a database, a datum in a row or column, a row or column in a table, a row or column in a database, a datum in a table, a table in a database, metadata in a database, an item in a hub or repository, an item in a database, an item in a table, an item in a column, an item in a row, a person in an organization, a sender or recipient of a communication, a user on a network, a system on a network, a device on a network, a person in a family, an item in a store, a dish on a menu, a product in a product line, a product in a
 35 product offering, a course or step in an educational or training program, a location on a map, a location of an item, a division of an organization, a person on a team, a rule in a system of rules, a service in a service suite, an entity in an organizational hierarchy of an enterprise, an entity in a supply chain, a customer in a market, purchaser in a purchasing decision, a price of a good or service, a cost of a good or service, a component of a product or system, a step of a method, a member of a group, or many others.

40 Referring to Fig. 16A, a high level schematic view of an architecture depicts how a plurality of services may be combined to operate as an integrated application that unifies development, deployment, operation, and life-

cycle management of a data integration solution. The unification of data integration tasks into a single platform may eliminate the need for separate software products for different phases of design and deployment. Although presented in a unified view, it should be understood that the individual modules, processes, services, and functions can each be provided separately, such as by invoking each of them independently as services in a services oriented architecture.

The architecture 6430 may include a GUI/tool framework 6432, an intelligent automation layer 6403, one or more clients 6434, APIs 6438, core services 6440, product function services 6442, metadata services 6452, metadata repositories 6454, one or more runtime engines 6444 with component runtimes 6450 and connectors 6448. The architecture 6430 may be deployed on a service-oriented architecture, such as any of the service-oriented architectures described above.

Metadata models stored in the metadata repository 6454 provide common internal representations of data throughout the system at every step of the process from design through deployment. The common services may provide for batch processing, concurrent processing, straight through processing, pipelining, modeling, simulation, conceptualization, detail design, testing, debugging, validation, deployment, execution, monitoring, measurement, improvement, upgrade, reporting, system management, and administration. Models may be registered in a directory that is accessible to other system components. The common models may provide a common representation (common to all product function services) of numerous suite-wide items including metadata (data descriptive data including data profile information), data integration process specifications, users, machine and software configurations, etc. These common models may enable common user views of enterprise resources and integration processes no matter what product functions the user is using, and may obviate the need for model translation among integrated product functions.

The service oriented architecture (SOA) is shown as encompassing all of the services and may provide for the coordination of all the services from the GUI 6432 through the run time engine 6444 and the connections 6448 to the computing environment. The common models, which may be stored in the metadata repository 6454, may allow the SOA to seamlessly provide interaction between a plurality of services or a plurality of models. The SOA may, for example, expose the GUI 6432 to all aspects of data integration design and deployment by use of common core services 6440, production function services 6442, and metadata services 6452, and may operate through an intelligent automation layer 6403. The common models and services may allow for common representation of objects in the GUI 6432 for various actions during the design and deployment process. The GUI 6432 may have a plurality of clients 6434 interfacing with SOA coordinated services. The clients 5204 may allow users to interface with the data integration design with a plurality of skill levels enabling users to work as a team across organizationally appropriate levels. The SOA 5201 may provide access to common core services 5210 and product function services 5212, as well as providing back end support to APIs 5208, for functions and services in data integration designs. Services may be shared and reused by a plurality of clients 5204 and other services. For example, a GUI 6432 may be the GUI for a client application that is designed specifically to work with a particular RTI service, such as exposing a particular data integration job as a service. Alternatively, the GUI 6432 may be a GUI for a product service 6442, such as a data integration service, such as extraction, transformation, loading, cleansing, profiling, auditing, matching, or the like. In other cases the GUI 6432 may be a GUI or client for a common service 6440, such as a logging or event management service. The clients 6434 may allow users to interface with the data integration design with a plurality of skill levels enabling users to work as a team across organizationally appropriate levels.

The SOA may provide access to common core services 6440, product function services 6442, and services related to metadata. The SOA may also include one or more APIs 6438 that expose the functions and services in the data integration platform to external applications and devices. Services may be shared and reused by a plurality of clients 6434, APIs, devices, applications and other services. The intelligent automation layer 6403 may employ
5 metadata and services within the architecture to simplify user choices within the GUI 6432, such as by showing only relevant user choices, or automating common, frequent, and/or obvious operations. The intelligent automation layer 6403 may automatically generate certain jobs, diagnose designs and design choices, and tune performance. The intelligent automation layer 6403 may also support higher-level design paradigms, such as workflow management or modeling of business context, and may more generally apply project or other contextual awareness
10 to assist a user in more quickly and efficiently implementing data integration solutions.

The common core services 6440 may provide common function services that may be commonly used across all aspects of the design and deployment of the data integration solution, such as directory services for one or more common registries, logging and auditing services, monitoring, event management, transaction services, security, licensing (such as creation and enforcement of licensing policies and communication with external
15 licensing services), and provisioning, and management of SOA services. The common core services 6440 may allow a common representation of functions and objects to the common GUI 6432. Any other service, such as the product function services 6442, RTI services, or other services, devices, applications or modules can access and act as a client of any particular common service 6440.

Other product specific function services 6442 may be contained in the product function services 6442 and
20 may provide services to specific appropriate clients 6434 and services. These may include, for example, importing and browsing external metadata, as well as profiling, analyzing, and generating reports. Other functions may be more design-oriented, such as services for designing, compiling, deploying, and running data integration services through the architecture. The product function services 6442 may be accessible to the GUI 6432 when an appropriate task is used and may provide a task oriented GUI 6432. A task oriented GUI may present a user only
25 functions that are appropriate for the actions in the data integration design.

The application program interfaces (APIs) 6438 may provide a programming interface for access to the full architecture, including any or all of the services, repositories, engines, and connectors therein. The APIs 6438 may contain a commonly used library of functions used by and/or created from various services, and may be called
recursively.

Fig. 16A additionally shows metadata and repository services 6454 that may control access to the metadata repository 6454. All functions may keep metadata represented by its own function-specific models in a common repository in the metadata repository 6454. Functions may share common models, or use metadata mappings to dynamically translate semantics among their respective models. All internal metadata and data used in data integration designs may be stored in the metadata repository 6454 and access to external metadata and data may be
35 provided by a hub (a metadata model) stored in the metadata repository 6454 and controlled by the metadata and repository services 6452. Metadata and metadata models may be stored in the metadata repository 6454 and the metadata and repository services 6452 may maintain metadata versioning, persistence, check-in and check-out of metadata and metadata models, and repository space for interim metadata created by a user before it is reconciled with other metadata. The metadata and repository services 6452 may provide access to the metadata repository
40 6454 to a plurality of services, GUI 6432, internal clients 6434 and external clients using a repository hub. Access

by other services and clients 6434 to the metadata repository 6454 may allow metadata to be accessed, transformed, combined, cleansed, and queried by the other services in seamless transactions coordinated by the SOA.

A runtime engine 6444, of which there may be several, may use adapters and connections 6448 to communicate with external sources. The engines 6444 may be exposed to designs created by a user to create
5 compiled and deployed solutions based on the computing environment. The runtime engine 6444 may provide late binding to the computer environment and may provide the user the ability to design data integration solutions independent of computer environment considerations. The run time engine 6444 orchestration with SOA services may allow the user to design without restrictions of run time compilation issues. The runtime engine 6444 may compile the data integration solution and provide an appropriate deployed runtime for high throughput or high
10 concurrency environments automatically. Services may be deployed as J2EE structures from a registry that provides access to interface and usage specifications for various services. The services may support multiple protocols, such as HTTP, Corba/RMI, JMS, JCA, and the like, for use with heterogeneous hardware and software environments. Bindings to these protocols may be automatically selected by the runtime engine 6444 or manually selected by the user from the GUI 6432 as part of the deployment process.

15 External connectors 6448 may provide access to a network or other external resources, and provide common access points for multiple execution engines and other transformation execution environments, such as Java or stored procedures, to external resources.

It will be appreciated that an additional functional layer may be provided to assist in selecting and using the various runtime engines 6444. This is particularly useful when provided in support of the high throughput or
20 high concurrency deployments. For example, the runtime engines 6444 may include a transaction engine adapted to parse large transactions of potentially unlimited length, as well as continuous streams of real time transactions. The runtime engines 6444 may also include a parallelism (or concurrency) engine adapted to processing small independent transactions. The parallelism engine may try to break up a process into pipeline functionality or some other partitioned flow, and works well with a large volume of similar work units. The parallelism engine may be
25 adapted to receive preprocessed input (and output) that has been divided into a pipelined or otherwise partitioned flow. A compilation and optimization layer may determine how to present processes to these various engines, such as by preprocessing output to the parallelism engine into small chunks. By centralizing connectors within the architecture, it is possible to more closely control distribution of processes between various engines, and to provide accessibility to this control at the user interface level. Also, a common intermediate representation of connectivity
30 in a transformation process enables deployment of any automation strategies, and selection of different combinations of execution engines, as well as optimization based on, for example, metadata or profiling.

The architecture 6430 described herein provides a high-degree of flexibility and customizability to the user's working environment. This may be applied, for example, to configure user environments around existing or planned workflows and design processes. Users may be able to create specific functional services by constructing
35 components and combining them into compositions, which may also serve in turn as components allowing recursive nesting of modularity in the design of new components. The components and compositions may be stored in the metadata repository 6454 with access provided by the metadata and repository services 6452. Metadata and repository services 6452 may provide common data definitions with a common interface with a plurality of services and may provide support for native data formats and industry standard formats. The modular nature of the
40 architecture described herein enables packaging of any enterprise function(s) or integration process(es) into a package having components selected from the common core services 6440 and other ones of the product function

services 6442, as well as other components of the overall architecture. The ability to make packages from system components may be provided as a common core service 6442. Through this packaging capability, any arbitrary function can be constructed, provided it is capable of expression as a combination of atomic services, components, and compositions already within the architecture 6430. The packaging capability of the architecture 6430 may be combined with the task orientation of the user interface to achieve a user interface specifically adapted to any workflow or design methodology that a user wishes.

Figure 16B depicts, at a high level, another architecture for a data integration system that includes an SOA, which in an embodiment may be the IBM WebSphere Services Backbone from IBM. The architecture may include components similar to those described in connection with Fig. 16A, such as one or more GUIs 6434, which may include specific clients 6480 that are designed to interact with various RTI services, such as described throughout this disclosure. The GUIs 6434 may include various other GUIs, such as GUIs for a variety of data integration tools, such as IBM's WebSphere Datastage, Metastage, RTI, Datastage TX, and other tools, as well as tools from other vendors. Thus a specially designed GUI, such as an RTI client 6480, or a conventional GUI 6434, may facilitate interaction with the functions, processes, modules and services of the data integration platform. In embodiments the GUIs 6434 may be clients of services that are deployed in a services oriented architecture. Various types of services can be enabled in such an architecture. In addition to real time data integration services, or RTI services, as described above, the platform may include various other product services 6442, such as services that perform specific data integration functions. A wide range of product services 6442 can be exposed as services in an SOA to enable access to the functions without requiring them to be separately coded. Many embodiments of such product services 6442 are described in detail below. In addition, the architecture may include common services 6440, which include a variety of services that may be useful for a wide variety of applications, modules, processes or functions. As described below, the GUIs 6434, product services 6442, other common services 6440, and other applications can serve as clients of any of the common services 6440, invoking the common services 6440 as needed to perform common functions, such as logging, event management, monitoring, provisioning, security, and the like. Many embodiments of such common services 6440 are described below. An SOA may also interact with common model and repository data and metadata 6454, including to expose metadata related services in an SOA. The architecture may also include an API, such as to allow an external device or application to access the data integration functions of the platform. An SOA may also interact with and/or invoke metabrokers 6452, engines 6450 and connectivity applications 6448. Such as to perform data integration tasks, such as extraction, transformation, and loading of data and metadata.

Referring to Fig. 16C a schematic of the SOA environment shows how the SOA interfaces to other architecture 6400 clients and services. The core of the SOA may be the service binding 6468, SOA infrastructure 6470, and service implementation 6474. Service binding 6468 may permit binding of clients, such as GUI 6464, applications 6460, script orchestration 6458, management framework 6456, and other clients, to services that may be internal or external to the SOA. The bound services may be part of the common core services 5520 and the services binding 6464 may access the service description registry 6466 to instantiate the service. The service binding 6464 may make it possible for clients to use services that may be local or external using the same or different technologies. The binding to external services may expose the external services and they may be invoked in the same manner as internal services. Communication to the services may be synchronous or asynchronous, may use different communication paths, and may be stateful or stateless. The service binding 6464 may provide support for a plurality of protocols such as, HTTP, EJB, web services protocols, CORBA/RMI, JMS, or JCA. As described

herein, the service binding 6464 may determine the appropriate protocol for the service binding automatically according to the computer environment or the user may select the protocol from the GUI 6464 as part of the design solution 5304.

The management framework 6456 client may provide facilities to install, expose, catalog, configure, monitor, and otherwise administer the SOA services. The management framework 6456 may provide access to clients, internal services, external services through connections, or metadata in internal or external metadata.

The orchestration client 6458 may make it possible to design a plurality of complex product functions and workflows by composing a plurality of SOA services into a design solution 5304. The services may be composed from the common core services 6476, services external to the internal services 6480, internal processes 6484, or user defined services 6478. The orchestration of the SOA is at the core of the capability to provide a unified data integration designs in the enterprise environment. The orchestration between the clients, core services, metadata repository services, deployment engines, and external services and metadata enables designs meeting a wide range of enterprise needs. The unified approach provides an architecture to bind together the entire suite for enterprise design and may allow for a single GUI 6464 capable of the seamless presentation of entire design process through to a deployment design solution. This architecture also enables common models to be used at design and run time, and common deployment models leveraging the same services as the design GUI 6464.

The application client 6460 may programmatically provide additional functionality to SOA coordinated services by allowing services to call common functions as needed. The functions of the application client 6460 may enhance the capability of the services of the SOA by allowing the services to call the functions and apply them as if they were part of the service. The GUI client 6464 may provide the user interface to the SOA services and resources by allowing these services and resources to be graphically displayed and manipulated.

The SOA infrastructure 6470 may be J2EE based and may provide the facility to allow services to be developed independent of the deployment environment. The SOA infrastructure 6470 may provide additional functionality in support of the deployment environment such as resource pooling, interception, serializing, load balancing, event listening, and monitoring. The SOA infrastructure 6470 may have access to the computing environment and may influence services available to the GUI 6464 and may support a context-directed GUI 6464.

The SOA infrastructure 6464 may provide resource pooling using, for example, enterprise java bean (EJB) and real time integration (RTI). The resource pooling may permit a plurality of concurrent service instances to share a small number of resources, both internal and external.

The SOA infrastructure may provide a number of useful tools and features. Interception may provide for insertion of encryption, compression, tracing, monitoring, and other management tools that may be transparent to the services and provide reporting of these services to clients and other services. Serialization and de-serialization may provide complex service request and data transfer support across a plurality of invocation protocols and across disparate technologies. Load balancing may allow a plurality of service instances to be distributed across a plurality of servers. Load balancing may support high concurrency processing or high throughput processing accessing one or a plurality of processor on a plurality of servers. Event listening and generation may enable the invocation of a service based on observed external events. This may allow the invocation of a second service based on the function of a first service and if a specified condition may occur. Event listening may also support call back capability specifying that a service may be invoked using the same identifier as when previously invoked.

The service description registry 6466 may be a service that maintains all interface and usage specifications for all other services. The service description registry 6466 may provide query and selection services to create

instances of services, bindings, and protocols to be used with a design solution. As an example, instances of services may be requested by a client or other service to the SOA where the SOA will request a query or selection of the called service. The service description registry 6466 may then return the instance of the service for binding by the service binding 6464 and then may be used in the design solution.

5 The common core services 6476 may contain a plurality of services that may be invoked to create design solutions and runtime deployed solutions. The common core services 6476 may contain all of the common services for design solutions therefore freeing other services from having to maintain the capabilities of these services themselves. The services themselves may call other services within the common core services 6476 as required to complete the design solution. A plurality of clients may access the common core services 6476 through the service
10 binding 6464, SOA infrastructure 6470 and service description registry 6466. Common core services may also be accessed by external services through metadata repository services 6452 and the SOA infrastructure 6470.

Additional external services may access any of the environments supported by the SOA infrastructure 6464 through the service implementation 6474. The service implementation may provide access to external services through use of adapters and connectors 6448. Through the service implementation 6474, services 6480
15 may expose specific product functionality provided by other software products for developing design solutions. These services 6480 may provide investigation, design, development, testing, deployment, operation, monitoring, tuning, or other functions. As an example, the services 6480 may perform the data integration jobs and may access the SOA for metadata, meta models, or services.

The service implementation 6474 may provide access for the processes 6484 to integration processes
20 created with other tools and exposed as services to the SOA infrastructure 6470. Users of other tools may have created these integration processes and these processes may be exposed as services to the SOA and clients.

The service implementation 6474 may also provide access to user defined services 6478 that may allow users to define or create their own custom processes and expose them as SOA services. Exposing the user-defined services 6478 as SOA services allows them to be exposed to all clients and services of the SOA.

25 Fig. 16D depicts the internal architecture of an SOA, such as the IBM WebSphere Services Backbone. A SOA, may incorporate or be composed of several different managers, such as a client invocation manager 6451 for managing the invocation of a client interface 6434, a policy manager 6453, that may manage service and binding policies, a J2EE manager 6455, a registry manager 6461, a persistence manager 6463, a service manager 6457 for managing the deployment of services, such as to add, modify or delete services, a binding manager 6465, a service
30 deployment manager 6459 for managing deployment of services and a binding deployment manager 6467 for managing deployment of bindings for services. An application server 6486, UDDI registry 6488 and a common repository 6490 may be associated with or part of the SOA. The SOA may provide common services 6440 and product services 6442. Each service may have a description 6477 associated with it. The description 6477, or the service itself, may have certain extensions associated with it. An extension may be used to link a service to other
35 services. An example of an extension would be to attach a "monitoring service extension" to a service. In the case of the monitoring service, this extension can consist, for example, of an m-bean that the service uses to track some values related to the service behavior. When this extension is found, the m-bean can automatically be registered with the monitoring service. In embodiments of the invention an administrator can define "metrics" that are calculated values created on top of the raw attribute values of the m-bean and can also define "monitors" that are
40 monitoring the m-bean to react to changes to the m-bean attribute values or to changes to the calculated values of the metrics. An example of a behavior associated to a monitoring service can be to generate an event (managed by

the event management service). In turn that event may call another service, or send an email or an alert to some specific users or administrators. An m-bean associated with a service description can capture values of attributes of the service, such as the number of times a service was invoked, or the like. In embodiments common services 6440, such as a monitoring service, can monitor the m-bean and calculate various metrics, such as averages, weighted averages, or the like, based on the values and attributes captured in the m-beans. The architecture can also include a service packager 6473 and a binding packager 6469. A binding factory 6479 can be used to build bindings 6468, such as bindings that are appropriate for various services. A service may have multiple bindings, which, as described below, may facilitate a variety of types of coupling between the service and various clients of the service.

Referring to Fig. 16E, in services oriented architectures one attaches bindings 6404 that allow the service to be accessed, such as through ports 6402. As described herein, various bindings, such as EJB, JMS, web services and JCA bindings can be used to invoke services in the various embodiments of services oriented architectures described herein. In embodiments, an API 13210 may be provided for assisting access to a service 6400. The API may provide various functions, such as selecting a particular binding for a service, where the selection is based on a condition or event, such as selecting a binding that is appropriate for a particular application. For example, bindings may vary in their flexibility, and an API 13210 may apply a tight or loose binding based on the conditions of the application or device that accesses the service. In embodiments the API 13210 may be a Java API or similar facility. In embodiments the same Java API 13210 may be used for different kinds of bindings. In embodiments, a smart client 13208 may be supplied for a service 6400. The smart client 13208 may be another layer on top of the API 13210 or may substitute for the API 13210. The smart client 13208 may be stored and accessed through a registry associated with a service. For example, an application may download the appropriate smart client 13208 based on the device using the application, the context of the application, or the like. For example, a smart client 13208 may be used to buffer certain information that is used by a service and send the information to the service in a package, rather than having an application access the service constantly. For example, when accessing a logging service, a user may wish to log only errors, rather than all events. By holding events until predetermined time periods, the user can reduce the number of calls to the server while still capturing all of the necessary events. The smart client 13208 can thus execute various rules that optimize the use of a service by a device or application. In embodiments the smart client 13208 can select a binding, either alone or by interaction with an API 13210, that optimizes the binding of the client-side device or application to the service 6400 based on the conditions of access, the capabilities of the device, the context of the access, or the like. The smart client 13208 or API 13210 can be used to store various access rules. For example, the rules might indicate that if a device or application is inside a firewall, then it can access a service using EJB bindings, while if the device or application is outside the firewall then it will access a service using a web service binding. Any such rules can be embodied in the API 13210 or may be included in a smart client 13208, which may optionally be listed in a registry with the service and downloaded by a client device or application that will access the service.

One of the benefits of a services oriented architecture is that it facilitates loose coupling between a client device or application that accesses a service and the code for the service itself; that is, a client device or application can invoke and use the service without knowing very much about the code for the service, needing to satisfy only certain predetermined inputs, such as what to input to the service (e.g., a file, an answer to a query, or the like). However, the absence of a tight coupling can result in performance problems, as context-dependent optimizing routines are omitted from the service description in order to make it more generically useful. An API 13210 and/or smart client 13208 can make up for diminished performance by ensuring that a service is accessed optimally, such

as by selecting a correct binding, caching data into batches, to avoid constantly invoking services for small jobs, or the like. Thus, a smart client 13208 provides effective performance in a loose coupling environment. The smart client 13208 thus bridges the gap between a tight coupling environment and a loose coupling environment and allows the user, application or device that accesses a service to choose a type of binding along the spectrum
 5 between loose coupling and tight coupling (such as EJB) according to the performance expectation or requirements. For example, EJB coupling may perform better than web services, because EJB couplings are by nature more tightly coupled between client applications and the server side. The smart client 13208 improves performance of both EJBs and web services by caching or buffering and sending things in appropriate batches. In situations where it is impossible or not desirable to cache or buffer items, a system can use a tight EJB binding to achieve good
 10 performance. In embodiments the API 13210 may hide the binding that the client device or application is using. With a smart client 13208, a user can tune the performance of the system by tuning the level of coupling between the client and the server.

In embodiments the runtime 13200 of a service in a services oriented architecture may be a client itself of another service, such one or more of the common services described in connection with Figs. 76 through 83. In
 15 embodiments the foregoing can be accomplished using AOP. In AOP, entities known as interceptors can associate a policy to a service. Inside the policy of the service, interceptors can be plugged into the policies, and the interceptors can be clients of the common services. For example, a policy in a service can include a plug-in that invokes the monitoring service 12500 of Fig. 77. Thus, AOP techniques can be used to insert code of interceptors into the code of various services described herein. In AOP, a user can create a piece of code and associate an
 20 "aspect" – a list of things to insert at runtime to the code as it is being executed. At that point in the code, the runtime program calls another piece of code, such as invoking a service, rather than doing what the code would normally do. At that point, the code calls another function that is compiled independently. Thus, when programmer looks at the source code for a runtime program, the programmer doesn't see the source code for the piece that is invoked by the interceptor. For example, in Java, the program can compile the source code to create
 25 the byte code, which is the runtime of Java, and a Java virtual machine reads the byte code. The program has the Java code and the aspect. The AOP compiler does byte code manipulation and calls other types of code, such as the services in the services oriented architecture. Thus, the methods and systems described herein include using common services either explicitly from an application or another service, or from an interceptor inserted in a service policy. That allows the same common service to be used by any service implementer and by the services oriented
 30 architecture framework transparently through the AOP sub-system.

Fig. 16F depicts a particular embodiment of an architecture for deploying a service in an SOA. As depicted in Fig. 16F, a variety of client-side and system-side components can be provided to enable the SOA. On the client side, various client-side applications 6480 or GUIs 6434, such as clients for RTI services, common
 35 services 6440 or product services 6442, can be developed and configured to access specific services. The client applications 6480 or GUIs 6434 can access the services directly through code that is designed to interact with various bindings, such as SOAP, EJB, JMS and web services bindings. Thus, depending on the capabilities, context and needs of the client application 6480, 6434 or device, a proper binding may be selected and enabled in the client application 6480, 6434, such as a tight EJB binding or a loosely coupled web services binding. The architecture may also include the API 13210, which may be designed to provide an interface to a particular service that is
 40 suitable for a particular type of client application, device, communication protocol, or the like. In embodiments a client invocation framework can automatically generate proxy, such as a C# or a C++ proxy, for either the

generated client API 13210 or for a registered smart/rich client application. The benefits of such a proxy are that: (i) a service through the client API 13210 can use any of the defined bindings transparently, according to business rules, without requiring special coding to interface with the bindings; (ii) additional smart /rich clients can be created on top of the generated API 13210 to optimize the use of the particular service, and (iii) proxies, such as C# or C++ proxies, can be generated to provide access to these generated clients or rich/smart clients in environments different from that of the API 13210, such as a non java environment in the case of a Java API. The system may include specific clients, such as SOAP clients 6407, EJB clients 6409, JCA clients 6411 and JMS clients 6413. The architecture may also include a WSDL layer 6415. Thus, multiple clients can exist to access a given service through various bindings, with a particular application or device being able to select the appropriate client, API 13210 or binding to access the service. The system also includes various ports 6402 with appropriate bindings 6404, which perform the functions described above. Referring still to Fig. 16F, the SOA runtime 13200 can enable many services, such as the various common services 6440 (such as logging, monitoring, provisioning, security, event management, administration, auditing and the like), product services 6442 (including metadata services 6452, RTI services, user-defined services, and the like). Services may also include connector access services, job execution services, metadata services, job browsing services, job deployment services, services related to workflow, job compilation services, logging services, security services, auditing services, monitoring services, licensing services, event management services and session management services.

Referring to Fig. 16G, the methods and systems described herein may include methods and systems for developing and deploying a wide range of data integration modules, tools, facilities, functions, services, jobs and processes, or combinations of these, as services in a services oriented architecture for data integration. Services oriented architectures can take various forms, such as those disclosed in connection with Figs. 16A through 16F. Referring still to Fig. 16G, a data integration module 6400, which could be any module, tool, facility, function, service, process, client application or other item that can be accessed by one or more pre-defined ports 6402 such as ports accessible through a computer network, a programming interface, or any other hardware or software connection or interface. Each port can have an associated binding 6404, which allows a user to access the module 6400 through the port 6402, as described above in connection with various embodiments of SOA. The module 6400 may include various operations 6408, which can be performed by the module 6400 when accessed through the bindings 6404 and ports 6402. A client interface 6410 may invoke or interact with services. One or more client interfaces 6410 may be invoked by or interact with the data integration service, module or facility 6400. The client interface 6410 may be a C++, C#, Java or any other application. Each module 6400 may include an interface 6414, such as for incoming and outgoing messages and other interactions with the service. The module 6400, possibly through one or more bindings 6404 may invoke or interact with service policies and/or interceptors 6412. The service policy 6412 may be a logging service, event management service, installation service, provisioning service, licensing service, monitoring service or auditing service. An interceptor 6412 may associate a policy to a service. Any one or more of a client interface 6410, port 6402, binding 6404, service policy or interceptor 6412 may form or be part of a services oriented architecture, such as the IBM WebSphere Services Backbone, common Services 6440 or product services 6442. Messages can have various parts, corresponding to the requirements of the definition of the module 6400, such as those described above in connection with various embodiments of services oriented architectures. For example, an incoming message can be in a format suitable for a given binding and can include input triggers for triggering operations of the particular module 6400. The module 6400 may include various

operations 6408, connected to or creating an abstract interface 6414, which can be performed by the module 6400 when accessed through the bindings 6404 and ports 6402.

Once a module 6400 is defined, including a definition of the appropriate port type, binding, and interface 6414, the module 6400 can be published in a registry, such as a registry for web services, to be identified and accessed by one or more users to accomplish the functions or operations defined in the definition of the module 6400. The code for those operations may be any conventional code for data integration platform functions, or any other code useful in data integration platforms of various vendors, such as IBM and others.

Many examples of modules 6400 are contemplated by this disclosure. For example, the modules 6400 can include product services 6442 for providing a wide range of functions, such as an extraction function, a data transformation, a loading function, a metadata management function, a data profiling function, a mapping function, a data auditing function, a data quality function, a data cleansing function, a matching function, a probabilistic matching function, a metabroker function, a data migration function, an atomic data repository function, a semantic identification function, a filtering function, a refinement and selection function, a design interface function, or many others.

Referring to Fig. 17, the module 6400 can be a data extraction module 6500. The data extraction module 6500 may extract data or metadata from a database 112 or other data facility 112 for use in a hub, in a data facility, or by a tool or other application. For example, the data extraction module 6500 may extract data from a customer database to a hub for use by a metabroker. Thus, the methods and systems described herein include providing a module for a data extraction function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 18, the module 6400 can be a data transformation module 6600. The data transformation module 6600 may transform data from a form provided from a data facility 112 into a form for storage in a data target, such as any database, data facility, or process, or combinations of these. The data transformation module 6600 may take the form of any of those described herein and may include, for example, one or more hubs or atomic data repositories, bridges, parallel execution engines, metabrokers, pipelining facilities or other facilities for moving data in batch or real-time transformations. For example, the transformation module 6600 may transform data from an XML or similar data format into the native format for a database or process, such as a supply chain database using SAP or Oracle. It will also be appreciated that, while a data transformation may be understood to include certain specific data integration operations, the data transformation module 6600 may perform additional operations incidental to a data transformation, such as extracting, loading, or cleansing. Thus, the methods and systems described herein include providing a module for a data transformation function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 19, the module 6400 can be a data loading module 6700. The data loading module 6700 may load data into one or more databases, processes, or other targets. A loading module 6700 may be a batch loading facility or a real-time loading facility, such as a loading facility that uses pipelining or similar functionality. The loading module 6700 may be used to load data in parallel to more than one data integration process, module, system, data facility or other element. For example, a loading facility may load data that is stored on or associated with a product tracking system simultaneously into a database for tracking the physical location of goods and into a database for tracking metadata associated with the goods, such as metadata entered by users at the time of collection

of the physical location data, such as data indicating that the order was received at a given time in acceptable condition. Thus, the methods and systems described herein also include providing a module for a data loading function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 20, the module 6400 can be a metadata management module 6800. The metadata management module 6800 may allow for storage and manipulation of metadata associated. The metadata management module 6800 may, for example, take the form of any metadata facility described herein. For example, the metadata management module 6800 may include a metabroker, an atomic data repository, a migration engine and/or other metadata facility. The metadata management module 6800 may be constructed to provide a variety of metadata functions that can be specified when the module 6800 is invoked as a service, or the metadata management module 6800 might perform a single, dedicated metadata management function. The metadata management module 6800 may allow a user to store, add, annotate and otherwise manipulate metadata. For example, a marketing manager may modify the metadata associated with a particular product to account for the fact that the product is currently the subject of a marketing campaign in a particular region. As another example, an engineer may modify the metadata associated with a part to reflect a change from metric units to English units, or vice versa, or to add a new characteristic for existing inventory such as RFID or UPC identification codes. Thus, the methods and systems described herein also include providing a module for a metadata management function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 21, the module 6400 can be a data profiling module 6900. The data profiling module 6900 may be used to profile data that is stored in a data facility or associated with a system. For example, the data profiling module 6900 may determine the content of columns or tables of data or metadata or assess the quality of the data or metadata. The data profiling module 6900 may generate a metadata model for one or more data sources to facilitate automation of subsequent data integration tasks. The data profiling module 6900 may also provide recommendations for constructing a target database from a source being profiled, such as keys and table normalizations. Thus, the methods and systems described herein also include providing a module for a data profiling function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 22, the module 6400 can be a data auditing module 7000. The data auditing module 7000 may be used to audit data that is stored in a data facility or associated with a system. For example, the data auditing module 7000 may determine the origin of a column of a table and track the job function of each user who modified the data. The data auditing module 7000 may also perform tasks such as validation of data ranges, calculations, value combinations, and so on. Thus, the methods and systems described herein also include providing a module for a data auditing function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 23, the module 6400 can be a data cleansing module 7100. The data cleansing module 7100 may cleanse data or metadata that is received from a database or system. The data cleansing module 7100

may take the form of any data cleansing facility, and may provide any data cleansing operations, such as any of those provided by the WebSphere QualityStage product from IBM. The data cleansing module 7100 may rapidly perform cleansing operations, such as de-duplicating records, so that any processes, systems, functions, modules, or the like that depend on the data have good data, rather than, for example, duplicate or erroneous data. Thus, the methods and systems described herein also include providing a module for a data cleansing function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 24, the module 6400 can be a data quality module 7200. The data quality module 7200 may assess the quality of data or metadata. The data quality module 7200 may provide any data quality functionality, such as functions provided by the WebSphere QualityStage product from IBM. The data quality module 7200 may determine the extent of duplication and erroneous data and may correct such errors. Thus, the methods and systems described herein also include providing a module for a data quality function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 25, the module 6400 can be a data matching module 7300. The data matching module 7300 may match data or metadata associated with an item to another item, such as a process, identifier, element, business process, business object, subject, data facility, rule, system or the like. For example, a matching module 7300 may match product data with a particular process, so that the product data or metadata is stored in the correct process. Thus, the methods and systems described herein also include providing a module for a data matching function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the data matching function may be a probabilistic matching function.

Referring to Fig. 26, the module 6400 can be a metabroker module 7400. A metabroker module 7400 may convert or transform metadata from one format or language to another, or between metadata models even if they use the same database technology. For example, a metabroker module 7400 may convert metadata associated with a particular line of products from SAP format to a format that can be used with an Oracle database. As another example, a company using its own metadata model for inventory may acquire another company that uses a different metadata model for inventory. The metabroker module 7400 may be used as a translator for combining or sharing data between inventory databases of the two companies. Thus, the methods and systems described herein also include providing a module for a metabroker function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the metabroker function maintains the semantics of a data integration function across multiple data integration platforms.

Referring to Fig. 27, the module 6400 can be a data migration module 7500. A data migration module 7500 may move data from one data facility 112 to another data facility 112 or hub. For example, a data migration module 7500 may move data from a customer database to a hub, where it may be acted upon by a metabroker module 7400, and then migrated or otherwise transferred to a finance database. Thus, the methods and systems described herein also include providing a module for a data migration function, providing a registry of services,

providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 28, the module 6400 can be an atomic data repository module 7600. An atomic data repository module 6400 may provide one or more fundamental data operations, such as read or write, for communicating with a repository using atomic data structures of the repository. The atomic data repository module 7600 may be employed for simple data transactions with a metadata model or other item stored in a repository, or may be combined with other modules 7600 to provide core repository services such as querying metadata models and the like. The methods and systems described herein also include providing a module for an atomic data repository, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 29, the module 6400 can be a semantic identification module 7700. A semantic identification module 7700 may identify an object, table, column or other item based on its relationship with other objects, tables, columns and other items. For example, a semantic identification module 7700 may create a string that may be acted upon by a data transformation module 6600. Thus, the methods and systems described herein also include providing a module for a semantic identification function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 30, the module 6400 can be a filtering module 7800. A filtering module 7800 may filter data, metadata, objects, items or instances of an item based on the associated level of abstraction or other properties. For example, a filtering module 7800 may filter the physical instances of the columns of a table in a hub from the logical instances based on the level of abstraction associated with each instance. Thus, the methods and systems described herein also include providing a module for a filtering function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the filtering is based on a level of abstraction. In embodiments the level of abstraction can be at least one of a physical level of abstraction and a logical level of abstraction.

Referring to Fig. 31, the module 6400 can be a refinement and selection module 7900. A refinement and selection module 7900 may filter data, metadata, instances or other items at the database, hub, query or other levels or stages of a process. For example, a refinement and selection module 7900 may allow a transformation operation to be performed on a query before it is sent to the relevant database. Thus, the methods and systems described herein also include providing a module for a refinement and selection facility, providing a registry of services, and identifying the facility in the registry, wherein the facility can be accessed as a service in a services oriented architecture. In embodiments the refinement and selection facility allows the system to distinguish between a logical level of abstraction and a physical level of abstraction.

Referring to Fig. 32, the module 6400 can be a database content analysis module 8000. A database content analysis module 8000 may analyze and summarize the content of a database and suggest possible related databases. For example, a database content analysis module may analyze a customer database and summarize salient information regarding the top twenty-five customers. As another example, the database content analysis module 8000 may provide a statistical analysis of numerical data in columns of a database, or report on the frequency of empty records, or report the number and size of tables, and so on. The database content analysis module 8000 may

also characterize database structure, and provide metadata relating to, for example, keys, column names, table names, and hierarchical or other relationships among the foregoing. More generally, the database content analysis module 8000 may provide any quantitative or qualitative analysis of a database than can be expressed in program code, and may provide corresponding reports or metrics that may be used by other modules 6400 or designers to characterize and apply the database contents. The database content analysis module may also, or instead, combine functions of modules described below for analyzing tables, columns and rows of databases, or employ those modules in analyzing a database. Thus, the methods and systems described herein also include providing a module for analyzing the contents of a database, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 33, the module 6400 can be a database table analysis module 8100. A database table analysis module 8100 may analyze and summarize the content of a table. For example, a database table analysis module 8100 may provide the hierarchical position of one table of a database with respect to other tables of the database. Thus, the methods and systems described herein also include providing a module for analyzing a table of a database, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 34, the module 6400 can be a database row analysis module 8200. A database row analysis module 8200 may analyze and summarize the content of a row of a table. For example, a database row analysis module 8200 may suggest other rows and/or tables that may be related to a row of interest. The database row analysis module 8200 may also, or instead, evaluate the validity of records within a row according to information about database structure. Thus, the methods and systems described herein also include providing a module for analyzing a row of a database, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 35, the module 6400 can be a data structure analysis module 8300. A data structure analysis module 8300 may analyze the overall structure of the data or metadata associated with the data relating to a row, column, table or data facility 112, or any combination of these. For example, a data structure analysis module 8300 may generate a report summarizing the number and hierarchical relationship of the rows, columns and tables composing a particular database 112. Thus, the methods and systems described herein also include providing a module for analyzing a data structure, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 36, the module 6400 can be a recommendation module 8400. A recommendation module 8400 may recommend a target data facility for an operation or process. For example, a recommendation module 8400 may locate and recommend an unused hub for a process involving a metabroker module 6600. As another example, the recommendation module 8400 may recommend a target database for an ETL operation based upon known characteristics of potential target databases such as access time, fault tolerance, capacity, and so on. The recommendation module 8400 may also, or instead, provide a number of different recommendations for the structure of a target database using techniques analogous to those employed by IBM's WebSphere ProfileStage and AuditStage products. Thus, the methods and systems described herein also include providing a module for

recommending a target data facility, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 37, the module 6400 can be a primary key module 8500. A primary key module 8500 may use dependency information from table analysis to identify primary key candidates for a table under analysis. For example, the primary key module 8500 may determine that the customer name column should be a primary key for a customer information table. This information may be used to assist in designing a target database for an ETL operation or other data integration process requiring a data target. Thus, the methods and systems described herein also include providing a module for providing a primary key for a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 38, the module 6400 can be a foreign key module 8600. A foreign key module 8600 may analysis a data structure to identify foreign keys. This information may be useful in, for example, preserving the integrity of relationships between tables, and in locating a primary key table with a data structure. Thus, the methods and systems described herein also include providing a module for providing a foreign key for a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 39, the module 6400 can be a table normalization module 8700. A table normalization module 8700 for a data integration function may transform or a split a table to eliminate dependencies and/or remove redundant data and anomalies. Normalization may provide significant performance improvements in a database including faster queries and improved data integrity. Thus, the methods and systems described herein also include providing a module for providing a table normalization for a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 40, the module 6400 can be a source-to-target mapping module 8800. A source-to-target mapping module 8800 for a data integration function may create a data transformation mapping for mapping data or metadata from the source system to one or more target data facilities. For example, a mapping facility may map product location data collected by a sensor to a new database combining all information about products. Or a mapping may be between a supply chain database and an inventory database, or more generally from any source to any target. While mapping typically connotes literal transfer between two locations, the source-to-target mapping module may also specify transformations with a mapping, such as combinations, filters, or other conversions or transformations. For example, the mapping may specify a coincident transformation from minutes to hours or days. Thus, the methods and systems described herein also include providing source-to-target mapping for a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 41, the module 6400 can be an automatic data integration job generation module 8900. An automatic data integration job module 8900 may automate the creation of a data integration job by generating a data integration job using a profile or specification provided to the module 8900. The data integration job may be provided as another module 6400 that may be registered for subsequent use throughout an enterprise, and the

automatic data integration job generation module 8900 may return a specification of where and how to access the newly created job module. For example, an automatic data integration module 8900 may generate a commonly used data integration job for a stored profile for that type of data integration job. The commonly used data integration job may be the integration of customer credit information with information regarding the customer's business. This job may need to be performed for each new customer. Thus, the methods and systems described herein also include providing a module for automatically generating a data integration job from a profile for a data integration job, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 42, the module 6400 can be a defect detection module 9000. A defect detection module 9000 may detect defects in a data facility, process or other operation. For example, a defect detection module 9000 may determine that a data integration process was performed incorrectly resulting in a table with mismatched entries. Thus, the methods and systems described herein also include providing a module for defect detection, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 43, the module 6400 can be a performance measurement module 9100. A performance measurement module 9100 may measure the performance of a data integration process. For example, a performance measurement module 9100 may record the time and processor load for a given data integration operation. The performance measurement module 9100 may also assist with the optimization or modification of data integration processes. Thus, the methods and systems described herein also include providing a module for measuring the performance of a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 44, the module 6400 can be a data de-duplication module 9200. A data de-duplication module 9200 may remove duplicate entries, rows, columns, tables, and databases from a data facility 112 or subset of a data facility 112. For example, a data de-duplication module 9200 may remove two identical address entries for Bob Smith. While de-duplication of identical records is straightforward, more subtle forms of de-duplication may also be employed using, for example, information about names (e.g., "Bill" = "William" or "GE" = "General Electric") and abbreviations, as well probabilistic matching or other techniques that may catch minor variations due to spelling errors or data entry errors. Thus, a data de-duplication module 9200 may also determine that the entry for Robert A. Smith at 55 Any Road, is the same as the entry for Bob Smith at 55 Any Rd., and remove the duplicate information. De-duplication may be an important preliminary quality enhancement step in an ETL operation, or any other data integration process involving an extraction of data from a database. Thus, the methods and systems described herein also include providing a module for data de-duplication, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the de-duplication module matches data items based on a probability. In embodiments the de-duplication module discards duplicate items.

Referring to Fig. 45, the module 6400 can be a statistical analysis module 9300. A statistical analysis module 9300 may perform tests and gather statistics relating to data, metadata or the processes and operations being

performed on the data and metadata. For example, a statistical analysis module 9300 may generate a relationship function describing the relationship between the number of units of a product sold and the age of the customer. A statistical analysis module 9300 may also provide process metrics, such as determining the average time it takes to perform a certain data integration operation with a certain processor configuration. More generally, the statistical analysis module 9300 may perform any statistical analysis on data within a data source, metadata for one or more data sources, or processes operating on data or metadata. Thus, the methods and systems described herein also include providing a module for statistical analysis of a plurality of data items, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 46, the module 6400 can be a data reconciliation module 9400. A data reconciliation module may reconcile data and metadata from disparate data facilities 112. For example, a data reconciliation module 9400 may join similar product entries from a company's product databases corresponding to two different geographic regions allowing for the creation of master records. In another aspect, a data reconciliation module 9400 may reconcile multiple instances of an identical or nearly identical record. For example, a customer may have two different records with different addresses. These records may be reconciled, such as by using a creation date or a most recent transaction date, into a single record. Other reconciliations may be useful in a data integration system, such as a reconciliation of database backups or a reconciliation of versions of a metadata model, and may be performed using a data reconciliation module 9400. Thus, the methods and systems described herein also include providing a module for reconciling data from a plurality of data facilities, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 47, the module 6400 can be a transformation function library module 9500. A transformation function library module 9500 may provide access to a library of transformation functions. For example, common transformation functions, such as integration of customer credit and purchasing information, or transformation of data between units (e.g., Celsius to Fahrenheit or quarts to liters), or revision of exchanges for telephone numbers, may be maintained in a library so that a user does not need to create the operation from scratch each time the user wished to perform the operation. Other more fundamental transformations may also be used, such as character strings to numerical values or vice versa, or change of numerical value types (e.g. byte, word, long word). Thus, the methods and systems described herein also include providing a module for accessing library of transformation functions, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 48, the module 6400 can be a version management module 9600. A version management module 9600 may assist in the management of different data integration jobs stored in a library or may assist in the creation and execution of data integration jobs. For example, a version management module may allow a user to maintain multiple versions of the customer credit and purchasing data integration job described above. It may be the case that customers often have two or three accounts that require integration, so a separate version of the data integration job may be maintained for jobs dealing with two or three transactions. Similarly, the version management module 9600 may be used to select a version of a metadata model, metabroker, or other repository object, or to query a registry or repository about what versions of these objects exist. The module 9600 may also support version-related functions, such as branching and reconciliation of multiple versions. Thus, the methods and

systems described herein also include providing a module for managing versions of a data integration job, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

5 Referring to Fig. 49, the module 6400 can be a version management module 9700 of a different type. The version management module 9700 of Fig. 50 may control versions of data or metadata used in a data integration process. Thus while the module 9600 of Fig. 48 may control versions of tools and processes, the module 9700 of Fig. 49 may control versions of data or metadata that the tools are applied to. Thus, the methods and systems described herein also include providing a module for managing versions of a data integration job, providing a
10 registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the module allows a user to share a version with another user. In embodiments the module allows a user to check in and check out a version of a data integration job in order to use the data integration job.

15 Referring to Fig. 50, the module 6400 can be a parallel execution module 9800. A parallel execution module 9800 may allow for the dynamic execution of data integration jobs in parallel. The parallel execution module 9800 may analyze processing and data dependencies of portions of an execution task to generate an appropriate parallel execution order, or may receive explicit parallelism instructions along with the identification of a task for execution. Thus, the methods and systems described herein also include providing a module for parallel
20 execution of a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 51, the module 6400 can be a data partitioning module 9900. A data partitioning module 9900 may break up a source record set into several sub-sets. For example, for a data integration job involving a
25 table, the table may be broken into several sub-tables, each having its own data, index, and so forth, and the data integration job performed on each sub-table simultaneously. This process may result in shorter processing times. Thus, the methods and systems described herein also include providing a module for partitioning data, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented
30 architecture.

Referring to Fig. 52, the module 6400 can be a partitioning and repartitioning module 10000. A partitioning and repartitioning module 10000 may function as a partitioning module 9900 with the added functionality of being able to recombine the original or transformed subsets. For example, after the data integration job described in the example of Figure 51 has been performed a partitioning and repartitioning module 10000 may
35 join the sub-tables to create a transformed table resembling the source table. Thus, the methods and systems described herein also include providing a module for partitioning and repartitioning data, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 53, the module 6400 can be a database interface module 10100. A database interface
40 module 10100 may allow a user to interact with a database and/or perform data integration jobs. For example, a database interface module 10100 may allow a user to view certain entries in a database, such as the sales

performance history for a certain employee. The database interface module 10100 may provide atomic user interaction, such as an individual query, read, write, or other transaction. The database interface module 10100 may also, or instead, provide more general database connectivity through which a data integration job or other process may operate continuously on a database. Thus, the methods and systems described herein also include providing a
5 database interface module, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the interface module facilitates an interface to databases of a plurality of database vendors.

Referring to Fig. 54, the module 6400 can be a data integration module 10200. A data integration module
10 10200 may allow for the creation or execution of data integration jobs. For example, a user may create and schedule certain transformation jobs using the data integration module 10200, or investigate what data integration processes are available in modules 6400 using the data integration module 10200. Thus, the methods and systems described herein also include providing a module for a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module
15 in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 55, the module 6400 can be a synchronization module 10300. A data synchronization module 10300 may synchronize data from disparate sources. For example, a data synchronization module 10300 may align similar entries in different databases, perform cross-linking analysis and remove any duplicative or erroneous records. Thus, the methods and systems described herein also include providing a module for
20 synchronizing data, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the module facilitates synchronization of data across a plurality of hierarchical data formats. In embodiments the module facilitates synchronization of data across a plurality of transactional formats. In embodiments the module facilitates synchronization of data across a plurality
25 of operating environments. In embodiments the module facilitates synchronization of Electronic Data Interchange format data. In embodiments the module facilitates synchronization of HIPAA data. In embodiments the module facilitates synchronization of SWIFT format data.

Referring to Fig. 56, the module 6400 can be a metadata directory supply module 10400. A metadata directory supply module 10400 may serve as a glossary or definitional database that provides insight into the types
30 of information recorded by an enterprise. For example, user in the sales department can access a metadata directory using the metadata directory supply module 10400 to learn about the types of data recorded by the production department. The user may learn that the production department defines units in lots, while the sales department defines units in hundred-lots. As a result, the user can adjust her supply forecasts accordingly. Thus, the methods and systems described herein also include providing a module for supplying a metadata directory, providing a
35 registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 57, the module 6400 can be a graphical depiction module 10500. A graphical depiction module 10500 may depict in graphical format the effects of a modification to a data integration job. For example, a
40 graphical depiction module 10500 may show a user the larger table that may result if the data normalization step is skipped in a data integration process. The graphical depiction module 10500 may be particularly useful, for

example, to support a strongly separated user interface for interacting with a data integration system. Thus, the methods and systems described herein also include providing a module for graphical depiction of the impact of a change to a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be
5 accessed as a service in a services oriented architecture.

Referring to Fig. 58, the module 6400 can be a metabroker module 10600. A metabroker module 10600 may provide metadata concerning metabrokers registered in a system. For example, the metabroker module 10600 may permit queries over available metabrokers to assist in a manual or automated selection of metabrokers for design of a data integration process. Thus, the methods and systems described herein also include providing a
10 module for creating a metabroker, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 59, the module 6400 can be a metadata hub repository module 10700. A metadata hub repository module 10700 may allow for the transient storage of metadata so that operations may be performed on
15 the metadata. For example, the metadata hub repository module 10700 may allow metadata to occupy a hub in such a way as to allow a metabroker to convert the metadata to an SAP compatible format. Thus, the methods and systems described herein also include providing a module for a hub repository of metadata, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In
20 embodiments the hub stores semantic models for a plurality of data integration platforms.

Referring to Fig. 60, the module 6400 can be a packaged application connectivity kit (PACK) module 10800. A PACK module 10800 may allow for the interchange of data and metadata between disparate applications. For example, a PACK module 10800 may allow data and metadata generated and/or stored using Informatica PowerCenter to be accessed and used by SAP BW. More generally, a PACK may enable connectivity to or
25 between any database, application, or enterprise running on any operating system and/or hardware. The PACK module 10800 may be particularly useful, for example, when integrating legacy data systems into an enterprise, or when integrating data across previously separated divisions of a business that use different database management technologies. Thus, the methods and systems described herein also include providing a PACK, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412 for the PACK,
30 and identifying the PACK in the registry, wherein the PACK can be accessed as a service in a services oriented architecture.

Referring to Fig. 61, the module 6400 can be an industry-specific data model storage module 10900. An industry-specific data model storage module 10900 may allow for the storage of industry-specific data models. For example, companies in the trucking industry may record certain characteristics about shipments. An industry-
35 specific data model storage module 10900 may allow for the storage of a template that can be used by trucking companies. Certain industries employ widely adopted or legally required standards for data storage and communication. For example, HIPAA mandates certain transaction types and privacy standards that must be used by health care providers. SWIFT is commonly used for transactions in financial industries. These and other similar standards may be managed and deployed within a data integration system using the industry-specific data model
40 storage module 10900. Thus, the methods and systems described herein also include providing a module for storing an industry-specific data model, providing a registry of services, providing one or more client interfaces 6410,

service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. The model may be a manufacturing industry model, a retail industry model, a telecommunications industry model, a healthcare industry model, a financial services industry model or a model from any other industry.

5 Referring to Fig. 62, the module 6400 can be a template module 11000. A template module 11000 may allow a user to build and store templates for certain type of data integration jobs. A template may combine tasks and functions of other modules 6400 described herein, or any other tasks and functions suitable for a data integration system, to capture a particular design solution for use, reuse, and refinement. For example, a user may build and store a template that integrates customer credit and order information. The user may make this template
10 available to other users through the transformation function library module 9500. Thus, the methods and systems described herein also include providing a template for building a data integration function, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412 for the template, and identifying the template in the registry, wherein the template can be accessed as a service in a services oriented architecture.

15 Referring to Fig. 63, the module 6400 can be a business rule creation module 11100. A business rule creation module 11100 may provide any business rule or business logic capable of formal expression, and may include comparisons, conditional evaluations, mathematical evaluations, statistical analyses, Boolean operations, and any other operations that may be performed in the context of providing a business rule. For example, a company may require a minimum credit score before issuing credit to a customer, and this may be formalized as a
20 business rule. A company may have predetermined programs for salaries and pensions that may be applied to payroll calculations in a human resources department, or a company may maintain different hiring criteria for different departments, or a company may be required to report sales to a local government agency. The scope and complexity of possible business rules is unlimited. Any such rule that can be programmatically expressed may be created using the business rule creation module 11100 and subsequently applied in data integration processes.
25 Thus, the methods and systems described herein also include providing a module for creating a business rule, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 64, the module 6400 can be a validation table creation module 11200. A validation table
30 creation module 11200 may allow for the creation of a validation table for other data integration functions. Thus, the methods and systems described herein also include providing a module for creating a validation table, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

35 Referring to Fig. 65, the module 6400 can be a data integration module 11300. It will be noted that a data integration module 10200 has been described in reference to Fig. 66. That data integration module 10200 related to the creation and/or execution of prepackaged data integration jobs. The module 11300 described here relates instead to a module that executes a specific data integration job, task, or function. Thus, a data integration job created with the data integration module 10200 may be executed as a prepackaged job in the data integration
40 module 11300 described here. The data integration module 11300 may perform any data integration job, task, or process. The data integration module 10200 may also be associated with a control in a graphical user interface

labeled to indicate the nature of the data integration function. In this manner, a strongly separated user interface may have access to any user-defined data integration function through a button, drop-down menu item, or other control, which may be conveniently labeled for user identification. Thus, the methods and systems described herein also include providing a module for a data integration function, providing a registry of services, providing one or
5 more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 66, the module 6400 can be a business metric creation module 11400. A business metric creation module 11400 may allow for the creation of certain business metrics to be associated with a business or subset of a business. For example, the business may be a consumer products business and the business metric
10 creation module 11400 may help to create a metric measuring increased sales per dollar of advertising. The business metric creation module 11400 may also collect the necessary data for computation of the metrics or work with other modules and systems to this end. The module 11400 may enable creation of a metric using any mathematical, logical, conditional, or other function, or combinations thereof. Thus, the methods and systems described herein also include providing a module for creating a business metric, providing a registry of services,
15 providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 67, the module 6400 can be a target database definition module 11500. A target database definition module 11500 may assist in the definition of a target database, including the type and structure of the database. For example, the target database definition module 11500 may receive recommendations from profiling
20 and auditing modules, and prepare a database definition for a target database suitable for a particular data source and transformation. The module 11500 may allow for interactive control at various decision points, or may function deterministically without user intervention. Thus, the methods and systems described herein also include providing a module for defining a target database, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the
25 module can be accessed as a service in a services oriented architecture.

Referring to Fig. 68, the module 6400 can be a mainframe data profiling module 11600. A mainframe data profiling module 11600 may allow for the profiling of mainframe data. A computer mainframe may have particular data formats, connectivity requirements, security layers, and so on. The mainframe data profiling module 11600 may be designed to address all of these issues for a particular mainframe or type of mainframe to accelerate design
30 of data integration systems using such a mainframe. Thus, the methods and systems described herein also include providing a module for profiling mainframe data, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 69, the module 6400 can be a batch processing module 11700. A batch processing
35 module 11700 may allow for the processing of data integration jobs in batch. For example, with certain processor configurations it may be desirable to process transactions in batch. As another example, it may be desirable to concentrate processing away from peak computer-use times, such as from 1:00 a.m. to 3:00 a.m. Batch processing may facilitate the execution of large data integration jobs and processes at user-programmable times, or on user-selectable machines. The batch processing module 11700 may aid facilitate processing in this manner, or any other
40 controlled manner. Thus, the methods and systems described herein also include providing a module for batch processing a batch of data, providing a registry of services, providing one or more client interfaces 6410, service

policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 70, the module 6400 can be a cross-table analysis module 11800. A cross-table analysis module 11800 may allow for the analysis of relationships and linkage between tables, which may yield significant benefits in the construction of target databases. For example, a cross-table analysis module 11800 may allow a user to determine the degree of relatedness between two customer data tables. Based on this information a user may decide to integrate the information in the tables. Thus, the methods and systems described herein also include providing a module for cross-table analysis, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 71, the module 6400 can be a relationship analysis module 11900. A relationship analysis module 11900 may analyze the relationship between any two or more rows, columns, tables, databases, or combinations of these and other data source items. For example, a relationship analysis module 11900 may determine the relationship between a column and a table. This information may be used to validate other data in the database, or identify keys or other structural information for a database that has not yet been fully characterized. Based on the relationship analysis a user may decide to take responsive steps in designing a data integration process or a target database, such as joining tables, partitioning tables, eliminating columns, and so on. Thus, the methods and systems described herein also include providing a module for relationship analysis, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 72, the module 6400 can be a data definition language code generation module 12000. A data definition language (DDL) code generation module 12000 may generate DDL code for a database, either to create a new target database, or modify a source or target database. The data definition language code generation module 12000 may generate DDL code in response to other structural database descriptions provided to the module, or as a parameter accompanying some other data integration process. DDL code may be applied directly to a database, such as an SQL database, to affect structural changes therein. Thus, the methods and systems described herein also include providing a module for DDL code, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture. In embodiments the methods and systems may further include using the module to create a mapping between source and target data facilities.

Referring to Fig. 73, the module 6400 can be a design interface module 12100. A design interface module 12100 may provide a user interface for the creation and design of data integration jobs. A design interface module 12100 may include a graphical user interface. The design interface module 12100 may be strongly separated, providing only the low-level controls and layout for an interface, while being associated with other modules 6400 or code that performs functions within a data integration system. As an example of operations that can be performed through the design interface module 12100, a design interface module 12100 may allow a user to link various operations on a screen to create a data integration job. In another embodiment, the design interface module 12100 may provide only functional access to a design, such as a metadata model or data integration job, by providing suitable programmatic control over storage, retrieval, and modification of the design. The design interface module 12100 may in turn connect the programmatic control to a client such as a program or a graphical user interface. Thus, the methods and systems described herein also include providing a design interface module for designing a

data integration job, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 74, the module 6400 can be a data integration job development module 12200. A data integration job development module 12200 may allow for the development of a data integration job. For example, a user may use the data integration job development module 12200 to build upon pre-existing data integration jobs. The data integration job development module 12200 may provide functional support for development features of a strongly separated graphical user interface. Thus, the methods and systems described herein also include providing a module for developing a data integration job, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

Referring to Fig. 75, the module 6400 can be a data integration job deployment module 12300. A data integration job deployment module 12300 may facilitate the deployment of data integration jobs, and address any implementation issues arising at run time. The data integration job deployment module 12300 may deploy data integration jobs on a scheduled basis, or under control of a client of the module 12300. The module 12300 may also suggest the scheduling of additional data integration jobs. The data integration job deployment module 12300 may deploy multiple data integration jobs simultaneously across disparate data facilities 112. Thus, the methods and systems described herein also include providing a module for deploying a data integration job, providing a registry of services, providing one or more client interfaces 6410, service policies and/or interceptors 6412, and identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

In various embodiments the modules, facilities, tools, jobs, services, processes and functions described herein may be accessed through various input and output facilities, including bindings and similar facilities, such as EJBs, JMS, web services, SOAP and other bindings.

In embodiments the methods and systems described herein may include a client-side facility for optimizing access of a module, facility, job, service, process, function or the like by a client device. In embodiments the methods and systems described herein may include a server-side facility for optimizing access of a module, facility, job, service, process, function or the like by a client device.

Referring to Fig. 76, in embodiments the services in a services oriented architecture for a data integration platform or process may be services that are useful for a wide range of integration and computing tasks, including modules that perform functions that are required or beneficial for many common tasks. Thus, for example, a logging service 12400 may be deployed, such as for logging events. A user who wishes to log events (for any reason related to any task, such as in connection with data integration job or task) may invoke the logging service by accessing it through a services registry in a services oriented architecture. Thus, a programmer need not create a new logging service for logging events, but instead may invoke a pre-coded logging service through the services registry.

Referring to Fig. 77, a monitoring service 12500 may be deployed as a service in a services oriented architecture. For example, the monitoring service 12500 may be invoked by a user to monitor some aspect of the performance of a data integration job or task, or to monitor an event or process. A monitoring service 12500 may allow for the generation of specific events and metrics, such as counters, averages and sums, for monitoring purposes. For example, a data integration system may have a service called a job execution service, the purpose of which is to run a job, such as a batch job. Using a monitoring service 12500, a user can monitor how many times

the job execution service has been run, how long it took to run, the minimum execution time, maximum execution time, average execution time and other statistics. The user can accomplish all of those functions without seeing the code of the underlying job execution service. The fact that all monitoring services are deployed as services means that inside the execution of the job a user can ask, for example, how many databases have been touched or other monitoring items that are specific to the semantics of the job execution service. Thus, the job execution service can itself be a client of the monitoring service. Thus, through a monitoring service 12500, the system can tell what is happening inside the implementation of another service. In embodiments, each common service, such as the monitoring service 12500 and the other services described in connection with Figs. 76 through 83, various areas can be established for each service, such as what to monitor, the runtime of the service, and an administration part. To invoke the monitoring service 12500, the user may be queried as to what to monitor. Thus, the monitoring service 12500 can be used by services in a services oriented architecture to monitor what the services do or may be used to conduct domain-specific monitoring for other events and conditions.

Referring to Fig. 78, a security module 12600 or service may be deployed as a service in a services oriented architecture for providing a security capability, such as in connection with a data integration job or task. When a user requires a security facility, such as password protection, encryption, tracking access, restricting access, or the like, the user can invoke a security module 12600 as a service in a services oriented architecture, so that the user does not have to create a separate security facility for each data integration job or task.

Referring to Fig. 79, a licensing module 12700 may be deployed in a services oriented architecture, for enabling licensing functions when invoked by a user. For example, a job designer may cause a data integration job to invoke the licensing service to determine whether a particular task to be executed at runtime does or does not comply with license restrictions, such as license restrictions related to the number of machines, number of users, or the like. The user avoids the need to prepare separate licensing code for each data integration job or task the user creates. A licensing module may be used in connection with an installation and/or provisioning service.

Referring to Fig. 80, an event management module 12800 may be deployed in a services oriented architecture for tracking and managing events when invoked by a user through a services registry. The user may access the event management module 12800 for any event management required for a data integration job or task, such as tracking events in order to determine when to execute a process or function. The user avoids the need to create separate event management code for each different data integration task or job. An event management module 12800 may allow for event subscription by application and may incorporate a callback mechanism.

Referring to Fig. 81, a provisioning module 12900 may be deployed in a services oriented architecture, allowing a user to enable provisioning functions by accessing the provisioning module 12900 through a services registry. A provisioning module 12900 may allow for the provision of components to multiple machines, may maintain a history of the components and version installed on different machines, push or distribute software or patches, may trigger the installation of a security service, may assist with or allow for authorization and/or authentication, may maintain internal and external user directories and may assist with or allow for single sign-on functionality.

Referring to Fig. 82, a transaction module 13000 may be deployed in a services oriented architecture that allows a user to access the transaction module 13000 through a services registry, avoiding the need to create separate transaction management code for each application created by the user, such as for a data integration job or task.

Referring to Fig. 83, an auditing module 13100 can be deployed in a services oriented architecture that allows a user to access the auditing module 13100 through a services registry, avoiding the need to create separate auditing code for each application created by the user, such as for a data integration job or task. Thus, by accessing the auditing module 13100 by invoking the service, the user can audit events, such as auditing what users have accessed a particular database or process, what events have taken place, and the like. An auditing module 13100 can allow a user to conveniently audit past events without having to generate separate code.

Thus, a wide variety of common tasks that are necessary or beneficial for data integration jobs or platforms can be created as modules and deployed as services in a services oriented architecture.

In the various embodiments of modules and services that are described herein, techniques of AOP can be used to implement services in a services oriented architecture. For example, various metadata functions and modules can be implemented as services with AOP. In embodiments, bindings for services, such as EJBs (such as EJB 3.0) may use AOP.

While the invention has been described in connection with certain preferred embodiments, it should be understood that other embodiments would be recognized by one of ordinary skill in the art, all of which are intended to fall within the scope of this disclosure.

CLAIMS:

What is claimed is

1. A method, comprising:
 - 5 providing a module for a data integration function;
 - providing a registry of services;
 - providing an interface for the module; and
 - identifying the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.
- 10 2. The method of claim 1 wherein the data integration function includes one or more of data extraction, data transformation, data loading, metadata management, data profiling, data auditing, data cleansing, data quality, data matching, data migration, and filtering.
3. The method of claim 1 wherein the data integration function includes one or more of an atomic data repository function, a semantic identification function, a filtering function, a refinement and selection function, a design
 - 15 interface function, an analysis function, a targeting function, a primary key provision function, a foreign key provision function, a table normalization function, a source to target mapping function, an automatic generation of data integration job functionality, a defect detection function, a performance measurement function, a data deduplication function, a statistical analysis function, a data reconciliation function, a library function, a version management function, a parallel execution function, a partitioning function, a partitioning and repartitioning
 - 20 function, an interface function, a synchronization function, a metadata directory function, a graphical impact depiction function, a hub repository function, a packaged application connectivity kit functionality, an industry-specific data model storage function, a template function, a business rule function, a validation table function, a business metric function, a target database definition function, a mainframe data profiling function, a batch processing function, a cross-table analysis function, a relationship analysis function, a data definition language code
 - 25 generation function, a data integration job design function, a data integration job deployment function and a data integration job development function.
 4. The method of claim 1 wherein the data integration function includes a metabroker function, the metabroker function maintaining the semantics of a data integration function across multiple data integration platforms.
 5. The method of claim 1 wherein the data integration function includes a refinement and selection function,
 - 30 wherein the refinement and selection function allows the method to distinguish items based on differentiating characteristics.
 6. The method of claim 1 wherein the module analyzes contents of a database.
 7. The method of claim 7 wherein the module analyzes one or more of a column of the database, a table of the database, and a row of the database.
 - 35 8. The method of claim 7 wherein the module recommends a target database for an item of data.
 9. The method of claims 1-8 wherein the module is accessible by one or more of an Enterprise Java Bean, a Java Messaging Service, and a web service.
 10. The method of claim 1 wherein a policy is associated with the service using aspect oriented programming.
 11. The method of claims 1-10 wherein the module includes a real time service.
 - 40 12. A system, comprising:
 - a module for a data integration function;

a registry of services;
 an interface for the module; and
 identification of the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

- 5 13. The system of claim 12 wherein the data integration function includes one or more of data extraction, data transformation, data loading, metadata management, data profiling, data auditing, data cleansing, data quality, data matching, data migration, and filtering.
- 10 14. The system of claim 12 wherein the data integration function includes one or more of an atomic data repository function, a semantic identification function, a filtering function, a refinement and selection function, a design interface function, an analysis function, a targeting function, a primary key provision function, a foreign key provision function, a table normalization function, a source to target mapping function, an automatic generation of data integration job functionality, a defect detection function, a performance measurement function, a data deduplication function, a statistical analysis function, a data reconciliation function, a library function, a version management function, a parallel execution function, a partitioning function, a partitioning and repartitioning function, an interface function, a synchronization function, a metadata directory function, a graphical impact depiction function, a hub repository function, a packaged application connectivity kit functionality, an industry-specific data model storage function, a template function, a business rule function, a validation table function, a business metric function, a target database definition function, a mainframe data profiling function, a batch processing function, a cross-table analysis function, a relationship analysis function, a data definition language code generation function, a data integration job design function, a data integration job deployment function and a data integration job development function.
- 15 15. The system of claim 12 wherein the data integration function includes a metabroker function, the metabroker function maintaining the semantics of a data integration function across multiple data integration platforms.
- 20 16. The system of claim 12 wherein the data integration function includes a refinement and selection function, wherein the refinement and selection function allows the method to distinguish items based on differentiating characteristics.
- 25 17. The system of claim 12 wherein the module analyzes contents of a database.
18. The system of claim 12 wherein the module analyzes one or more of a column of the database, a table of the database, and a row of the database.
- 30 19. The system of claim 12 wherein the module recommends a target database for an item of data.
20. The method of claims 12-19 wherein the module is accessible by one or more of an Enterprise Java Bean, a Java Messaging Service, and a web service.
21. The method of claim 12-20 wherein the module includes a real time service.
22. A computer program product comprising a computer useable medium including computer readable program code, wherein the computer readable program code when executed on one or more computers causes the one or more computers to:
 - provide a module for a data integration function;
 - provide a registry of services;
 - provide an interface for the module; and
 - 40 identify the module in the registry, wherein the module can be accessed as a service in a services oriented architecture.

23. A method, comprising:
 taking a module of computer executable code that performs a data integration task; and
 deploying the module as a service in a services oriented architecture.
- 5 24. The method of claim 23 wherein the data integration task includes a common service that can be accessed by other services within the services oriented architecture.
25. The method of claim 23 wherein the common service is selected from the group consisting of a monitoring service, a transaction service, a provisioning service, an event management service, a security service, an auditing service and a logging service.
- 10 26. The method of claim 23 wherein the service is a real time service.
27. A system comprising:
 a services oriented architecture platform; and
 a data integration service deployed on the services oriented architecture platform, the data integration service having more than one bindings;
15 wherein the services oriented architecture platform automatically generates a binding from among the more than one bindings based on an access characteristic.
28. The method of claim 27 wherein the access characteristic is based on at least one of a device, an application and a service that invokes the service.
29. The method of claim 27 where the access characteristic is based on a tightness of the binding between an
20 invoking entity and the service.
30. The method of claim 27 further comprising invoking the service, wherein an entity invoking the service is at least one of a client, an application, a device and a service.
31. The method of claim 27 wherein the data integration service includes one or more of data extraction, data transformation, data loading, metadata management, data profiling, data auditing, data cleansing, data quality, data
25 matching, data migration, and filtering.
32. The method of claim 27 wherein the data integration service is a real time service.
33. A method comprising:
 providing a services oriented architecture for deploying services;
 deploying a service within the services oriented architecture, the service having a number of bindings
30 available in the services oriented architecture; and
 automatically selecting one of the number of bindings for the service.
34. The method of claim 33 wherein automatically selecting includes making a rules-based selection of one of the number of bindings for the service.
35. The method of claim 33 wherein automatically selecting includes making a selection based upon an access
35 characteristic of an entity invoking the service.
36. The method of claim 35 wherein the entity includes one or more of a device, an application, and a service.
37. The method of claim 33 further comprising providing a client invocation framework for invoking the service.
38. The method of claim 37 wherein the client invocation framework includes an interface for dynamically
40 calling any one of the number of bindings for the service.
39. The method of claim 38 further comprising generating a plurality of proxies for the interface.

40. The method of claim 39 wherein the plurality of proxies include at least one C++ proxy and at least one C# proxy.
41. The method of claim 37 wherein the client invocation framework is language independent.
42. The method of claim 37 wherein the client invocation framework is proxy-based.
- 5 43. The method of claim 33 wherein the service includes at least one data integration function.
44. A computer program product embodied in a computer readable medium including:
computer executable code for providing a services oriented architecture for deploying services;
computer executable code for deploying a service within the services oriented architecture, the service
having a number of bindings available in the services oriented architecture; and
10 computer executable code for automatically selecting one of the number of bindings for the service.
45. A system comprising:
a services oriented architecture for deploying services;
a service within the services oriented architecture, the service having a number of bindings available in the
services oriented architecture; and
15 a software module for automatically selecting one of the number of bindings for the service.

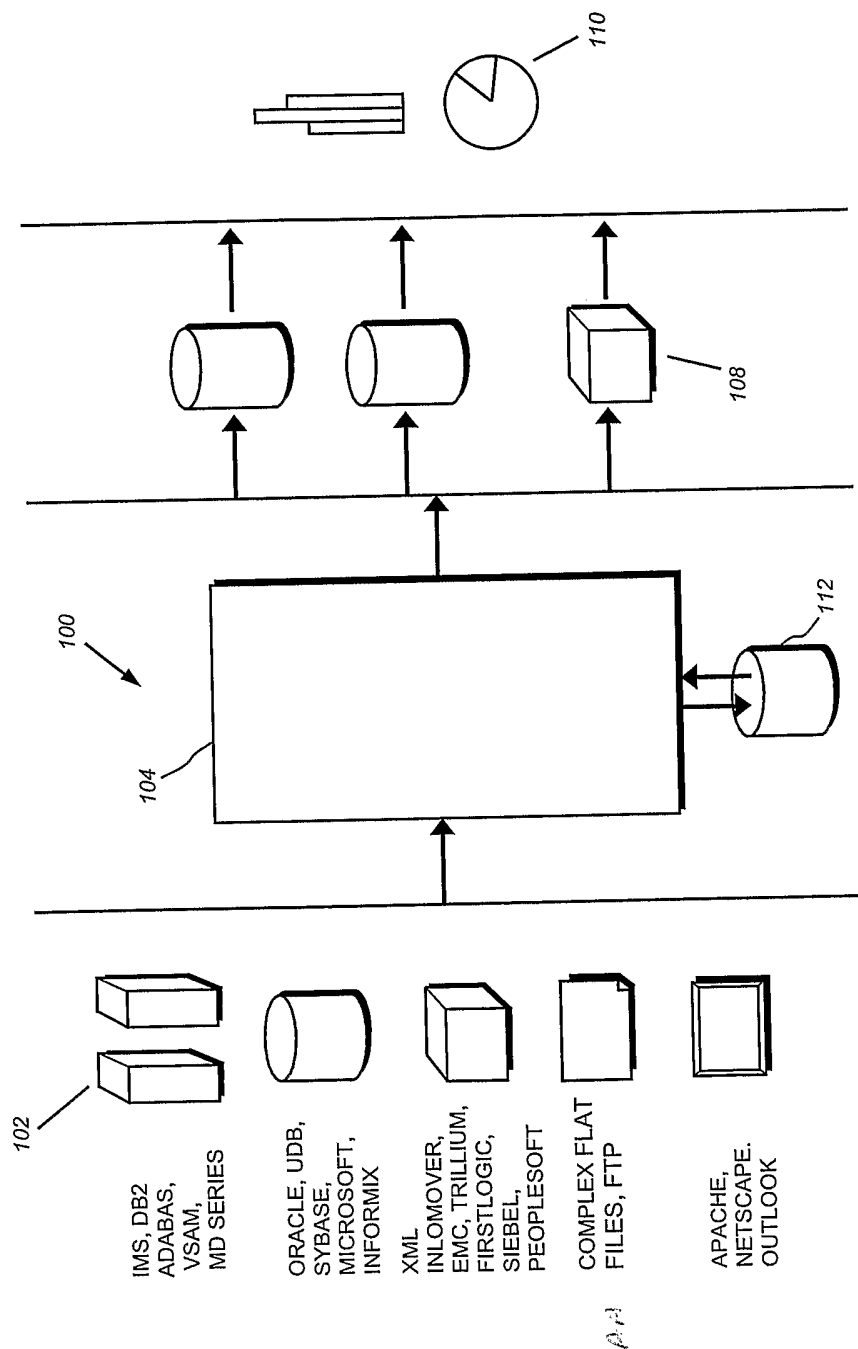


Fig. 1

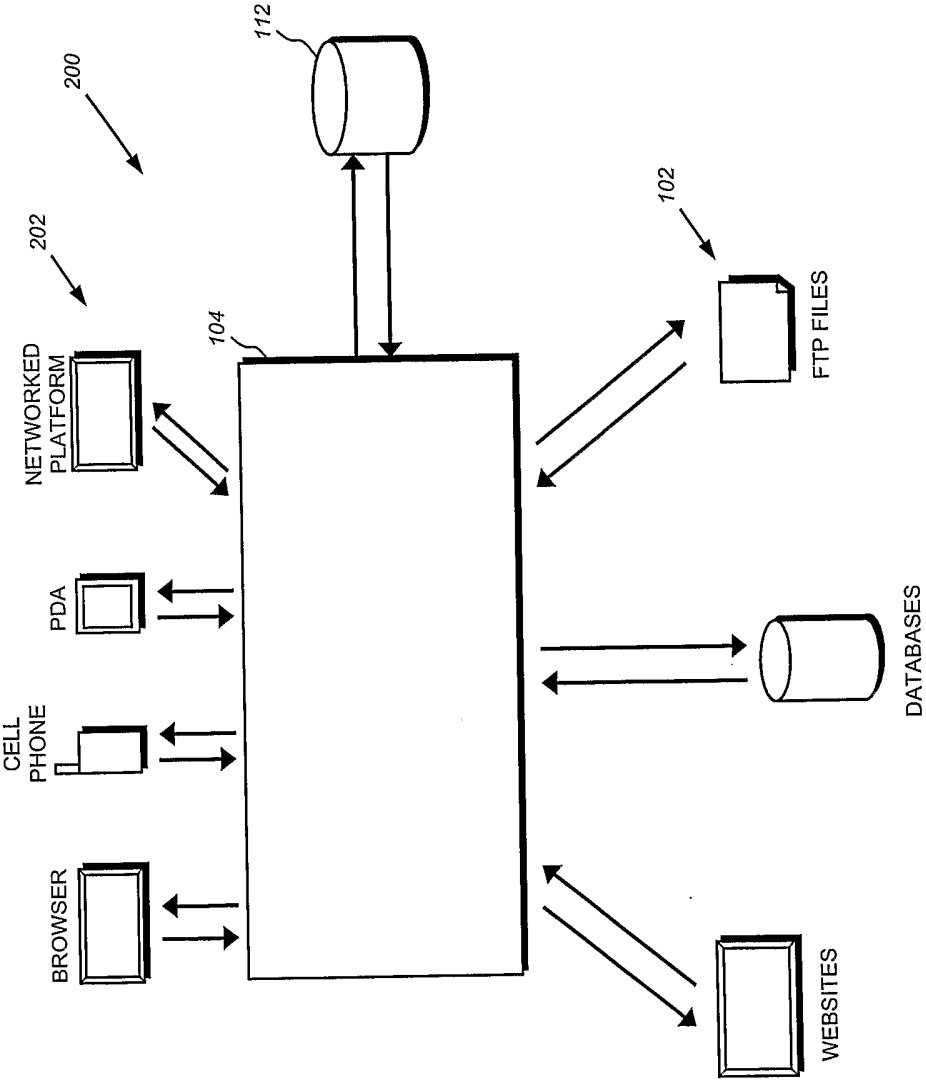


Fig. 2

3/91

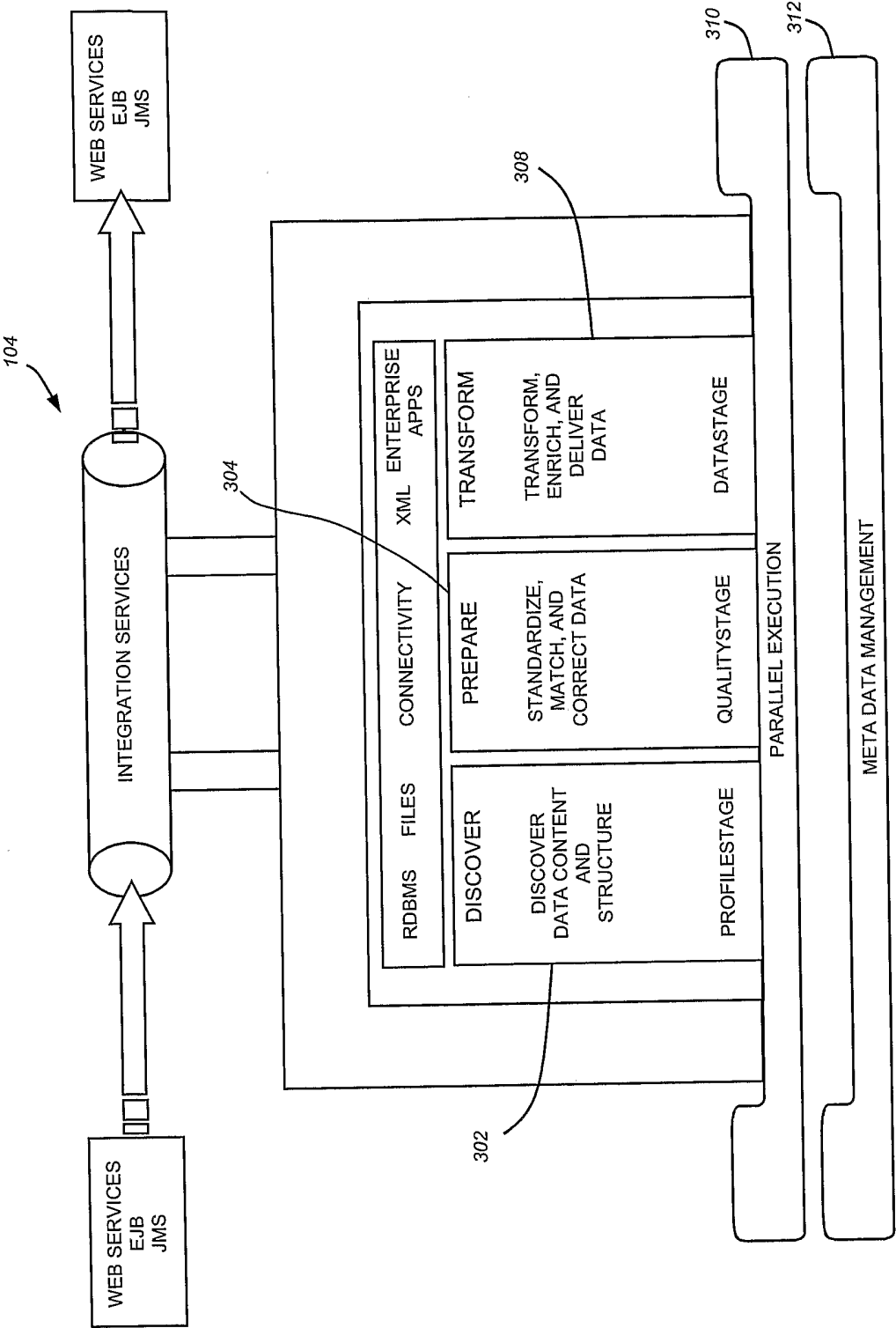
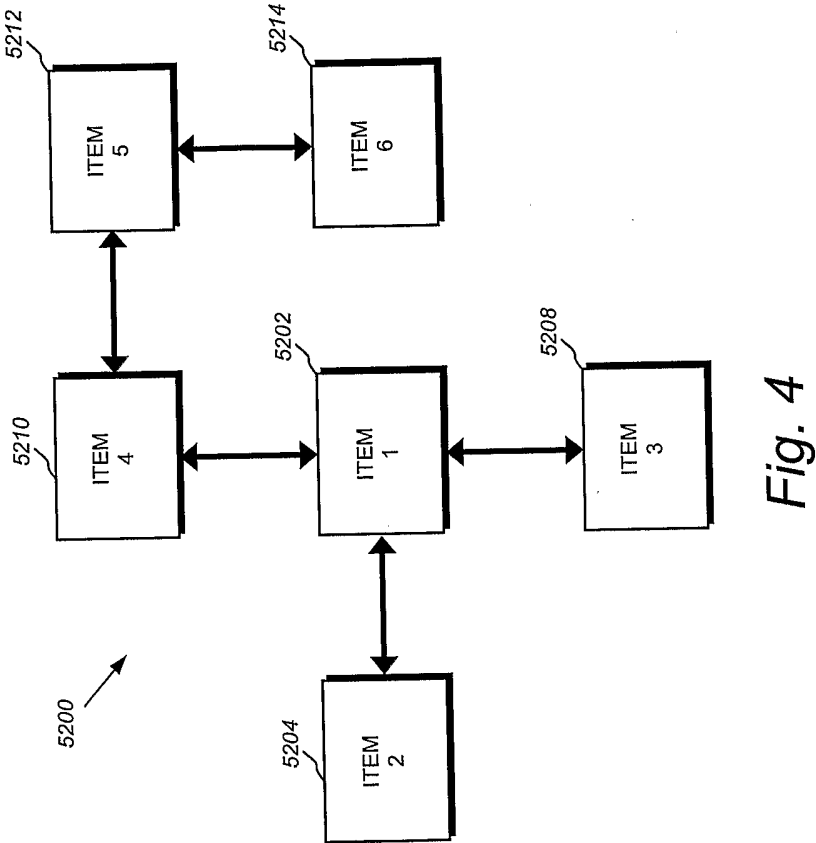
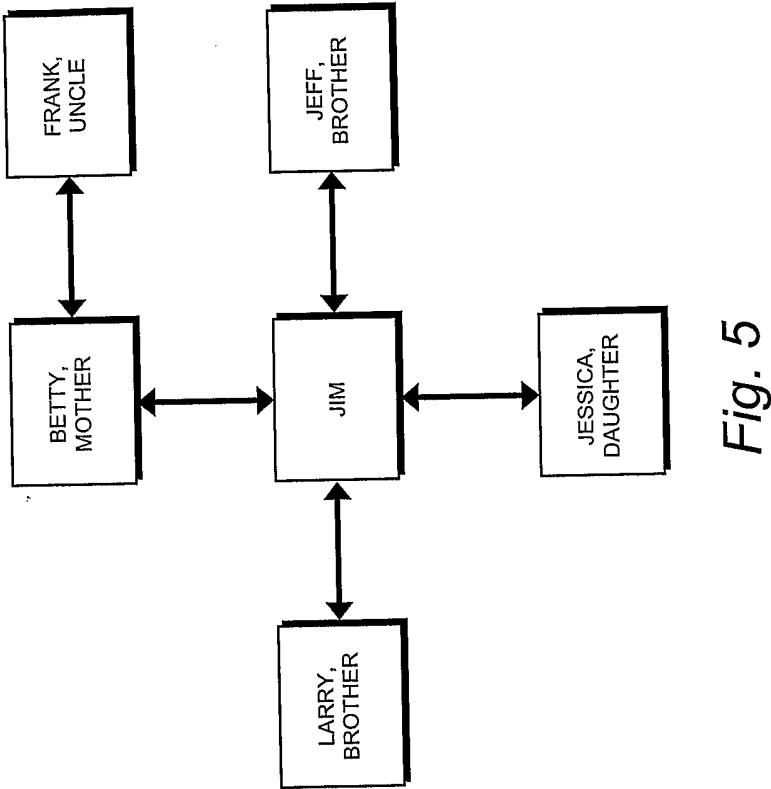


Fig. 3

4/91



5/91



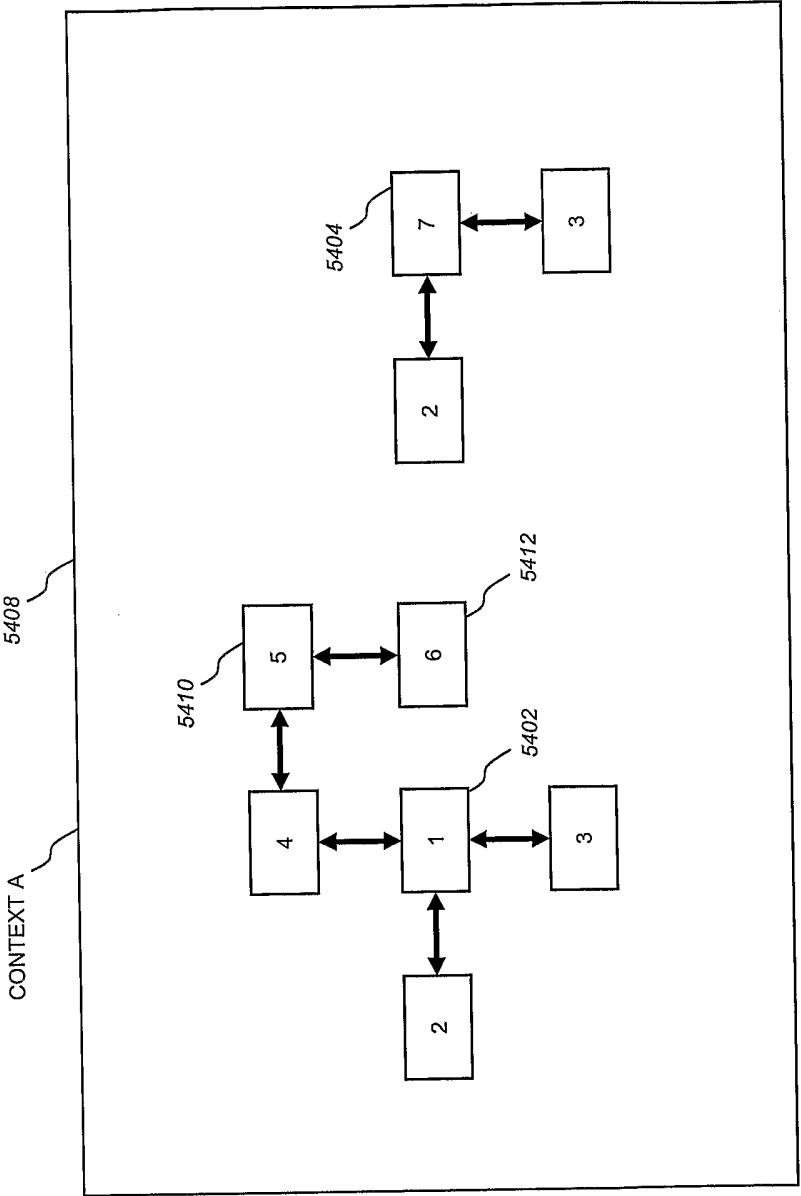


Fig. 6A

7/91

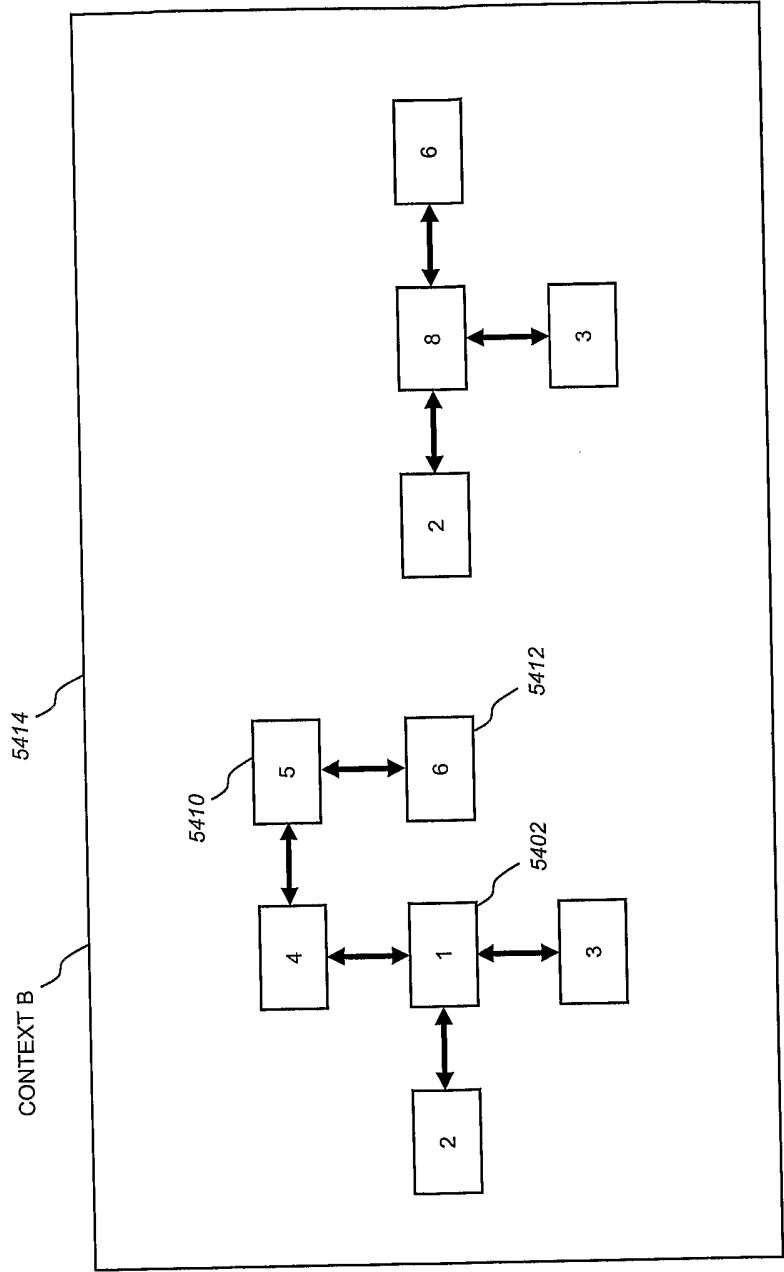


Fig. 6B

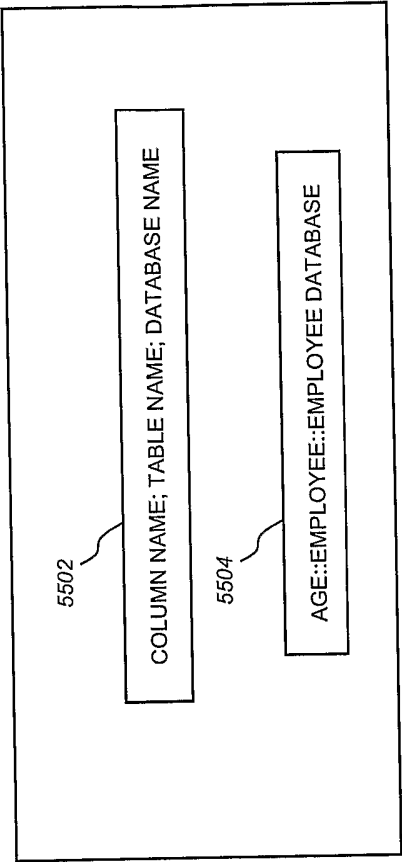


Fig. 7

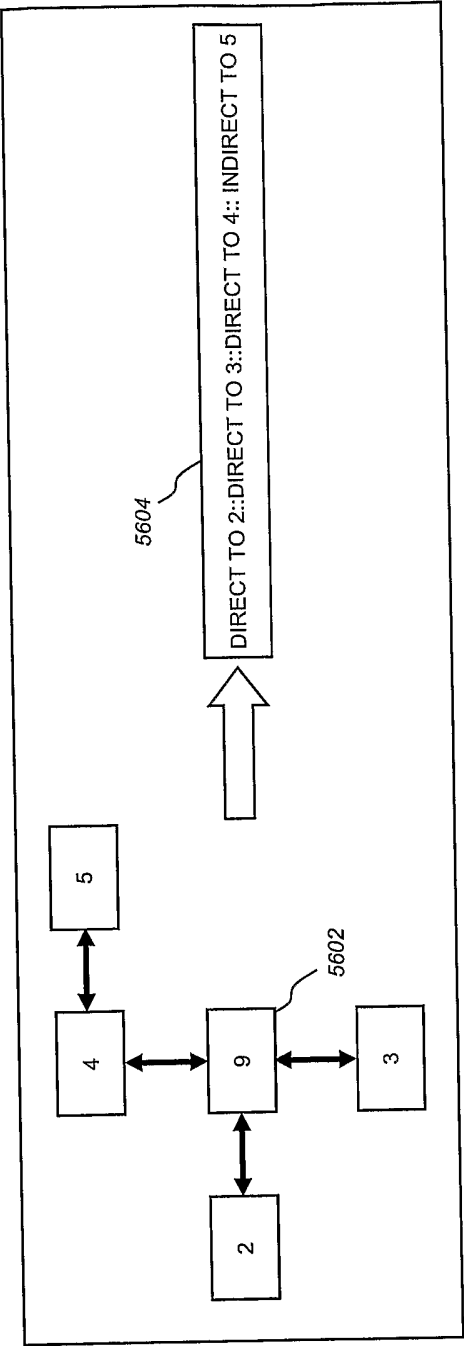


Fig. 8

9/91

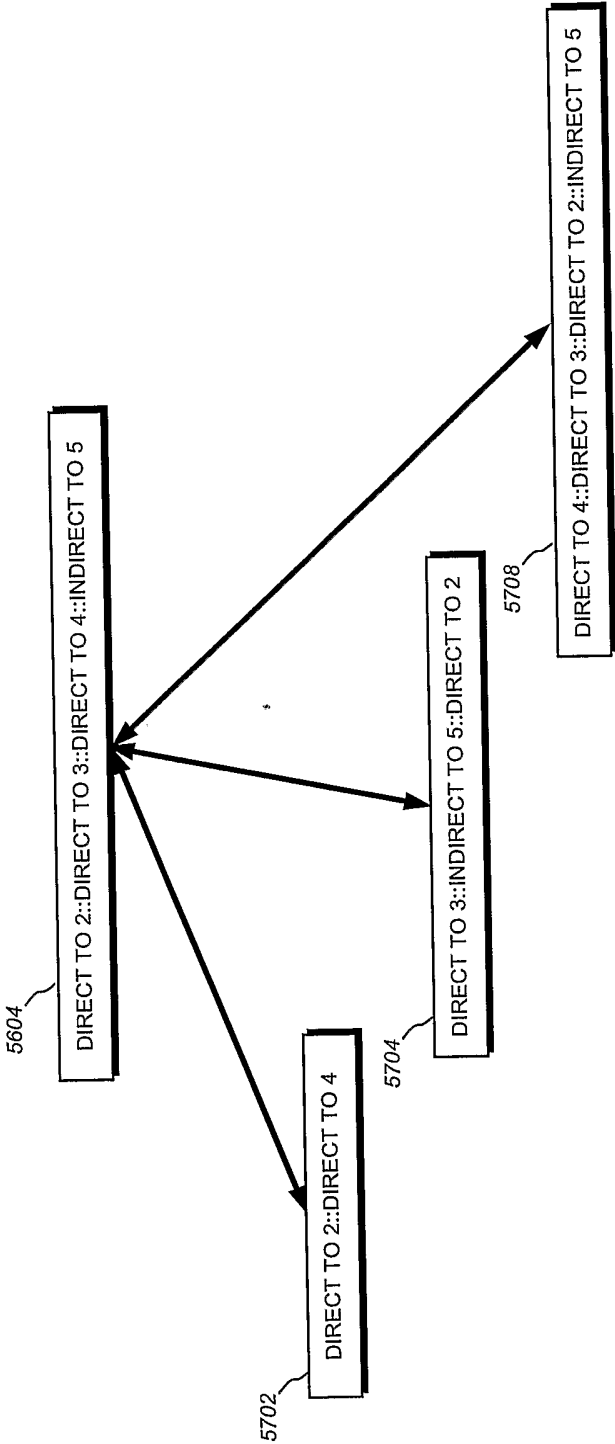


Fig. 9

10/91

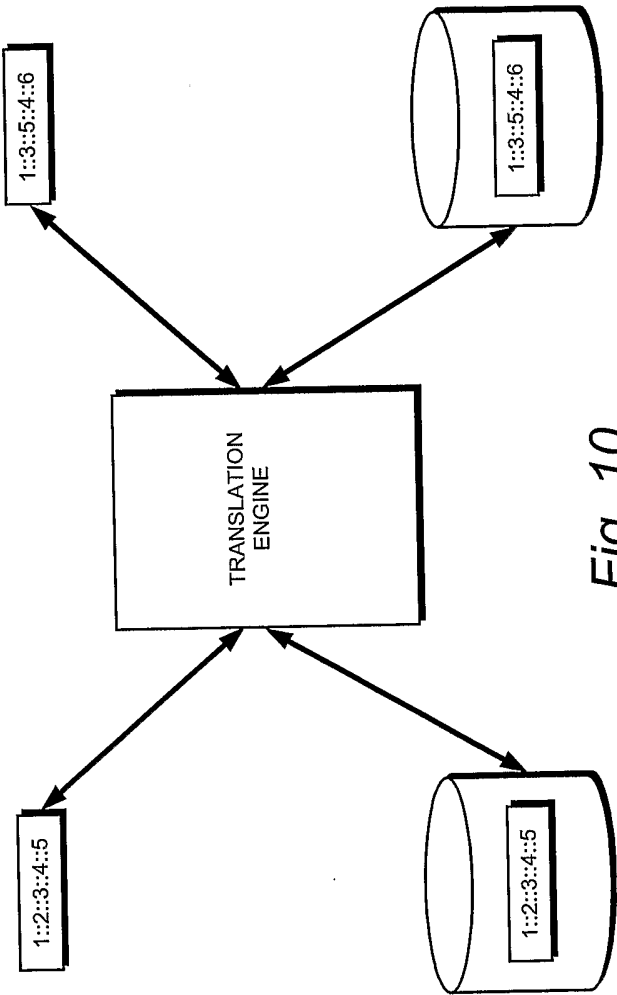


Fig. 10

11/91

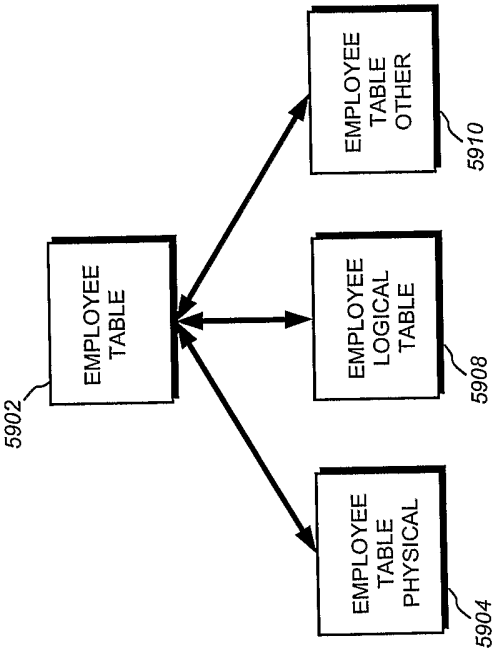


Fig. 11

12/91

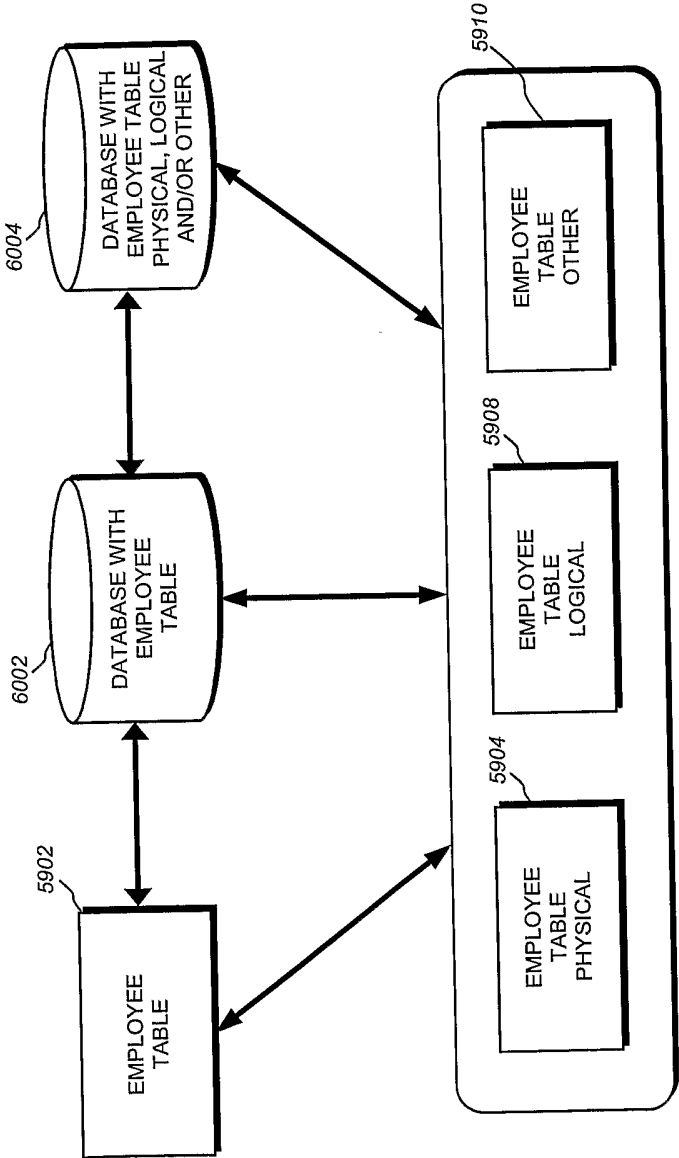


Fig. 12

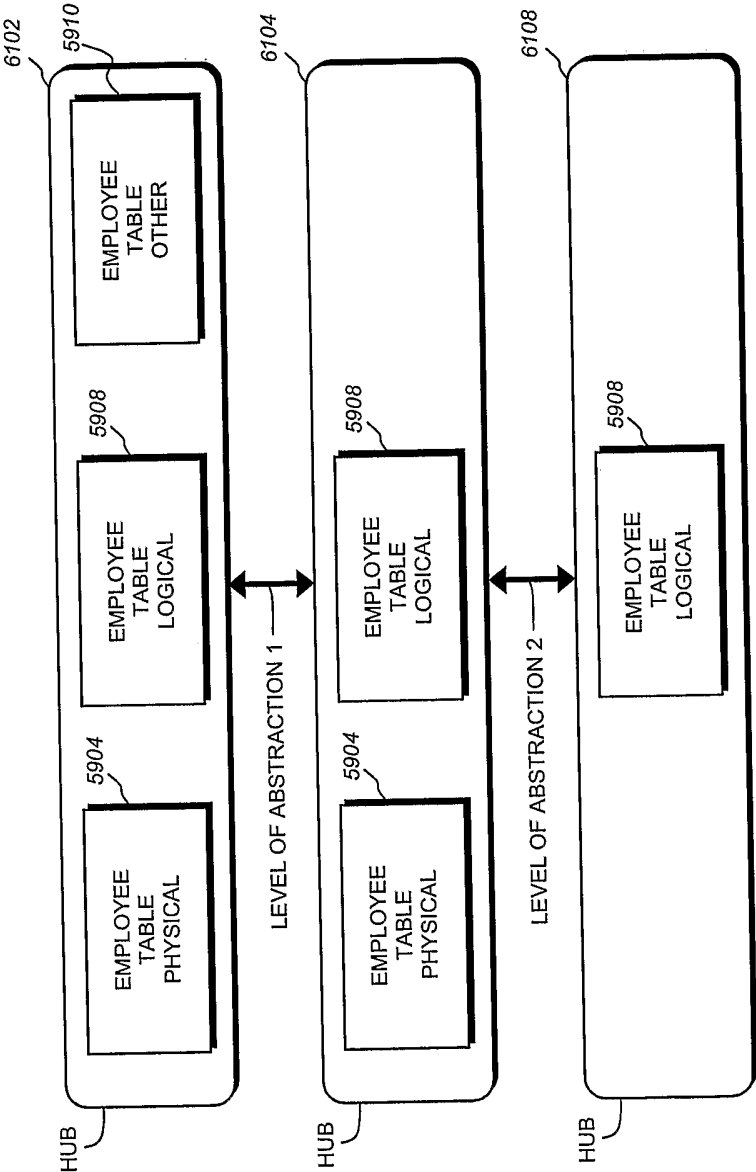


Fig. 13

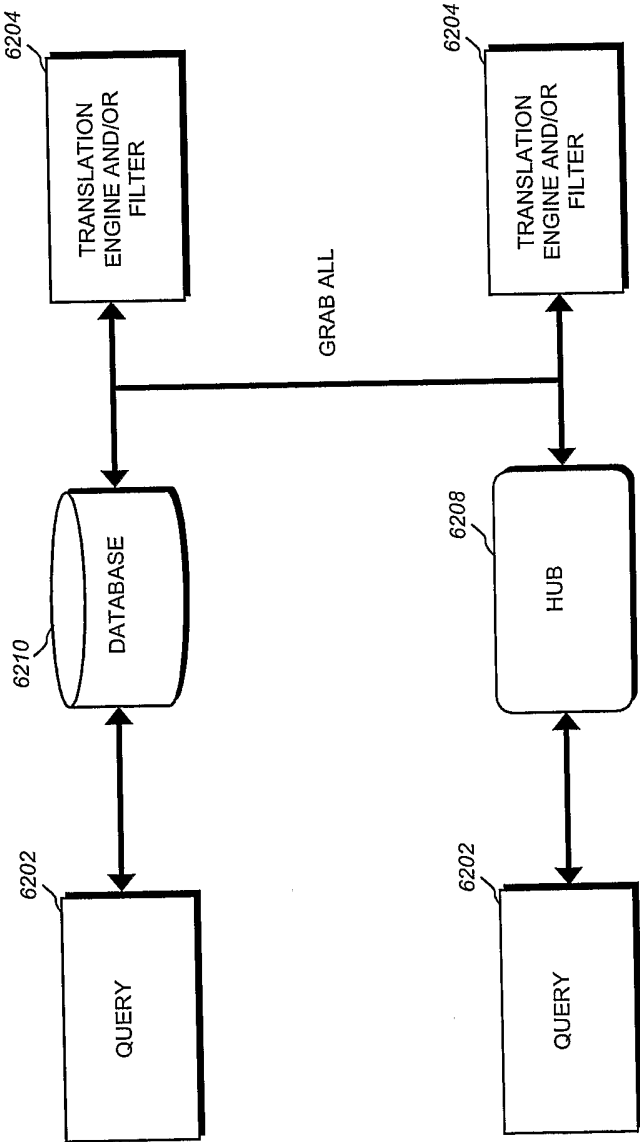


Fig. 14

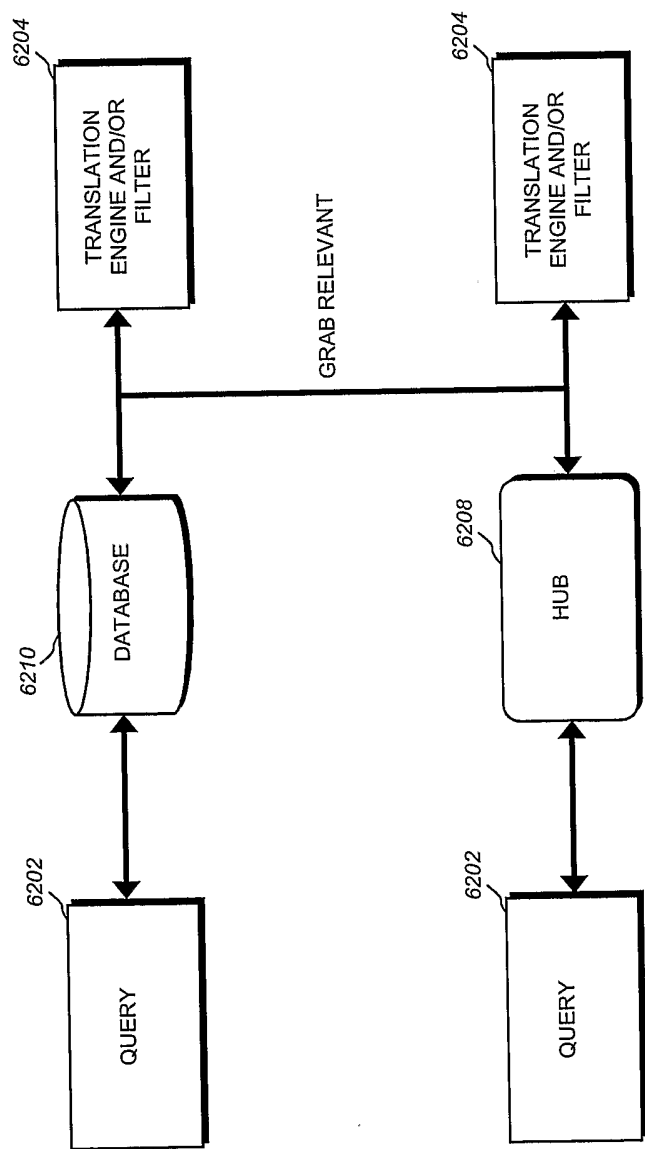


Fig. 15A

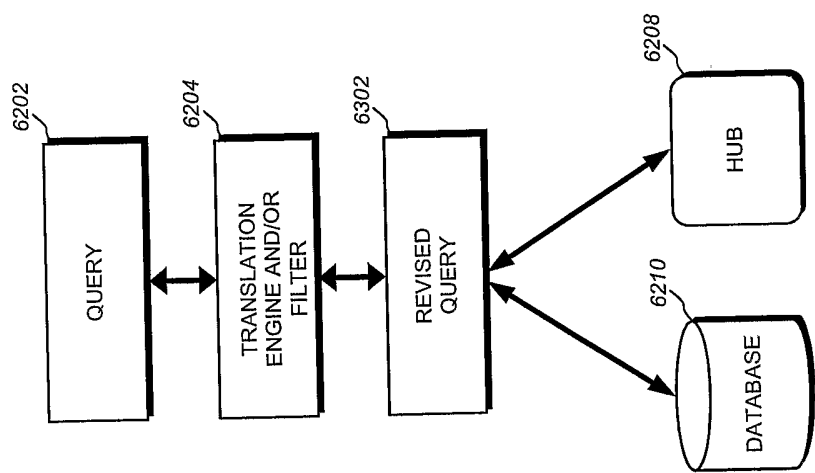


Fig. 15B

17/91

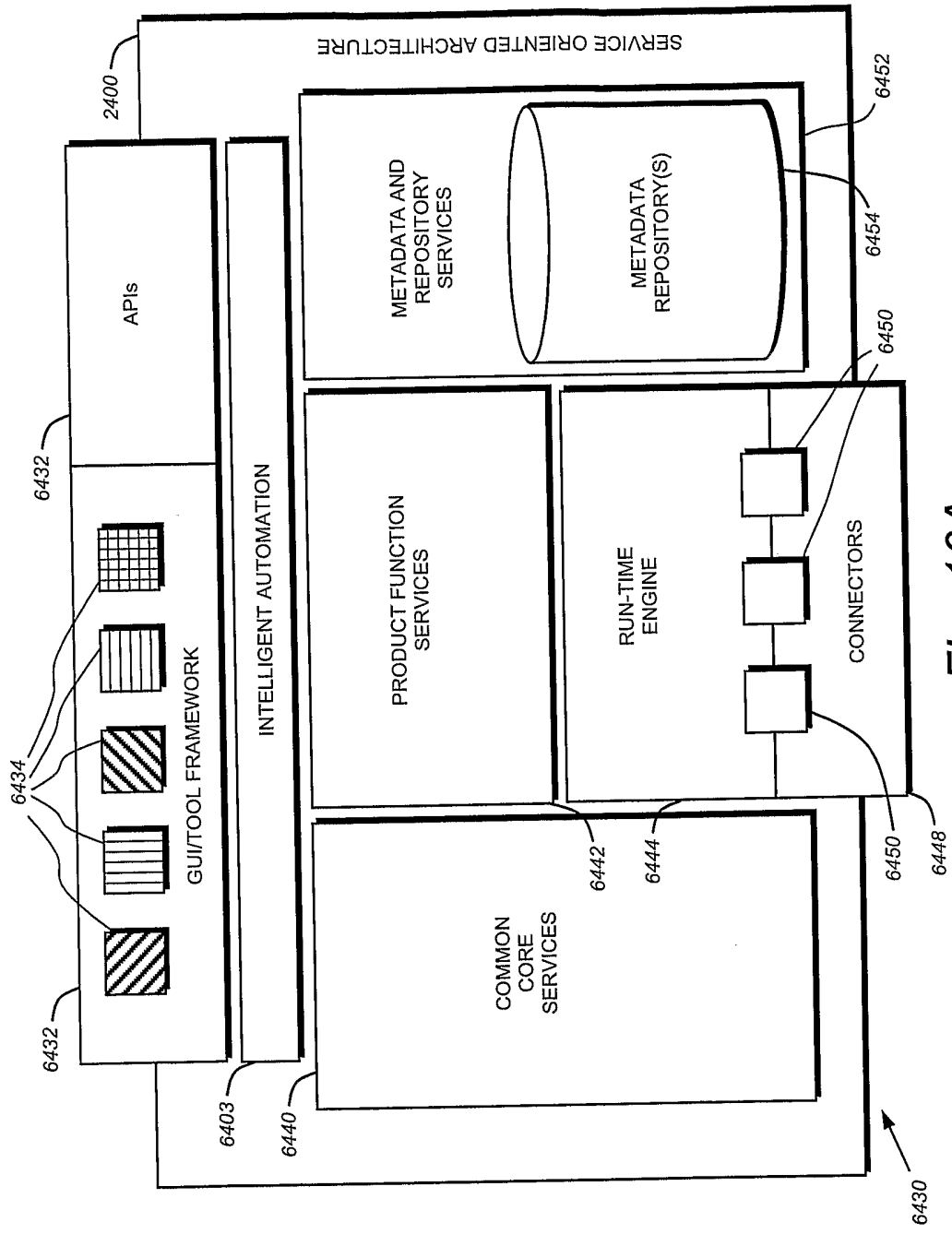


Fig. 16A

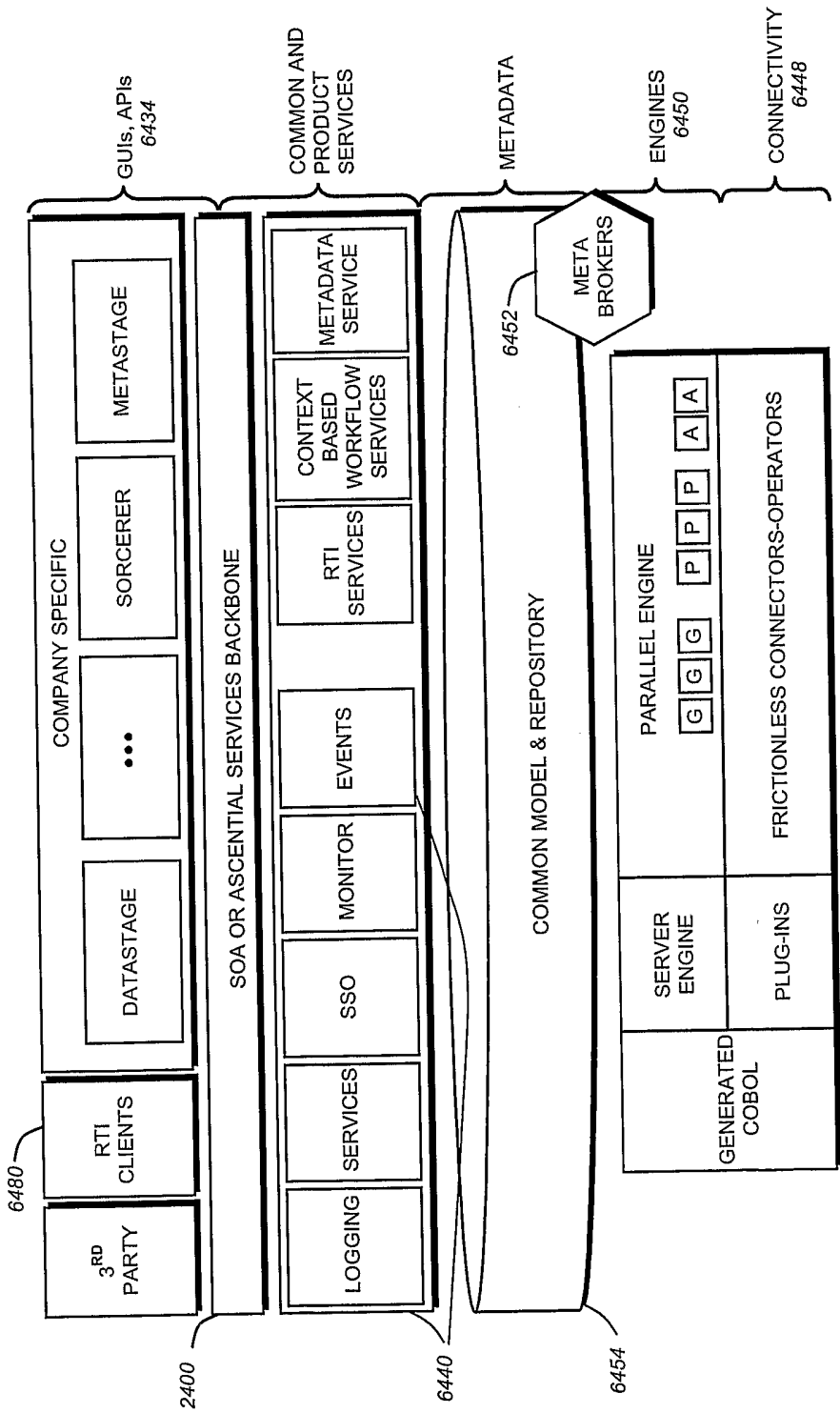


Fig. 16B

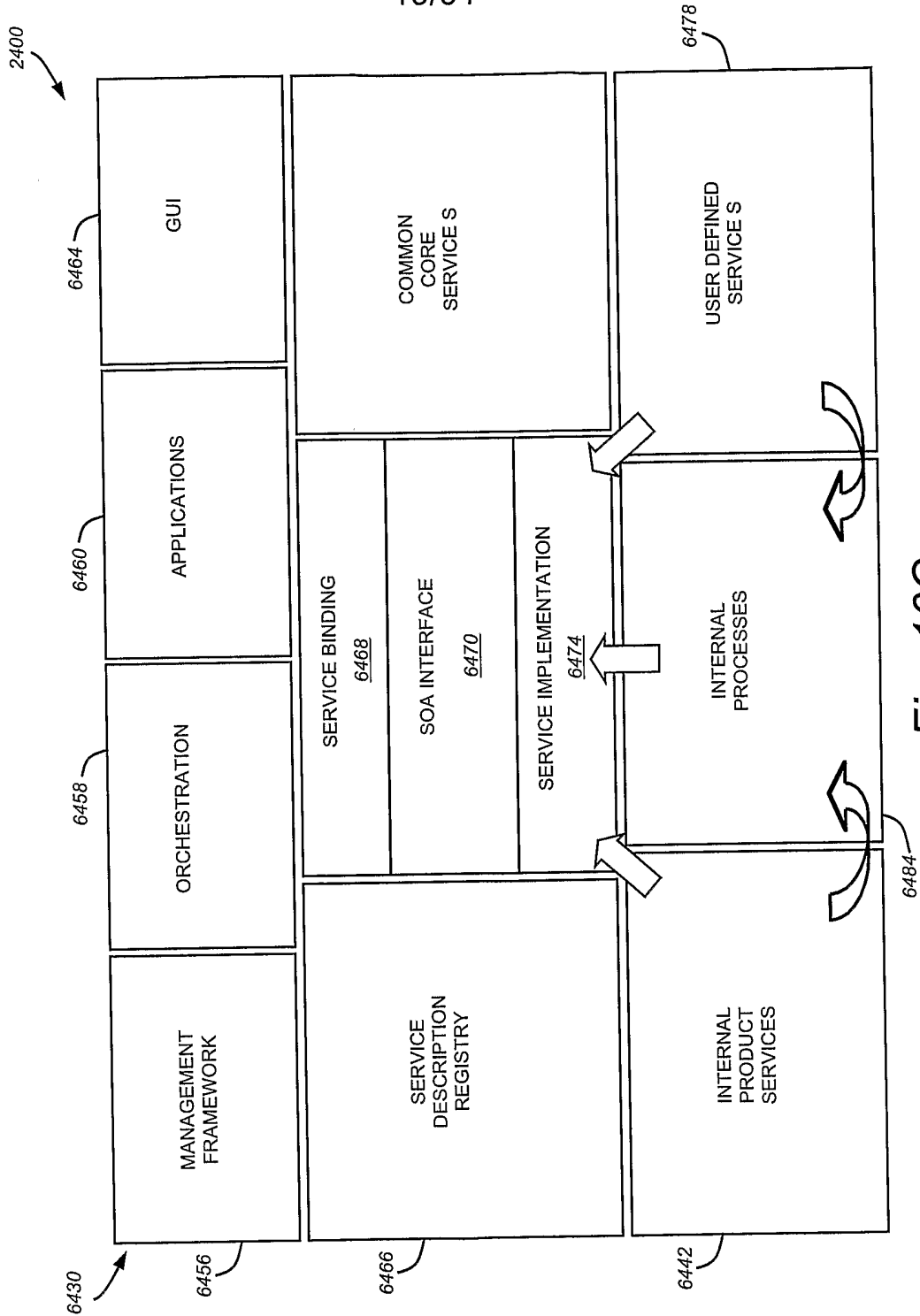


Fig. 16C

20/91

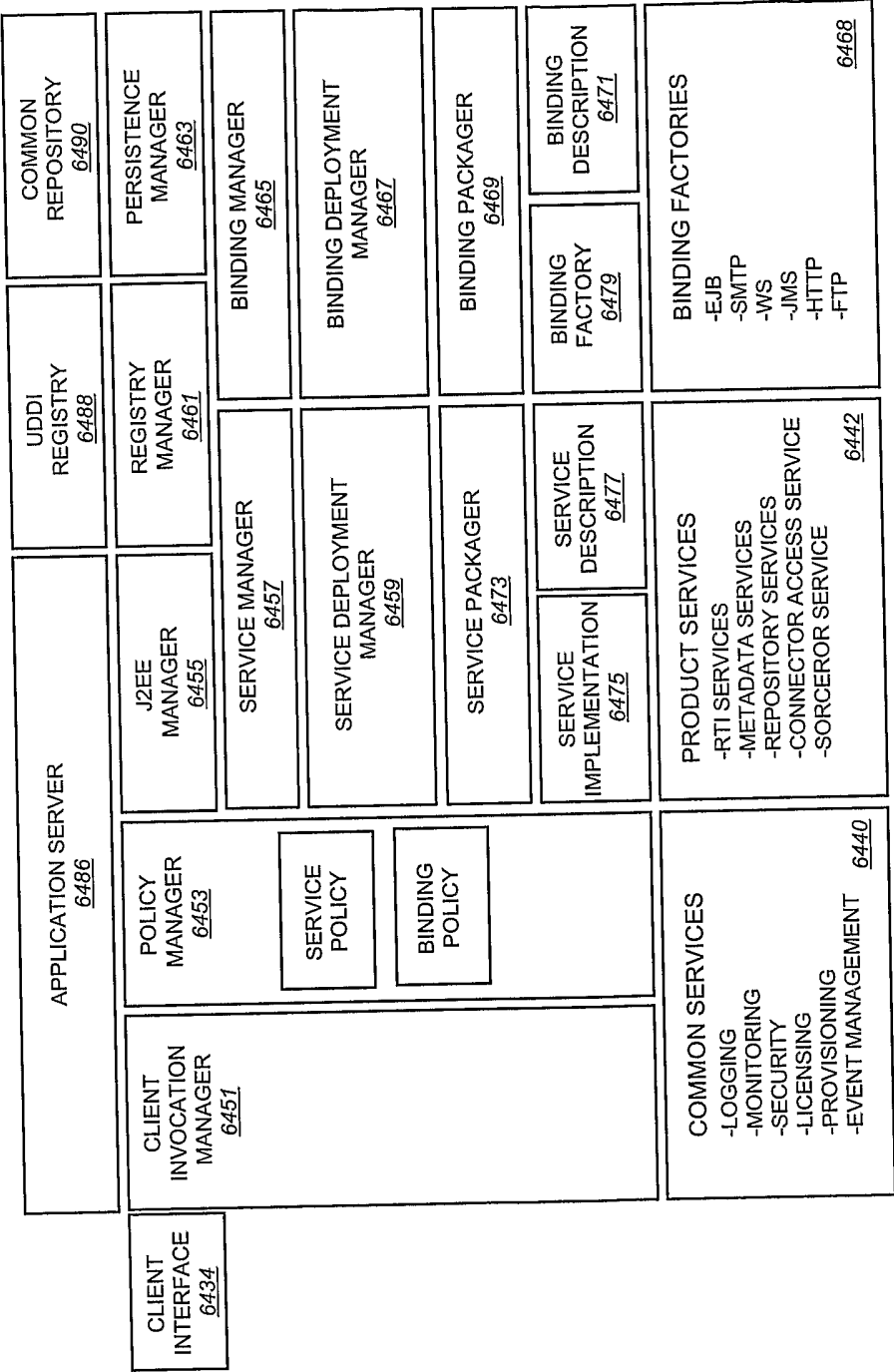


Fig. 16D

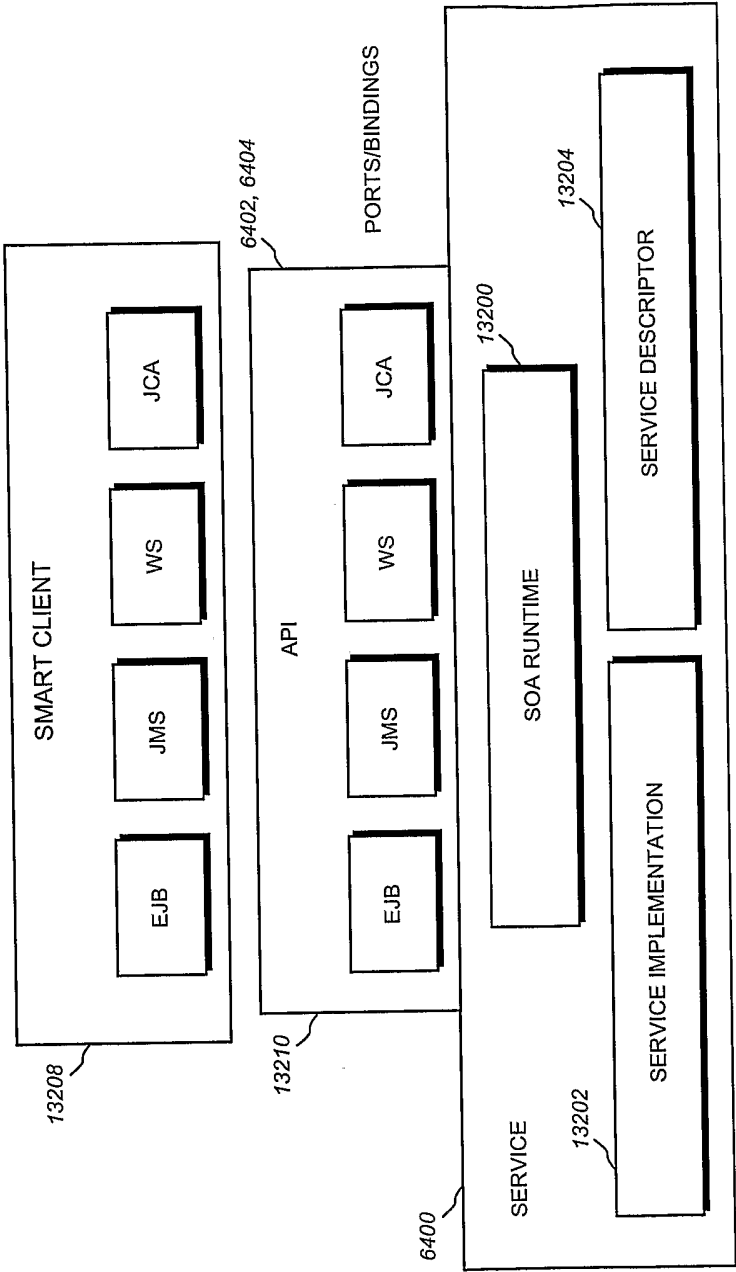


Fig. 16E

22/91

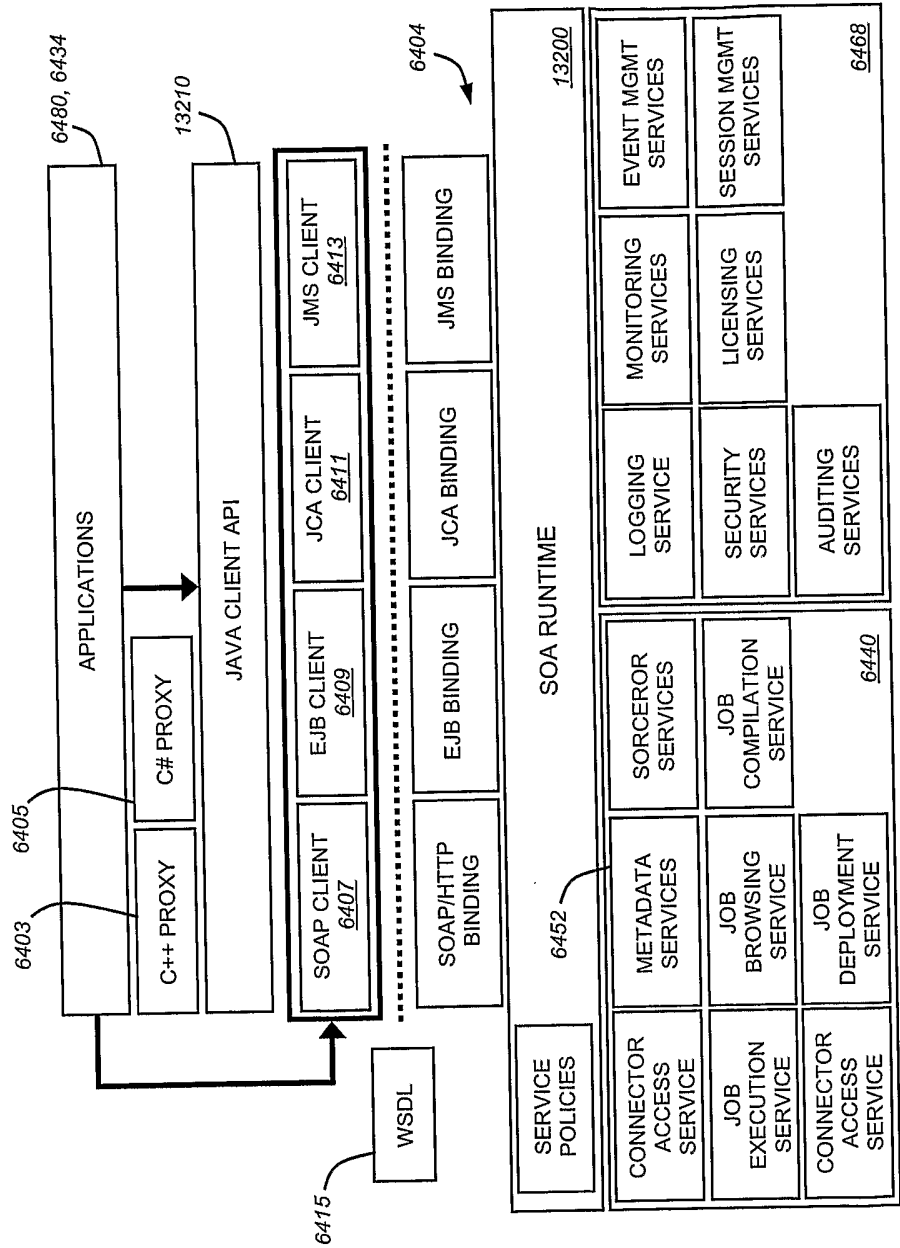


Fig. 16F

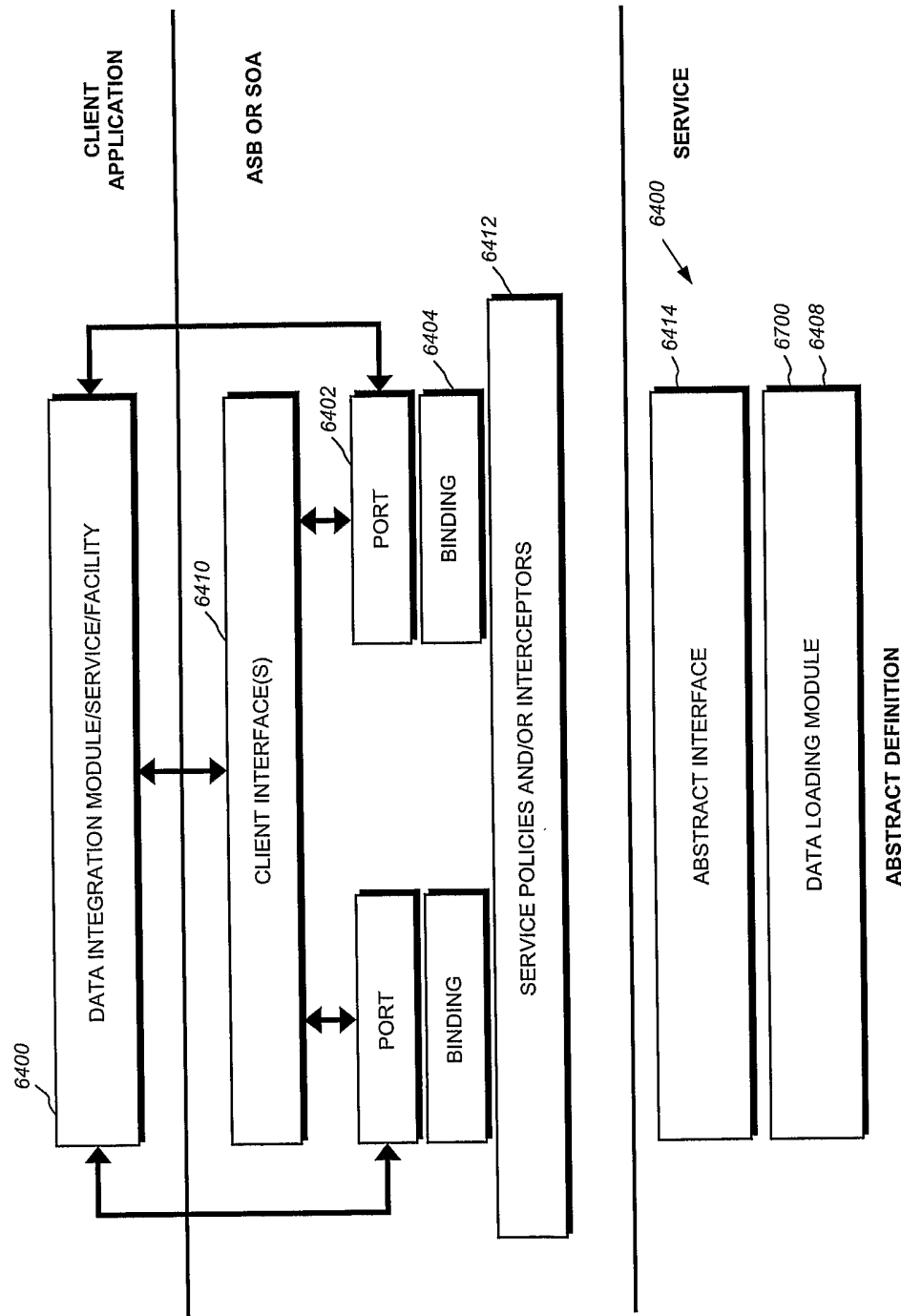


Fig. 16G

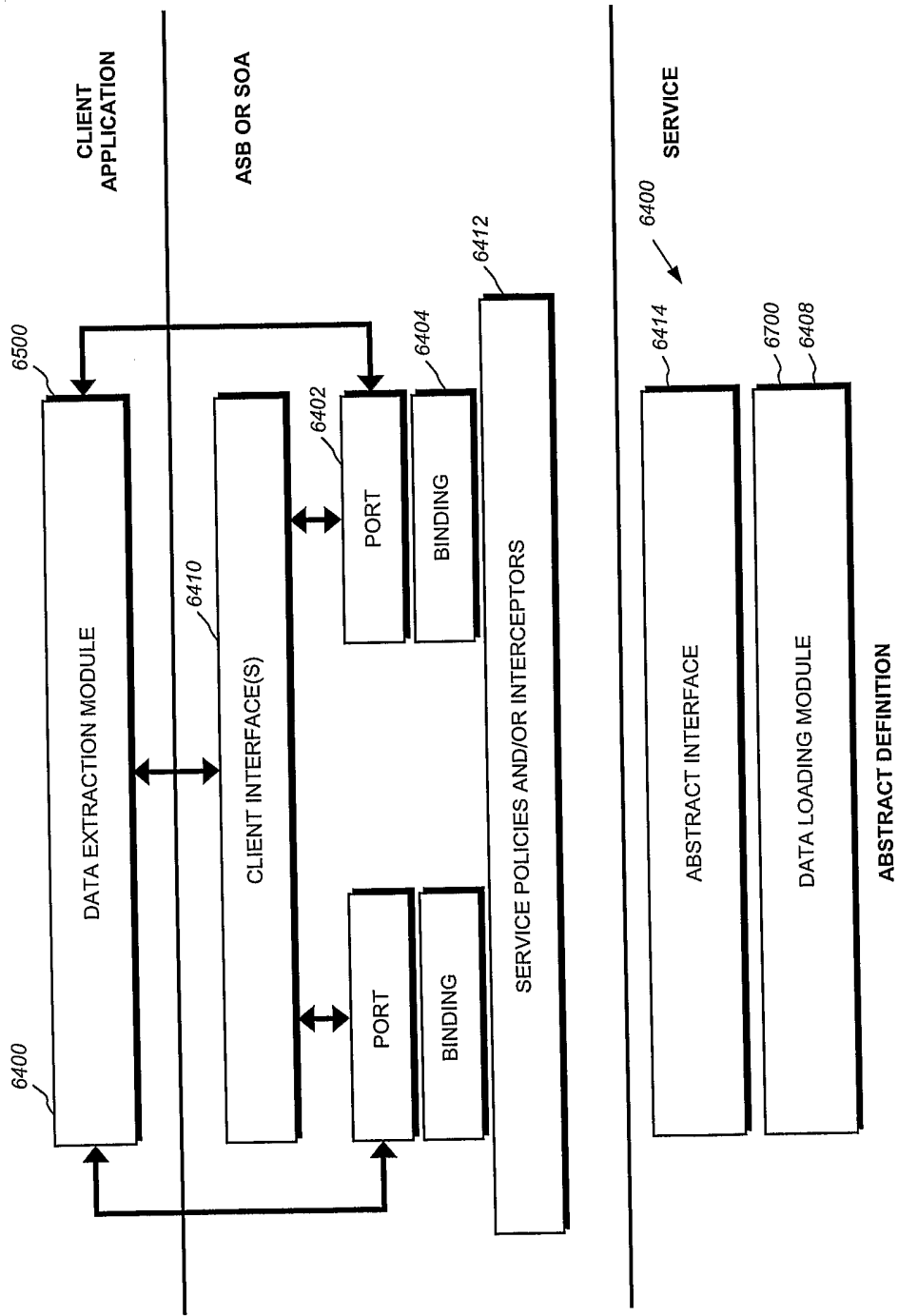


Fig. 17

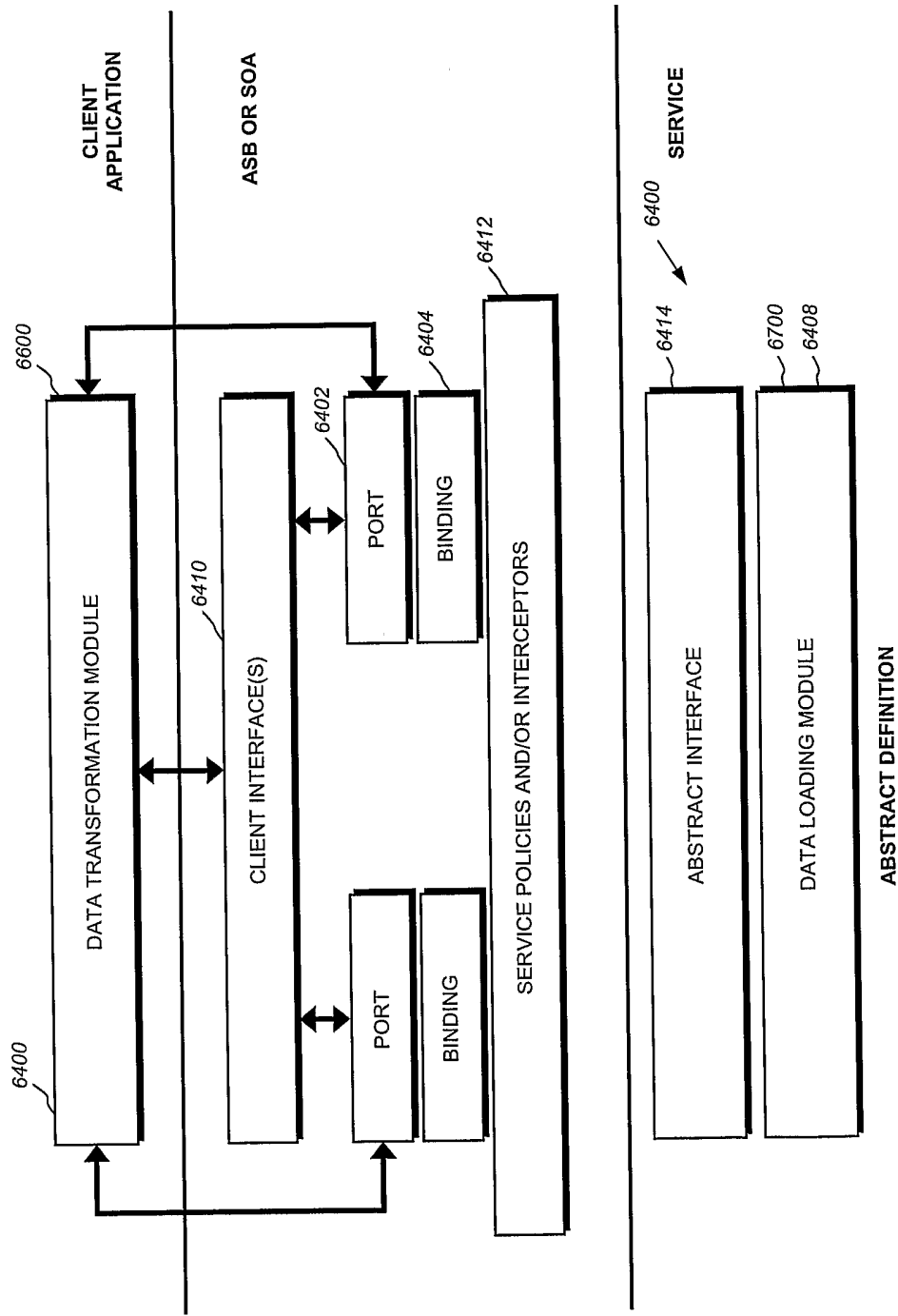


Fig. 18

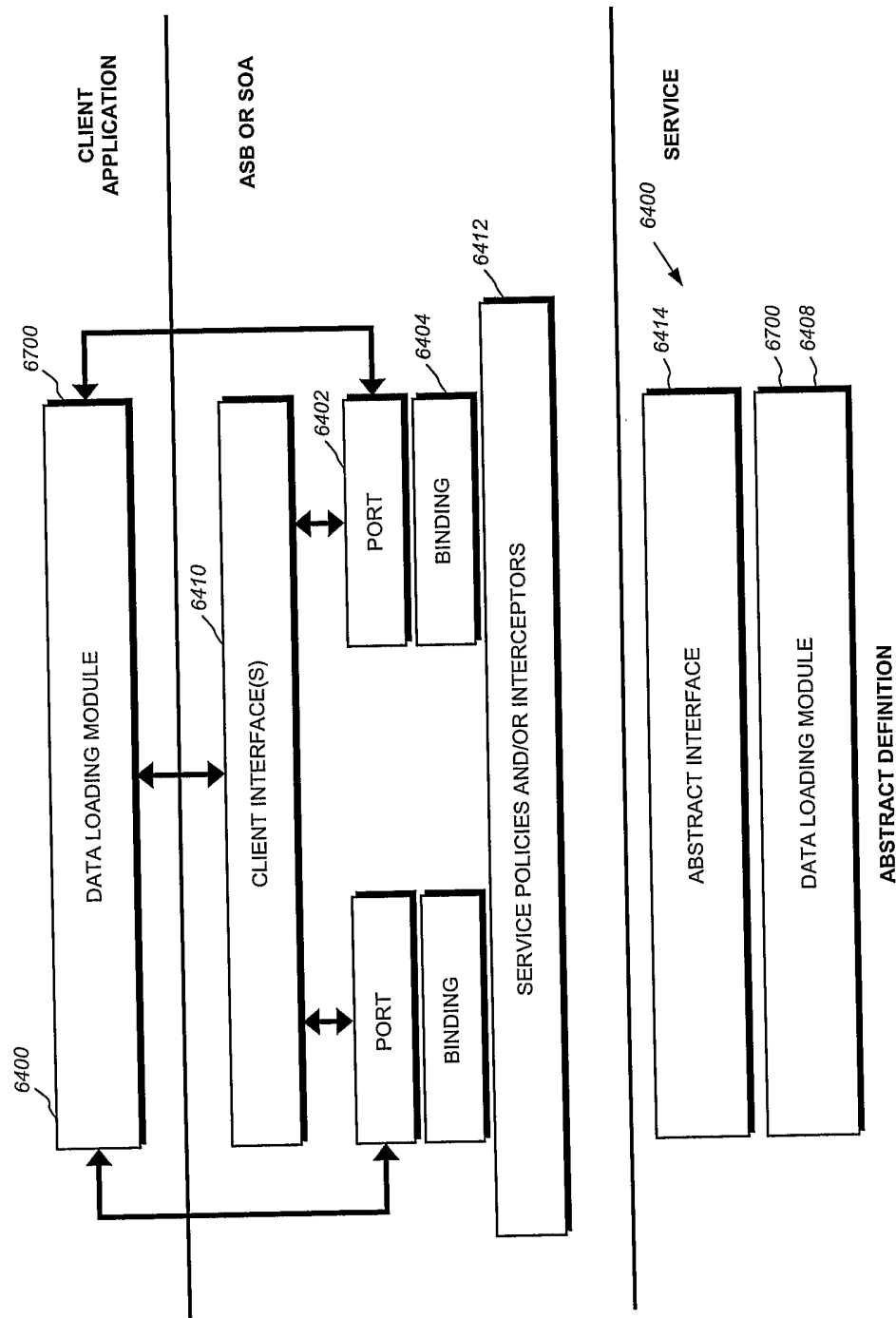


Fig. 19

27/91

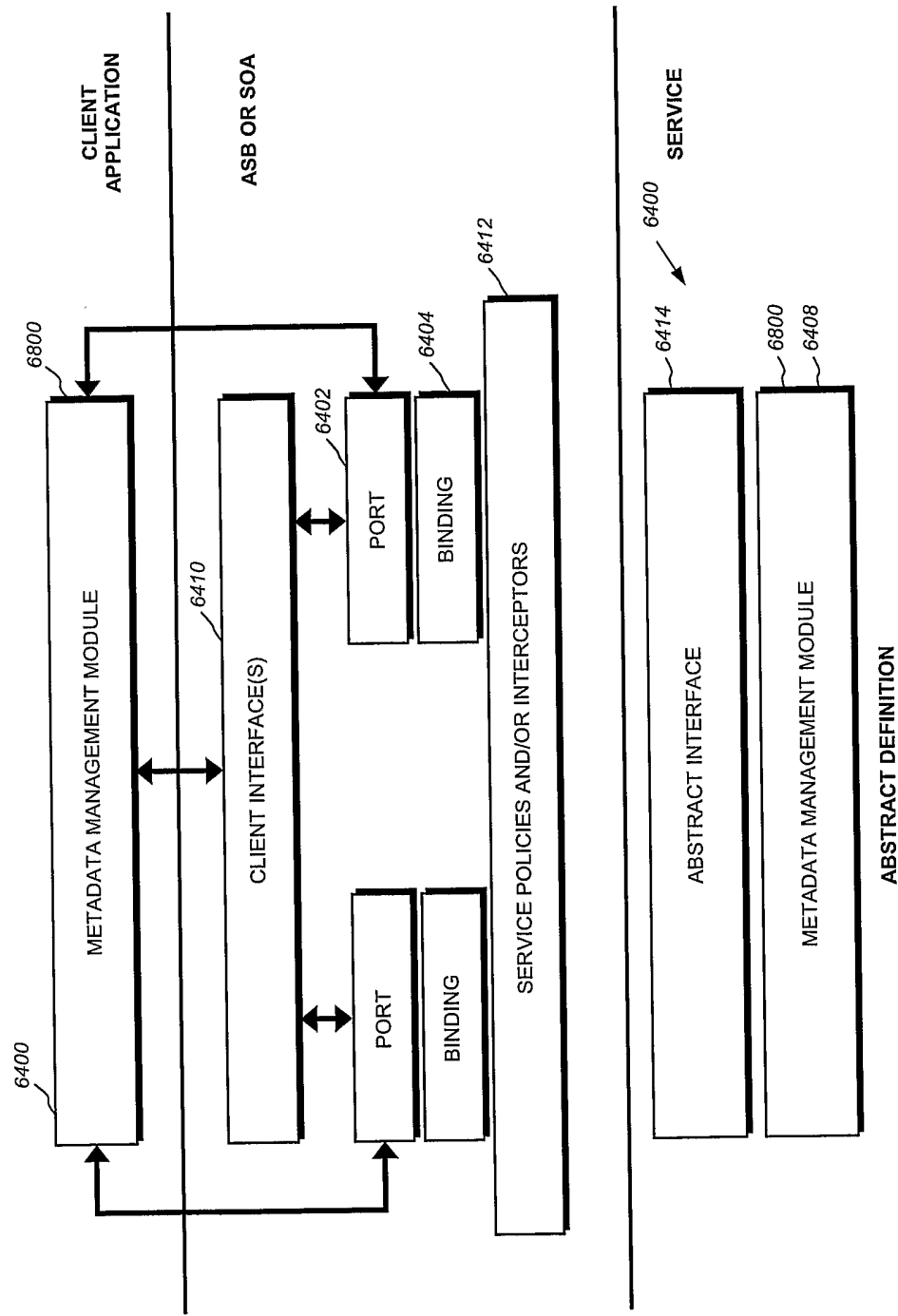


Fig. 20

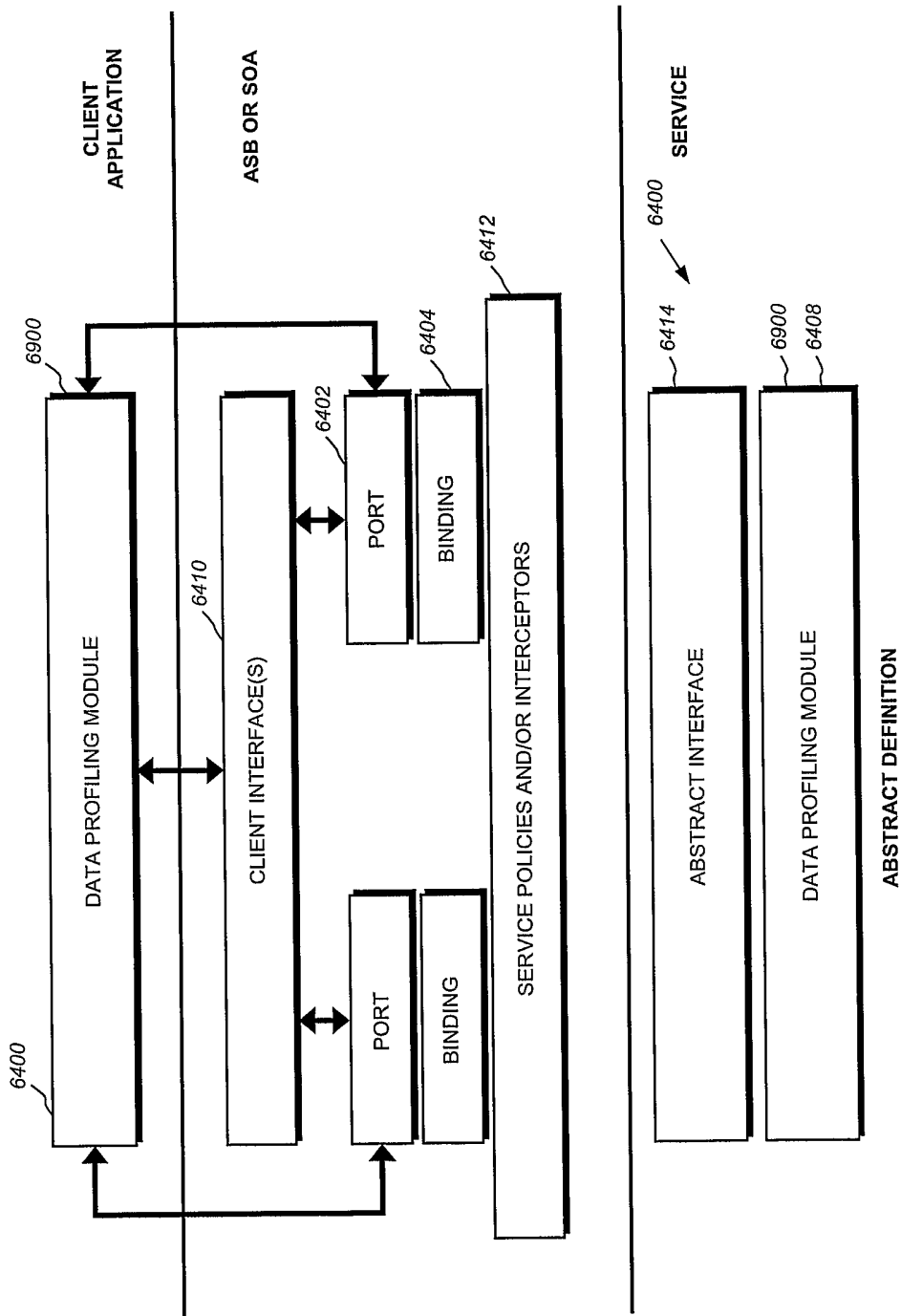


Fig. 21

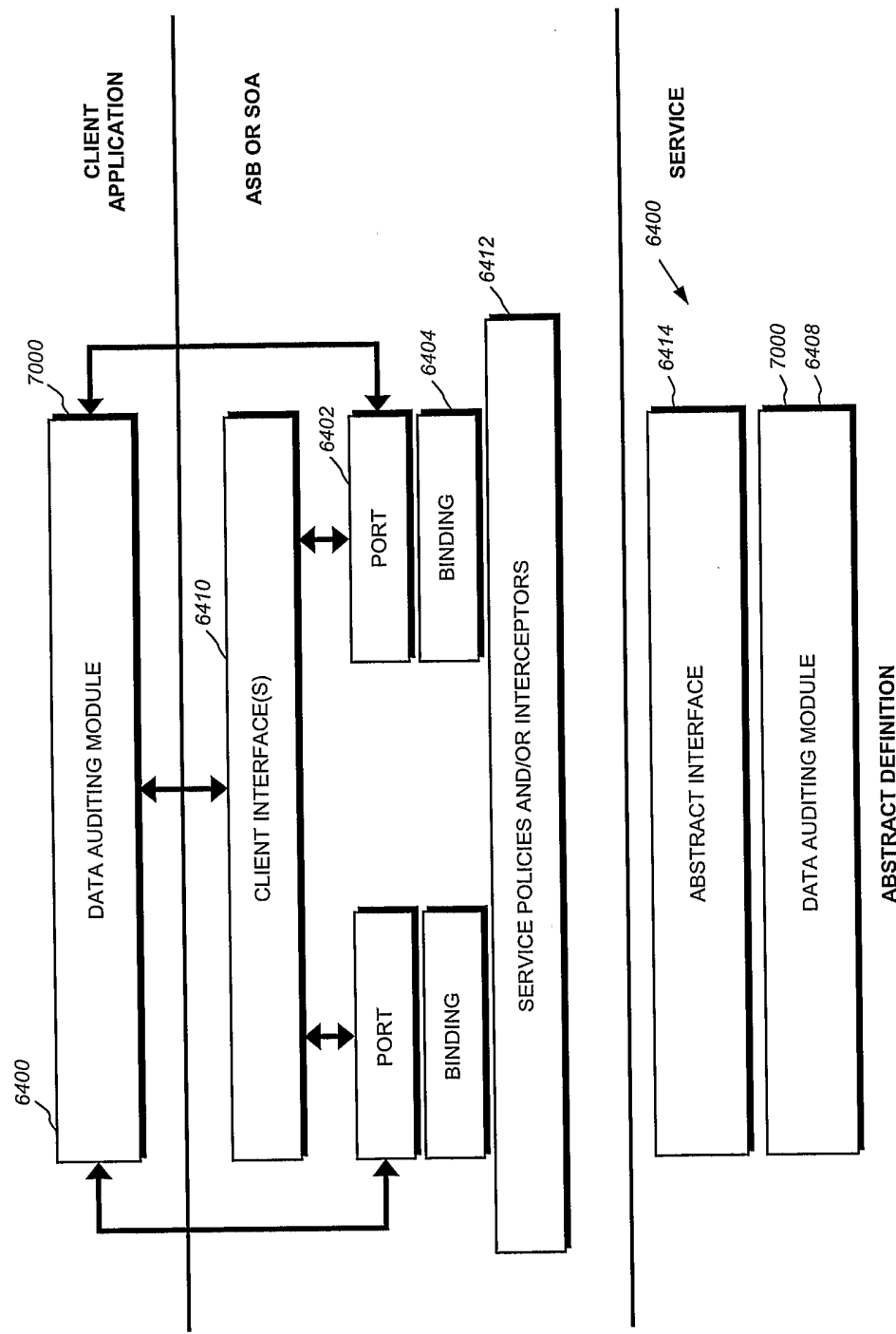


Fig. 22

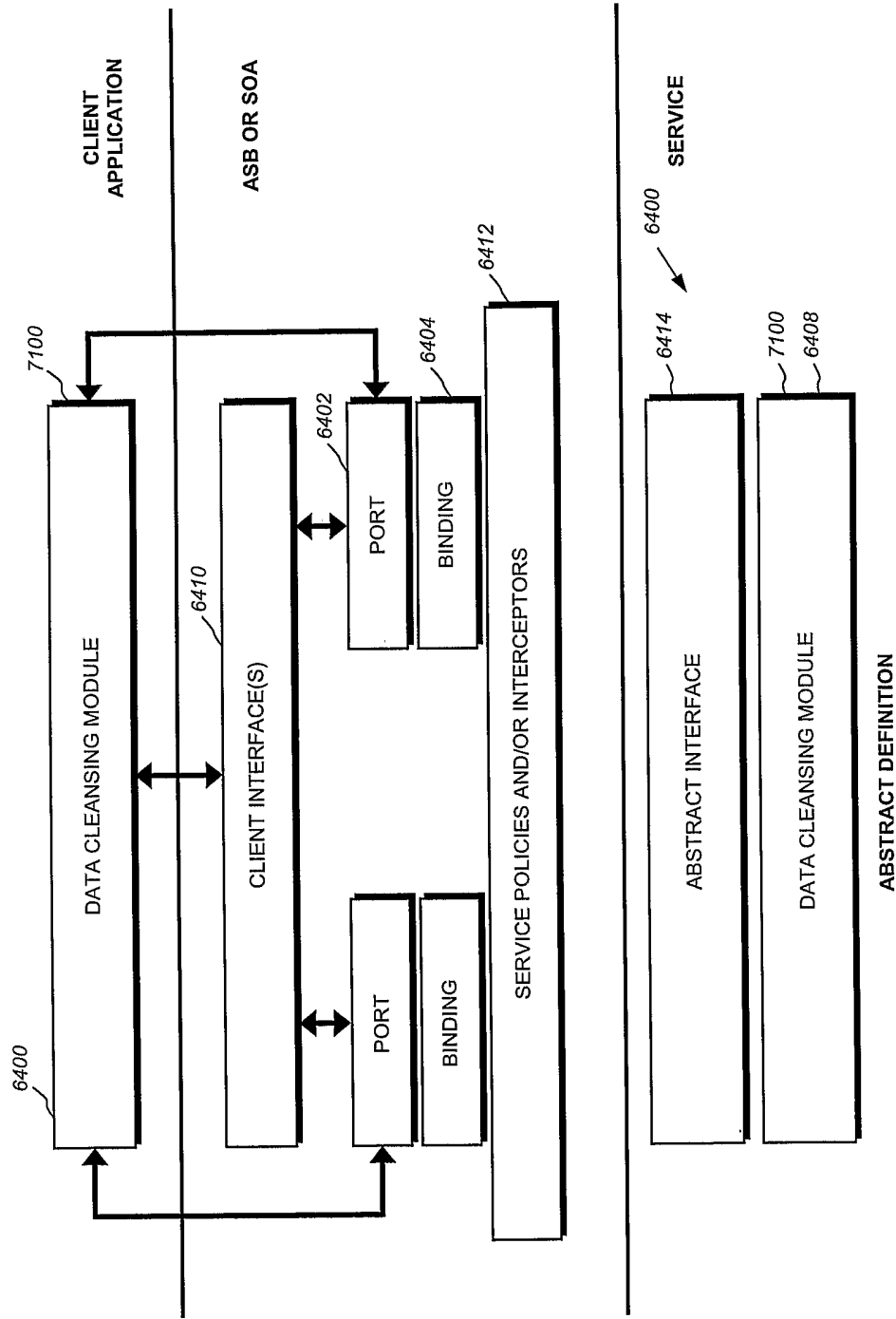


Fig. 23

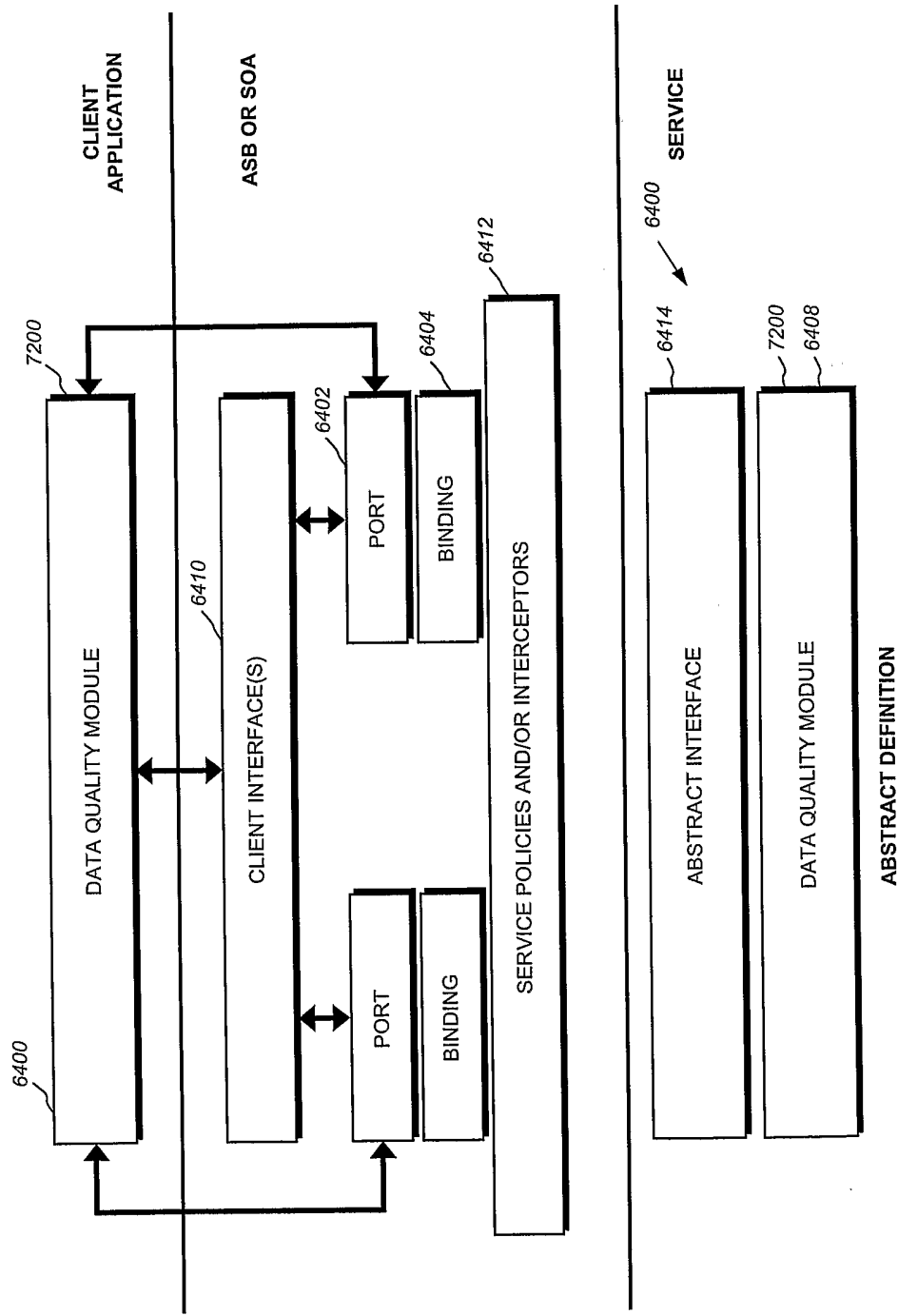


Fig. 24

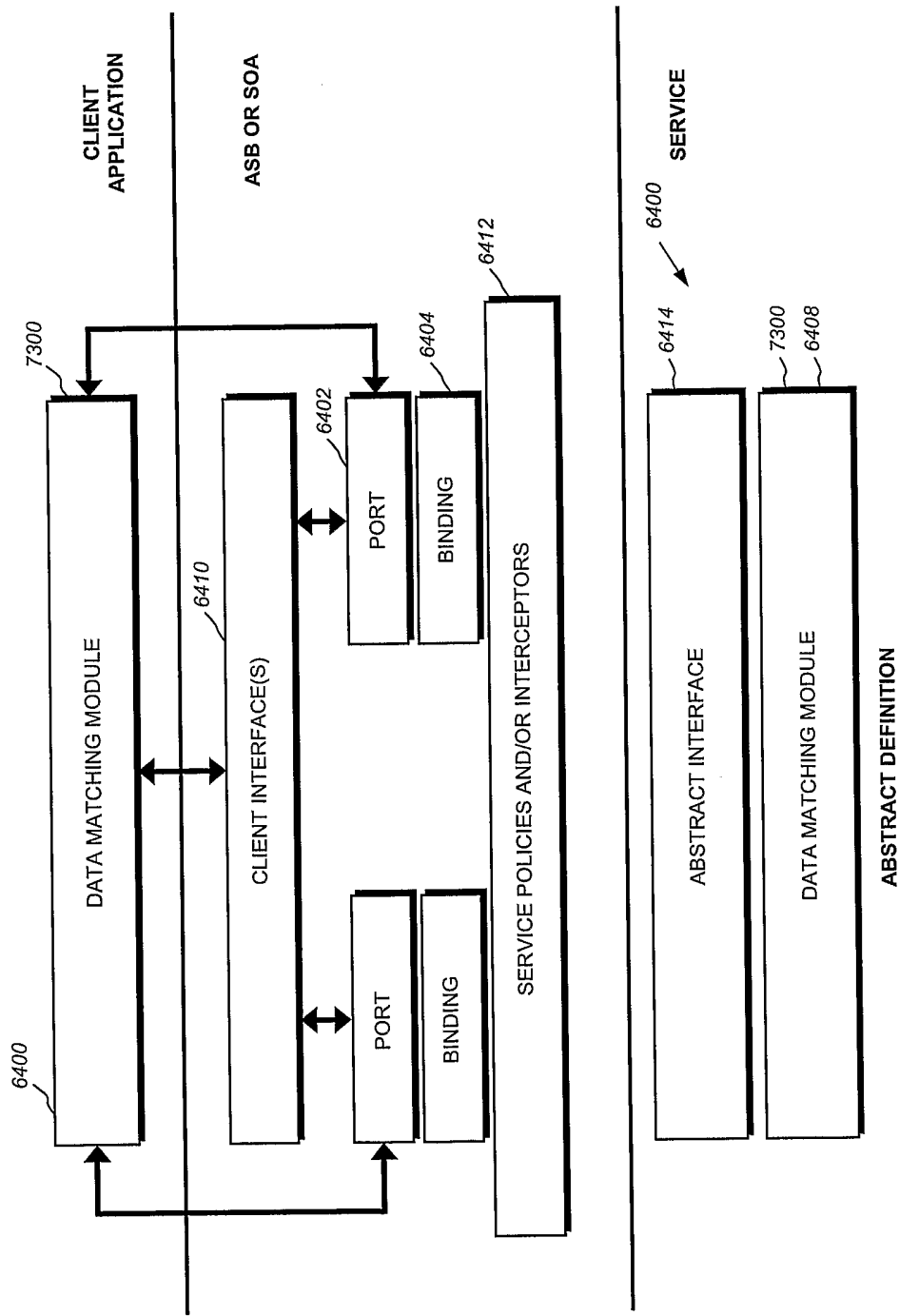


Fig. 25

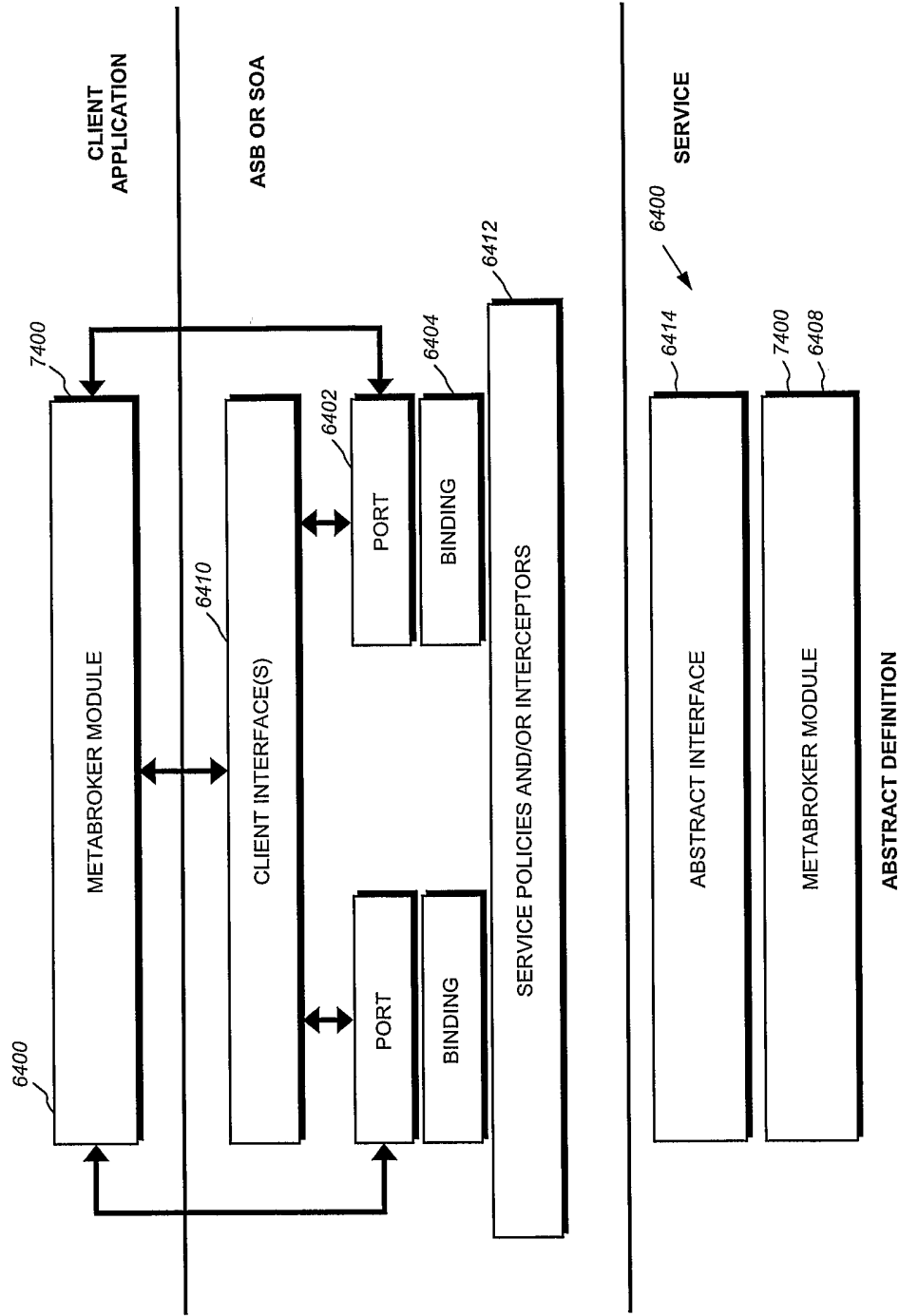


Fig. 26

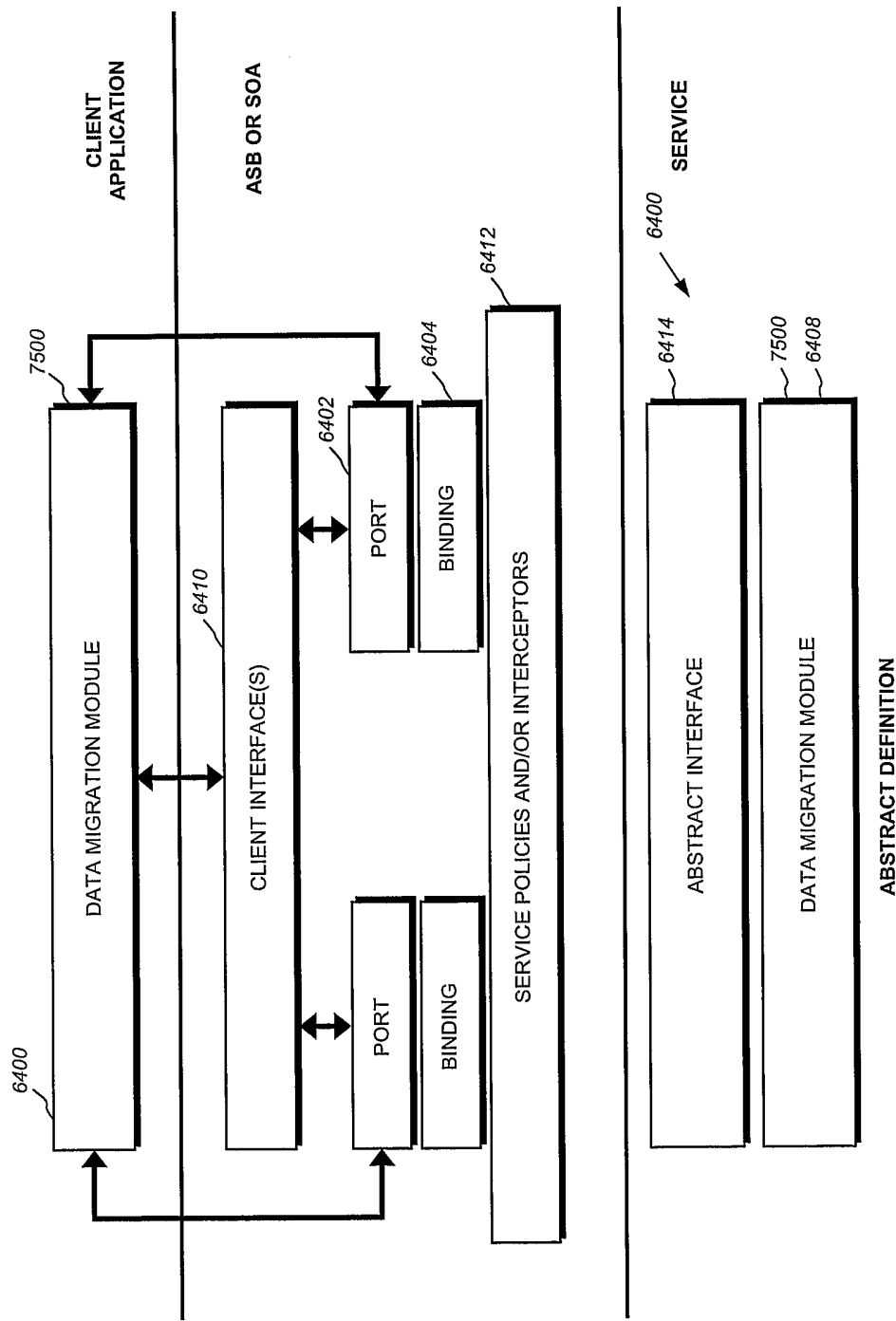


Fig. 27

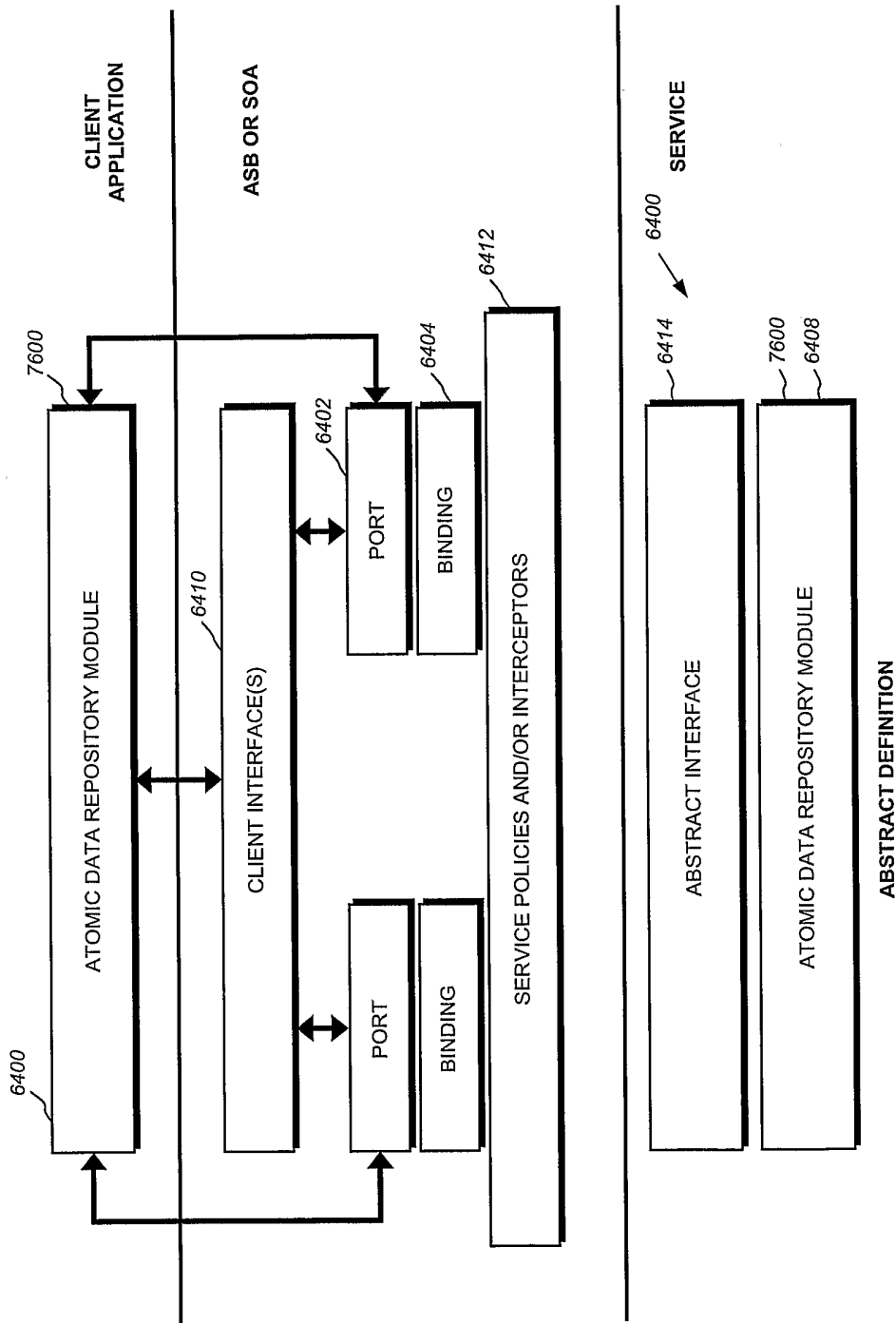


Fig. 28

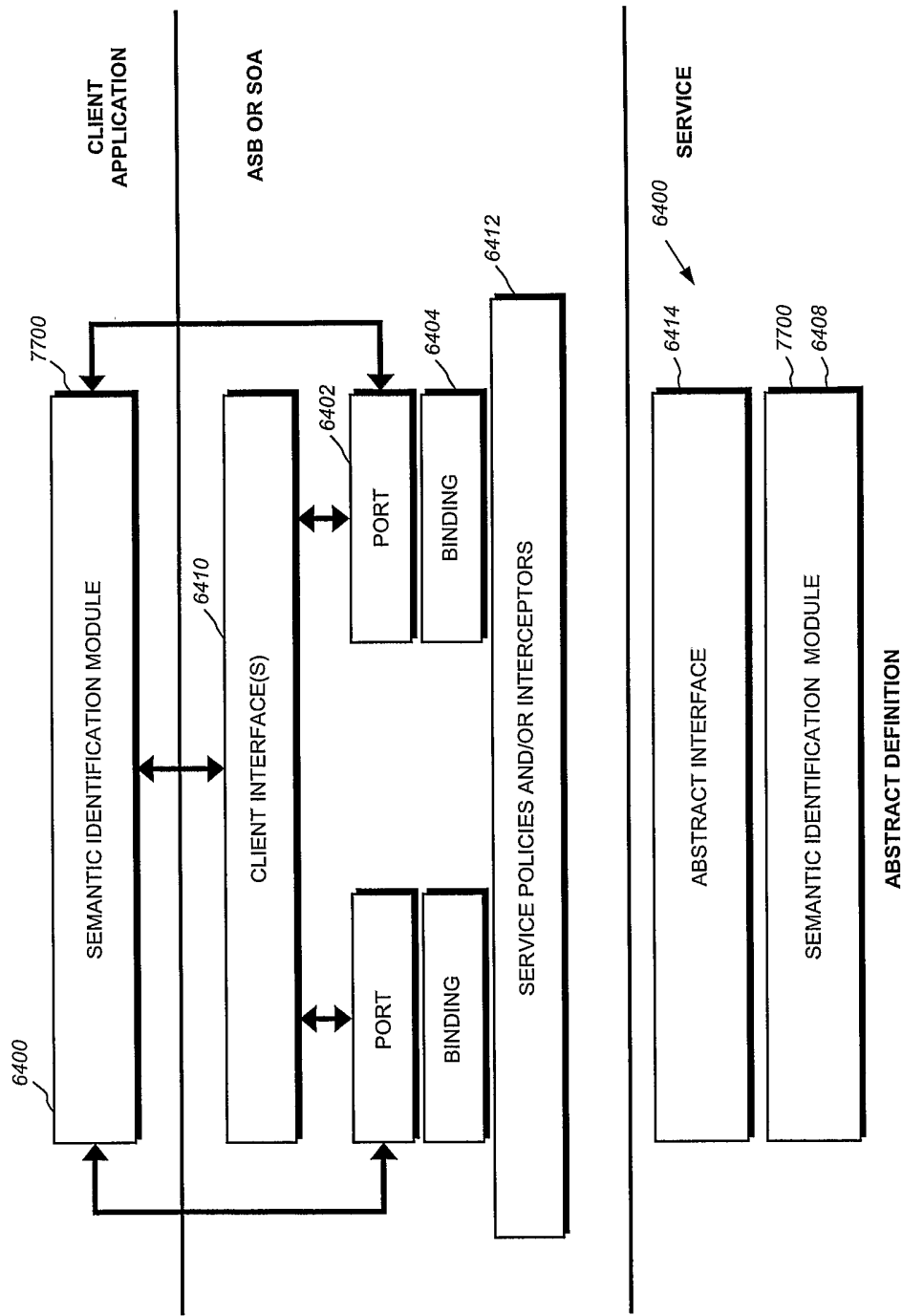


Fig. 29

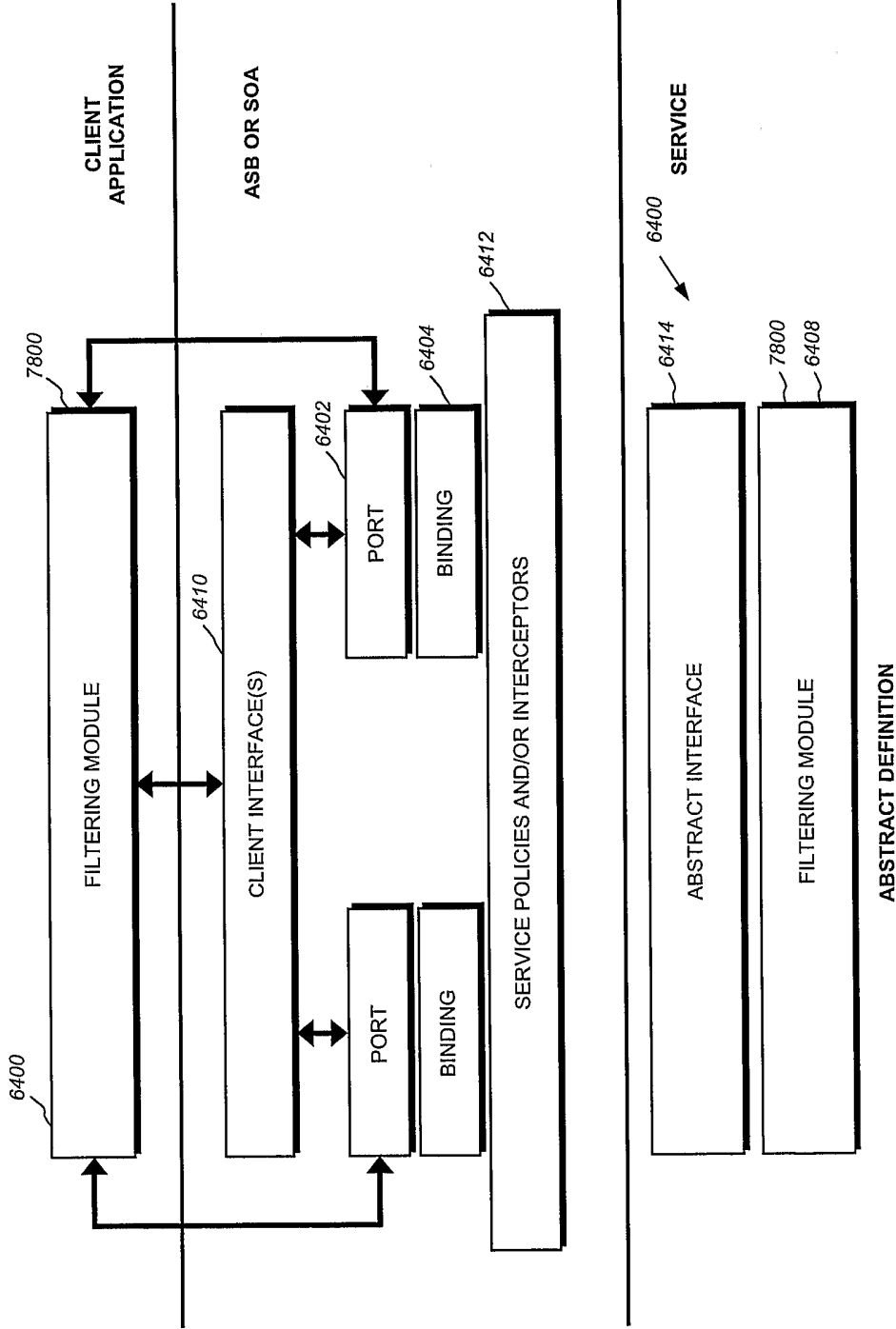


Fig. 30

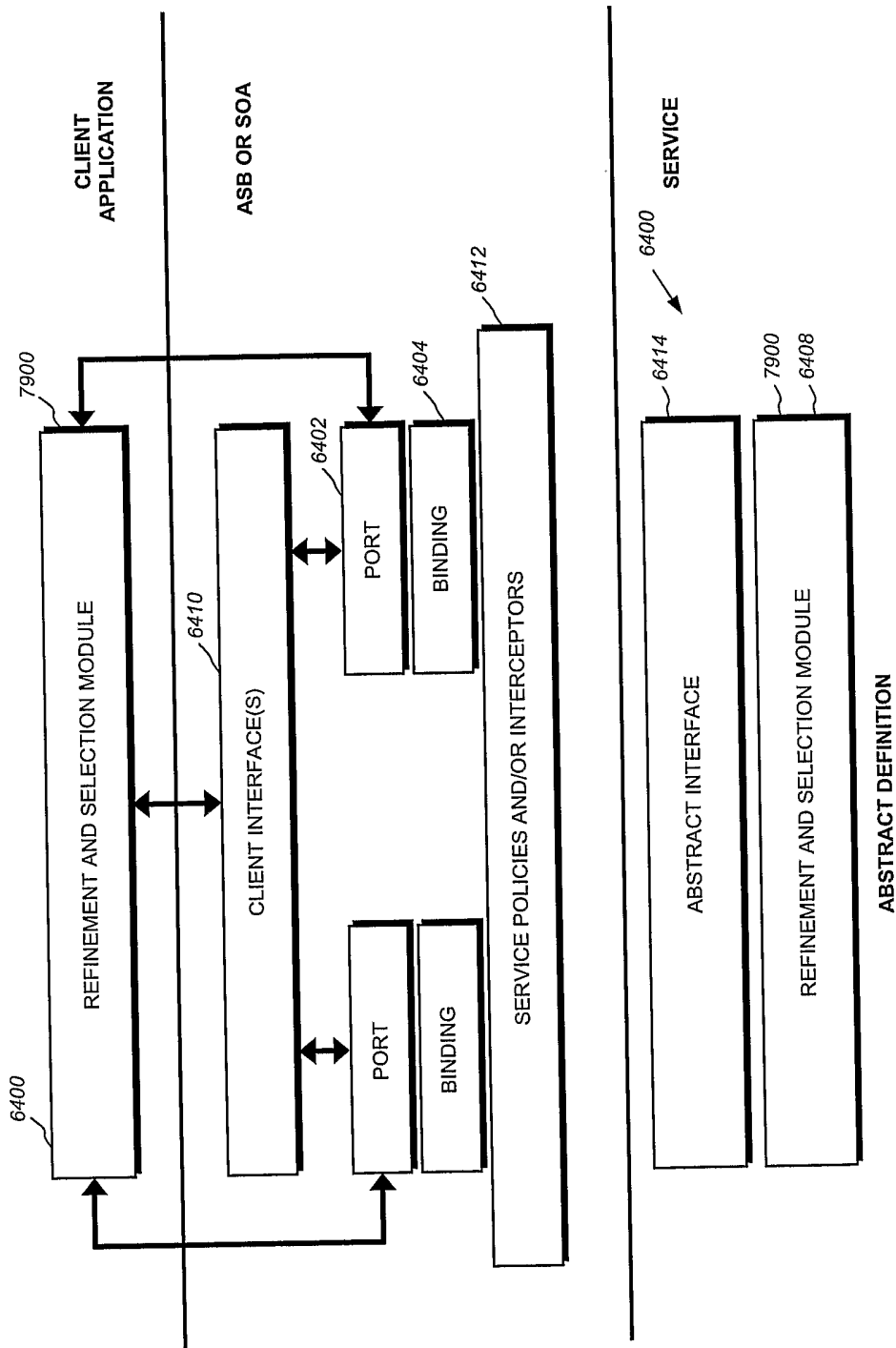


Fig. 31

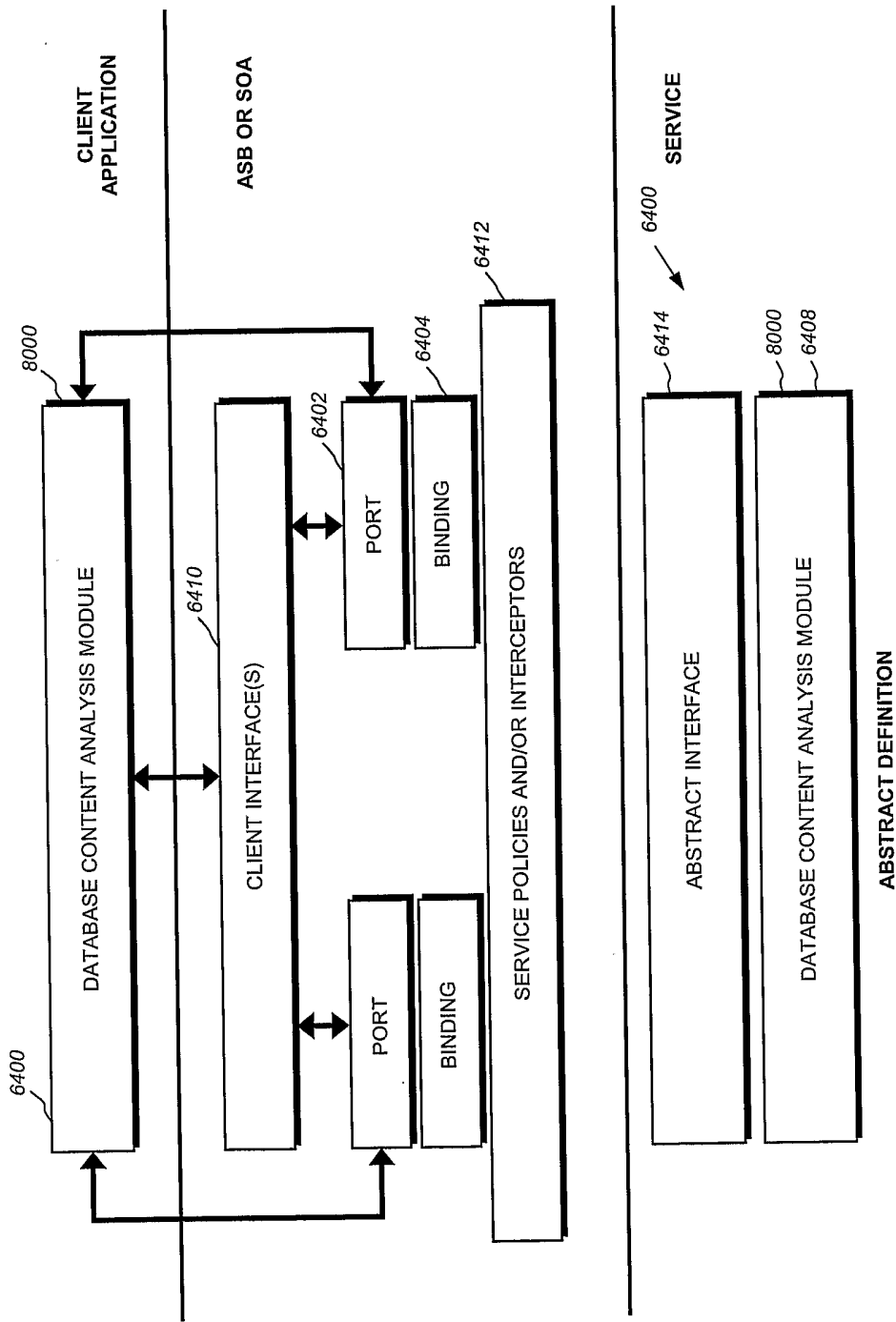


Fig. 32

40/91

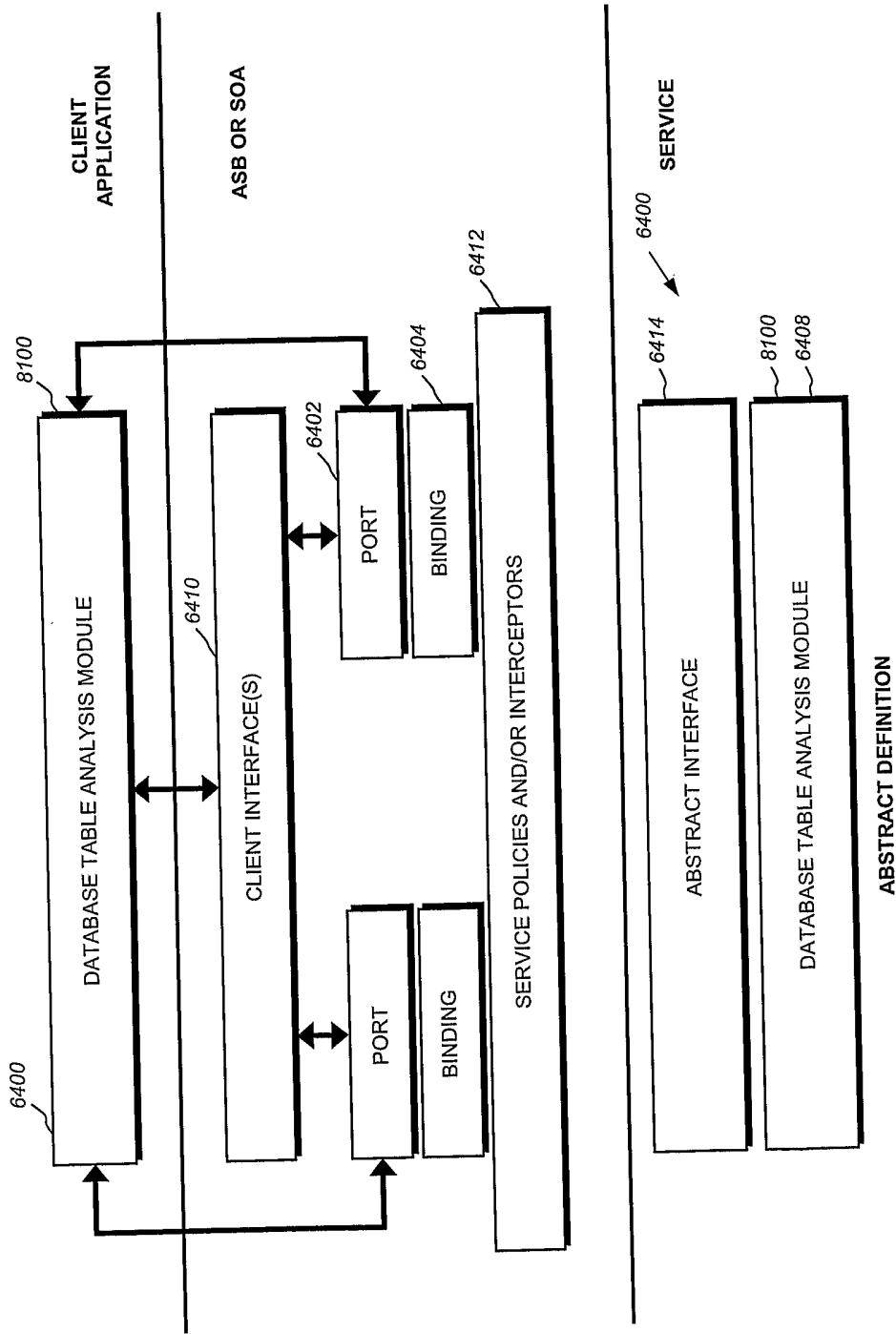


Fig. 33

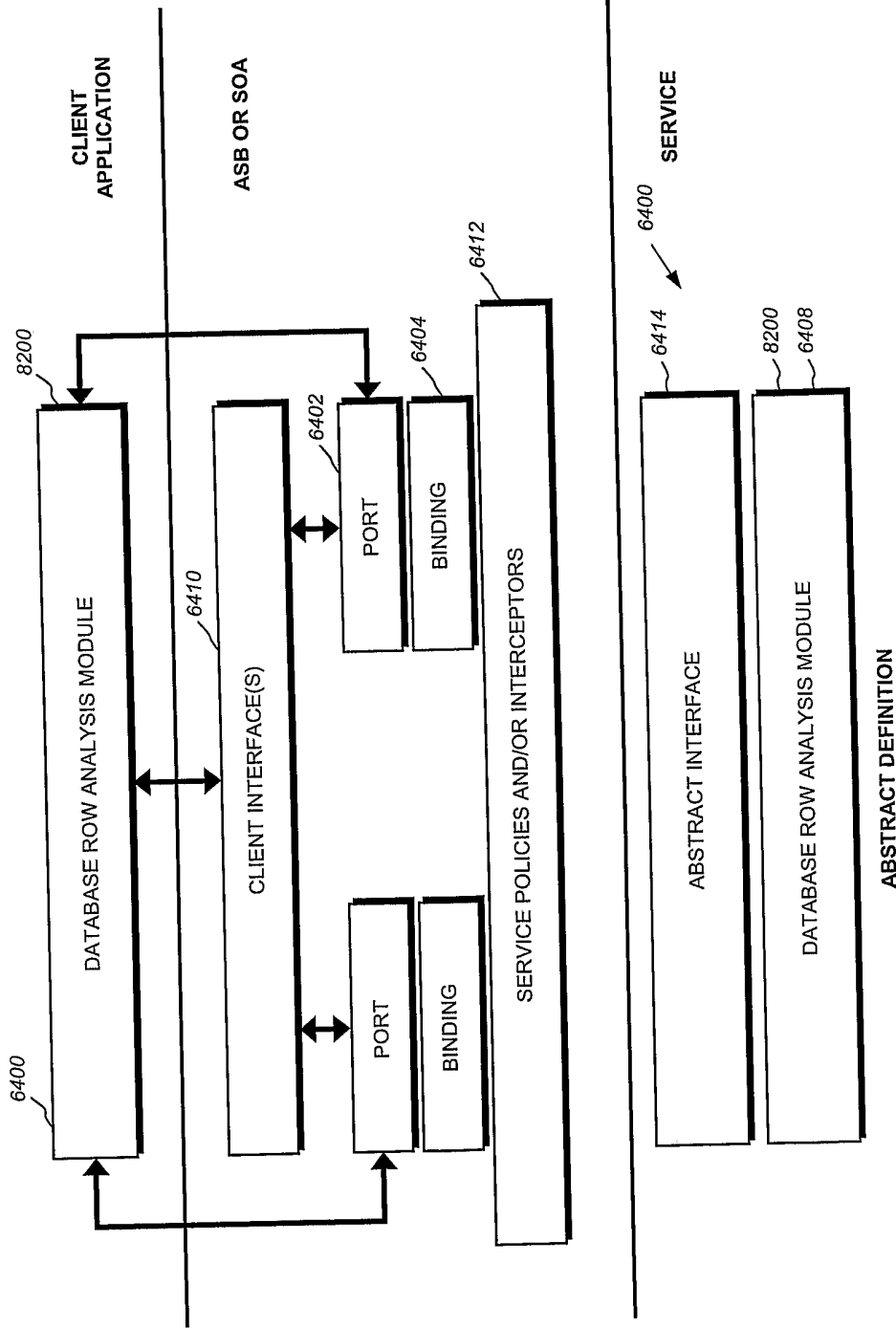


Fig. 34

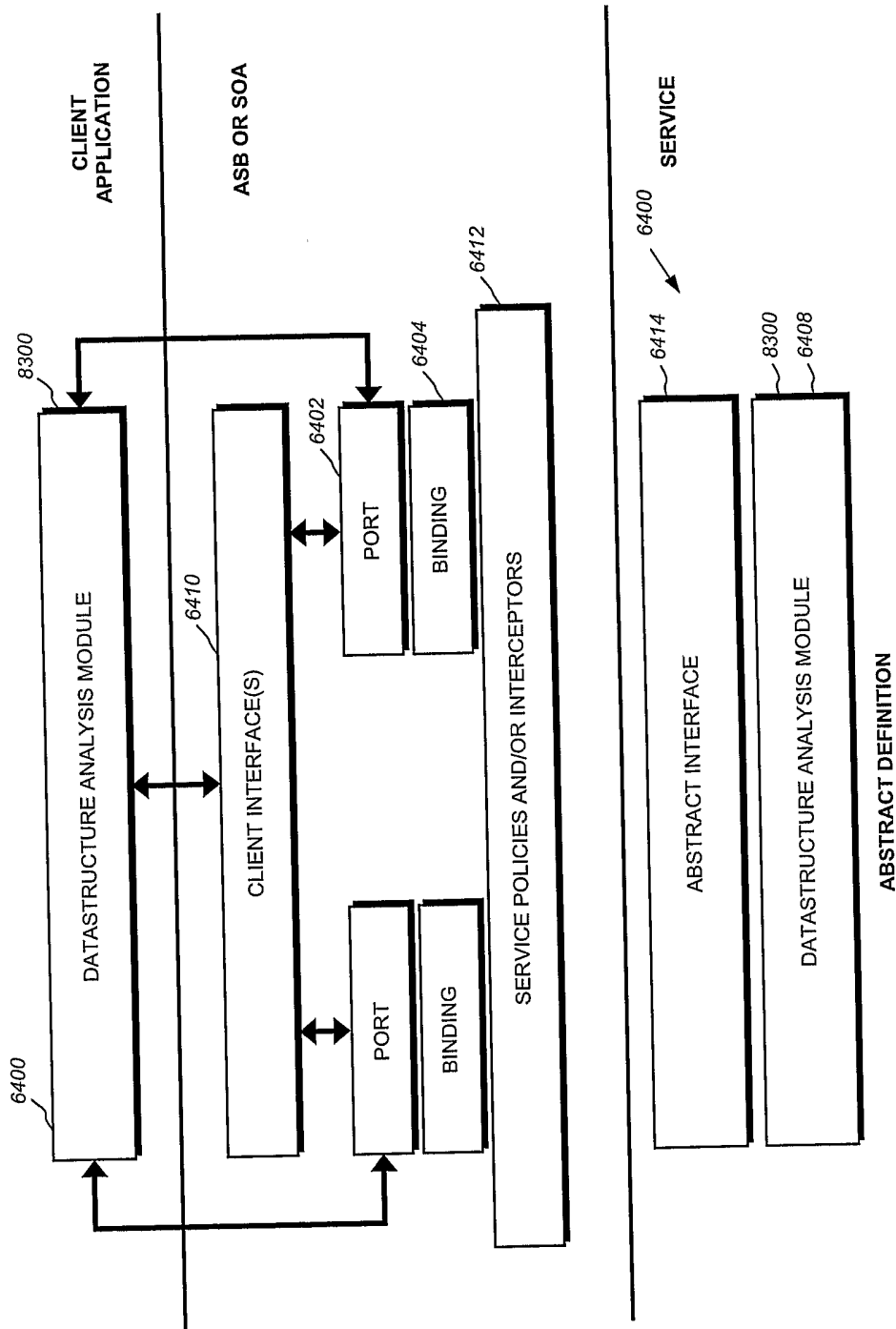


Fig. 35

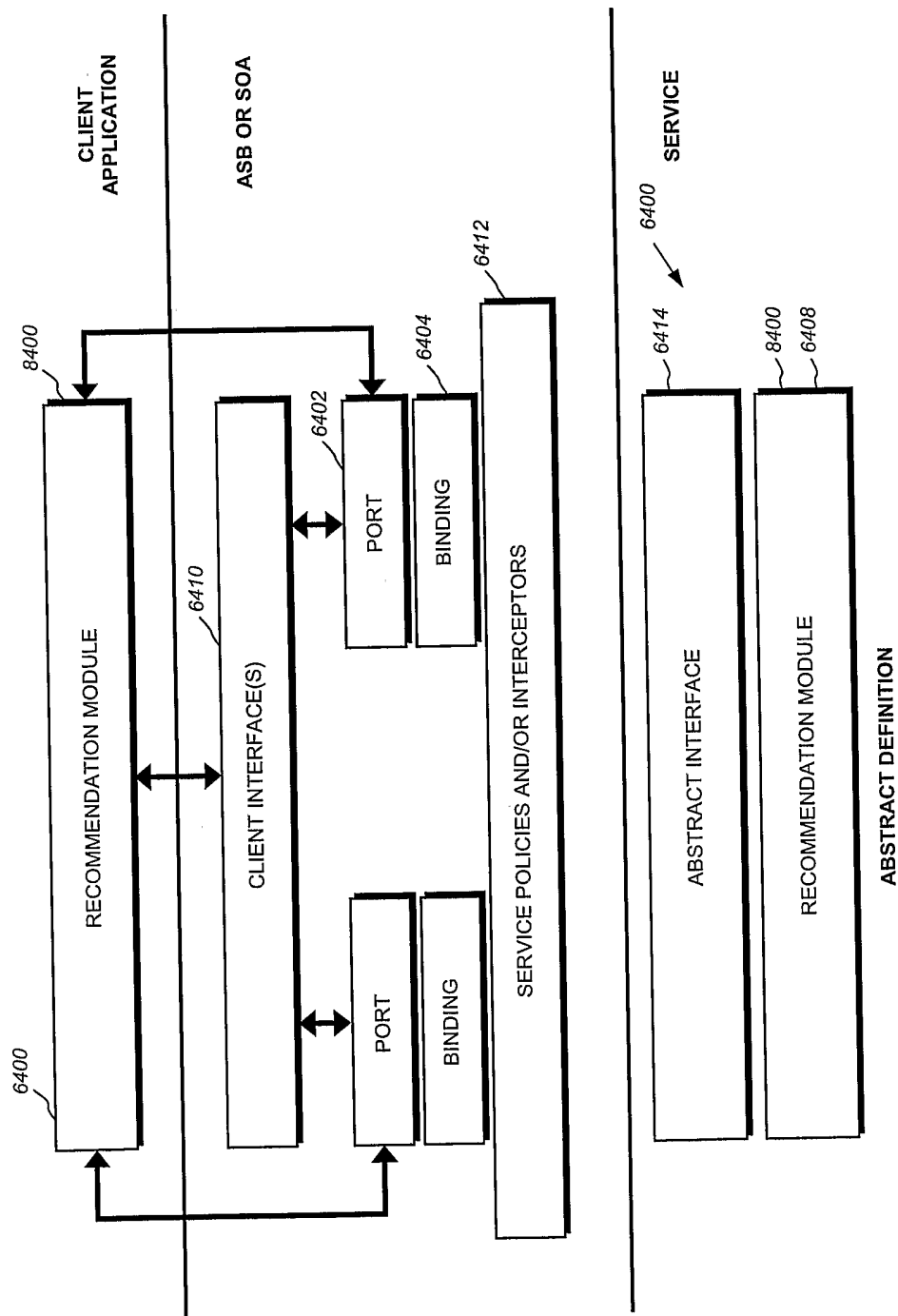


Fig. 36

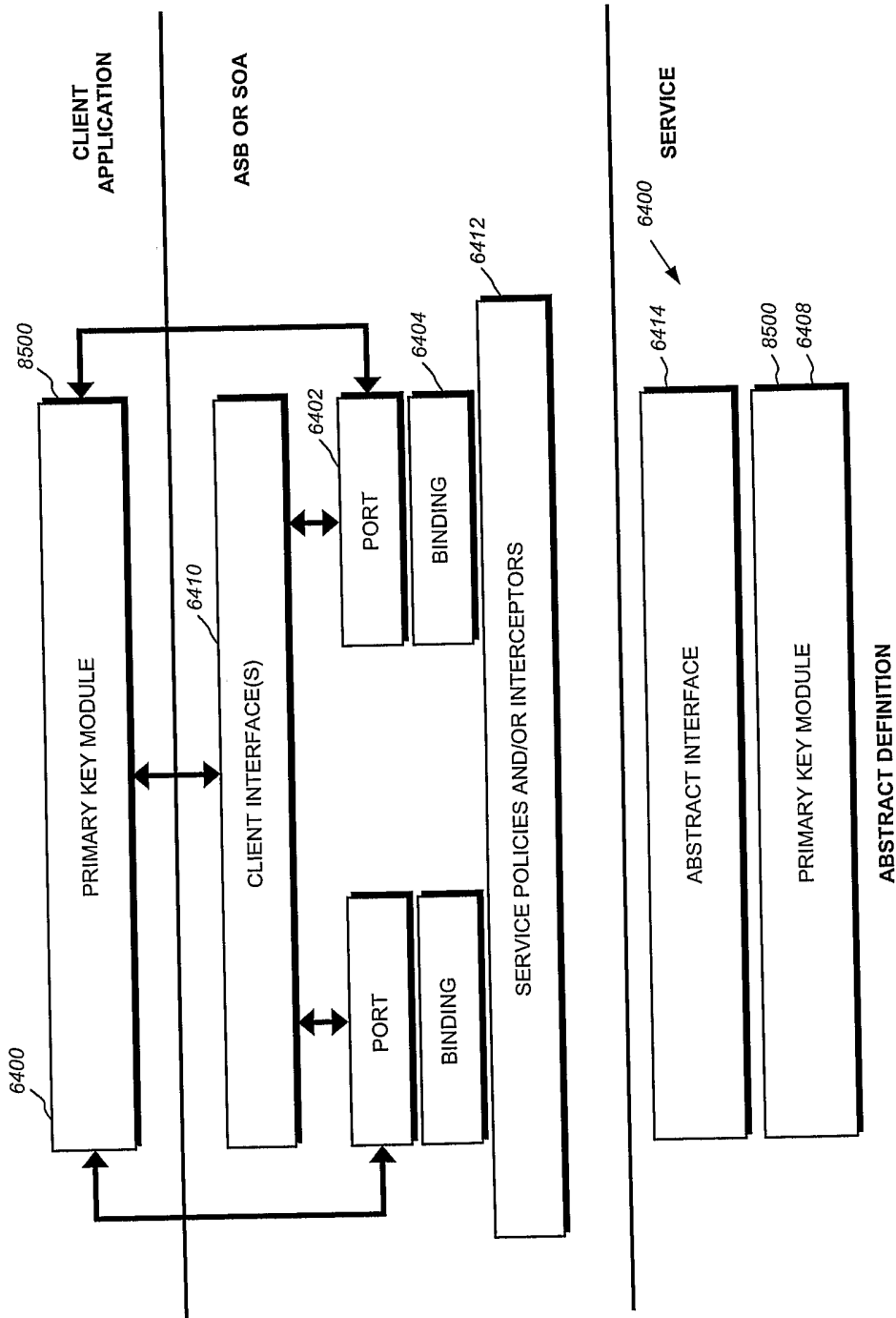


Fig. 37

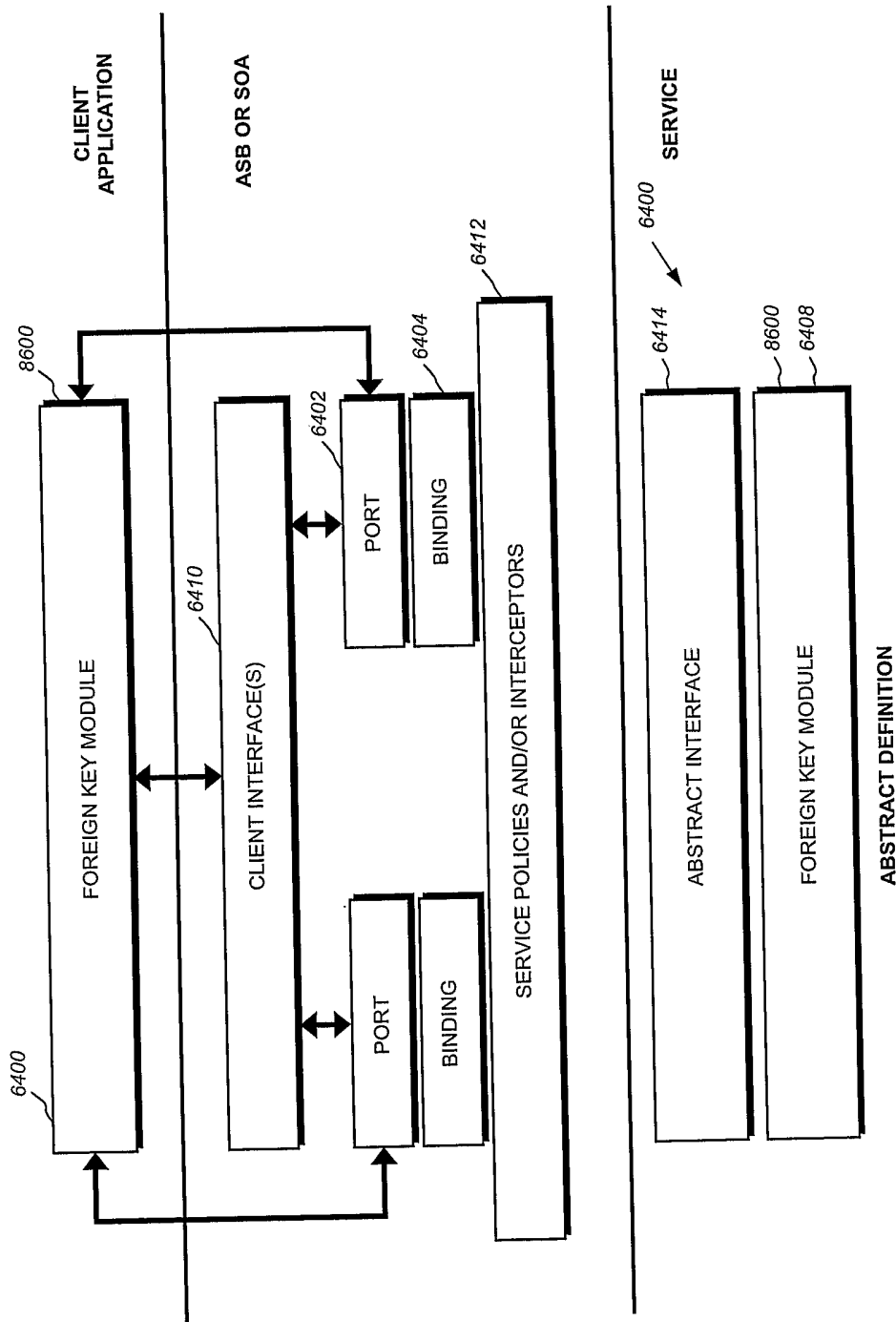


Fig. 38

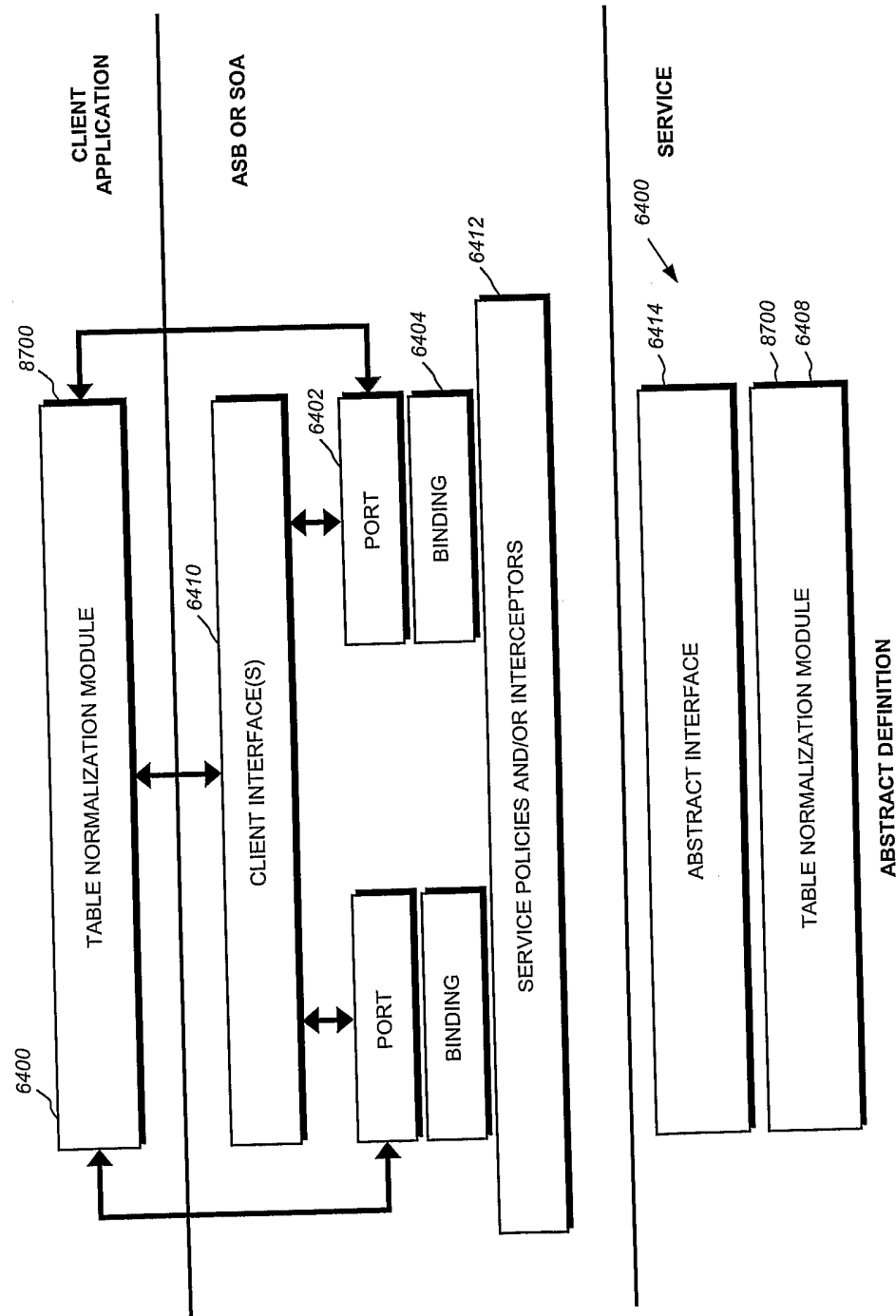


Fig. 39

47/91

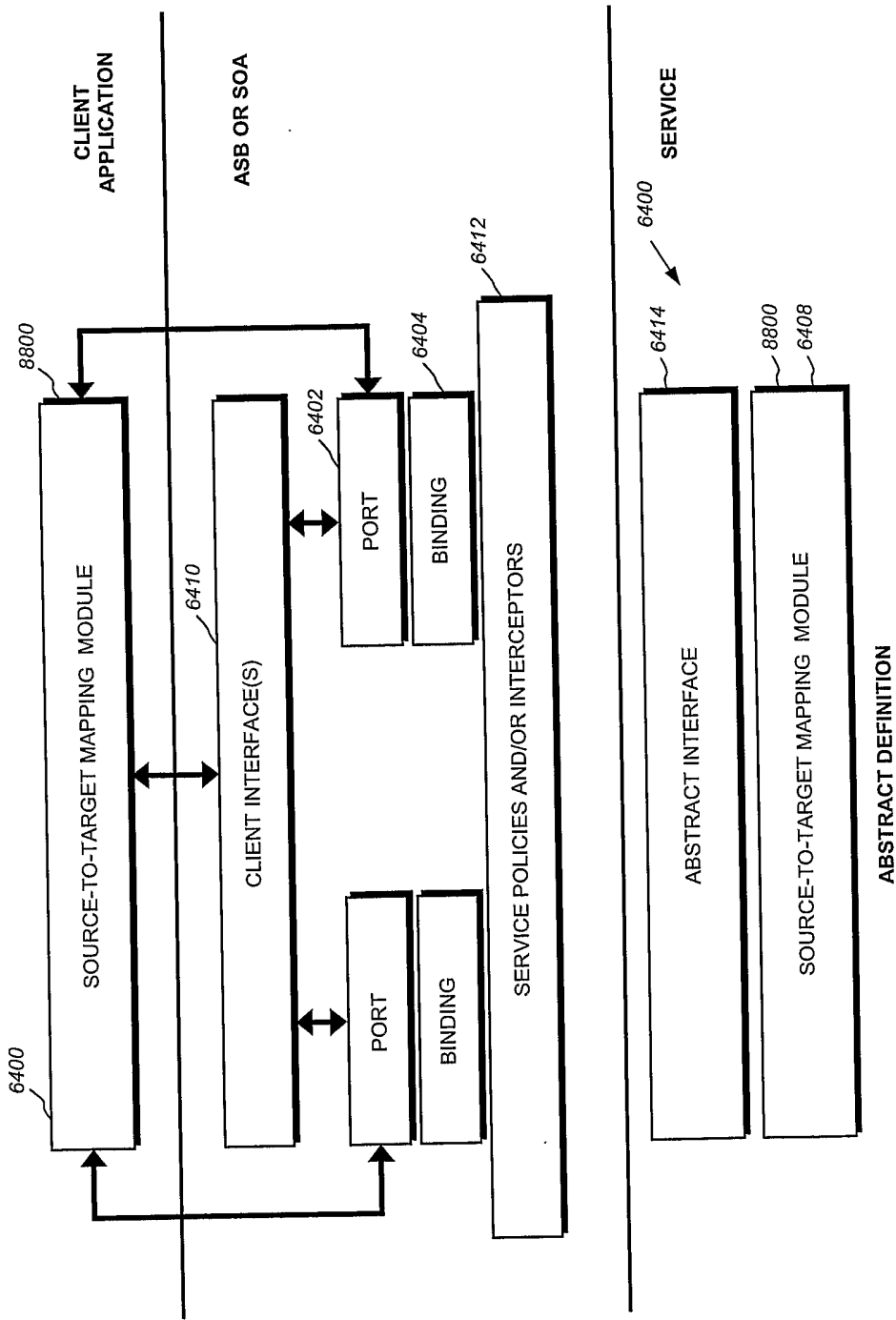


Fig. 40

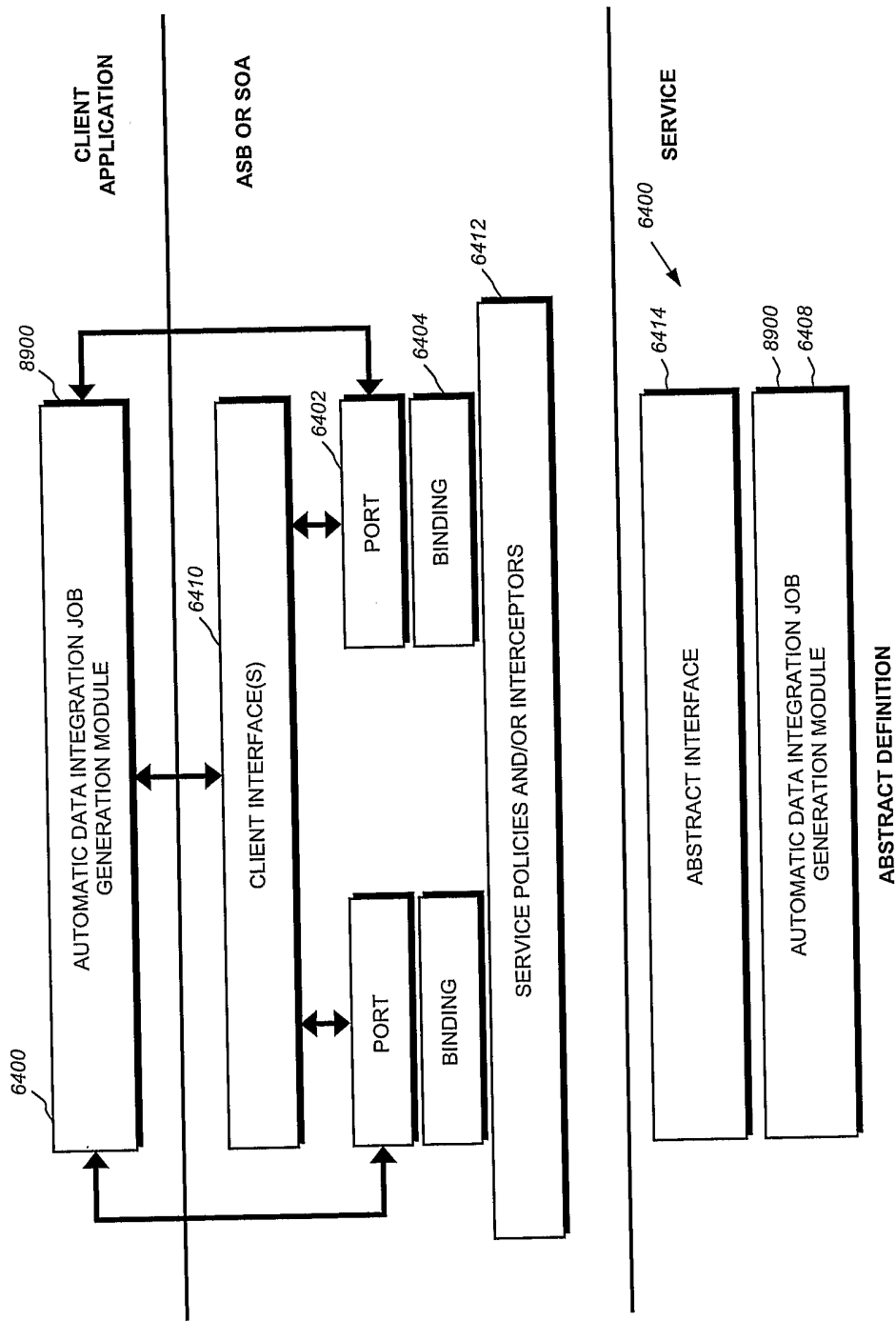


Fig. 41

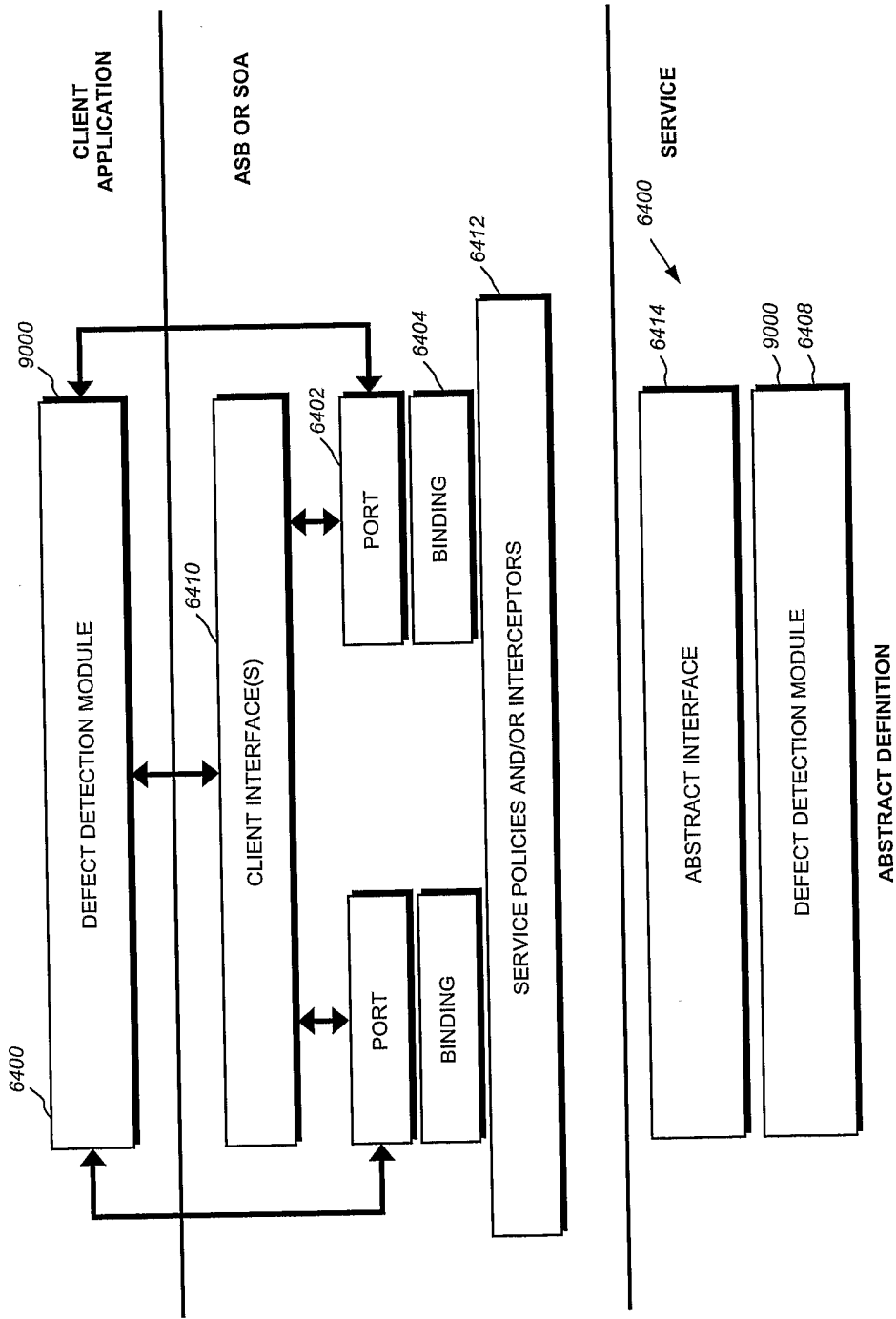


Fig. 42

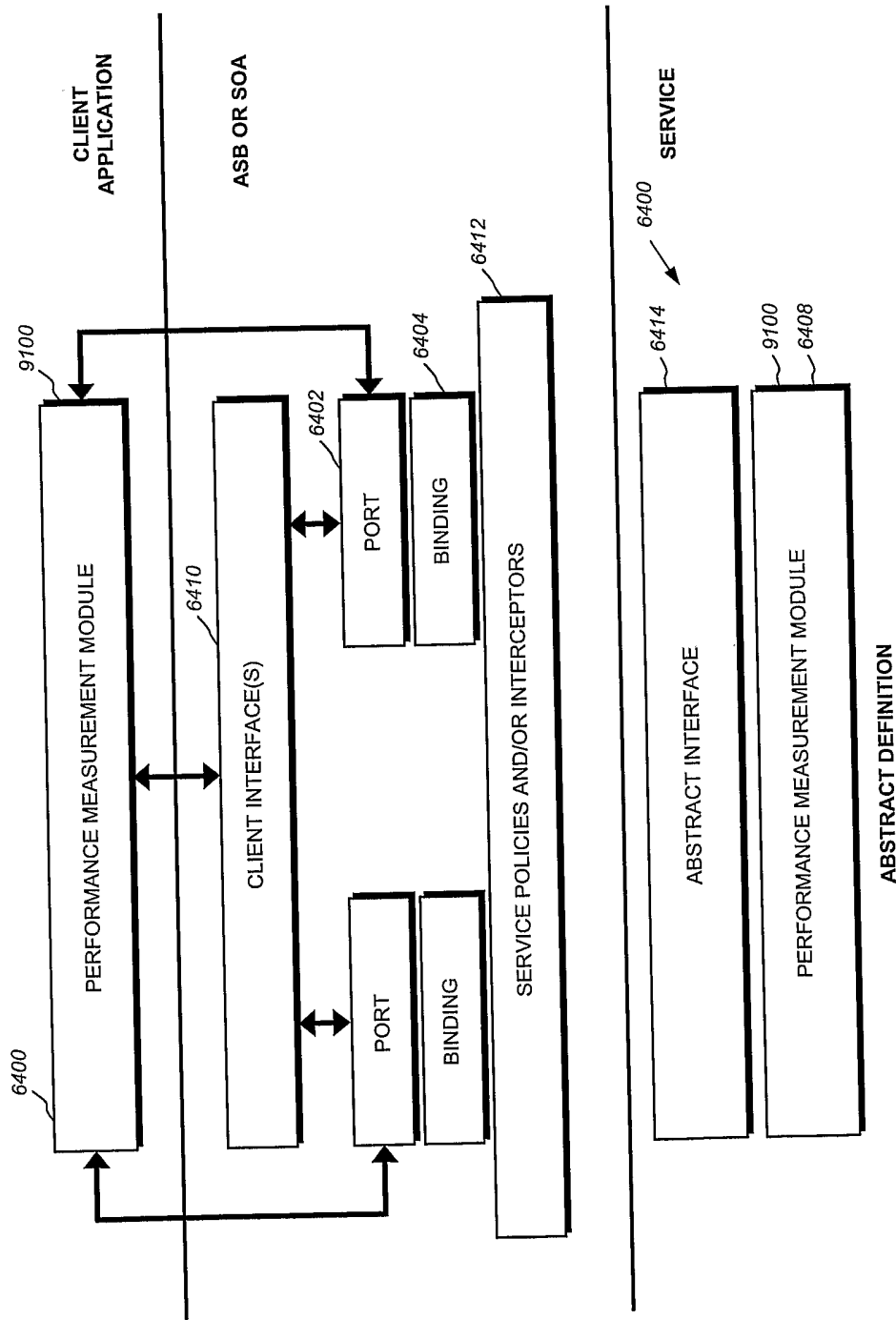


Fig. 43

51/91

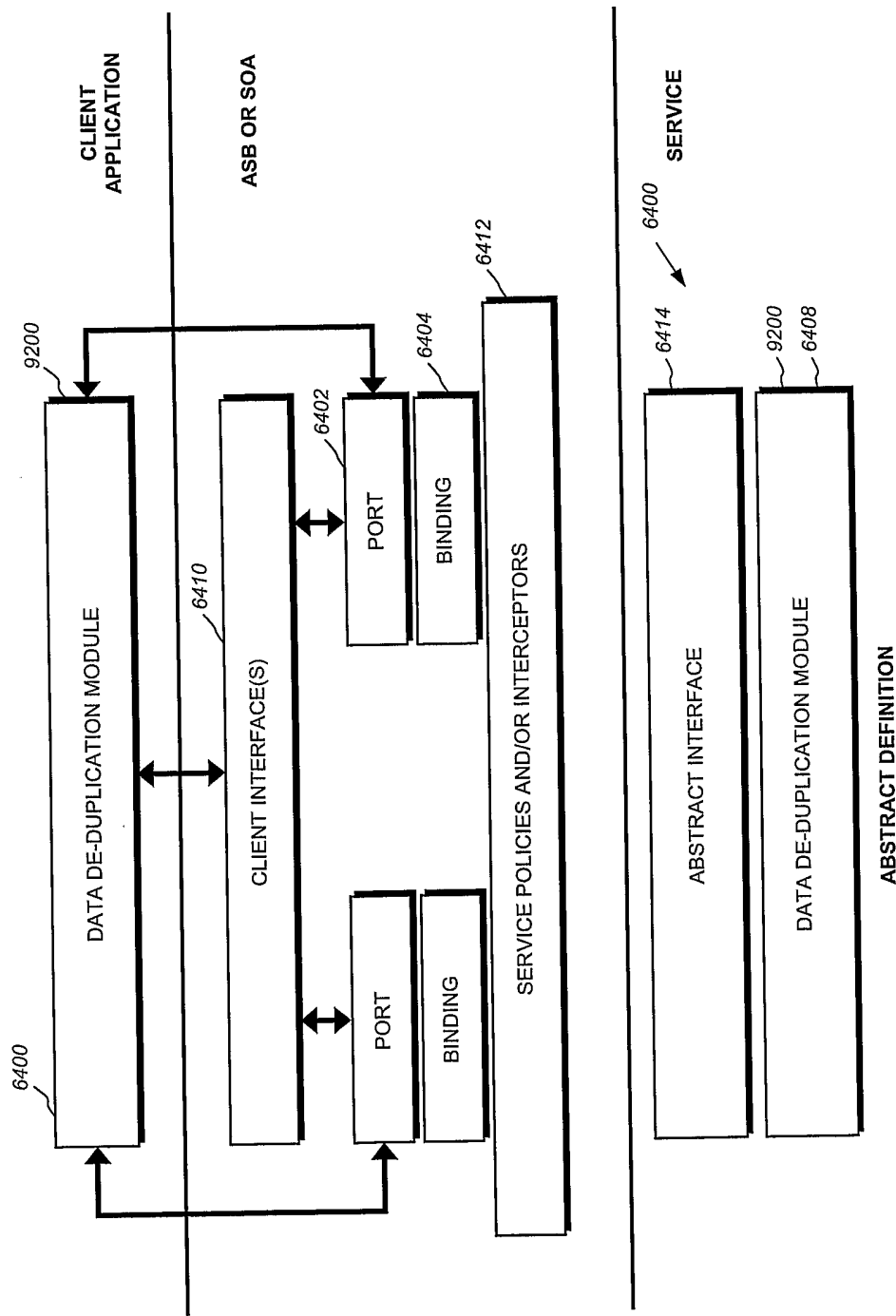


Fig. 44

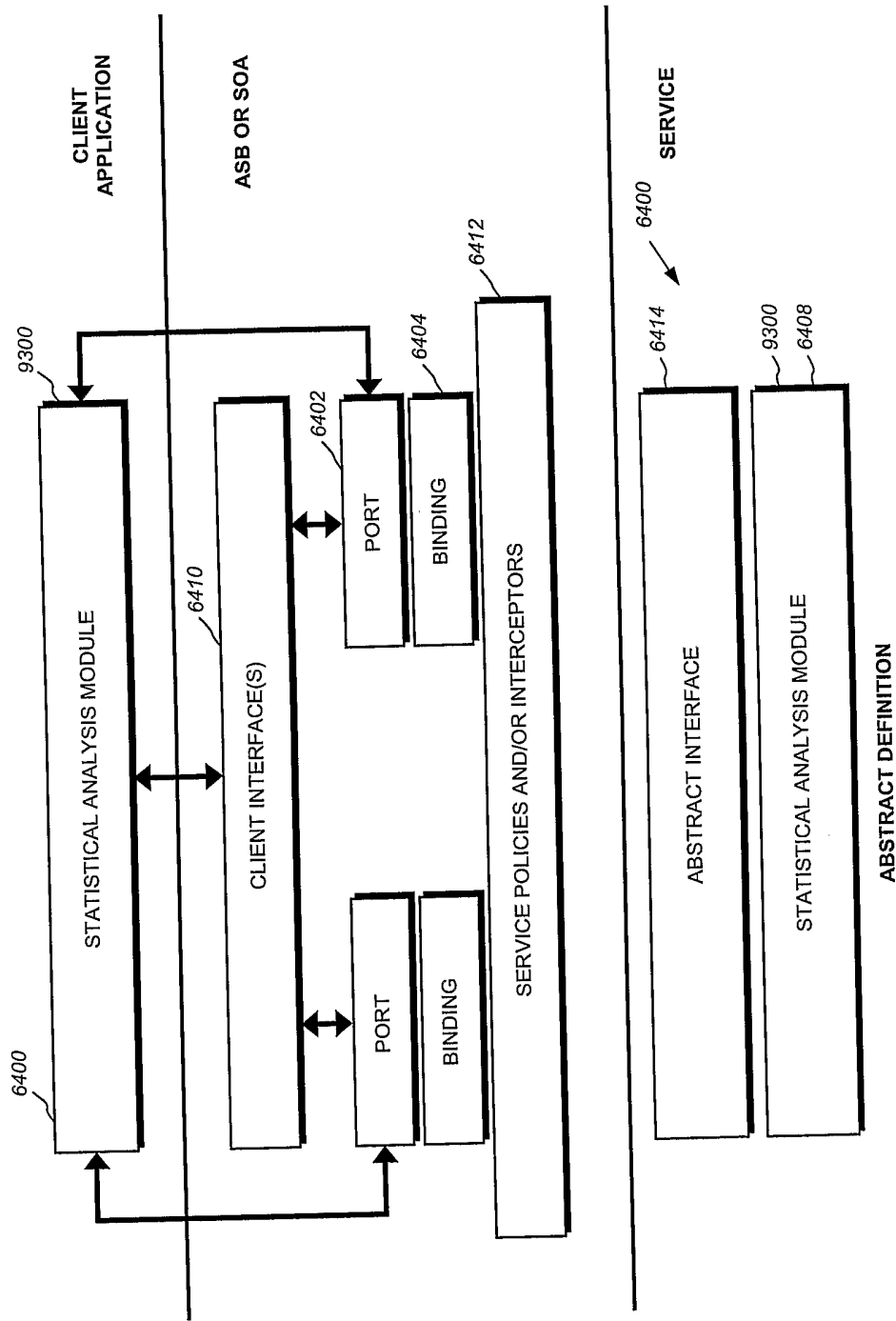


Fig. 45

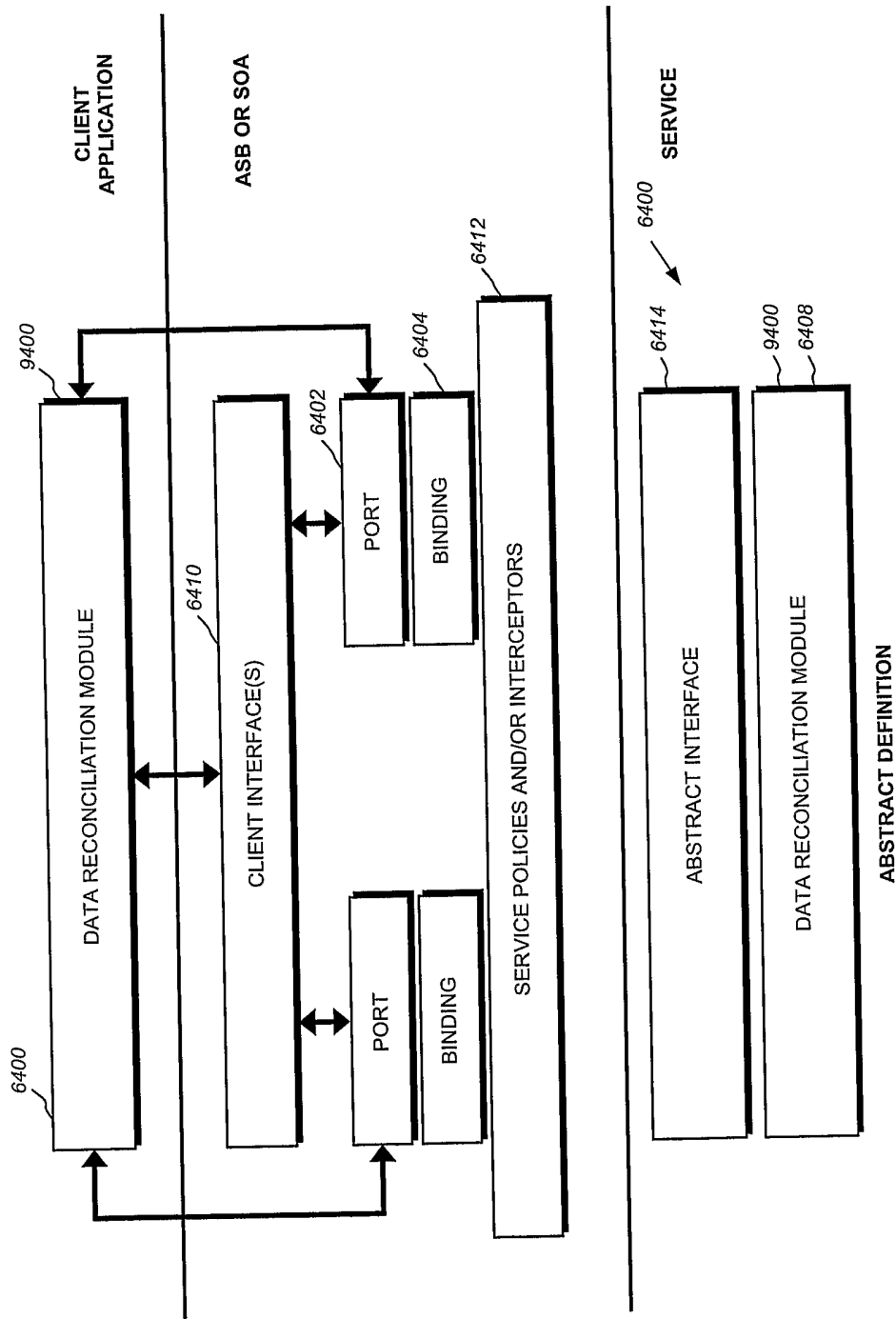


Fig. 46

54/91

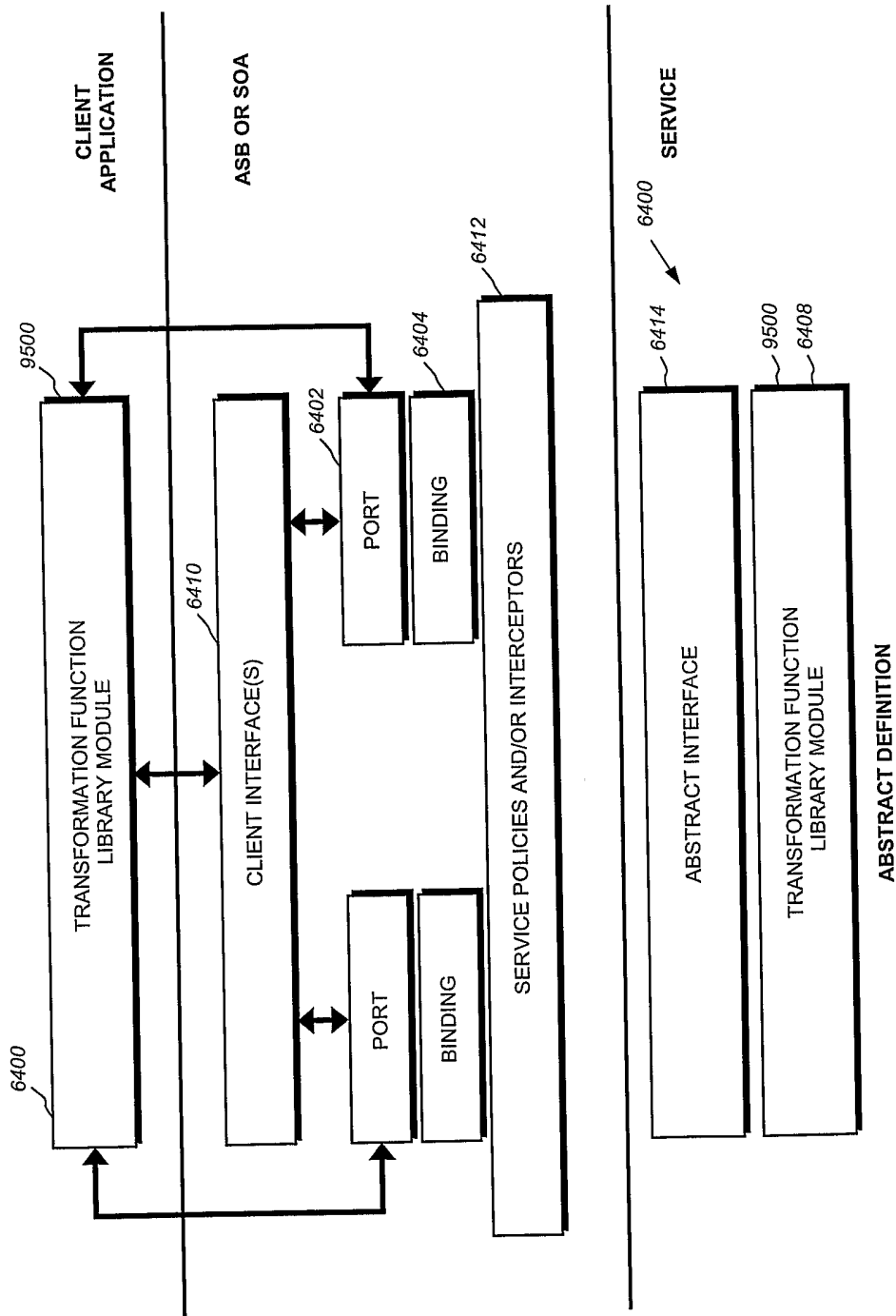


Fig. 47

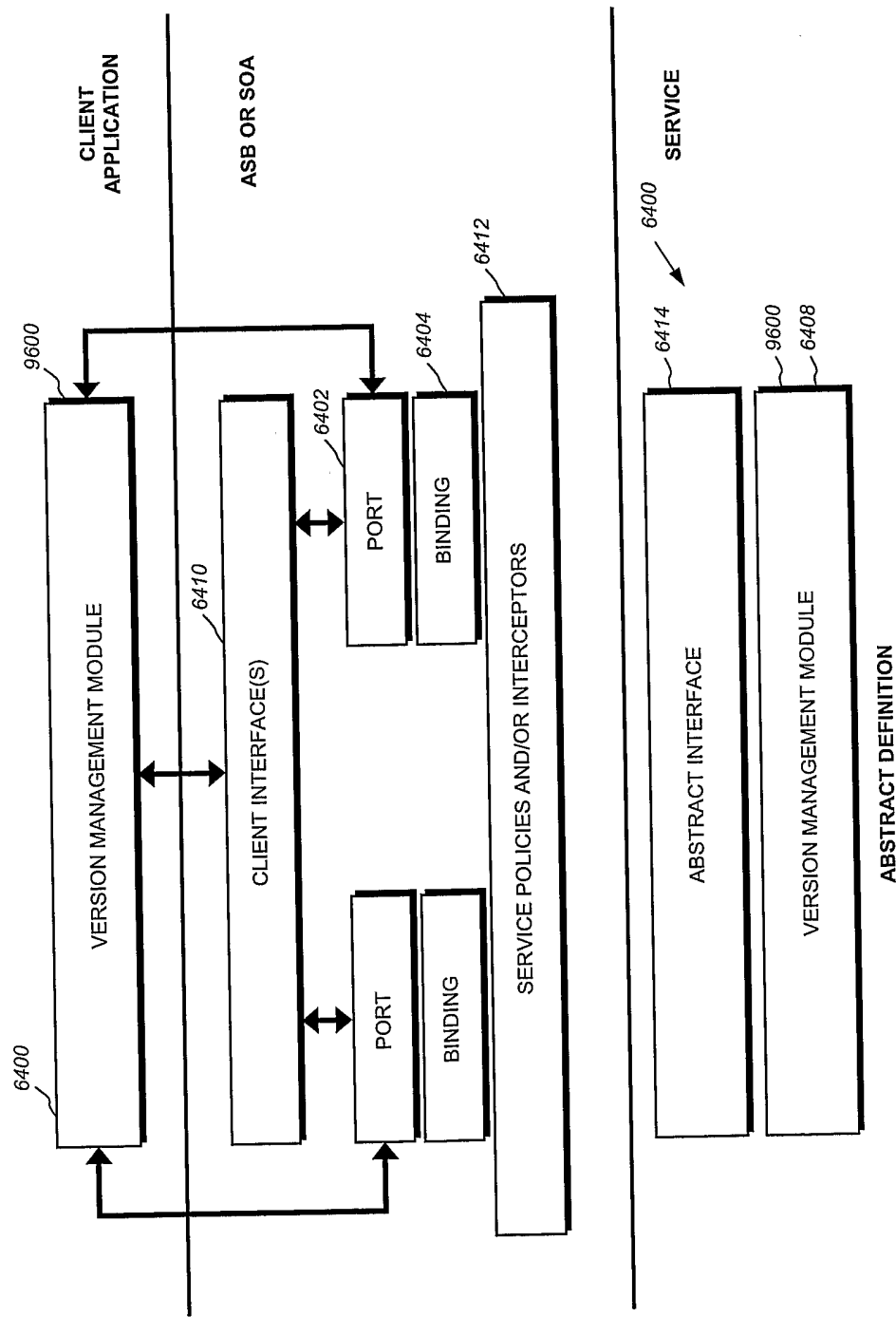


Fig. 48

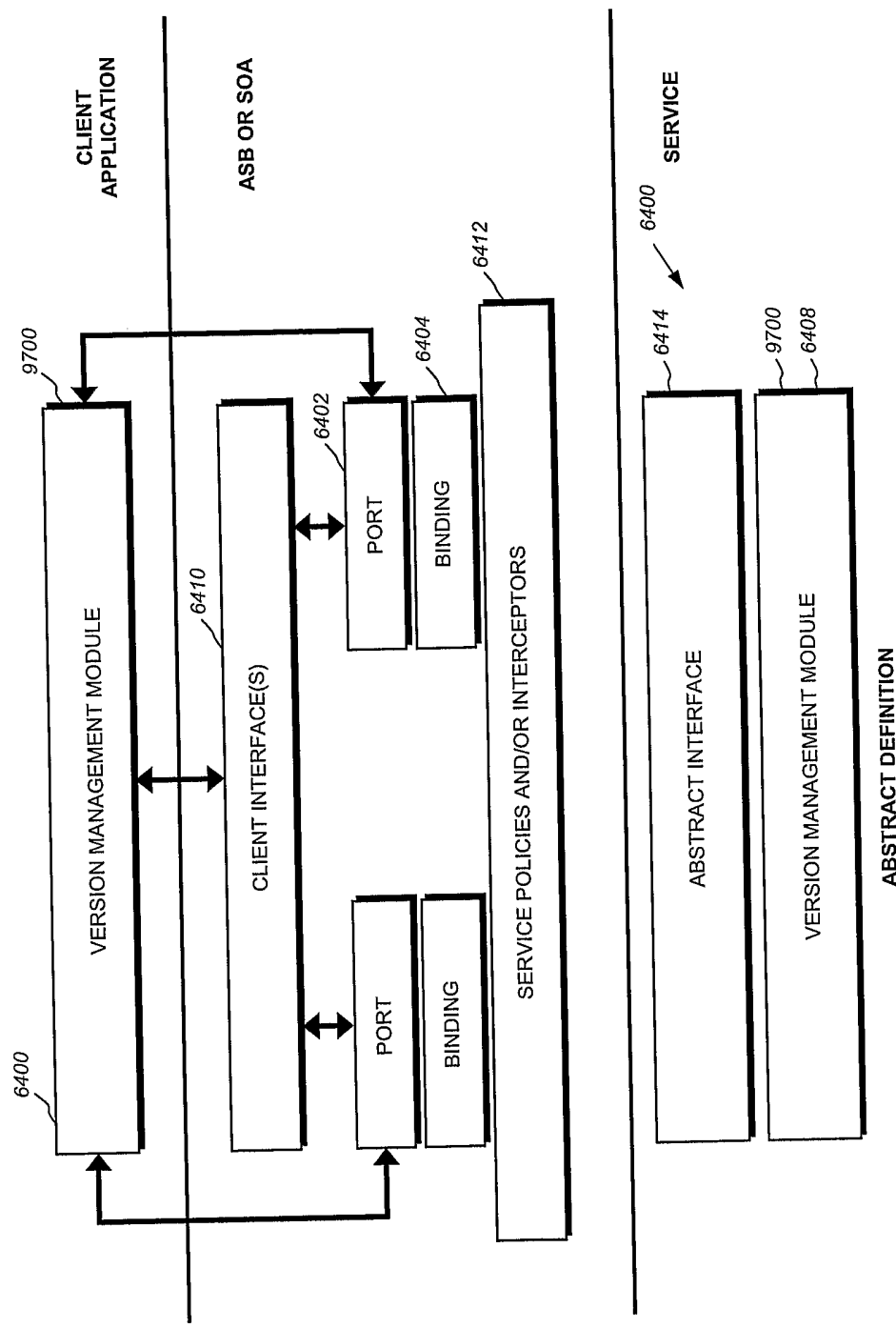


Fig. 49

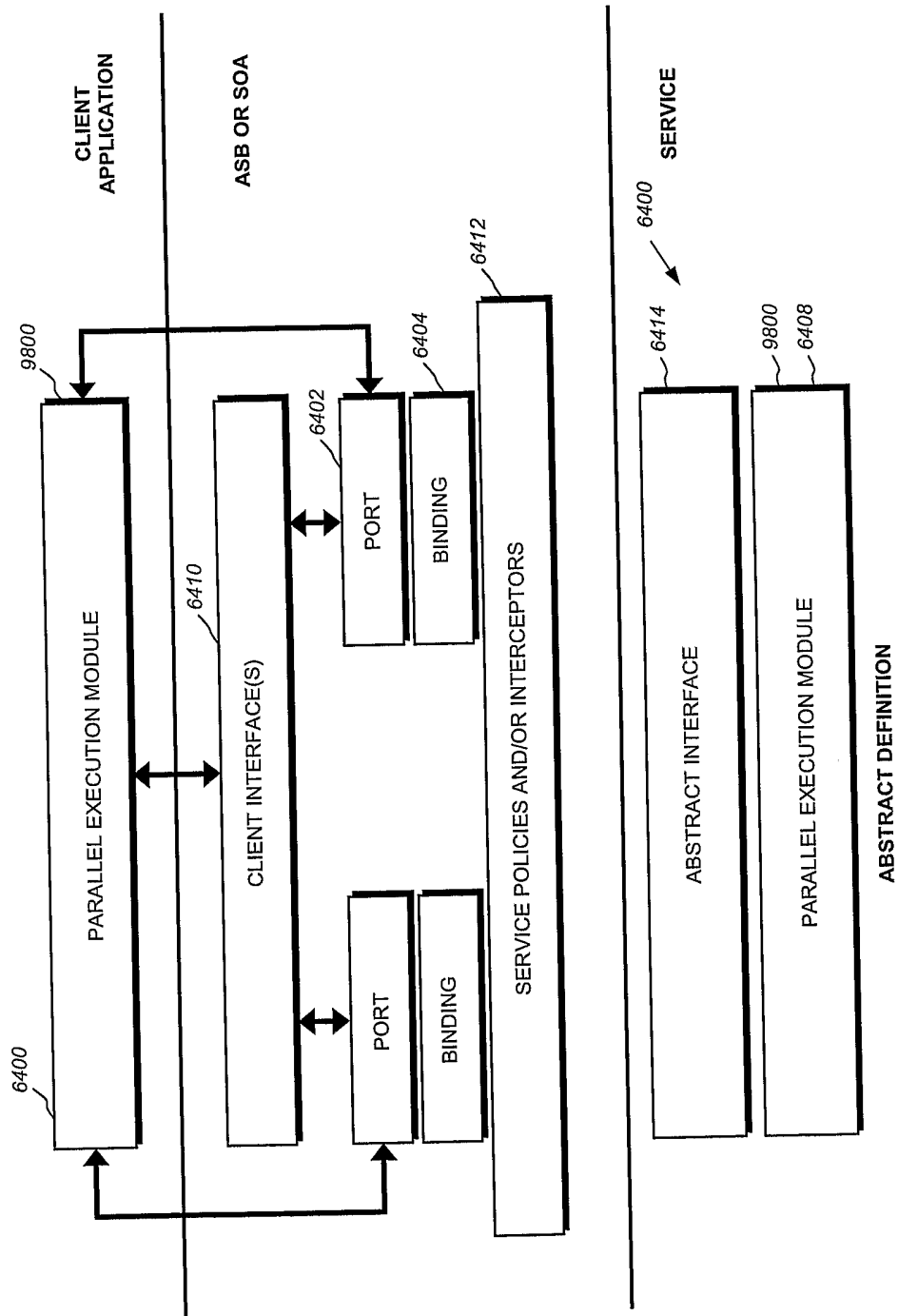


Fig. 50

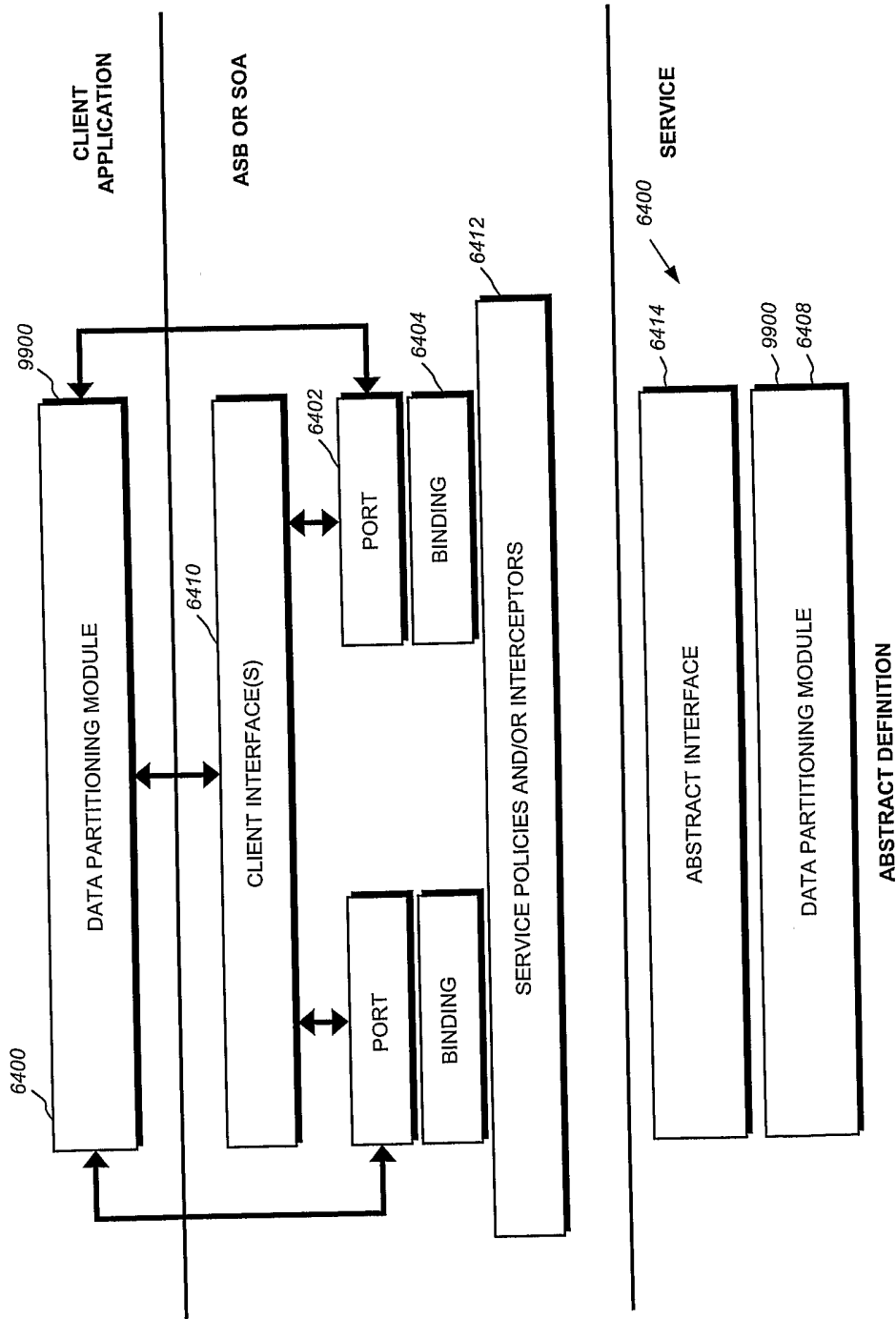


Fig. 51

59/91

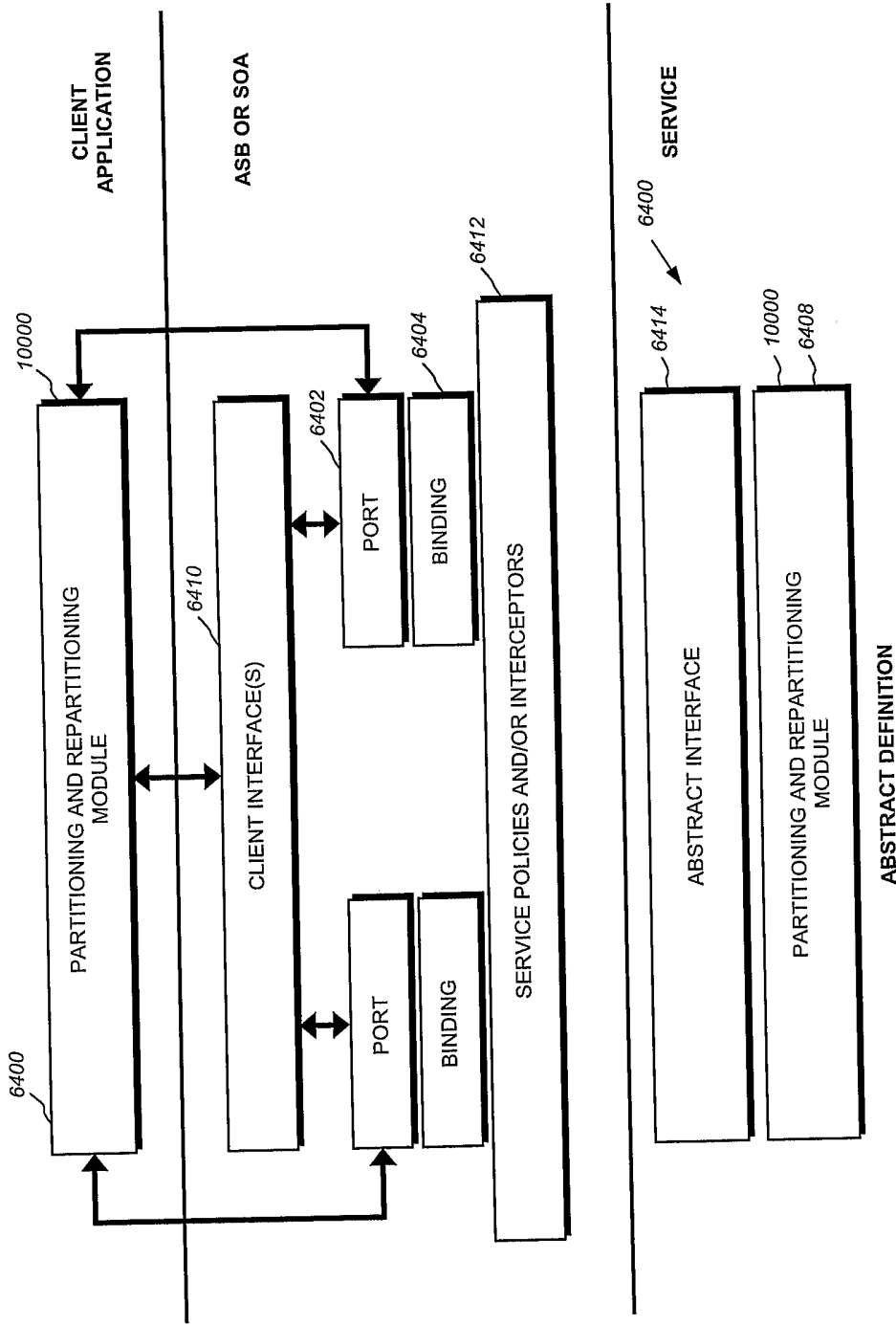


Fig. 52

60/91

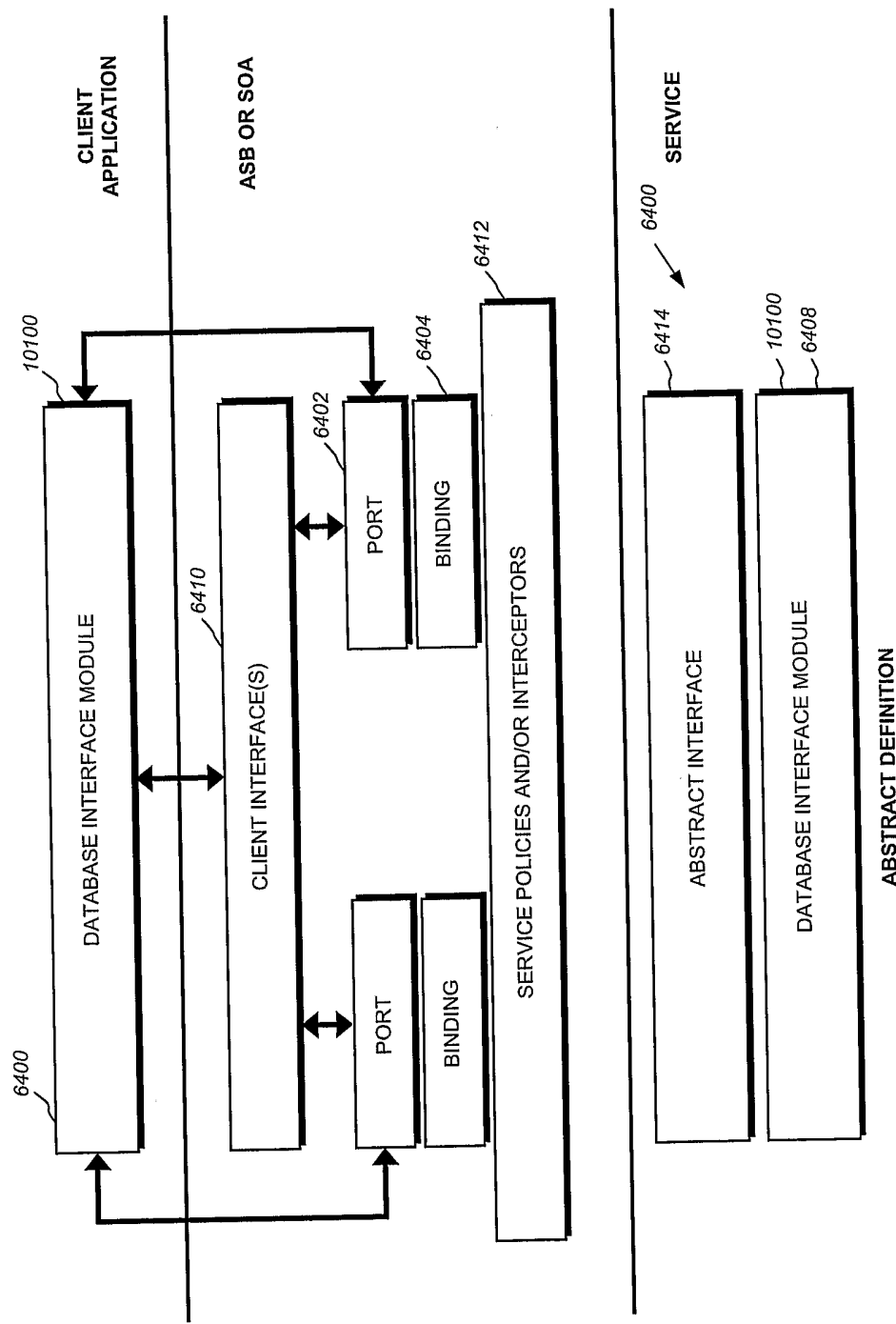


Fig. 53

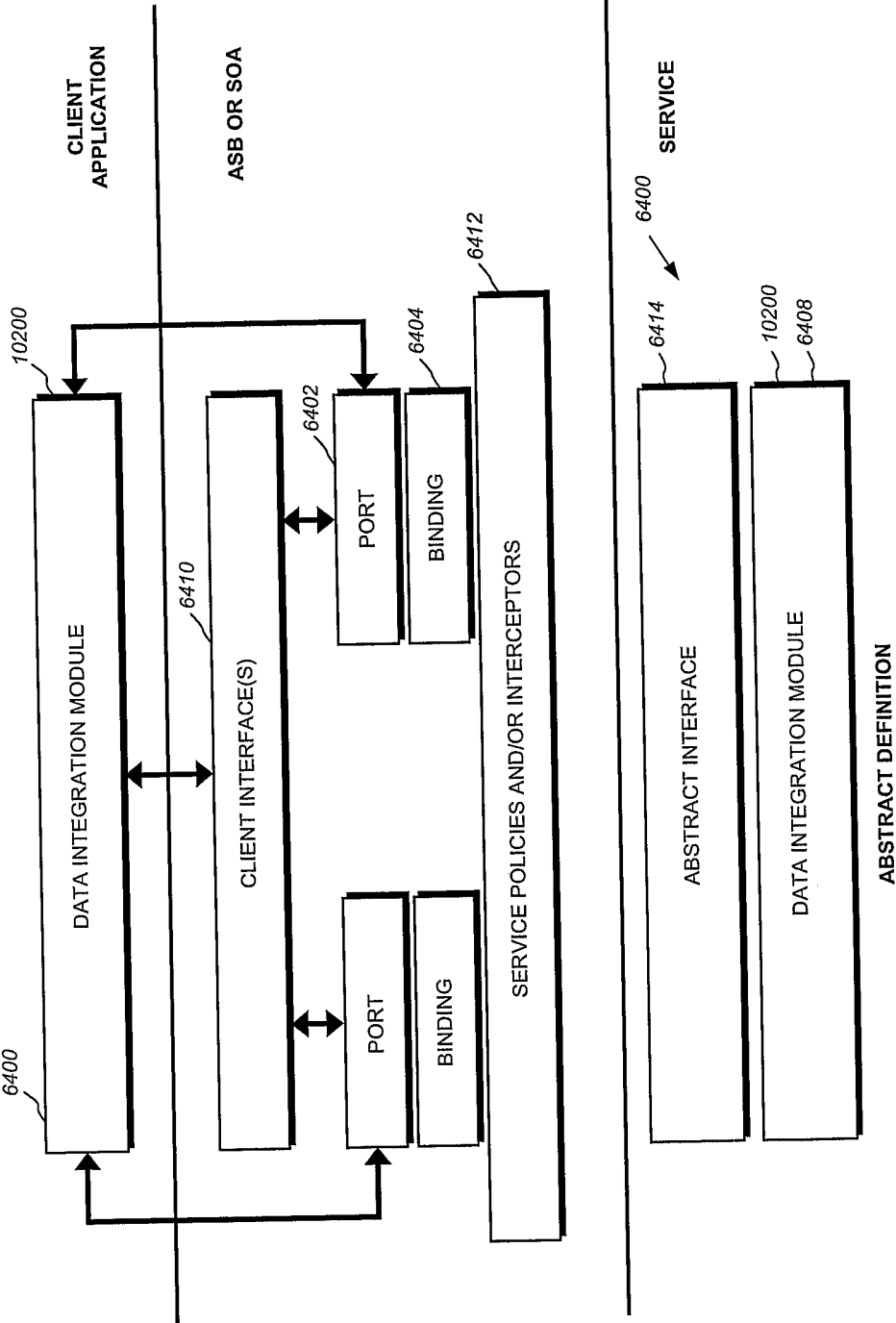


Fig. 54

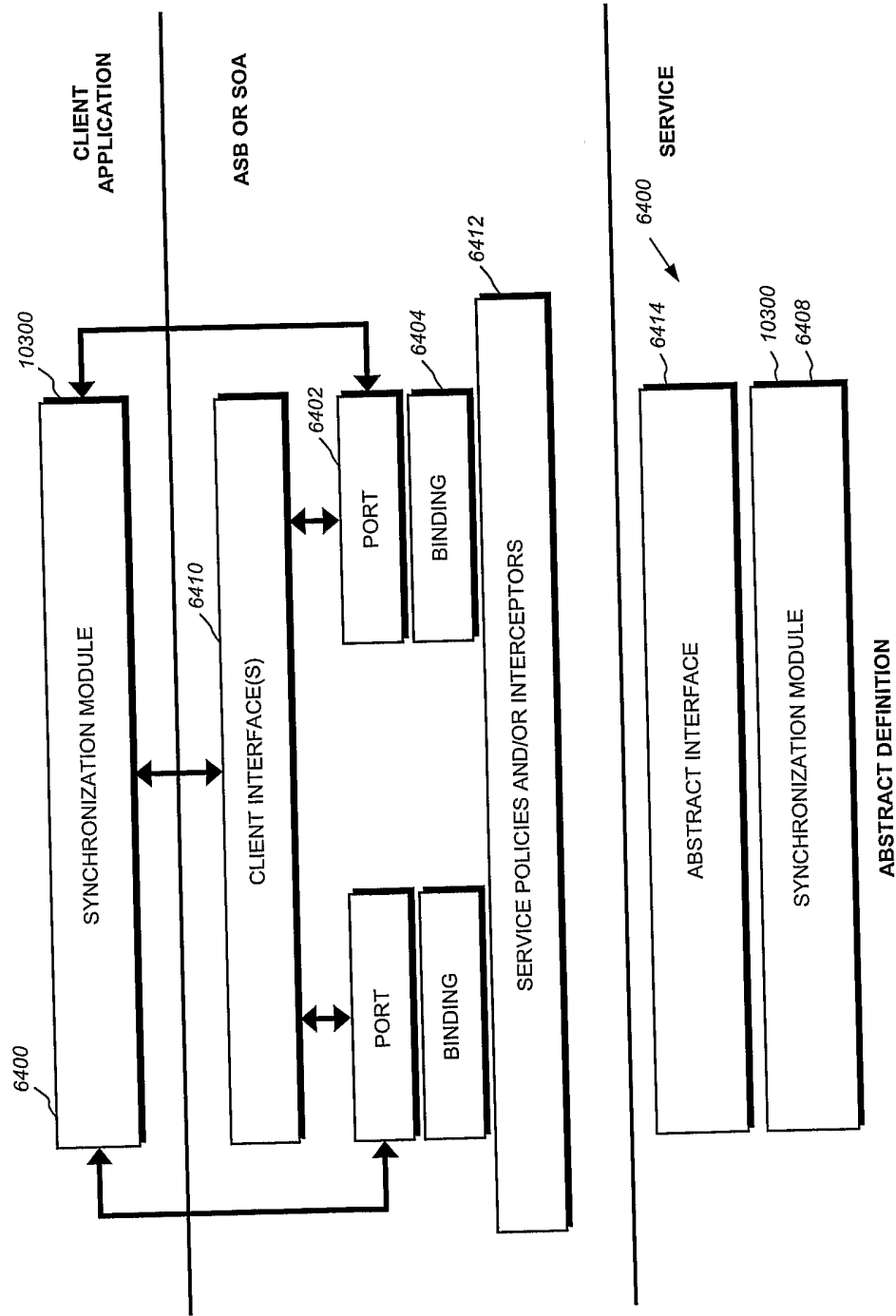


Fig. 55

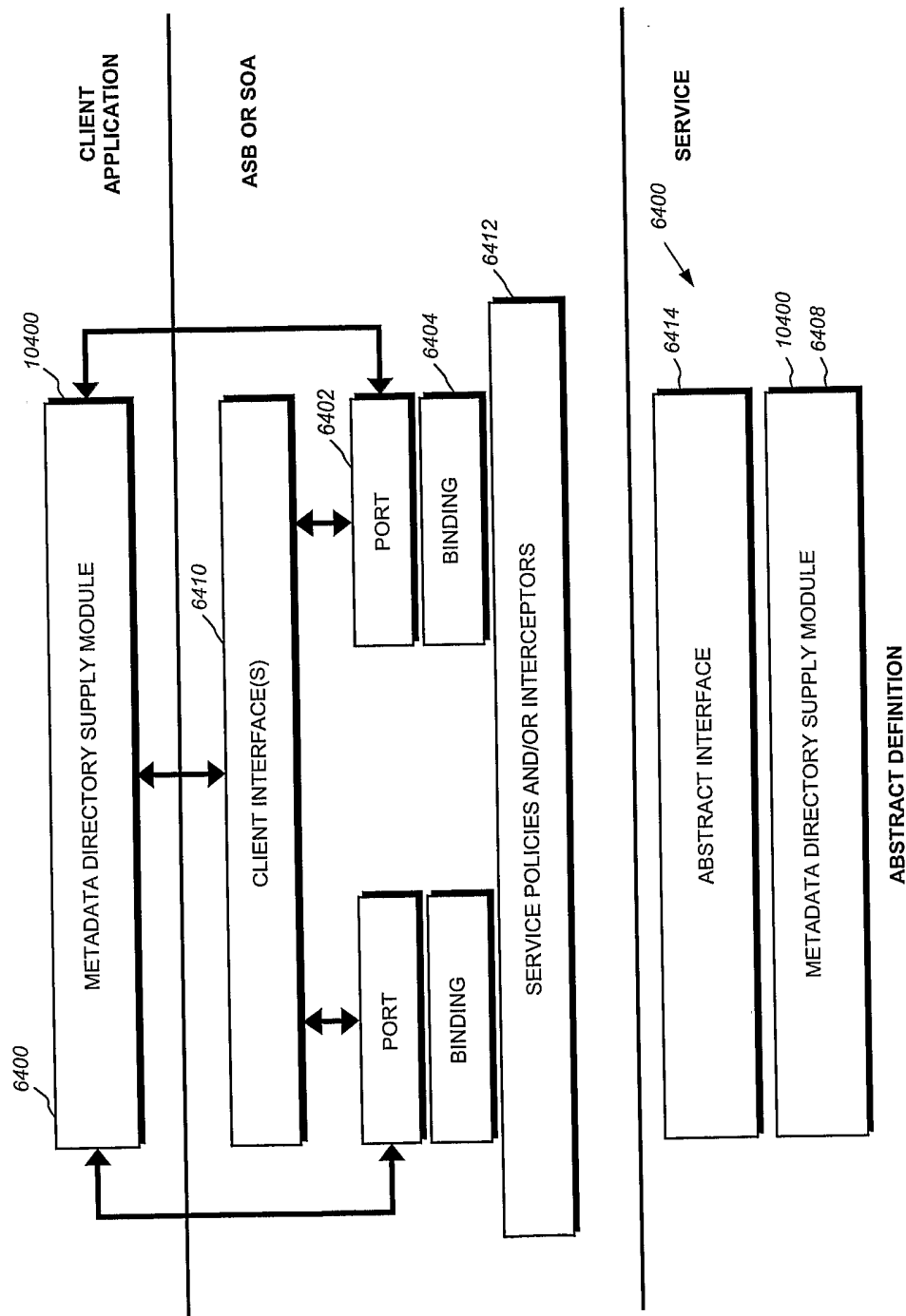


Fig. 56

64/91

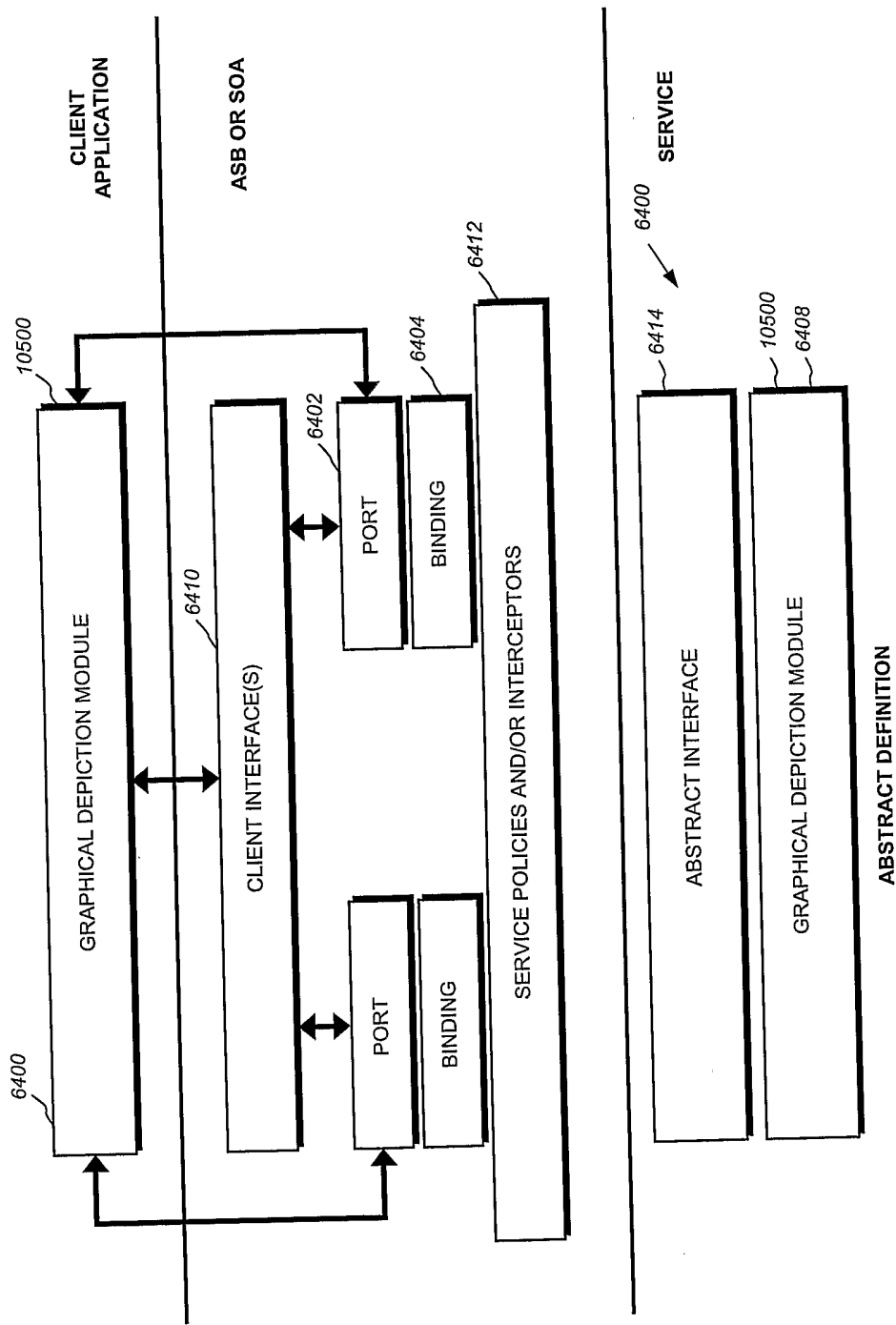


Fig. 57

65/91

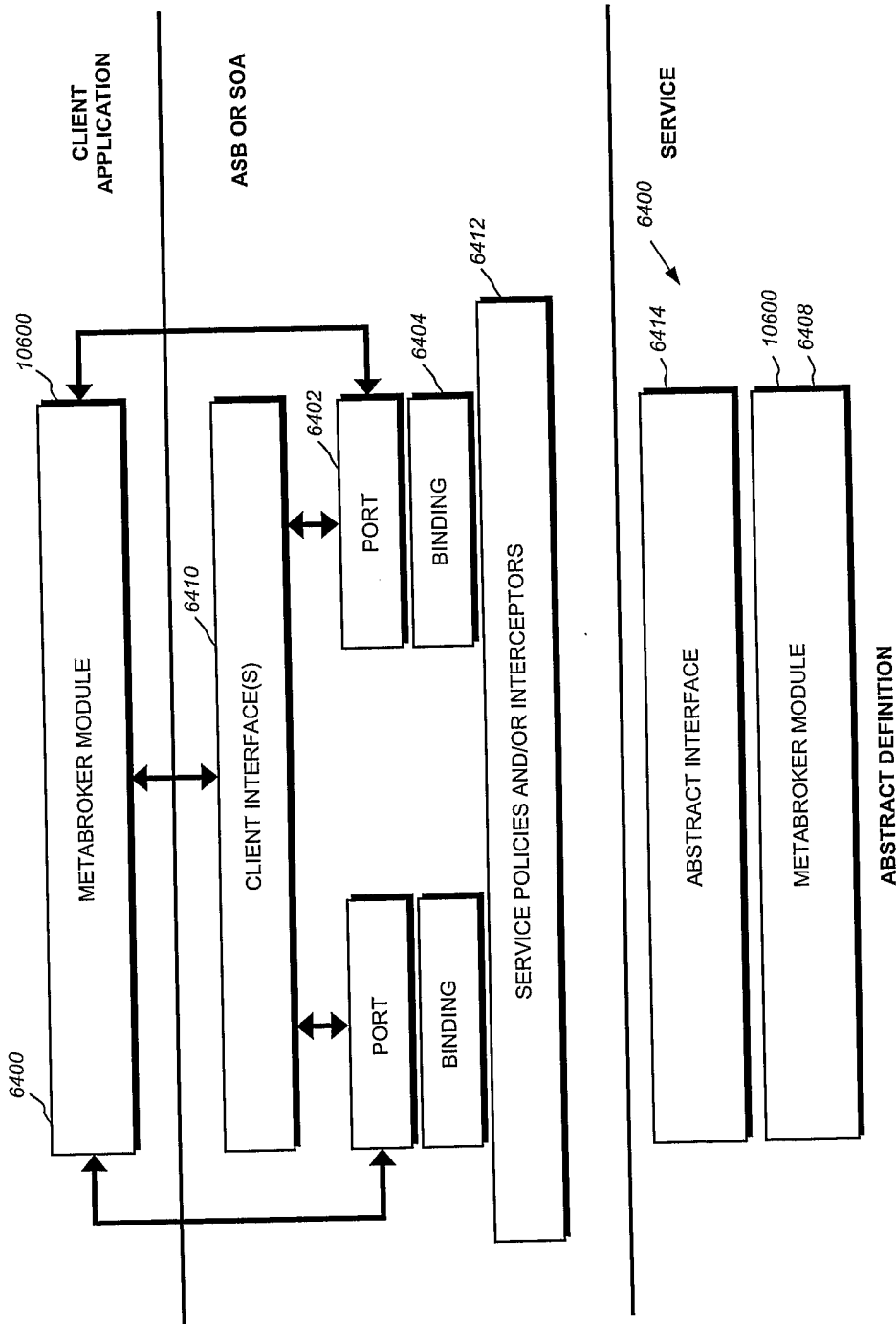


Fig. 58

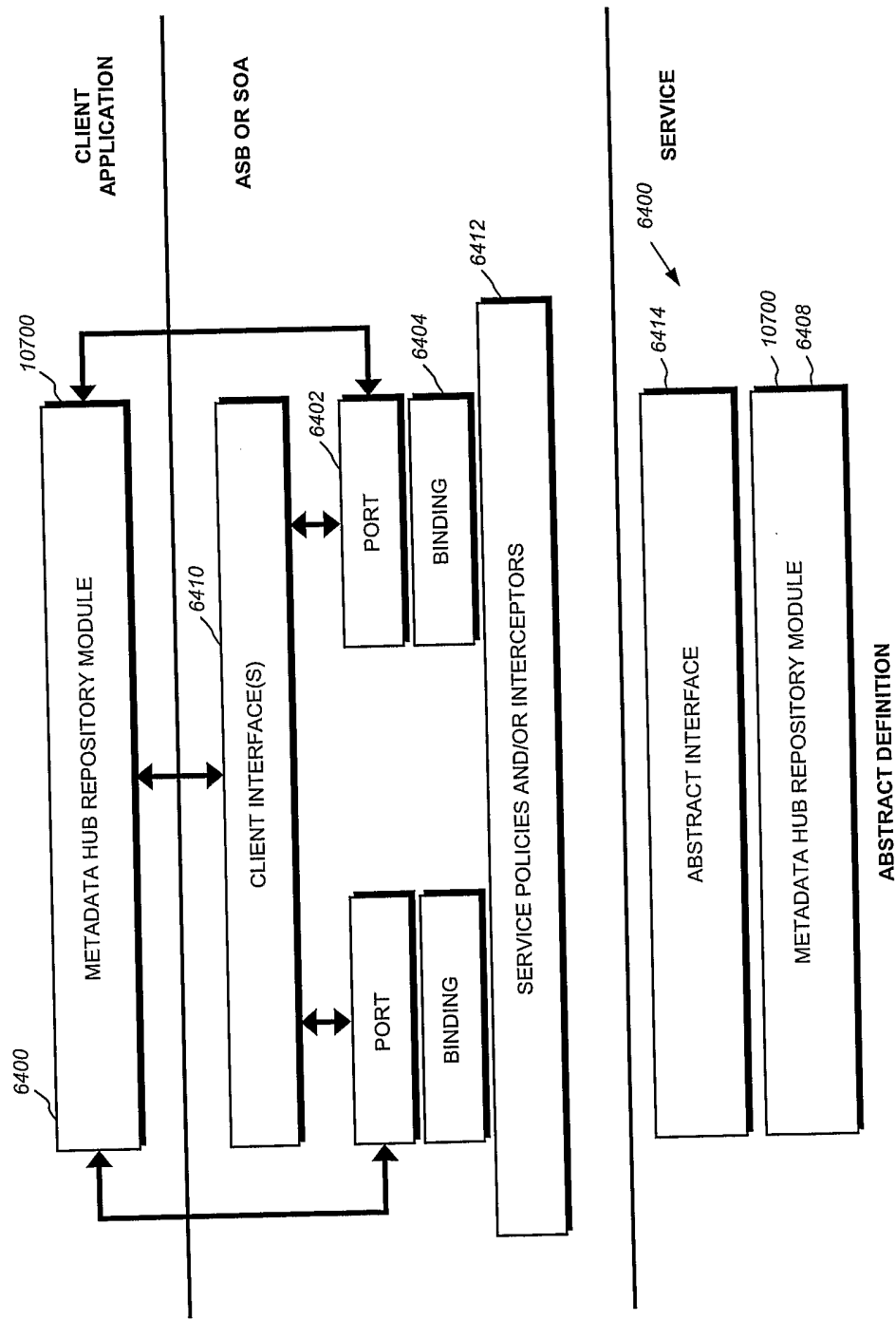


Fig. 59

67/91

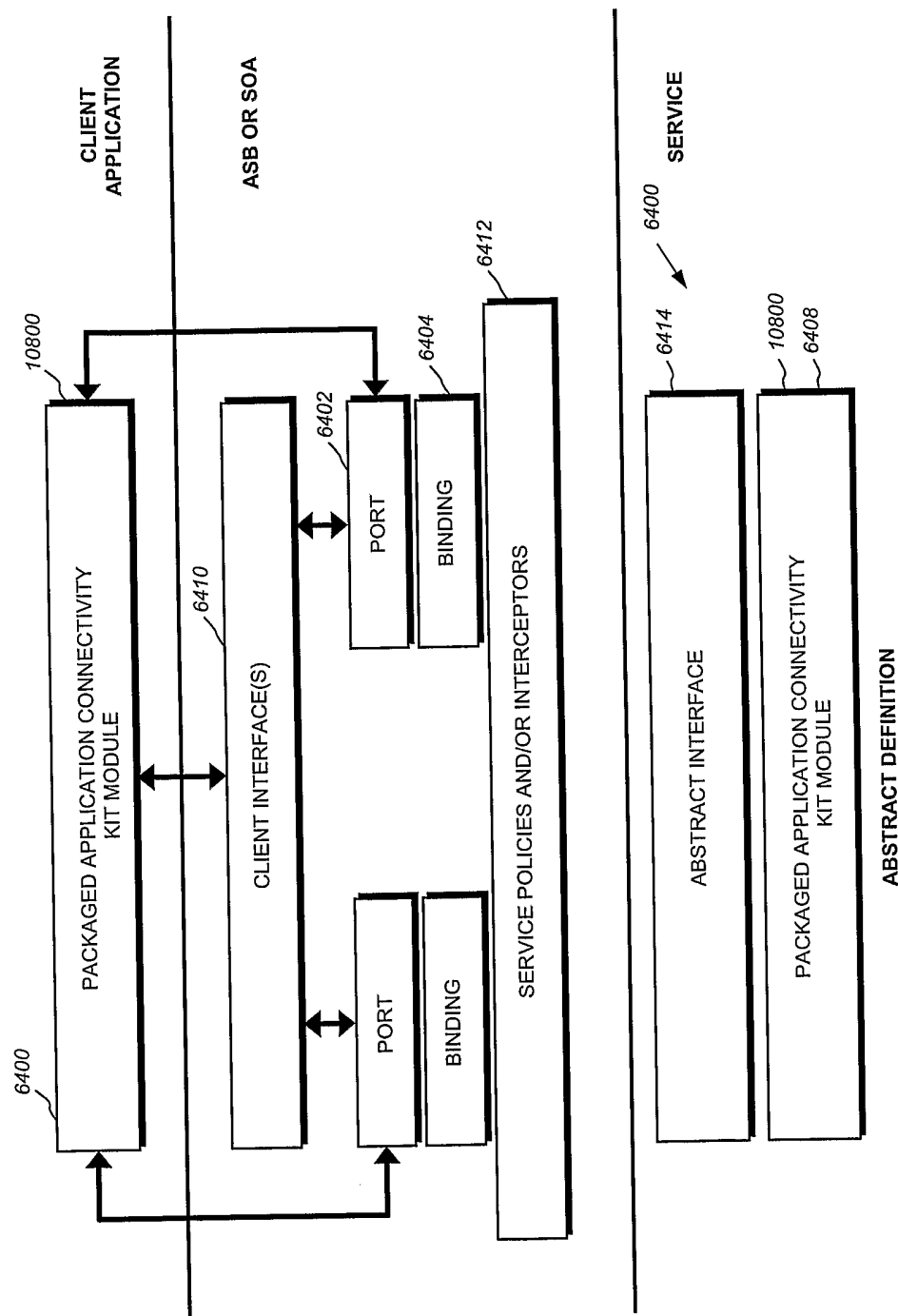


Fig. 60

68/91

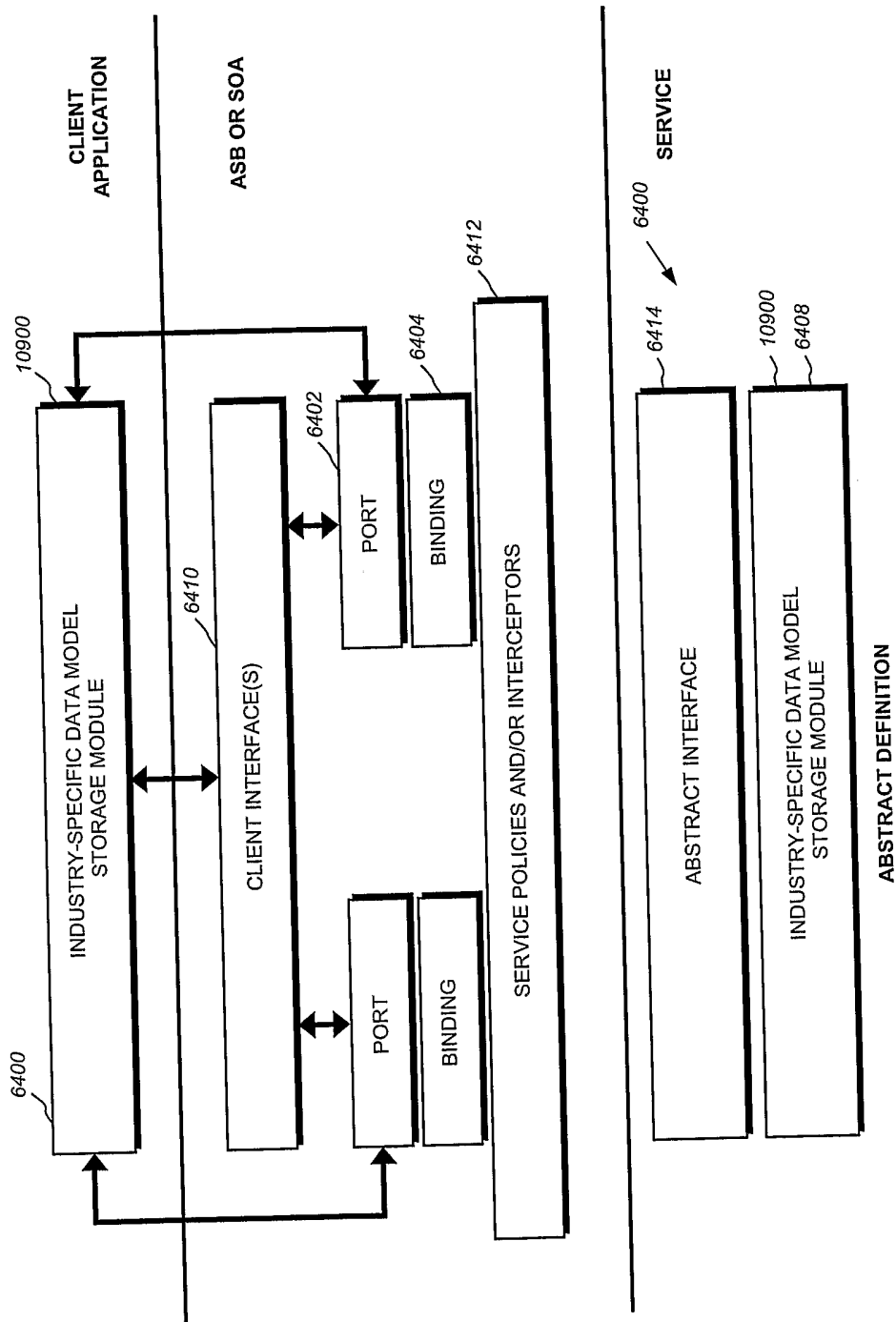


Fig. 61

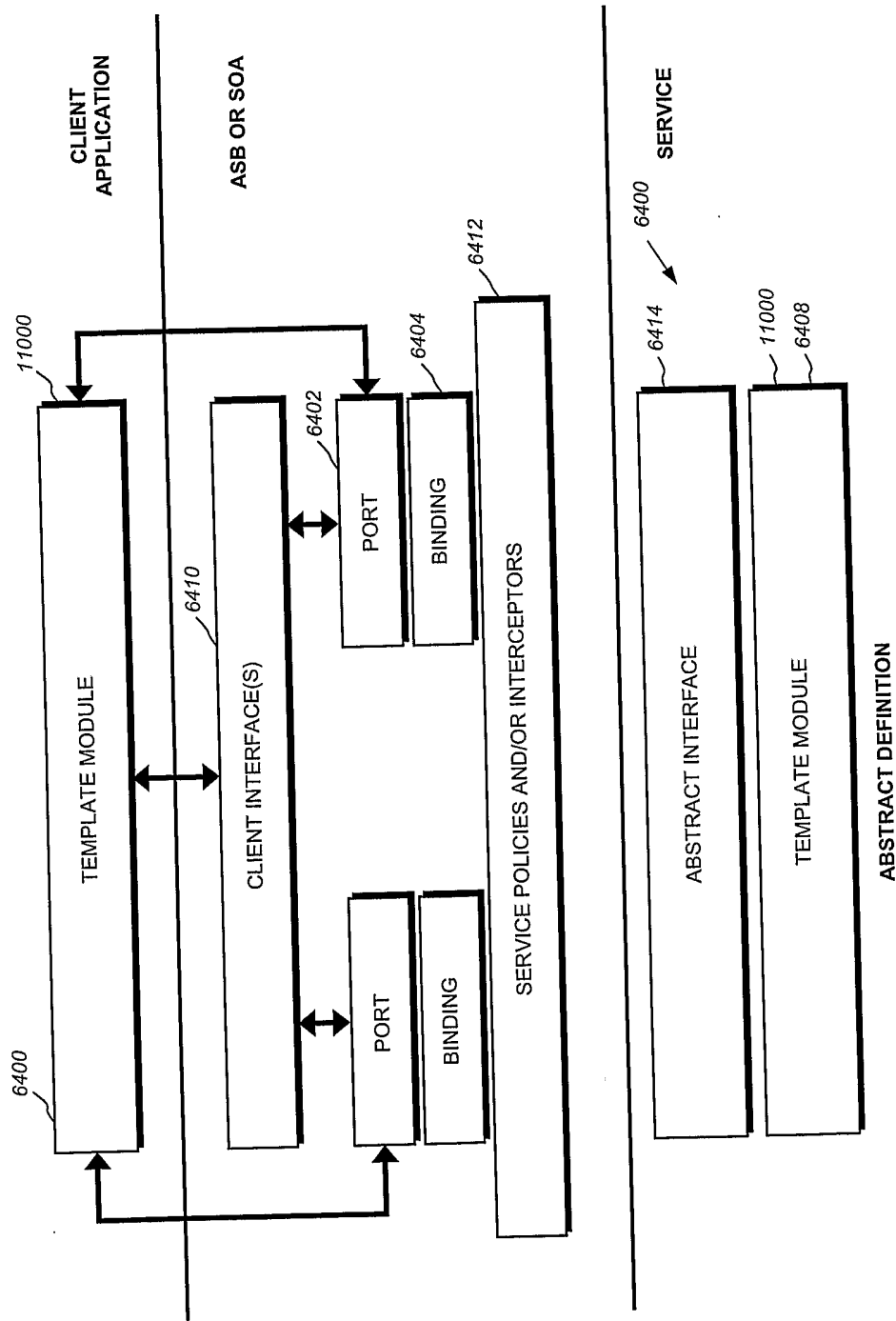


Fig. 62

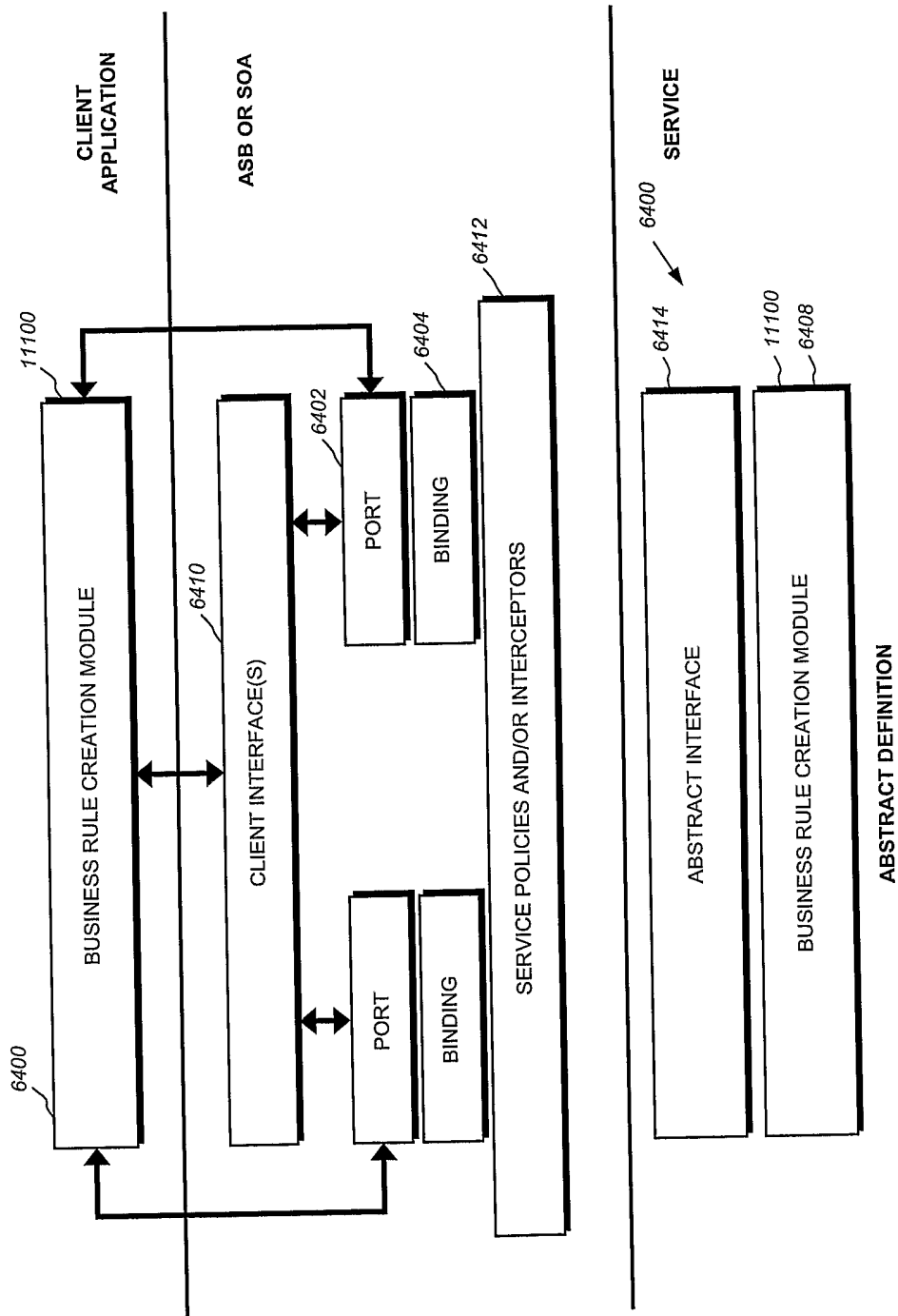


Fig. 63

71/91

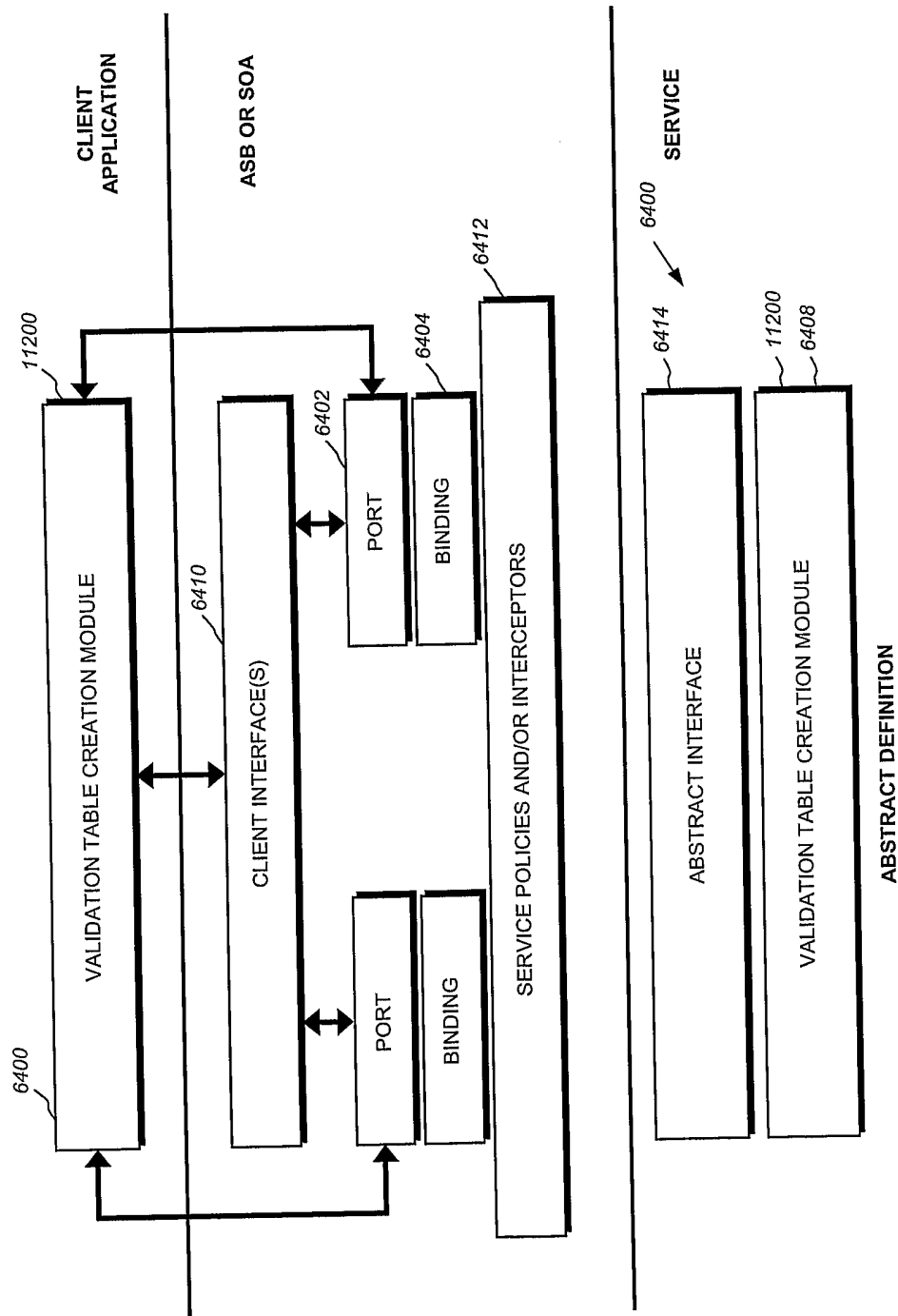


Fig. 64

72/91

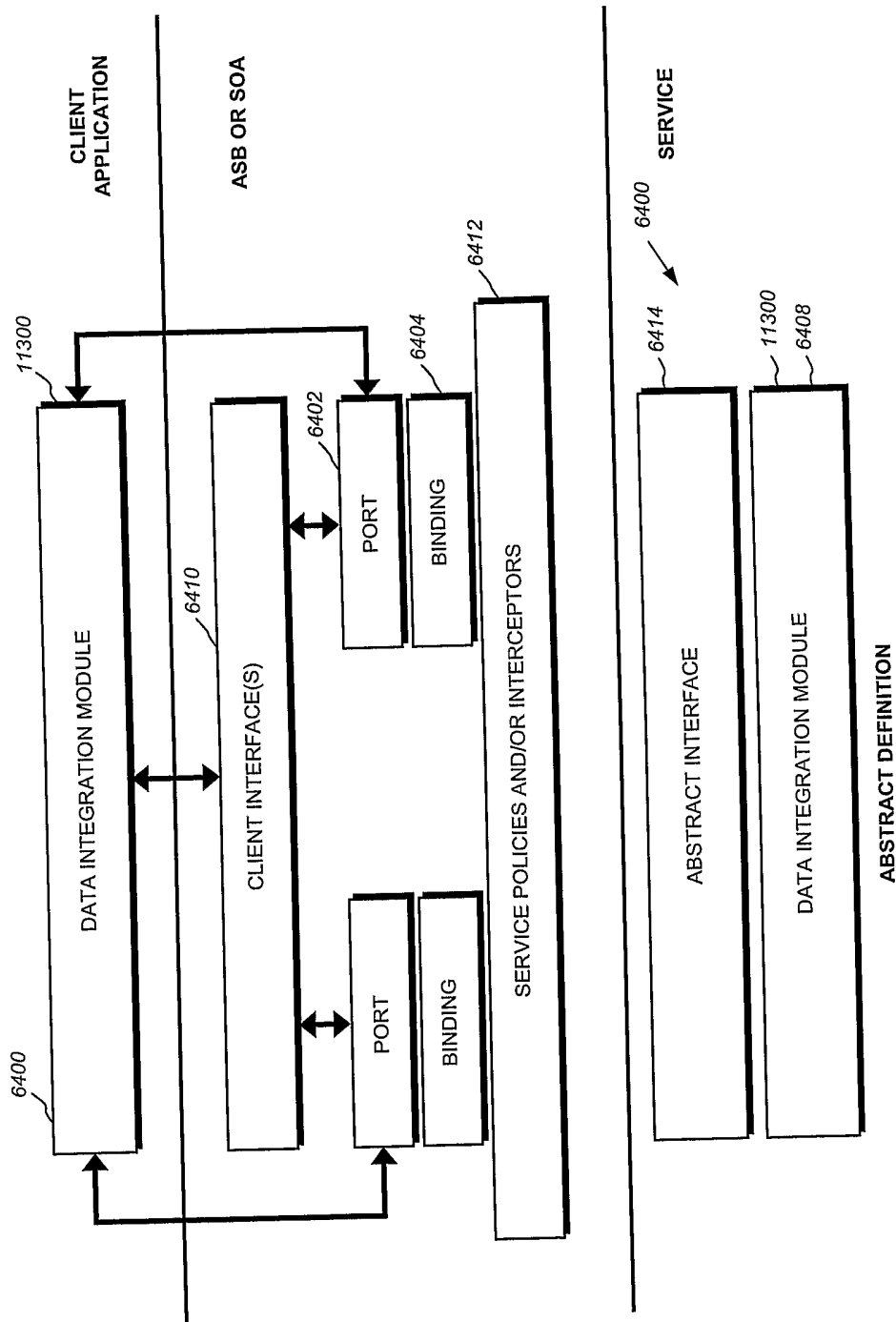


Fig. 65

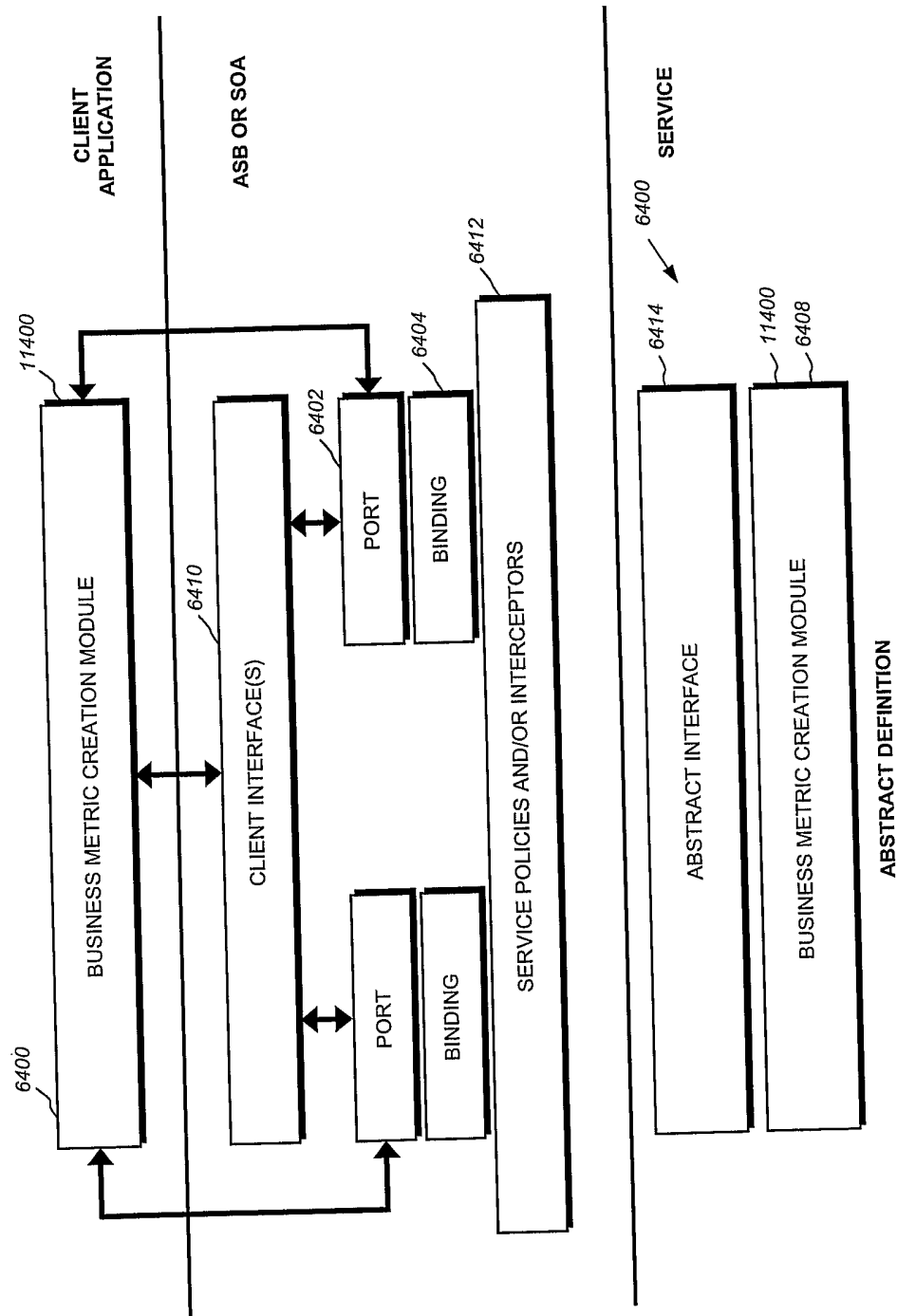


Fig. 66

74/91

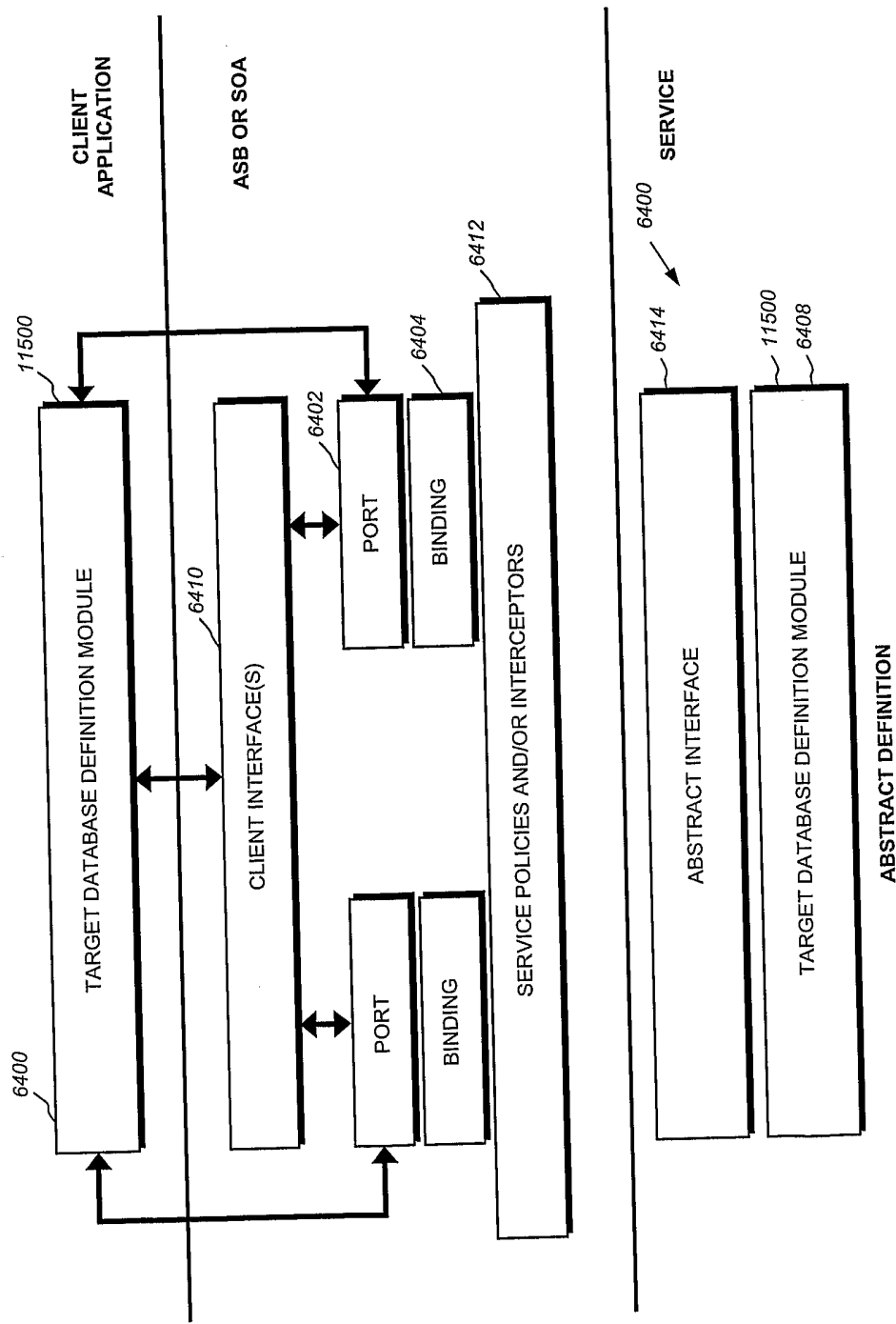


Fig. 67

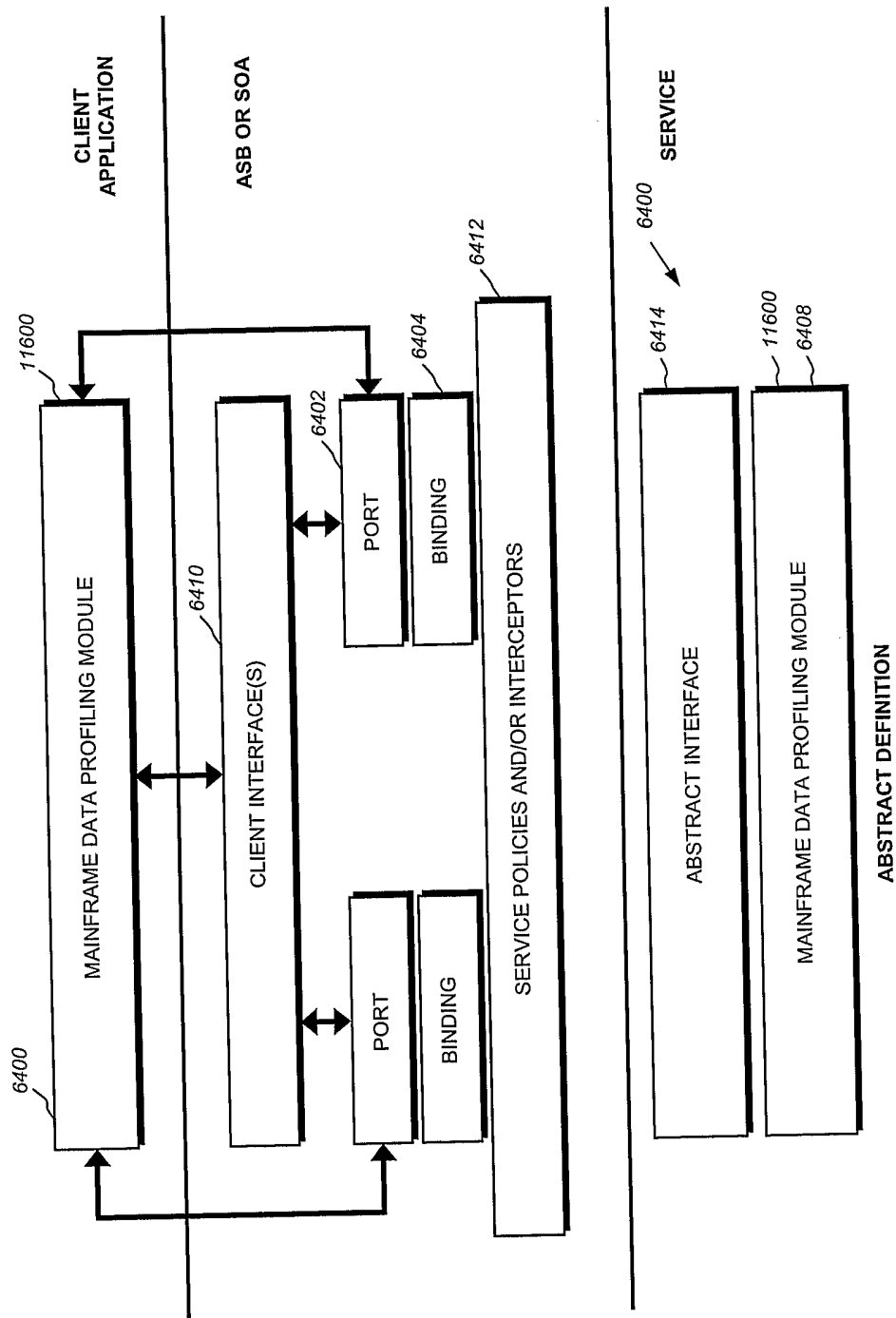


Fig. 68

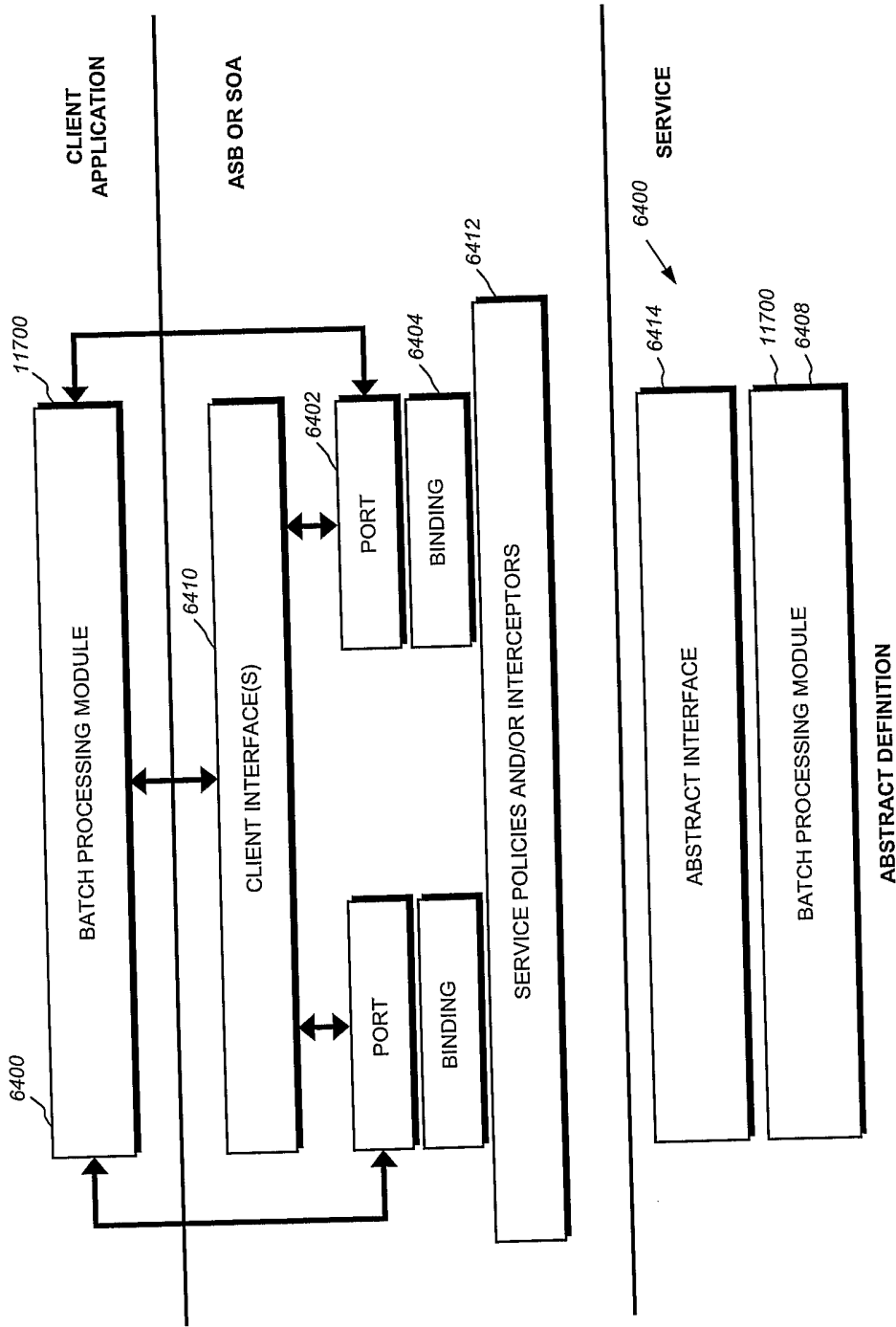


Fig. 69

77/91

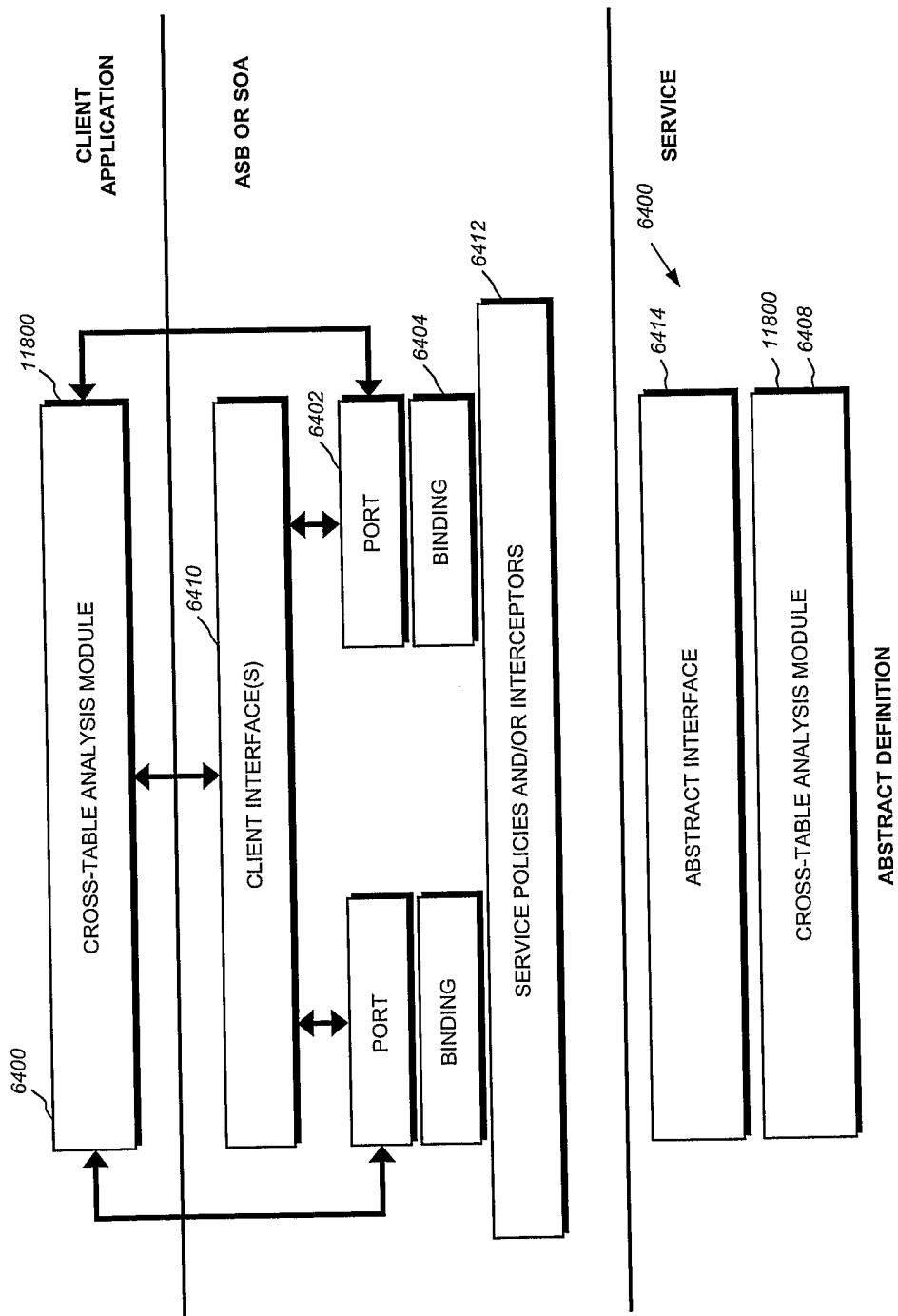


Fig. 70

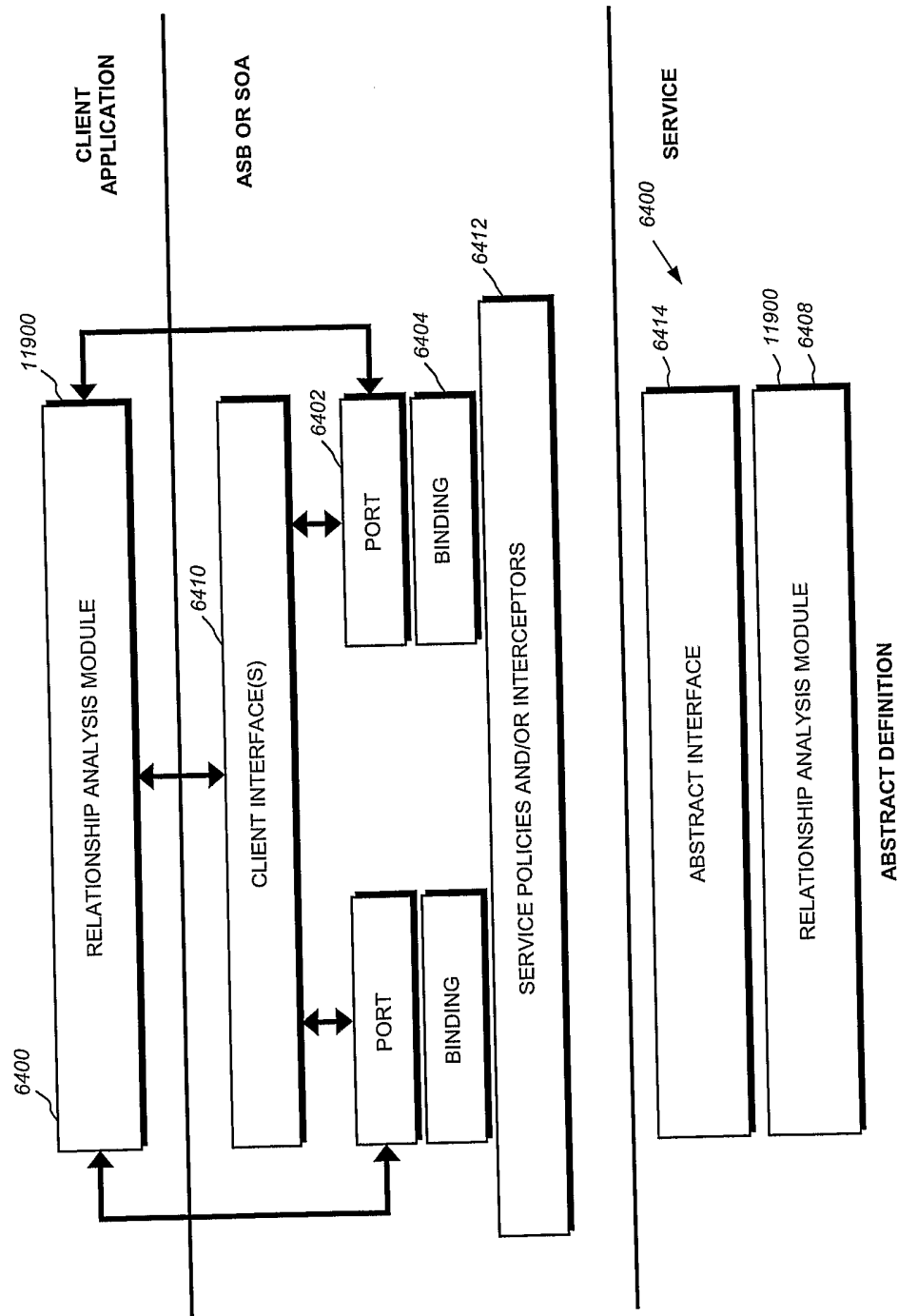


Fig. 71

79/91

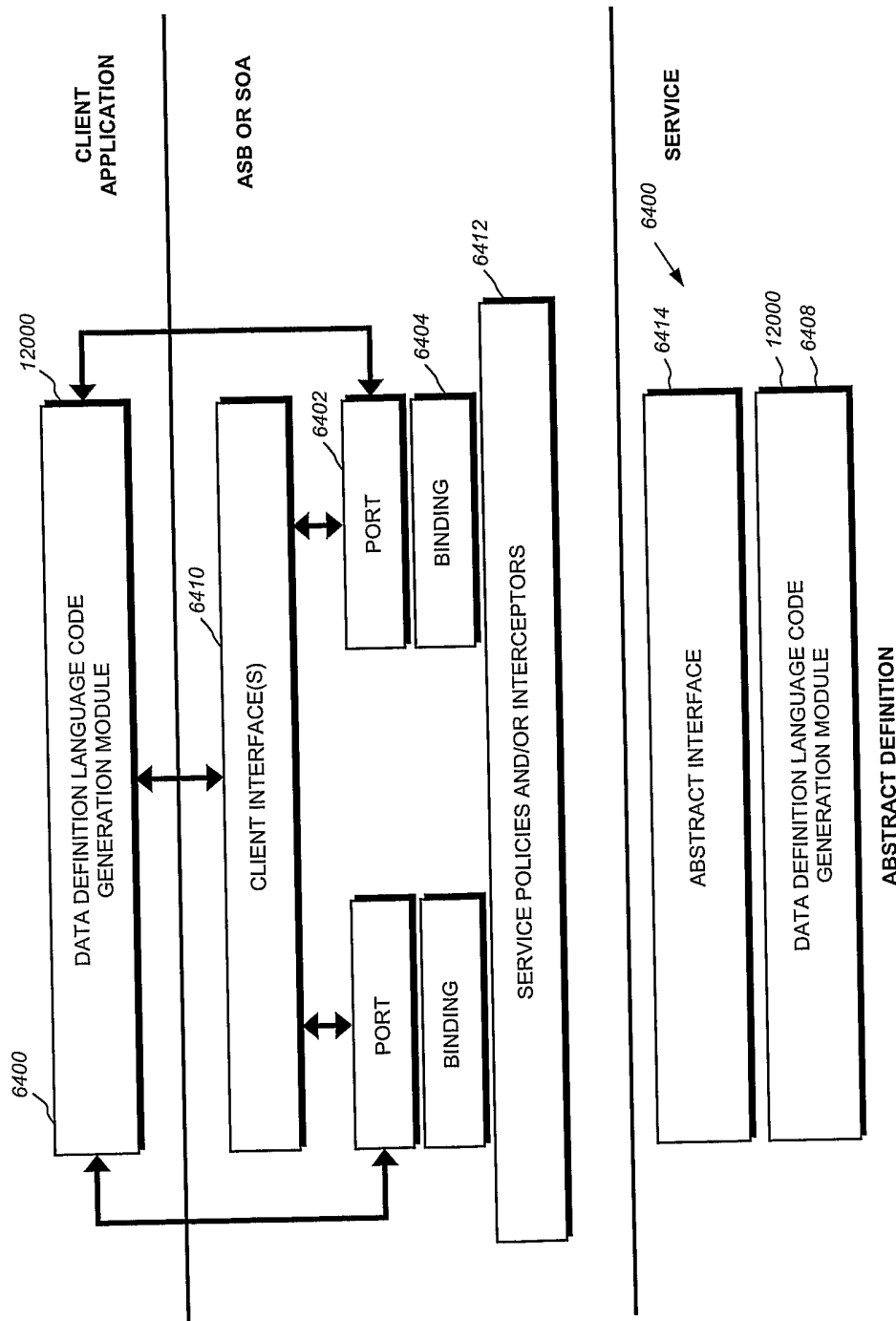


Fig. 72

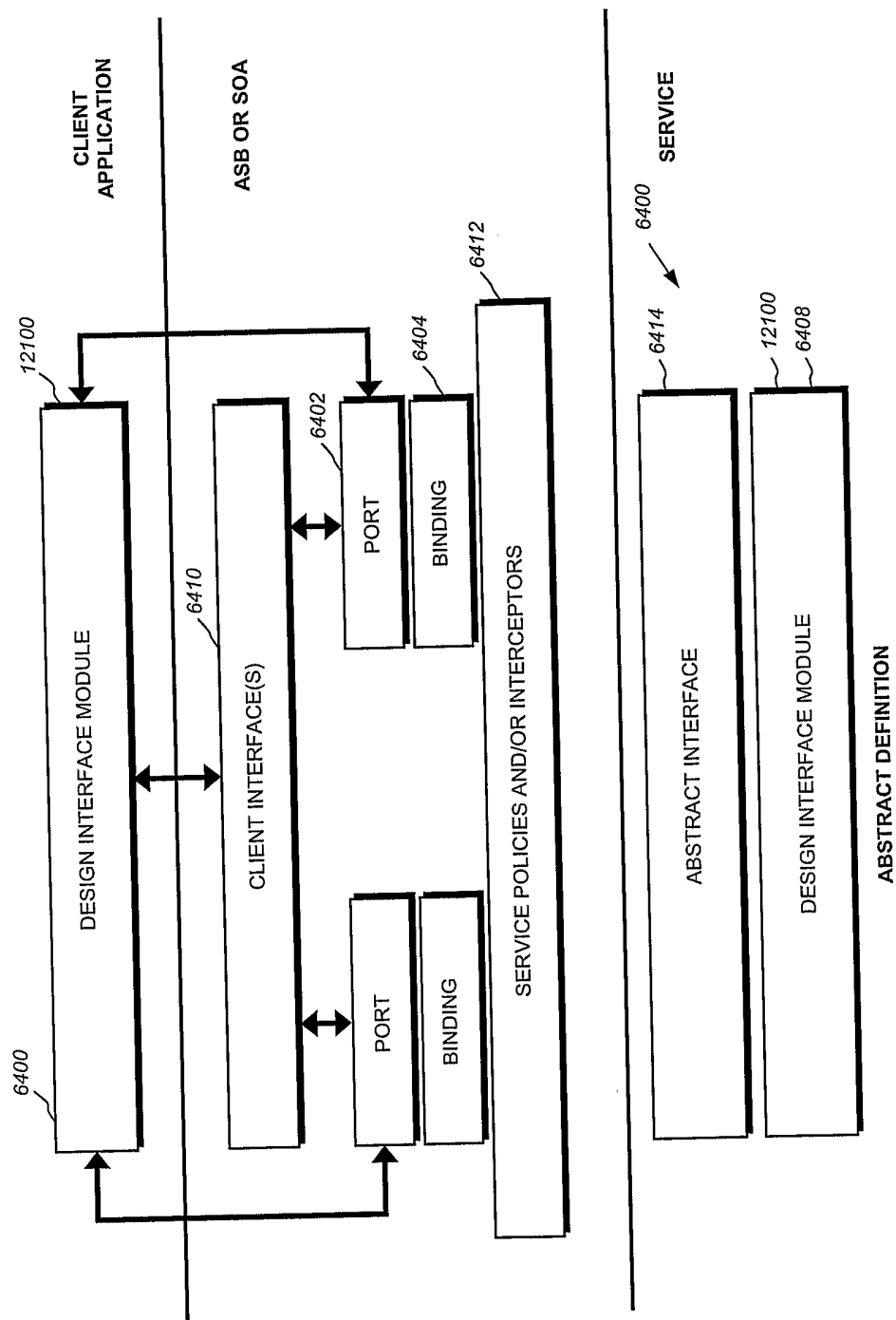


Fig. 73

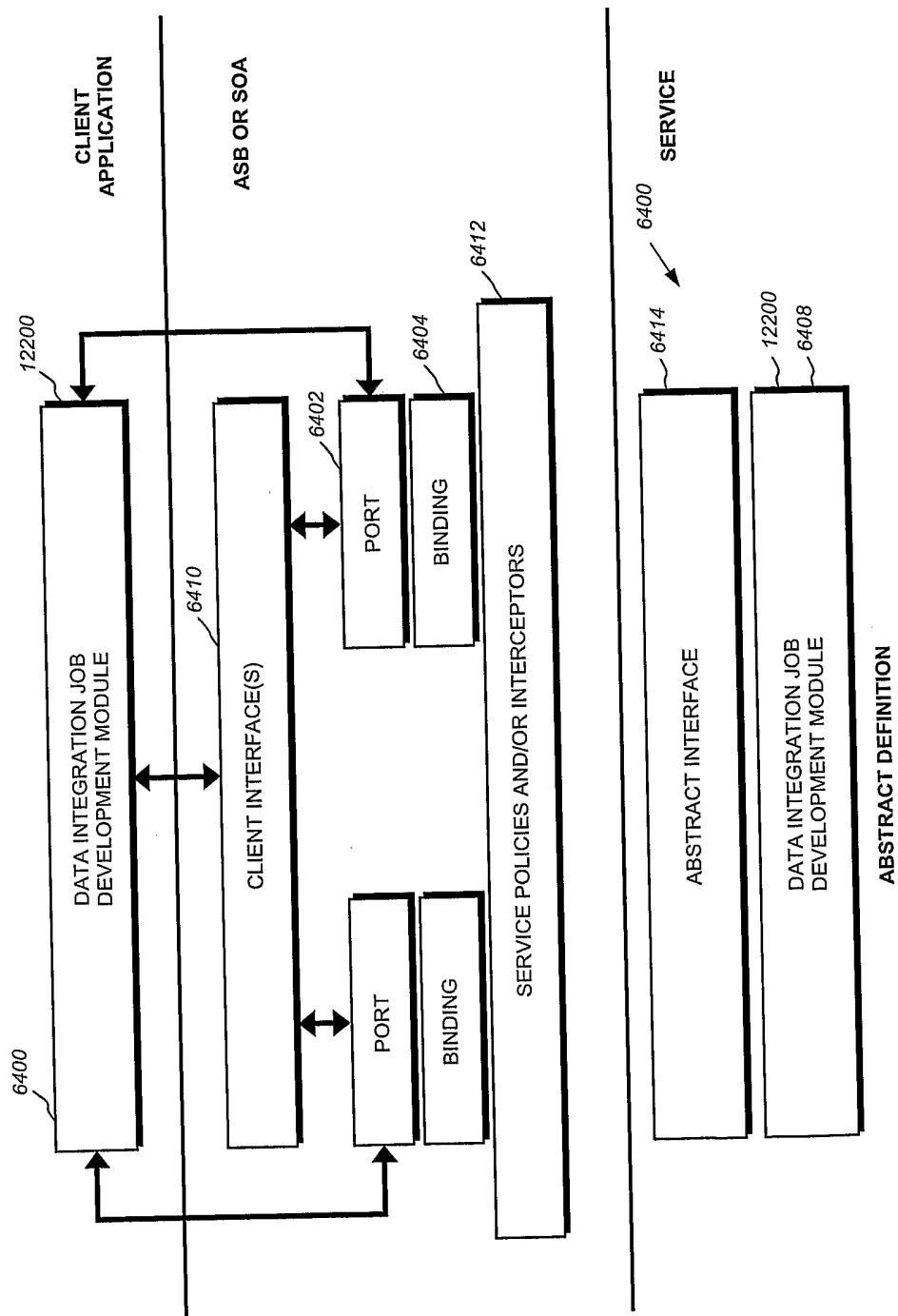


Fig. 74

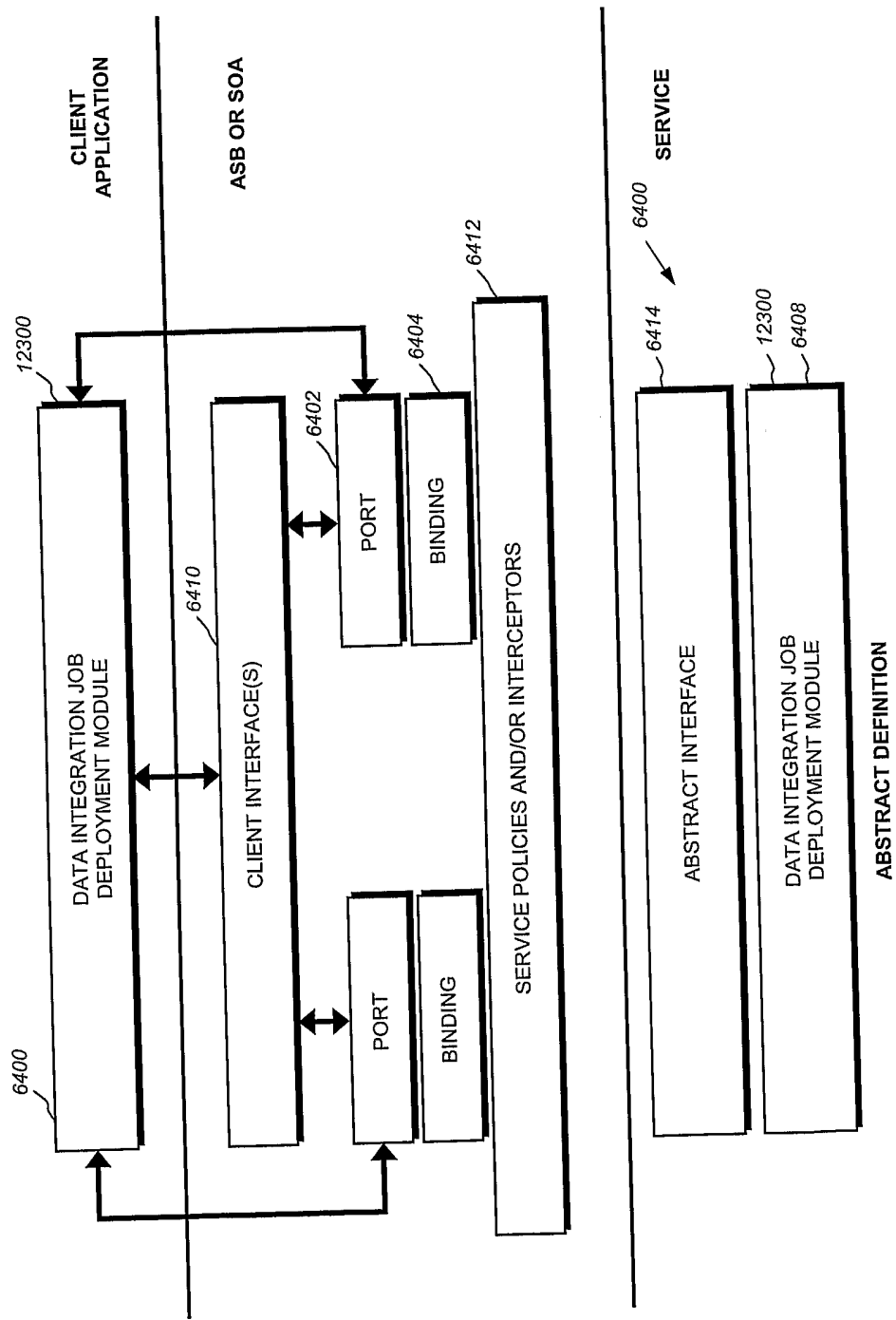


Fig. 75

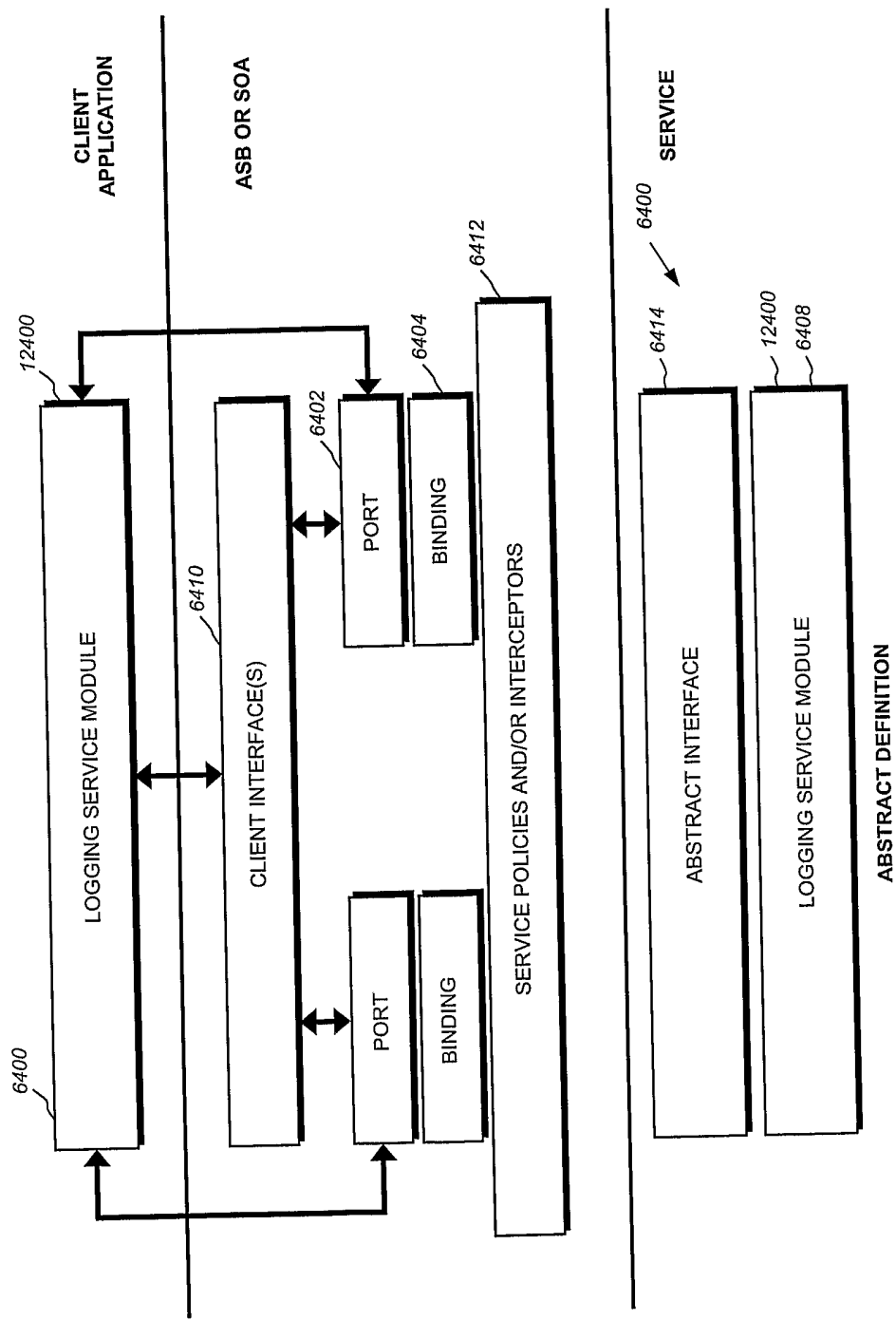


Fig. 76

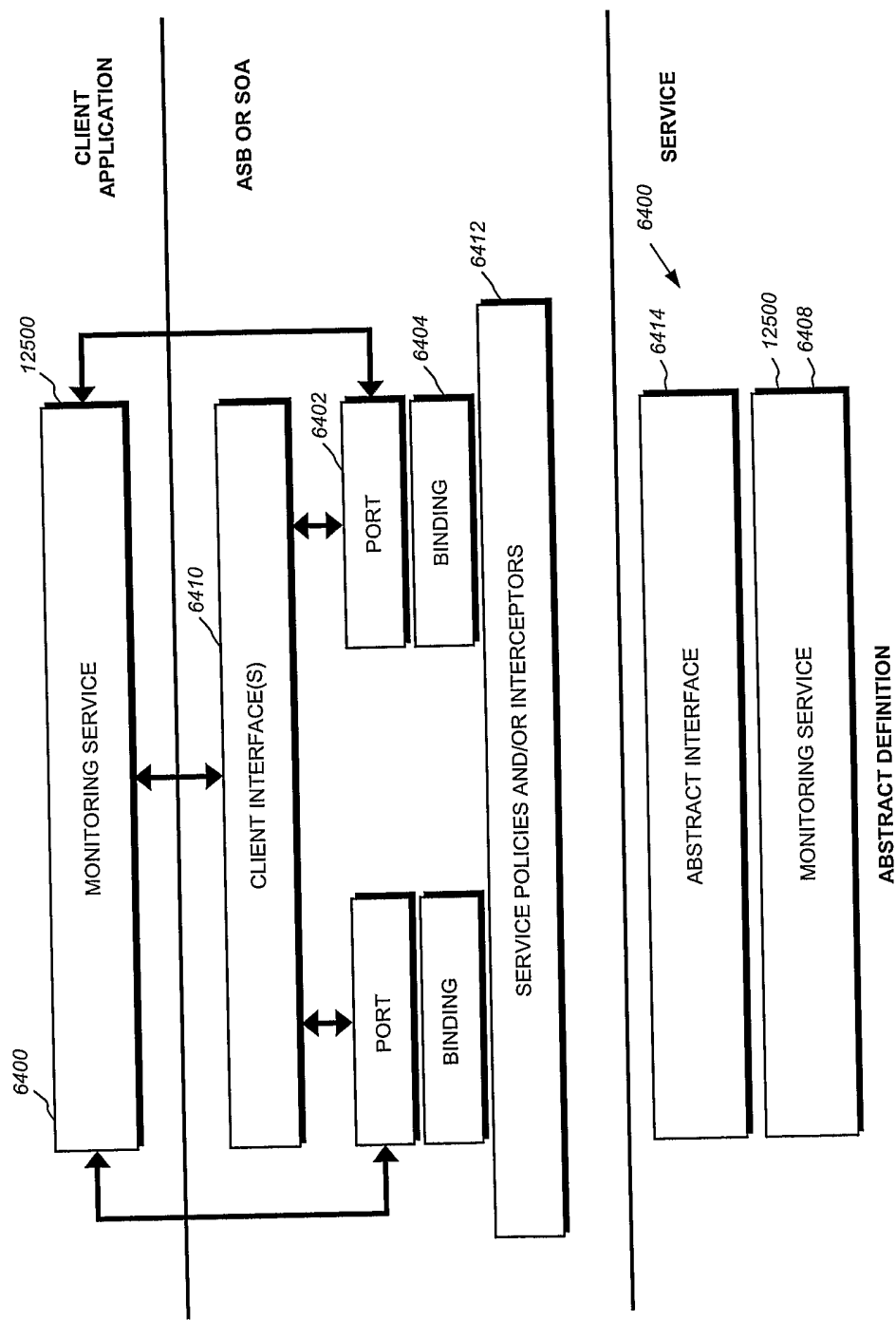


Fig. 77

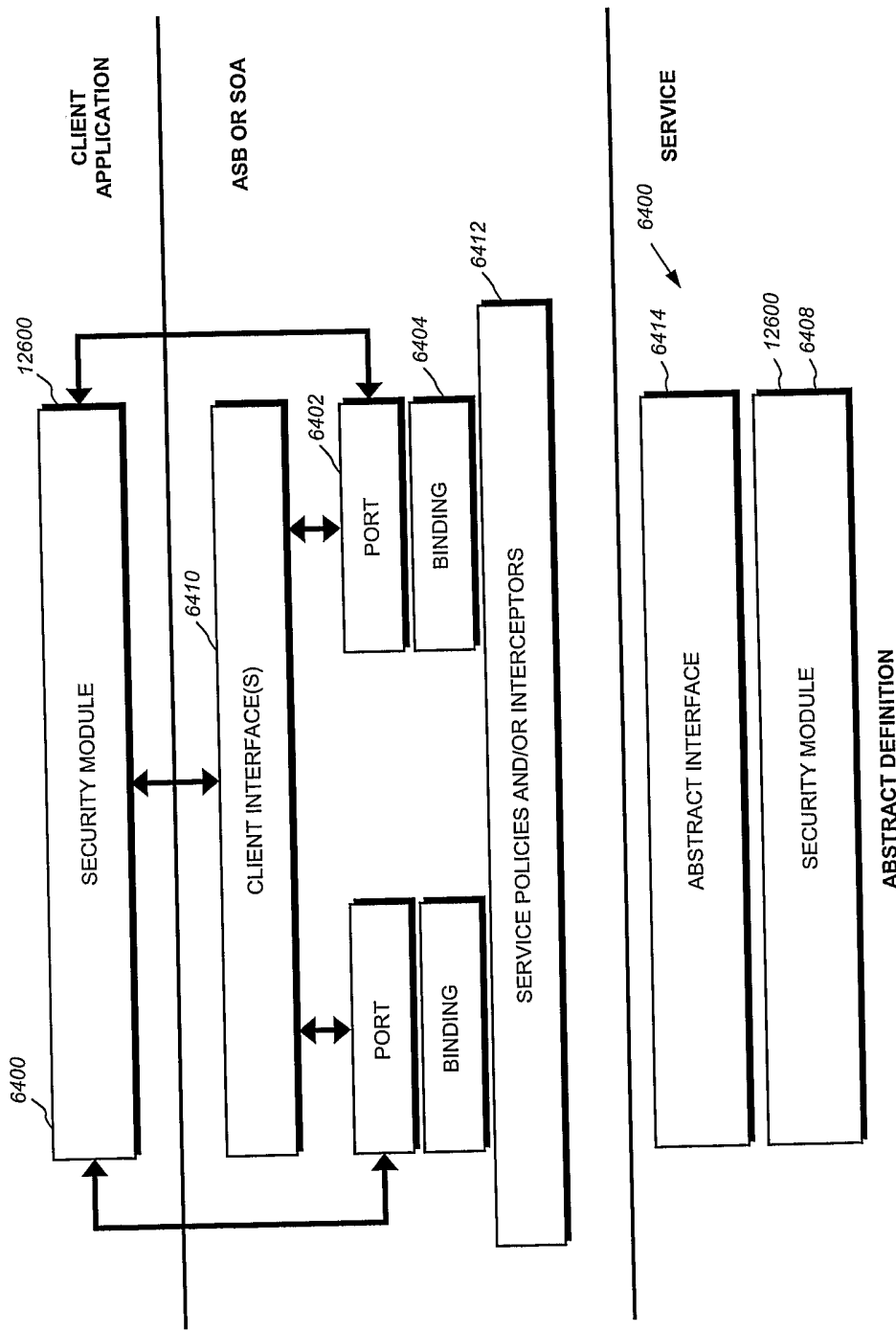


Fig. 78

86/91

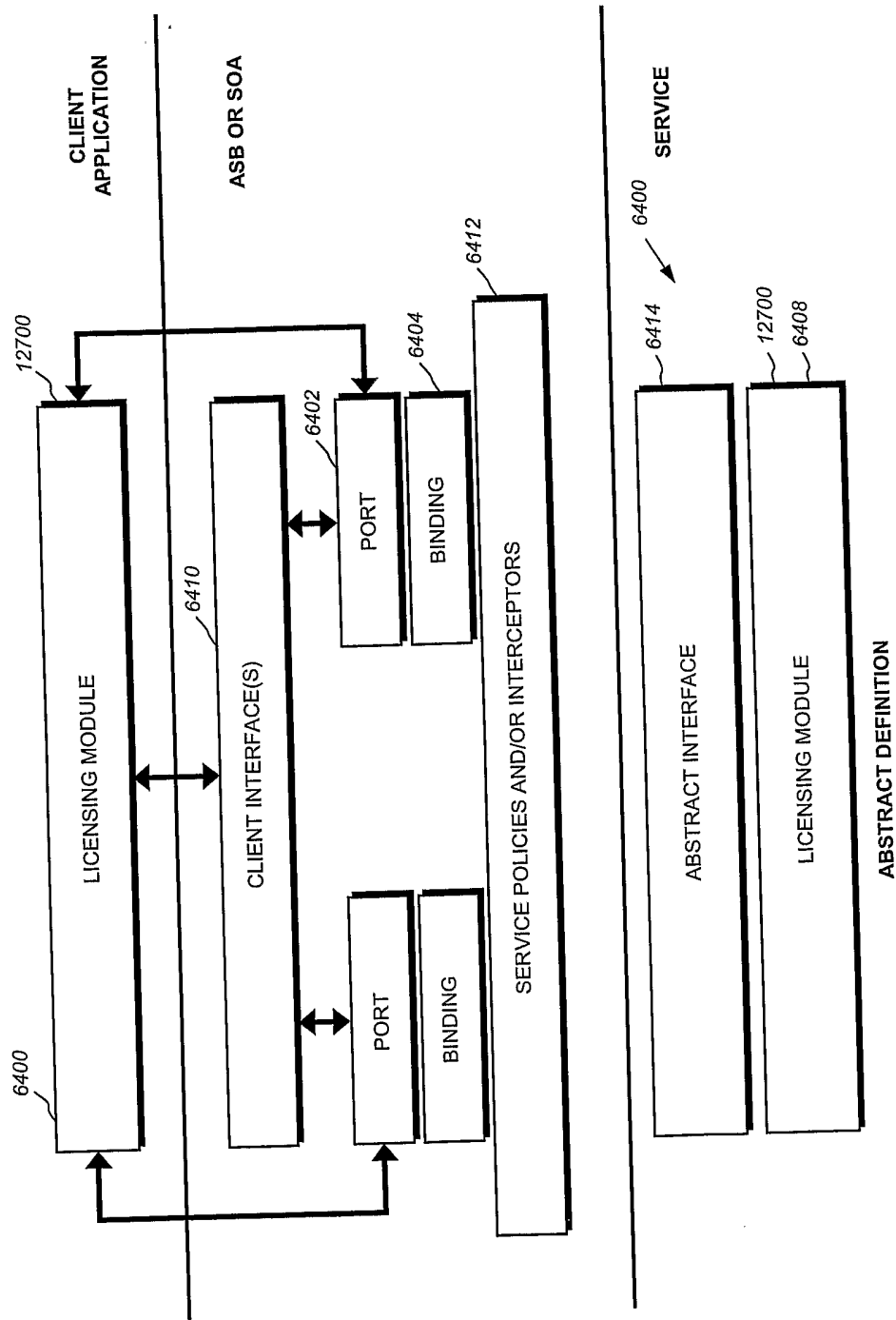


Fig. 79

87/91

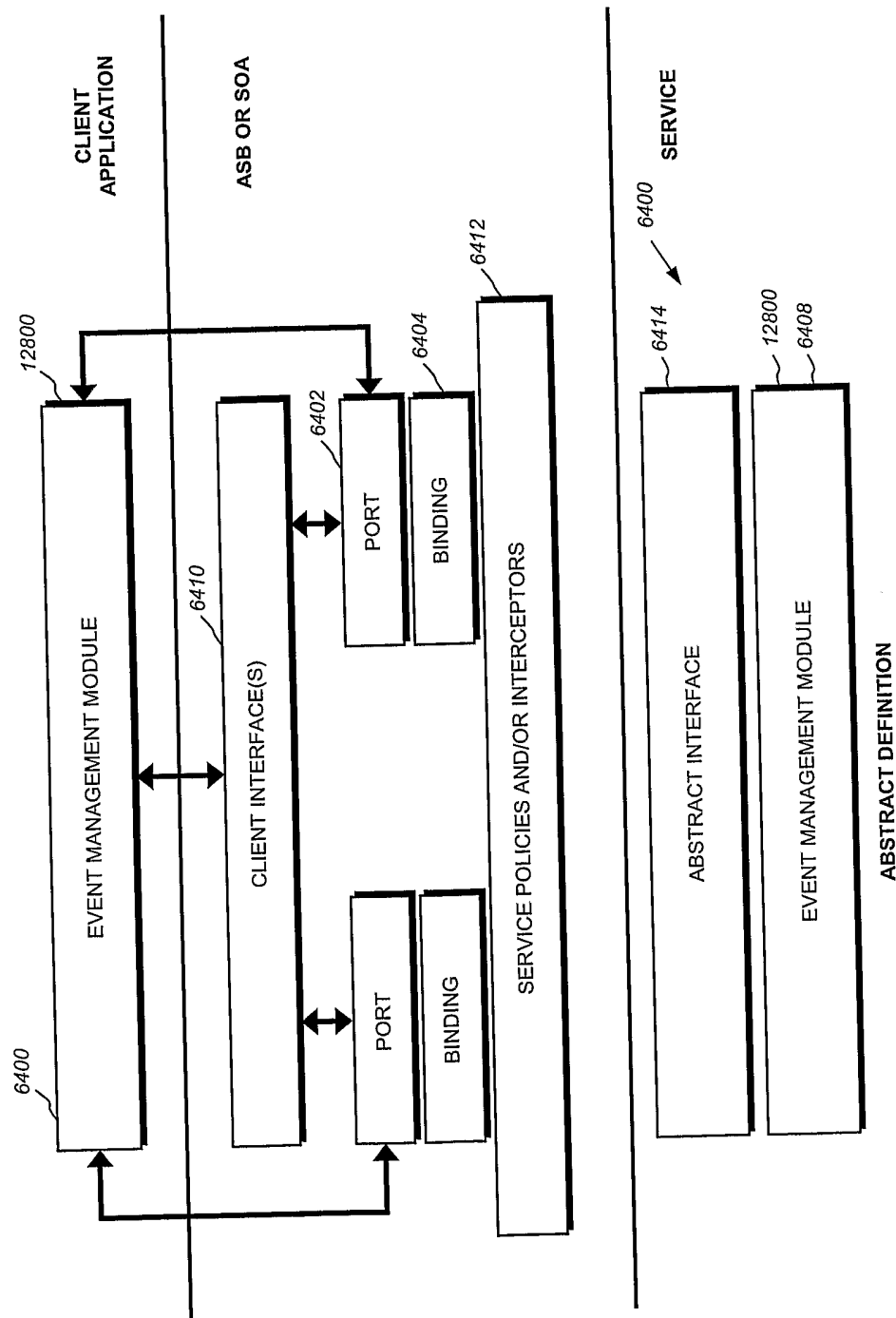


Fig. 80

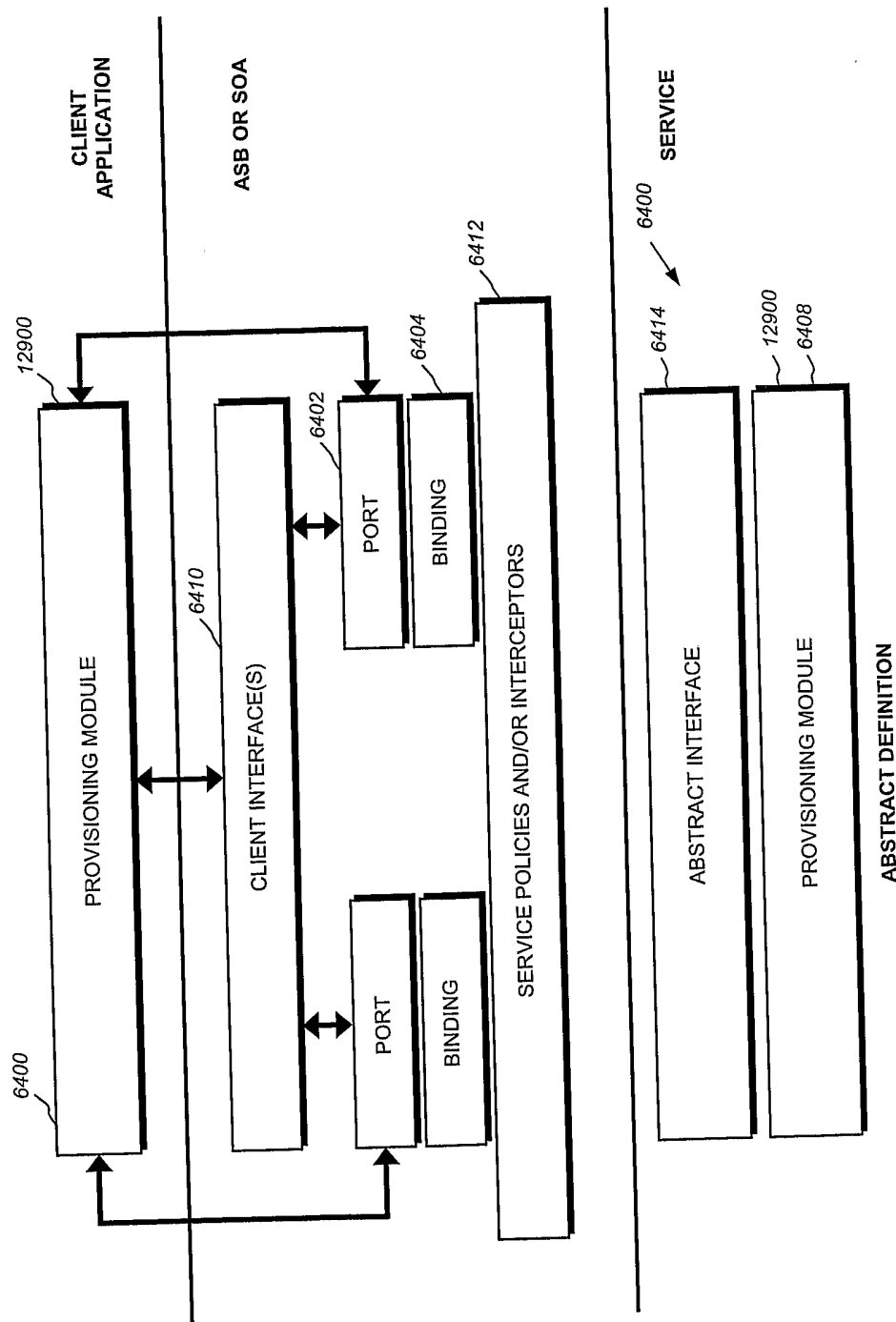


Fig. 81

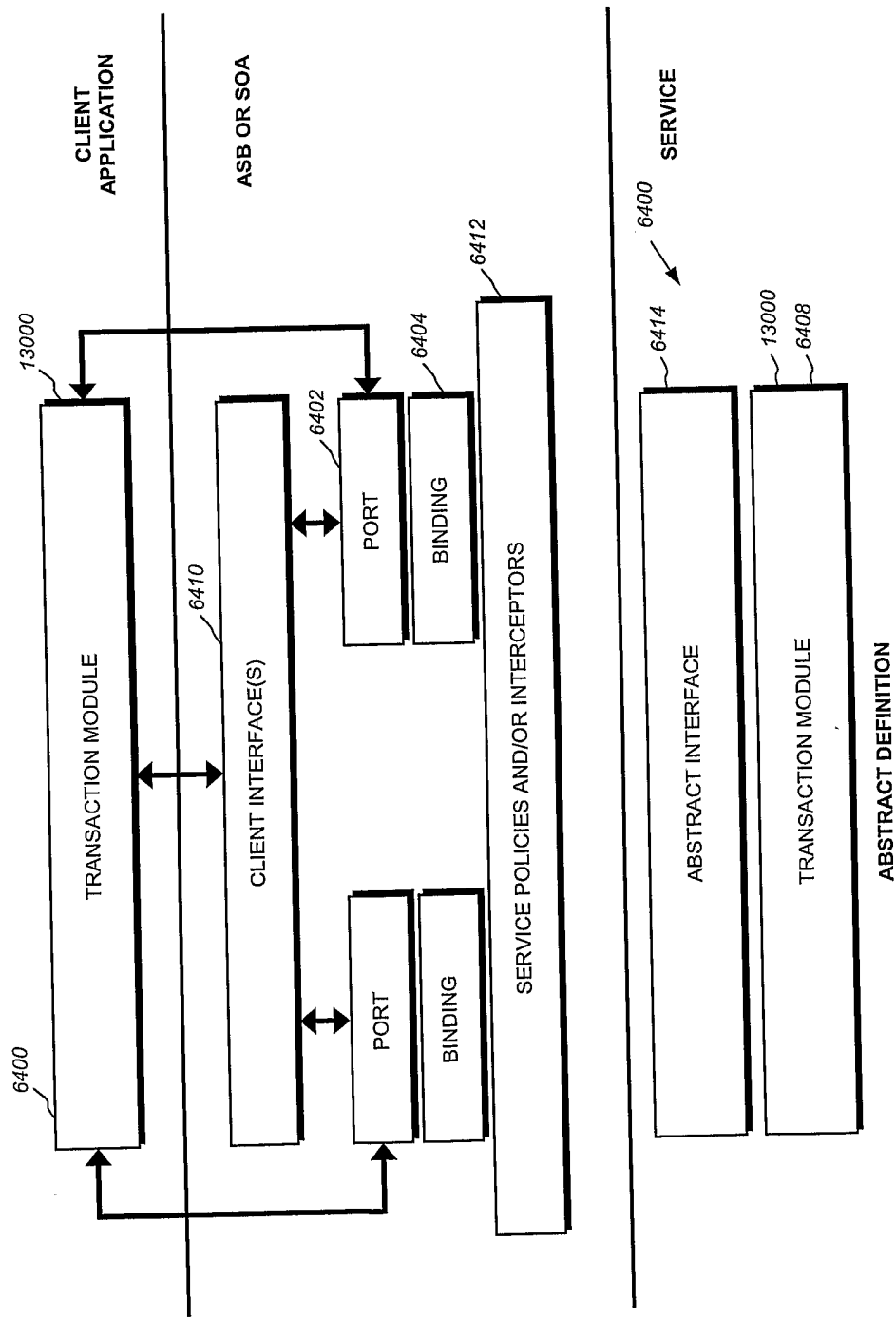


Fig. 82

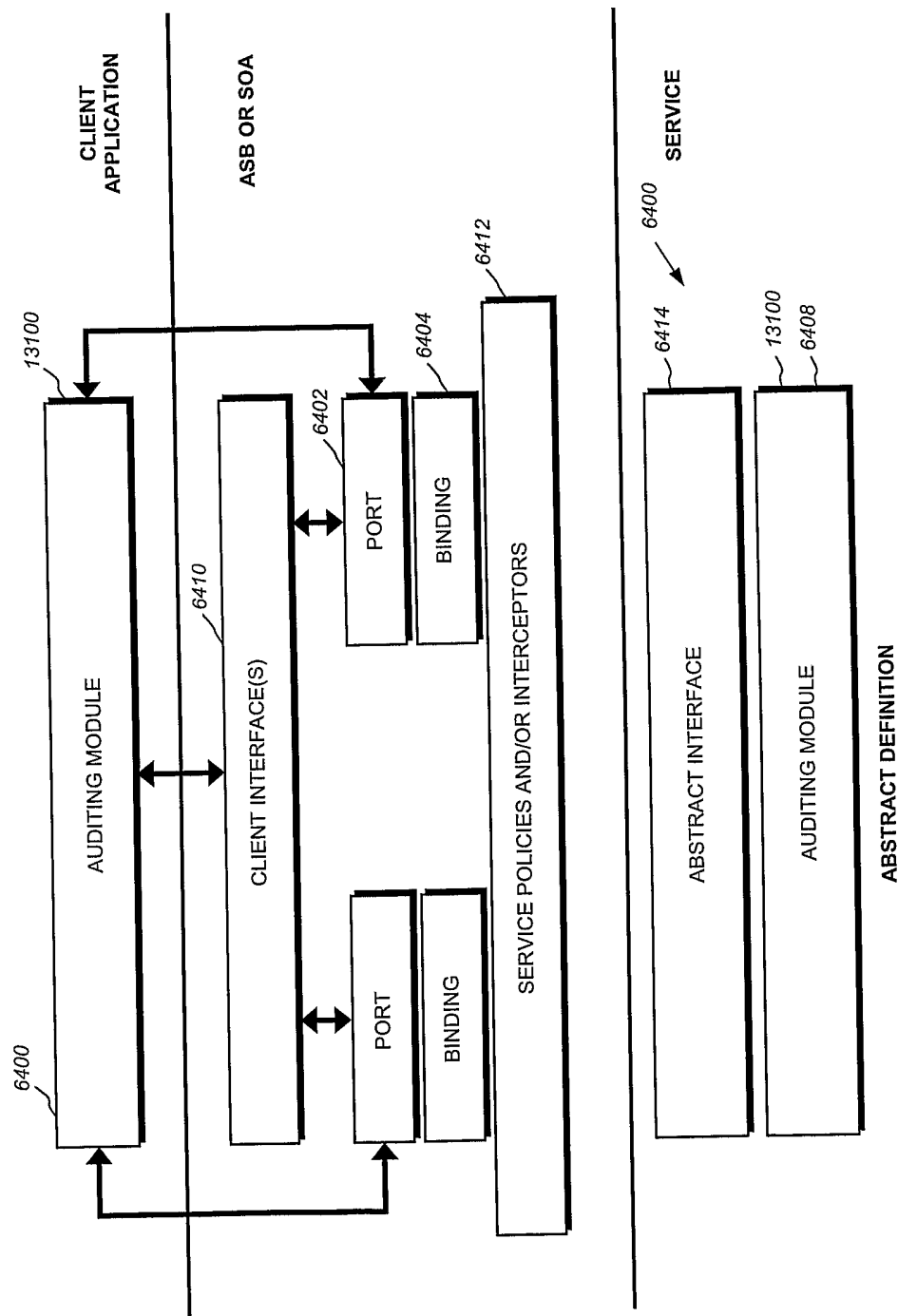


Fig. 83

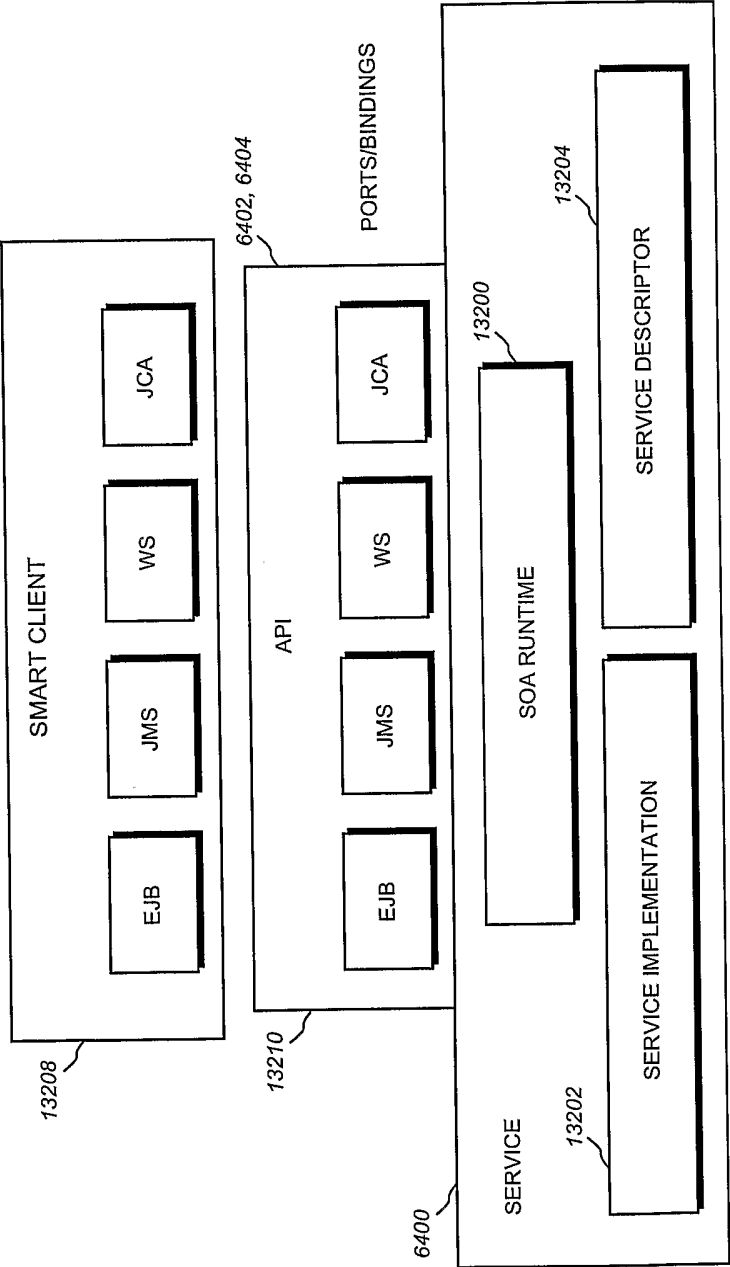


Fig. 84