It is common that a rookie will spend a lot of effort and make many mistakes before getting the hang of a huge project which contains a large number of highly coupled components and modules. Even a veteran is prone to cause seemingly irrelevant functions being affected after modifying functions of a module, because of complicated inter-dependence among components.

[0006] The SOA architecture features complex deployment and configuration. In real cases, a large application is usually divided into hundreds of independent services. For example, a famous e-commerce website (among the top 5 in China) which fully embraces SOA has divided their Web application into tens of hundreds of services. We can imagine the huge amount of workload required to deploy hundreds of servers within high availability environment where multiple active data centers exist, and to configure these servers to establish coordination relationships among them. For example, the recent network outage with ctrip.com was followed by slow recovery due to its huge SOA architecture which comprises tens of hundreds of services.

[0007] Inefficient is another major disadvantage of SOA. From the logic flow perspective, almost every complete request from the client needs to flow through multiple services before the final result is generated and then returned to the client. Flowing through each service (through messaging middleware) is accompanied by multiple times of network and disk I/O operations. Thus several requests will cause long network delay accumulatively, resulting in bad user experience and high consumption of resources.

[0008] Fig. 3 shows the conventional messy SOA dependencies (Image from the Internet).

[0009] The responsibility to implement the support for cross-service distributed transaction will fall on the application developers, no matter each service is connected to a different DBMS or all services are connected to the same distributed DBMS system. The effort for implementing distributed transaction itself is more complex

[0022] Fig. 2 is the schematic diagram of embodiment II of the present invention.

[0023] Fig. 3 is a conventionally schematic diagram of the messy dependencies of the SOA.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0024] The embodiments of the present invention are explained in combination with the drawings.

[0025] In order to make the purposes, the technical scheme and the advantages of the present invention clearer, the present invention is further explained in details in combination with the functional diagrams and flowcharts. The following illustrative embodiments and descriptions thereof are used to explain the present invention, but are not intended to define the present invention.

[0026] The service-oriented modular system architecture comprises plugins and API Nexus, wherein the plugins abstract different functions in servers into pluggable business modules (BMOD, generally, each business module corresponds to one plugin), and any number of API Dispatchers can be registered on the API Nexus, so that one business module in a corresponding plugin dynamically exposes the services, which can be provided at present, to the system and other business modules or other plugins belonging to the system in a manner of API.

[0027] It defines cross-platform plugin interfaces (IPlugin). Plugins are usually provided in the form of dynamic-link library (DLL), exposing a single interface like “extern "C" void* CreateInstance(void);”. Plugins can also be embedded into projects in the form of static library or source codes, without exposing any interface. Automatic plugins matching is implemented based on current environment factors such as processor, operating system, and release version (MBCS / UNICODE).

WHAT IS CLAIMED IS:

1. A service-oriented modular system architecture, comprising plugins and an application programming interface (API) Nexus, wherein the plugins abstract different functions in servers into pluggable business modules, and any number of API dispatchers are capable of being registered on the API Nexus, so that one business module in a corresponding plugin dynamically exposes services, which can be provided at present, to the system and other business modules or other plugins belonging to the system in a manner of API.
2. The service-oriented modular system architecture according to claim 1, wherein a dynamic API registering or unregistering mechanism is matched with a hot-plugging and hot-unplugging capability of a plugin interface.
3. The service-oriented modular system architecture according to claim 1, wherein configuration information carried by the plugins is divided into several parts including general configurations, advanced configurations and internal configurations, each part is capable of being customized according to the category of the plugins or the specific implementations thereof.
4. The service-oriented modular system architecture according to claim 1, wherein each plugin carries two virtual file systems (VFS) containing any types of resources, which include a resource virtual volume for external services and a private virtual volume for internal services.
5. The service-oriented modular system architecture according to claim 1, wherein the API Nexus comprises a dispatcher registering unit, a dispatcher unregistering unit, an API call request making unit, an API call request posting unit and an API dispatcher matching unit.