International Journal of Hybrid Information Technology
Vol.8, No.3 (2015), pp.195-212
http://dx.doi.org/10.14257/ijhit.2015.8.3.19

Cloud-Brain: A Knowledge-Based Development System for End-User in Cloud Computing

Rui Zhou, Guowei Wang, Jinghan Wang and Jing Li

School of Computer Science and Technology
University of Science and Technology of China, Hefei, 230026, China
{rayzhou, weiking, heiswjh }@mail.ustc.edu.cn, lj@ustc.edu.cn

Abstract

With the development of the cloud computing, increasing application and service based on cloud can be used by end-users. However, for these applications, the end-users can only use them passively. Without the programming experience, the end-users hardly involve their knowledge and awareness into the logic of applications. Cloud computing can provide large storage and computing resources for end-users on demand, and the Web of Things and sensor network can provide various kinds of information which is beyond the users' own sense. In our opinion, the end-users in cloud computing environment have the ability to develop and design the function of applications by themselves through their knowledge and awareness. Therefore, we proposed the Cloud-Brain, a knowledge-based developing approach for end-user in cloud computing. In our approach, the presentation of fact and knowledge in cloud is considered. The rule engine and service-oriented architecture are involved in processing the knowledge in the cloud. End-users can upload their knowledge and awareness in the form of rule into cloud. At the same time, the system collects facts and data from Web of Things or sensor network. The actions in these rules, which have matching facts, call the web service or other function for users. A prototype was implemented to verify the feasibility of this method. Here, we also present a case study to demonstrate the functionality, performance, and potential of the approach.

Keywords: *Cloud computing, rule-based, end-user development*

1. Introduction

In recent years, cloud computing has become a hot issue in both industry and scientific community. More and more researchers take part in exploring software architecture and developing model in cloud computing. At the meantime, more and more applications based on cloud computing emerge in the Internet [1, 2]. These applications can be used in a lot of aspects in life of users. However the most of them only provide some pre-designated functions and only can be used passively. There are few products that can include the user's knowledge and provide the user-designed functions. Therefore, we proposed the Cloud-Brain, a knowledge-based development approach for end-user in cloud computing. It can think with user's knowledge and provide services to end-users.

Cloud computing is a new computing mode based on the Internet. Through the cloud, shared resource of software and hardware can be provided to users and customers on their demands. Typical cloud service provider provides network business application in general. It can be accessed with browser, client or other web service. The program and data are stored and running in the server of cloud service provider. For the users, they can request the resources of computing and storage on demand from the cloud service, and pay the cost for the resources without buying any equipment and maintenance. So far, numerous projects from industry and academia have been proposed. For example, the Google App Engine provides Platform-as-a-Service (PaaS), which is a platform allowing users to run and host their web applications on Google's infrastructure [20]. In addition, the Amazon Elastic Compute Cloud provides Infrastructure-as-a-Service (IaaS) that provides resizable compute capacity in the cloud [21].

Thus, we believe that the end-users can obtain limitless resource and capability of computing and storage from cloud. Together with the development of the artificial intelligence research for the dozens of years, we also expect the cloud to help users store part of their knowledge and process some thinking instead of human brain. The result of the thinking can provide some actions or services for users. Users can get these results according to their own knowledge. This procedure also can be regarded as a method for end-user development.

In the other aspect, with the development of Web of things and sensor network, various kinds of data are produced from sensors in industry and personal life. More and more sensors are expected to appear in people's life in many scenes, for example to detect location, temperature, air pressure, and communication log in mobile devices. In houses, there are some smart household appliances with sensors which can collect status about these appliances. Some of these sensors and devices have processing capability. They can transform raw data into status information. The information represents some facts about users and the environment around them in a certain level. They also can be called contexts [3-5]. These contexts extend the ability of people's perception. In our view, these contexts can be organized as the basis of effective reasoning in cloud computing.

Therefore, a knowledge-based development approach is proposed in this paper. The system is also called Cloud-Brain. This approach is based cloud computing for end-users. With the approach, end-users can upload part of their knowledge to cloud environment in the form of rule. These rules become the basis of reasoning in cloud rule engine. Facts, collected by sensors, are inspected whether they match some facts by rule engine. When some facts are matched, the "THEN" term, which is a service invocation normally, will be executed by the system. End-users can get information or action from the executed service.

There are some advantages in the approach for end-users. Firstly, user's knowledge is stored in the cloud, and is accessible everywhere. Then, facts and context from sensors network and web of things are organized in the cloud. The process of reasoning with facts and knowledge is running in the cloud and produce some actions for user. Moreover, it also provides a method to composed web services for end-users, which have existed in the Internet. Finally, it can be regarded as an approach to develop application for non-programmer.

To achieve this target, a few of fields in computer science are involved into our work. Our approach is based on cloud computing environments. The processing and storing nodes in the system are based on virtual machine. Therefore, the approach can respond to multi-user concurrently. In order to represent knowledge and facts, some methods from artificial intelligent are applied. For the application development with knowledge, functions are implemented through service composition.

In this paper, we will describe the architecture, key points of Cloud-brain, and the prototype implementation we built. The paper is organized as follows: the section 2 will introduce the background and motivation. In section 3 and section 4, we explain the Cloud-Brain architecture and some primary component. In section 5, we demonstrate the Cloud-Brain prototype in action. In section 6, we show some experiments and demos in the Cloud-Brain. Finally, the future work will be proposed.

2. Related Work

Our research is based on the software engineering, cloud computing, artificial intelligence. This chapter will introduce the relevant background and the motivation of the research.

Since the time when computer software was born, the software development method has made tremendous change. It developed from procedure-oriented programming to object-oriented programming, from modularization programming to service-oriented development. It became easier with the development of computing science. However, the most of existing software development methods are still provided for professional development programmers rather than end-users. The end-users, who do not understand the principle of software, can hardly develop any application using programming languages. Some researches for end-user programming are introduced by Lieberman, *et al.*, [6]. They expected that the goal of interactive systems and services would evolve from just making systems easy to use to making systems easy to develop by end users. In this aspect, the most methods require the users' understanding the demand of software.

Recently, with the generation of cloud computing, end users have gained computing resource more easily. Therefore, we consider making use of cloud computing to provide a new development method to end users. For this target, some methods and researches for service composition have been investigated and surveyed in [7]. In [8, 9], automated web service composition methods were discussed. Some of methods used artificial intelligent to generate service composition plan in semiautomatic or automatic way. Then, some services were composed with the plan to achieve users' demands [12]. Inspired by these methods, service-oriented development was involved into our approach. In these existing service composition methods, the driving force was users' demands. We believe that these demand driven composition methods have certain hysteresis.

Research on using rules in service composition has been done in the software engineering domain in several systems. Some of them have related architecture with our approach. The most similar approach in the past research is "Ontology-based User-defined Rules and Context-aware Service Composition System" in 2011 [17]. It discussed a platform for context-aware service composition based on user-defined rules. But they only considered some simple context facts as conditions of rules. Its target was service composition rather than thinking with user's knowledge.

Another example to use rules for end user is Mayhem [22]. Mayhem was an open source application that provides a collection of triggers and reactions, like simple rules, allowing non-programmers to use their computers to do things automatically. Mayhem could easily tie disparate things together such as home automation devices, social networking sites, media devices, messaging, office productivity tools, webcams, musical instruments, etc. Its basic ideal was similar with our approach, but the rules and facts in our system were more complex to express complicated logic.

In [18], a knowledge-based approach to resource synthesis and service composition has been proposed. The approach they used exploits domain knowledge to guide the service composition process and provided advices on service selection and instantiation. In another system, SWORD [19], services were represented by rules by given certain inputs, and services were capable to produce particular outputs. A rule-based expert system was then used to automatically determine whether a desired composite service could be realized with existing services.

In recent years, some researches have been done for service composition in cloud environments. In [10], user-centric personal clouds based service cloud was proposed. A fast cloud-based web service composition was introduced in [11]. Its QoS was discussed in detail.

Compared with these researches on rule and service, our approach is different in a few points:

1. The targets are different. Our approach aims to provide a series of actions for end user rather than a generated service.
2. The processes are different. The process in service composition with rule was usually divided into planning with rule and execution with working flow. But in our system, the rules execute action and call service directly in runtime.

3. The roles of rules are different. In the previous research, rules always guided the composition and planning. But rules in our approach describe the way to use the facts and services.

3. Model and Architecture

3.1. Knowledge-based Development Model

In this study, we considered a method driven by user's knowledge, *i.e.*, the development progress for user is the progress of inputting his knowledge into system actually. Most of time, user may not know when the knowledge will be used, while the knowledge is input. In the real world, the usage of some knowledge depends on appropriate environment. Every piece of knowledge has its background and condition. For example, knowledge like common sense and tactics in the basketball may not be suitable for other sports. In other words, only in the environment of basketball, that knowledge is useful. Accordingly, in our approach, the knowledge of users includes the conditions in which this knowledge will be used. In order to structure these conditions, the information in web of things and users' context were introduced in our approach as conditions. With the development of mobile and web of things, increasing number of sensors can collect a mass of data and information for our system. In our opinion, these data and information in the system represent the world around users, and extend users' senses. As mentioned before, the cloud computing also extends the user's ability in thinking and memory. Above all, a knowledge-based development approach for end-user in cloud computing was explored in our study.

Under these circumstances, some factors were considered in our approach. Firstly, user's knowledge is stored in the cloud, and can be accessed everywhere. Then, facts and context from sensors network and web of things are organized in the cloud. The process of reasoning with facts and knowledge is running in the cloud and produce some actions for user. Moreover, it can be regarded as an approach to develop application for non-programming experiment users.

To this end, the most basic way of human thinking was observed. In a person's learning process during growth, the atomic knowledge can comes from "when something to do something", for example, when raining then take umbrella; when red light then stop; when feeling cold then wear clothes; and so on. This kind "the WHEN - THEN" type of rules can be used to describe knowledge in computer field.

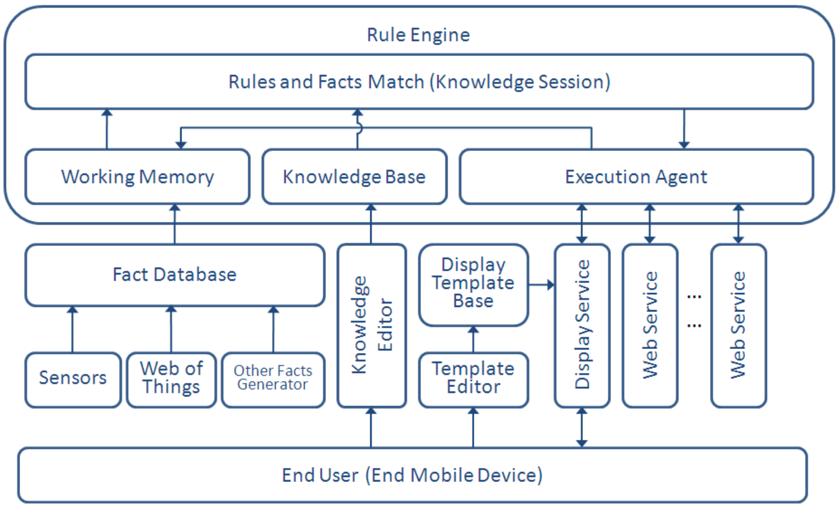


Figure 1. Framework of the Knowledge-Based Development Approach

A model was designed to resolve the problem. In cloud environment, a framework is proposed in figure 1. It includes end-device, sensors, database of facts, rule-base, rule engine, and web services. The end-device is an interface to an end-user where user can input rules and get services. The sensors detect everything around user as they can. The facts collected from sensors are stored in facts database. The rule-base stores knowledge in the form of rule. Rule engine is the core module that executes knowledge rules in a runtime Cloud environment. Web services provided by our system or other service providers are the components that execute the actions in the fired rules. The following parts will discuss these modules in detail.

3.2. Architecture in Cloud Environment

In the big data era, every user can produce a great quantity of rules and facts. Facts of every user may be produced every minute. With every fact, the rule engine searches the matching rules in user's rule-base. In some peaks of the facts producing, powerful processing ability is necessary in order to respond immediately. So it meets the demand of multi-user and high level concurrence. Consequently, we built our system in cloud environment. In this work, facts and rules for the same user are stored in the same virtual machine, and the reasoning process runs in the virtual machine.

The architecture of AppScale, an open source extension to the Google AppEngine (GAE) Platform-as-a-Service (PaaS) cloud technology, was referred to design our system in cloud environment [13]. The architecture of the Cloud-Brain system in cloud environment is shown in Figure 2. It consists of the load balance, one or more knowledge servers, a fact database management system and a controller for inter-component communication. The load balance distributes initial requests from user to knowledge servers. The fact database management system includes a fact database master and one or more fact database slaves. Knowledge servers are the execution engines for matching and action the rules. They interact with a fact database master for fact storage and access.

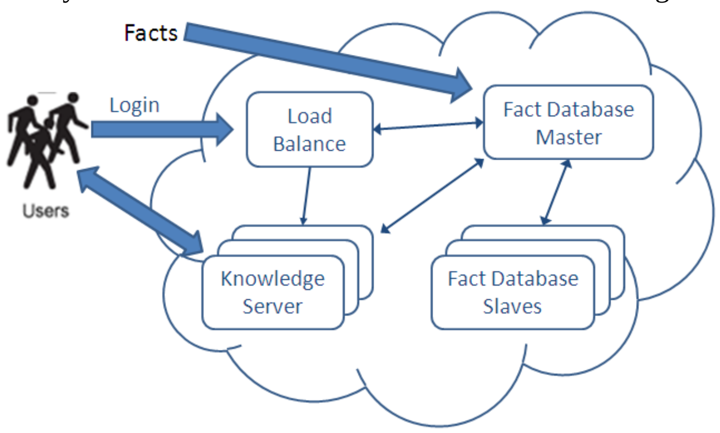


Figure 2. Architecture in Cloud Environment

A Cloud-Brain deployment consists of one or more virtualized operating system instances (guestVMs) as depicted in Figure 3. A node is an instance of a guestVM image. It implements a Controller for cross component interoperation and communication as well as at least one of the Cloud-Brain components: load balance, fact database master, knowledge server, or fact database slave; there can be multiple knowledge servers and fact database slaves in a deployment. A single node implements one or more components. The guestVM executes over a Xen virtual machine monitor (VMM). The VMM can execute one or more guestVMs using the same hardware resources.

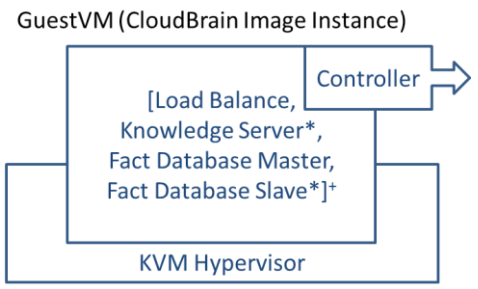


Figure 3. Cloud-Brain Image for Virtual Machine

4. Key Points of Cloud-Brain

4.1. Presentation of Facts

In our approach, facts include all kinds of information that can be got to describe something concerned by user. They mainly include:

- The data from end device. For example, the GPS location, the log of communication from mobile phone and some status of WIFI from laptop PC.
- The data from sensor network and web of things. Such as, temperature sensor in house, some status from smart household appliances.
- General facts, data and some common sense facts. For example, time and system status.
- Facts from services' outputs. Like weather information from weather web service, traffic information from navigation service.
- Other facts that uploaded manually by users. User can add facts to describe anything, such as person's current mood, perspective of something, and his demands.

These facts are in different forms and types. Moreover, while the system is running, facts are produced constantly. Since more and more facts need to be stored in the system, the ontology was involved to present these facts. With ontology, facts and relation between them can be described in memory and storage, and the scale of facts can extend easily.

In this approach, the relation between facts can be described at two levels. Firstly, facts can be distinguished in category. For example, the facts of temperature include body temperature, room temperature, water temperature, and so on. Secondly, for one kind of facts, it produces different data at different time or by different sensor. For instance, the temperature of bedroom should be recorded at different time, from morning to night. Therefore, these facts are stored and represented in two different ways. Correspondingly, the data describing facts are divided into two parts: metadata and data. In this approach, metadata represents the fact's definition and the semantic relation with other facts. It also stores the value of the fact or the link to the fact in database. Ontology is used to organize these metadata. Then, the common data, which record facts in different time and identities, are stored and organized in database. The access paths of these facts in database are indicated in the metadata.

A simple part in the metadata of our facts is show in Figure 4. The concept "FACT" is the general designation of all the facts. The arrow in the figure means "Is-a" or "Be a part of". In our system, the facts mainly include general facts, university facts, house facts, mobile facts, and any other classes of facts if necessary. The category can be redefined by users. In this case, the facts included in general facts describe general status in real world, like current time, and some common sense. The facts in other category like university, house, and mobile facts include the facts that reflect the information in our live environment. Every fact can contain a set of values to present the status of our world if needed, like the value in "current time" indicates the current time. If the values in a kind of facts are spread in time, like the bedroom temperature, the values can be considered as a set of facts which are stored in a table in database. At the same time, some information of the table in database should be described in the metadata.

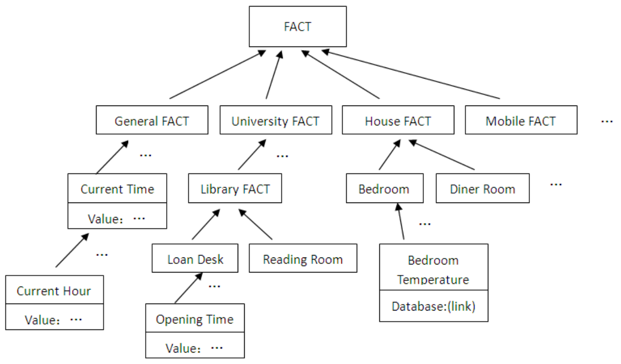


Figure 4. Fact Metadata of the Cloud-Brain

In the database, data collected from sensors are stored in Tables which are divided into classes of facts. The primary keys of the tables usually are the time or identities of the fact records. A fact record consists of the recording time, sensor identity, and data values. For example, a kind of facts in mobile facts is GPS location. This fact of GPS information changes with the movement of device. In order to record the changes and history data, the system collects data from the GPS sensor periodically and inserts the data into the GPS fact Table in database. As mentioned above, a GPS fact metadata contains the access of the Table. The columns in data table include time, identity, latitude, longitude, altitude, speed, and other information. These data are stored in our system permanently unless deleted by users.

Different users may concern different facts. Therefore, the structure and content of facts is usually specific for different user. Our system maintains a standalone set of facts for every user. The data for these facts are collected from the devices around end users and the sensors with users involved.

In rule engine, facts are used to match rules only when they exist in working memory. The working memory is where all facts are contained. It is often referred to the fact space. Facts can be asserted, modified and retracted from the working memory by using the methods in rule engine’s API. The system periodically detects the update of fact into database. The new facts and modified facts are added to a buffer list. Rule engine then updates its working memory with the buffer list regularly. In order to maintain these facts in working memory, the classes of facts should be defined in forms like JavaBean. The facts in working memory exist in the form of objects.

4.2. Lifetime of Facts

The lifetime is the remaining time of the data in the system. In this model, the lifetime of the data is the period when the input and output data stay in the working memory of rule engine. If some data still exist in the system after the correlative rules triggered, in some situation, they will make some incorrect operations by fire new rules. On the contrary, if some data are retracted from memory before invocated completely, the functions and the data will be damaged. For the input and output facts, the impacts of their lifetime are different. If the input event facts persist in the system after their function completed, they will match some rules again incorrectly when new relevant facts or rules are inserted into the system. It may be cause reduplicate response to user’s one operation. In other hand, if the input facts are removed earlier, it can cause some required services have no response. Because in this model, one operation event can match and trigger several rules and services, the removing fact in the former rules can causes the mismatch in the posterior rules. For the output data and facts, the output notices to users can be repeated, if these facts remain in system after being sent to users. For these output facts invoked by another service, if they disappear after output, the subsequent services are no longer able catch these data. Consequently, the lifetime of individual fact is significant for the correct reasoning.

In order to solve the problem of fact lifetime, two methods, extinction by system automatically and delete in rules, have been purposed to control the lifetime in these model. The first method, fact extinction by system, is automatic, according to the definition of fact. As mentioned before, the facts are defined and managed with ontology semantic description. In the description, user or service provider can define the lifetime of these facts. The system will control the extinction of facts according to the definition. In the other method, users can add the fact retract operation to rules. These rules will dispose facts when they are triggered. The retract operation can either be added into the RHS of rules or be involved in a separate rule with special conditions. When there are few of rules triggered by a set of facts simultaneously, different action sequences will influence the result of interaction. It makes mistake or function deficiency, when the retract fact operations exist in rules especially. In this situation, users can define the priority for rules to control the lifetime of facts and avoid the retract operation affecting other operations. When some rules have same priority or non-priority, their retract operation in them also does not affect other actions in the meantime.

These facts in the database will be stored permanently until the users want to delete them. But facts in rule engine are updated at any time. The older facts will be retracted in an interval with the retraction interval defined in facts ontology attribute.

4.3. Presentation of User's Knowledge

In order to integrate user's knowledge into the Cloud-brain model, we draw inspiration from expert system. We believe that the judgment to common things in human' brain is in the form of rule. For example, in our thoughts, if we arrive at a new place where we have never been before, we should check it in the map. Therefore, in our approach, user input knowledge with writing rules. Every rule includes two parts. One is Left Hand Side (LHS), which describes the conditions of the rule. It means the "IF" statement. The other part is Right Hand Side (RHS), which describes the actions. It is the "THEN" statement. If the condition in LHS matches with some facts, the rule will be fired, and the action in the RHS will be executed. In this model, the condition in LHS can be anything like facts. There are two kinds of actions in RHS:

1. Some operations to the facts like "insert", "retract", and "update". These operations will directly make the change in facts database. The operation object can be the fact which matches the condition in LHR. For example, a fact for user's mood can be altered by "$mood. Set (lucky); update ($mood)". This "$mood" means the mood fact matched the rule.

2. Service invocation and function call. In this kind of actions, functions from our system and web services are invoked. The facts matched the rule can be used in this functions as parameters. For example, the statement "getWeather($location.getCity)" means calling the weather service to obtain weather information of the location in facts database.

With more rules stored in the system, the knowledge accumulates in the Cloud-Brain. We model the rule-base as a set *R*, which presents the user's knowledge. In the knowledge set, $r \in R$, where $r=<C_r, A_r>$ is a rule. The $C_r=LogicExp(c_{r1}, c_{r2}, c_{r3},)$ presents the condition of the rule, and the $A_r = Seq(a_{r1}, a_{r2}, a_{r3},)$ presents a sequence of actions.

The condition is a logical expression that consists of some conditional elements c_{ri}, so called sub-conditions, and logical operations. The logic between sub-conditions may be AND, OR, NOT and so on. The sub-conditions in our approach are patterns for facts.

They potentially match on each fact that is inserted in the working memory. A pattern matches to a fact of the given type or class. All the action happens inside of the pattern parenthesis: it defines the constraints for that pattern. The constraints take effect to the property in fact objects. A constraint is an expression that returns true or false. The fact property can be used in the expression. If a fact makes the expression true, it matches the condition element successfully. The actions in the rule are some operations of working memory and some program codes to call services which run in the internet or local host.

4.4. Reasoning Process

In our approach, we expect to simulate the reasoning in human's brain with computing power in Cloud. The basis of the reasoning is user's knowledge. Therefore, we adopt the rule-engine like expert system in our reasoning process. In the reasoning process, several data sets are involved, in which, *F* is the fact set that is stored by working memory; *R* is the rule set that presents user's knowledge; and *A* is the action list where the fired actions will be inserted.

A complete reasoning process is as follows:

1. Process starts with changes in facts database or rule-base, when the sensors collect new data in facts database or rule-base is changed by user.
2. Matching check between facts and rules in rule engine. When existing *F'* is included in *F*, existing *r* belongs to *R*, and *F'* matches C_r, the corresponding A_r will be inserted into list *A*.
3. A_r from action list *A* is executed by agent in rule engine, and then will be extracted from the list.
4. If the result of step 3 has modified facts or rule-base, the process will repeat from step 2.
5. When the list *A* is empty and all the rules and facts are checked, this routine of reasoning is complete. The system returns to ready mode.

4.5. Interaction Mode

In our approach, many interactions should be considered between entities, such as the interactions between web services, and interactions between users and services. According to the feature of the approach, an interaction rule was designed to specially cope with interaction between services. With these rules, every entity can interact each other via a unified method.

Interaction rule can be divided into LHS and RHS as normal rules, and it represents user's logic and knowledge as well. This method takes full advantage of the Cloud-brain model. These interaction rules add condition, which matches the output fact of pre-service, in LHS. In the RHS, the statement contains the invocation of service that involves the output as a parameter. Of course, other components in normal rules can appear in interaction rule too. These services can connect each other with the method. In detail, an interaction rule is established as "IF output ($p:p==condition) THEN service S ($p)". In this rule, "output" is the condition's type, which indicates that the matched fact must be an output fact. The "$p" is a variable which is assigned with the symbol ":". The "p==condition" confines the condition of the fact in value. In the RHS, the service "S" is invocated with the parameter $p. With the system running, while an output "p" conforming to the condition is produced by one of the services, this rule will be fired. The system will call the service "S" with the parameter "p" to complete further process. This method can transfer parameter and invoke service effectively.

Although the interaction rule is designed as above, it is not perfect for our approach. For some situations following, it lacks the control in invocation of service.

1. When more than one service input on equal terms, *i.e.,* there are many rules which have the same LHS condition. At this time, when there is an output fact of service

in accordance with the condition of the rules, all the rules will trigger and execute RHS services. However, in many cases, we do not want to fire all the rules, but call the targeted service.

2. Some services do not need to input any parameter. For those who do not have input parameter service, if these services don't depend on other facts, then the present interaction rules will be difficult to describe the service conditions of the call.

Therefore, the "serviceReq", a fact type for service request, is proposed to solve the problem. This fact is similar to a request, and at the same time it is the general fact that appears in LHS of rule and the fact database. In the necessary, we add a call service request fact in LHS of the rules. For example, if we want to call the service B after the invocation of service A, we can add a statement "insert(serviceReq (B))" to produce a service request fact for service B after service A is called in RHS of rule. Moreover, the new sub-condition "serviceReq(B)" should be added in LHS of the rules which invocated service B. At this situation, the service B will be called, only when the request fact for service B is produced.

5. Prototype and System Implementation

5.1. Prototype in Single Server

In order to validate our approach, we implemented a proof-of-concept prototype system. It can be divided into mobile client side and server side. For the beginning, the system in server side is mainly deployed in a blade server with 2 Quad-Core AMD Opteron processors, 32 GB of RAM and a single 160GB SATA disk. It contains fact base, knowledge base, rules engine, service agent, and some web service.

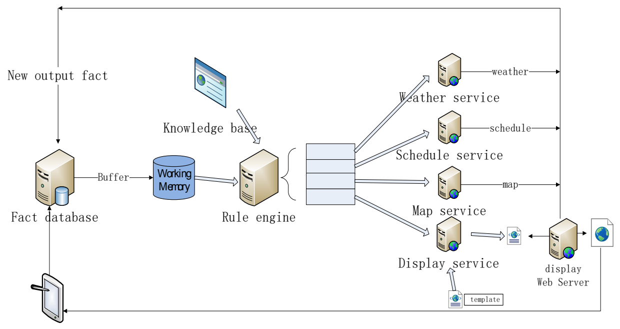


Figure 5. Architecture of the Prototype

The mobile client in our system is an android app, which mainly collects facts from mobile device and uploads to the server. These facts can be classified into sensors' data and behavior of user. The sensors' data includes GPS location, speed, atmospheric pressure, and other data that can be obtained from mobile phone. The behavior of user includes the user's contacts, recent call log, and other information input by user. Moreover, the mobile client also can display contents which are pushed from display service. In order to distinguish different users, the system requires users to log in.

On our server side of the system, the open source tool Jena [23] was involved to describe the category of facts in ontology [14]. In the prototype system, the category include the data from sensor, facts from user's information, schedules of facilities in our university, the output facts from services, the input facts from users, and other additional facts.

MySql was deployed as facts database that stored all the facts. In our approach, each user's knowledge is only for the facts around him, *i.e.*, each user has its private domain of facts and knowledge. Therefore, in the database, the data from different users' accounts are individual. It also ensures the safety of users' data.

The open source rule engine, Drools [15], was chosen as the core of the system. Drools are a Rule Engine that uses the rule-based approach to implement an Expert System and is more correctly classified as a Production Rule System [Drools Expert User Guide]. It includes working memory, knowledge base and rule executing agent. As mentioned before, working memory is the fact space. All the facts will be inserted into working memory before matching with rules. Frequently inserting of facts reduces the rule engine's performance. In order to synchronize the fact database and working memory, and stabilize the rule engine, a buffer list was used to cache the update facts. The facts in buffer are transferred to working memory at regular intervals.

In our prototype system, knowledge is input with a rule file by users. The file is in “.drl” format which is the description language of rules in Drools. These rules will be transferred into a RETE net that is the basis of RETE algorithm for rule matching [16].

For the rules matching facts, the agent executes their RHS. As described earlier, there are some operations of facts and some service invocations in RHS.

These services are mainly web services from internet. In order to call these web services easily, the encapsulations of the services were introduced to soap interface.

5.2. Cloud-Brain System in Cloud Platform

With the experiments in Section 6, a poor performance can be seen when the single-node rule engine system is in the large-scale data. Therefore, a distributed and elastic system is required. The Cloud architecture is employed to cope with big data and multi-user. A prototype system based Cloud has been developed to verify this architecture. The Cloud-Brain has been implemented on the USTC Cloud platform, which is an IaaS cloud environment based on Openstack cloud platform. As shown in Figure 6, there are three types of virtual machines in Cloud-Brain. Virtual machine in the role of master distributes the rules and facts to other virtual machines. The other virtual machines are in roles of fact database and rule engine slave. These three roles of virtual machines do not conflict with each other. In Cloud-Brain system, there is only one master node, but can be several fact database nodes and rule engine slave nodes. The number of the two kinds of nodes depends on the load and throughput.

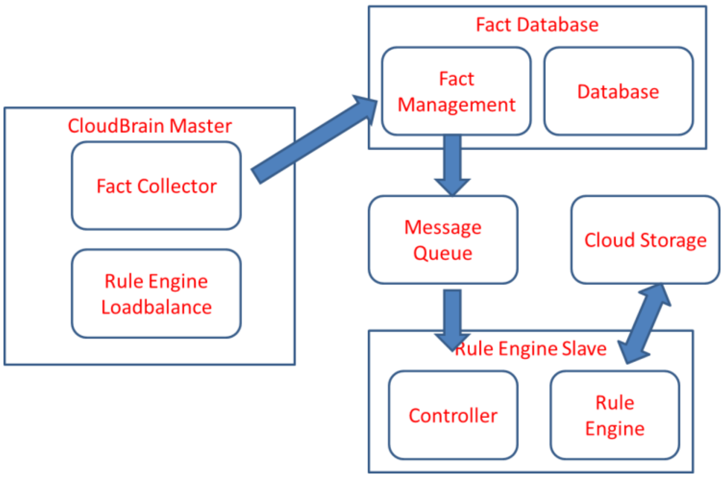


Figure 6. Cloud-Brain system in Cloud

In master node, Fact transporter collects facts from different sources and distributes them to fact database nodes for different users. The load balance of rule engine manages all the rule engine slave nodes in system. In fact database, the fact management maintains relevant facts based ontology and manages the facts data in database. The number of fact database can be expanded and contracted depending on the user throughput and the scale of facts. In the rule engine slave, a controller is designed to control the rule engine. When the rule engine is not accessed for offline with user, the rule-base in the engine will be serialized to cloud storage. If the user returns to this system again, controller can retrieve the rule engine from the slave node. This mechanism can solve the problem that these suspended rule engines consume gigantic memory. In the cloud storage, which is based on Swift of Openstack, the serialized rule-base can be backed up and recovered quickly. For the delivery of fact between fact database and rule engine, a message queue server based on RabbitMQ has been deployed.

6. Experiment and Demos

6.1. Experiment for the Performance of Rule Engine

In order to realize the performance of the prototype cloud-based system in virtual machines, a series of tests on the system response time for the changing quantities of rules and facts were conducted.

In these tests, the rules and facts are produced randomly by a program. In every rule, there are 4 conditions and 3 restrictions for properties. In addition, the facts may have the potential types and values matching rules.

Firstly, a group of tests were completed to measure the establish time of knowledge base for different quantities of rules. Some ".drl" files with different rules were produced randomly as rule sets. In order to make the results more accurate, we tested different rule sets in the same quantity, and calculated the mean value. The results of knowledge base establish time for different rule quantities are illustrated in Figure 7. The rule quantities in the test were from 10 to 100000. The results are shown in logarithmic coordinates. It can be seen that the establish time is almost in linear relationship with the rule quantity. The establish time is tolerable if user don't update rules largely and frequently.

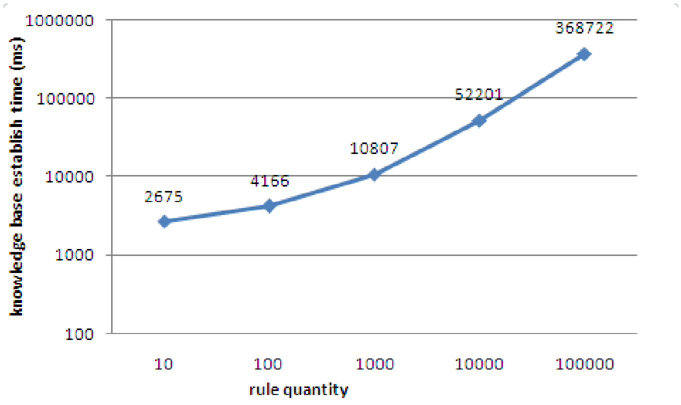


Figure 7. Knowlage Base Establish Time

Then, another group of tests aim to test the time of fact insert and match. The fact and rule quantities were both from 10 to 10000. The time of fact insert and match in every rule and fact scale was measured several times. The results of the average time in different groups of rule and fact scale are shown in Figure 8. They are compared in logarithmic coordinates. It can be seen that the effect of fact quantity is more than the effect of knowledge base scale.

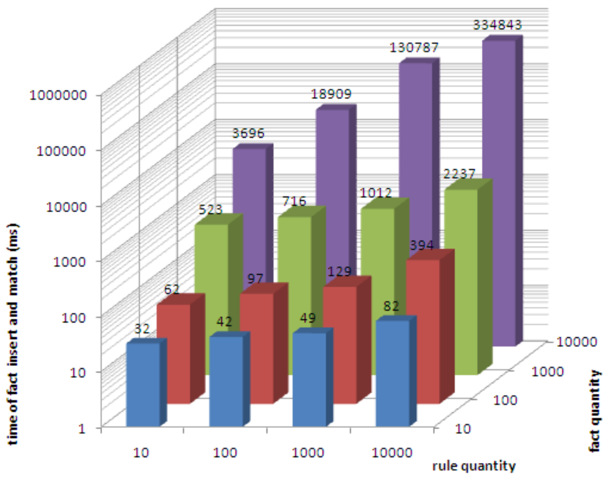


Figure 8. Matching Time in Different Scale of Rules and Facts

Based on the results of these tests, we also tested the response time of new fact in running system with different scale of rules and facts. With the some scale as above mentioned tests, a new fact was inserted into the working memory after all the fired rules executed. In these tests, the results of response time were usually lower than 10ms. Therefore, the response time for every fact is reasonable for end-users.

In these experiments, the consumption of memory was also monitored. In total, about 6Gbyte memory was consumed by 1000 rules and 10000 facts. The memory consumption was mainly in knowledge base construction, facts insertion and matching, and rules execution. When the rule quantity was increased to 10000, the memory for knowledge base construction was about 7Gbyte, and the whole memory consumption was nearly 20Gbyte. As the demand of memory is increased rapidly with the accumulation of rules and facts, distributing the rules in different virtual machines in cloud is necessary.

6.2. Demos of the End-user Development

The Cloud-Brain system has been put into use in our University. Common information, like notification of library, curriculum schedule, status of public athletic fields, was considered as facts in the system. The specific facts for users were also maintained in fact database, such as users' communication log, status of users' dorm, library records of users. Moreover, some services were provided to cater students' demands. These services cover many aspects of a student, for example, the curriculum schedule service, campus bus service, weather service, and map service. Students in the campus can develop their own knowledge in the Cloud-Brain. All the facts mentioned above can be matched as the LHS in the rules, and all the services provided in this system can be invoked by the RHS of the rules.

```
rule "WeatherFactDisplay"
    //include attributes such as "salience" here...
    when
        //conditions
        $fact:ContextFact(type == "WeatherFact", $info:info )
    then
        //actions
        DisplayServiceAgent.showDisplay($fact,"template.xml");
        System.out.println("WeatherFactDisplay\n");
        retract($fact);
end
```

Figure 9. The "Weather fact Display" Rule

With the prototype system, a demo was completed to verify the effectiveness of our approach. Some rules were written into a file. These rules describe some knowledge about how to remind schedule, query weather, call the map, find the appropriate campus bus line and interact with user. As an example, weather fact display interaction rule is shown in Figure 9. This is an interaction rule. Its LHS is the condition matching a weather fact. In the RHS, the system calls display service to display the weather information with the template in "template.xml". Then, the weather fact is retracted from working memory.

These rules have the following functions: 1. to remind user the schedule item, and display the schedule item to user with the customized template; 2. user can choose relevant information on the schedule item, like weather in this example; 3. find the appropriate campus bus line with the time and location of schedule item; 4. display these results to users.

The execution of demo is as follow:

• When the current time corresponds to the condition in rule "Schedule notice", the rule is fired. The rule executing agent will then call the schedule query service. If there are items like conference or memorandum in user's schedule, the service will return them. These items will be inserted into working memory as facts.

• When some schedule items are inserted into working memory, the rule "Schedule fact display" matches to them. The rule's RHS will then be executed. The display service will be called to push notice to user. The notice will be displayed in client app with a template, where potential interaction events can be described. In this example, the template defines a query weather event. User can choose the interaction event when the notice displays.

• If the user clicks the "query weather" button upon receiving the schedule notice, a weather query event will be produced and inserted into working memory. Then the rule "Interface event for weather" will match to the event. The agent will invoke weather query service and insert the weather information into working memory as a fact.

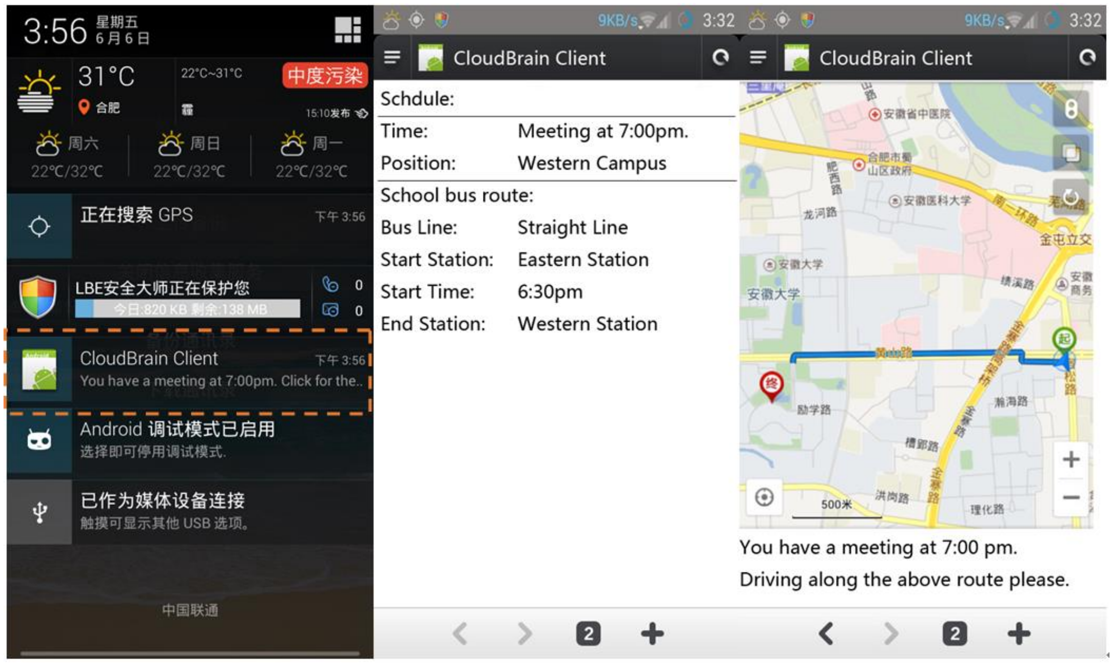
• When the weather fact matches the rule "Weather fact display", display service will be called again to deliver the weather information to the user.

• In other rules about bus line, the campus bus service is invoked. Campus bus timetable can be queried with the service. A series of reasoning judgment is carried to find the appropriate campus bus line with rules about schedule item, current location and campus bus.

• The campus bus line and location of schedule item can match some rules to invoke map service. The map service returns a piece of web page code, which can be used to display to users.

• The bus line and a map are displayed to users in Cloud-Brain client.

The figure 10 shows some of screenshots in this demo. In these screenshots, when a schedule item is produced by web service, a notice will displayed in notice bar. After user tap the notice, the reasoning result is displayed. A campus bus line is recommended by Cloud-Brain according to the current time, the location of user, schedule in calendar, and the campus bus service.


3:56 星期五 6月6日

31°C 合肥 22°C~31°C 霾 中度污染 15:10发布

周六 22℃/32℃ 周日 22℃/32℃ 周一 22℃/32℃

正在搜索 GPS 下午 3:56

LBE安全大师正在保护您 今日:820 KB 剩余:138 MB

CloudBrain Client 下午 3:56
You have a meeting at 7:00pm. Click for the..

Android 调试模式已启用
选择即可停用调试模式.

已作为媒体设备连接
触摸可显示其他 USB 选项。

中国联通

CloudBrain Client

Schdule:

Time:	Meeting at 7:00pm.
Position:	Western Campus

School bus route:

Bus Line:	Straight Line
Start Station:	Eastern Station
Start Time:	6:30pm
End Station:	Western Station

CloudBrain Client

You have a meeting at 7:00 pm.
Driving along the above route please.

Figure 10. Screenshots of Demo

With the system and demos above, a basis framework, which can help users to take full advantage of the resource in Internet and Web of Things, was proposed. Users can compile a series of rules as the knowledge in the system. This process also can be regarded as development by users.

7. Conclusions and Future Work

In this paper, we described Cloud-Brain, a knowledge-based development approach for end-user in cloud computing. Cloud-Brain provides a framework that allows user to store their knowledge, collects facts around user, and infers with knowledge and facts to execute actions for user. In our approach, the implementation of a framework and its prototype was described in detail. In the framework, the demand of multi-user and high concurrence, presentation of facts and knowledge, the reasoning, and the interaction were considered to support the approach. A demo and some performance tests has been discussed to show the function and insufficient.

To improve our prototype, future extensions of the system in cloud environment is needed. The management of facts, knowledge base and rule engine in different node will be considered. In addition, more facts and services will be involved to enhance and perfect the Cloud-Brain.

Acknowledgement

We thank the Supercomputing Center at University of Science and Technology of China (USTC) for their platform and technical support. This research was supported by the National Key Technology R&D Program under Grant No. 2012BAH17B03, the Continuing Learning and Education and Science Integration Project, and the Cloud Computing Joint Laboratory of USTC and Lenovo.

This paper is a revised and expanded version of a paper entitled "A Knowledge-based Development Approach with Fact and Service for End-user in Cloud Computing", presented at Proceedings of 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops, July 22, 2013 - July 26, 2013, in Kyoto, Japan.

References

[1] M. A. Vouk, "Cloud computing Issues, research and implementations," in Information Technology Interfaces, 30th International Conference on, **(2008)**, pp. 31–40.
[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin and I. Stoica, "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, **(2010)** pp. 50–58.
[3] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan and D. Riboni, "A survey of context modelling and reasoning techniques". Pervasive and Mobile Computing, vol. 6, no. 2, **(2010)**. pp. 161-180.
[4] P. Makris, D. N. Skoutas and C. Skianis, "A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments' Integration", Communications Surveys & Tutorials, IEEE, vol. 15, no. 1, (**2013**), pp. 362-386.
[5] L. Guan, X. Ke, M. Song and J. Song, "A survey of research on mobile cloud computing", Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science, **(2011),** pp. 387-392
[6] H. Lieberman, F. Paternò, M. Klann and V. Wulf, "End-user development: An emerging paradigm," in End user development, Springer, **(2006)**, pp. 1–8.
[7] G. Kapitsaki, D. A. Kateros, I. E. Foukarakis, G. N. Prezerakos, D. I. Kaklamani and I. S. Venieris, "Service Composition: State of the art and future challenges," 16th IST Mobile and Wireless Communications Summit, **(2007),** pp. 1–5.
[8] J. Rao and X. Su, "A survey of automated web service composition methods," in Semantic Web Services and Web Process Composition, Springer, (**2005)**, pp. 43–54.
[9] J. Cheng, C. Liu, M. Zhou, Q. Zeng and A. Yla-Jaaski, "Automatic Composition of Semantic Web Services Based on Fuzzy Predicate Petri Nets", (**2013**).
[10] L. Ardissono, A. Goy, G. Petrone and M. Segnan, "From Service Clouds to User-Centric Personal Clouds," 2009 IEEE International Conference on Cloud Computing, **(2009),** pp. 1–8.
[11] S. Wang, Q. Sun, H. Zou and F. Yang, "Particle swarm optimization with skyline operator for fast cloud-based web service composition", Mobile Networks and Applications, vol. 18, no. 1, (**2013**), pp. 116-121.
[12] O. Hatzi, D. Vrakas, M. Nikolaidou, N. Bassiliades, D. Anagnostopoulos and L. Vlahavas, "An integrated approach to automated semantic web service composition through planning", Services Computing, IEEE Transactions on, vol. 5, no. 3, **(2012),** pp. 319-332.
[13] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman and R. Wolski, "Appscale: Scalable and open appengine application development and deployment," in Cloud Computing, Springer, (**2010)**, pp. 57–70.
[14] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne and K. Wilkinson, "Jena: implementing the semantic web recommendations," in Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, **(2004)**, pp. 74–83.
[15] M. Bali, "Drools J Boss Rules 5.0 Developer's Guide", Packt Publishing Ltd, (**2009)**.
[16] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artificial intelligence, vol. 19, no. 1, **(1982),** pp. 17–37.
[17] V. Beltran, K. Arabshian and H. Schulzrinne, "Ontology-based user-defined rules and context-aware service composition system", In The Semantic Web: ESWC 2011 Workshops. Springer Berlin Heidelberg, **(2012),** pp. 139-155.
[18] L. Chen, N. R. Shadbolt, C. Goble, F. Tao, S. J. Cox, C. Puleston and P. R. Smart, "Towards a knowledge-based approach to semantic service composition," in The Semantic Web-ISWC 2003, Springer, **(2003)**, pp. 319–334.
[19] S. R. Ponnekanti and A. Fox, "Sword: A developer toolkit for web service composition," 11th World Wide Web Conference (Web Engineering Track), **(2002)**, pp. 7–11.
[20] Google App Engine, http://appengine.google.com.
[21] "Amazon Elastic Compute Cloud [URL]", http://aws.amazon.com/ec2, access on Nov. 2007.
[22] "Microsoft Mayhem", http://www.microsoft.com/appliedsciences/content/projects/mayhem.aspx.
[23] "Apache Jena project", http://jena.apache.org/.

Authors

Rui Zhou is a PhD student in Computer Science and Technology at the University of Science and Technology of China. He received his bachelor degree in Xidian University in 2009. His research interests include Cloud computing, service-oriented computing, mobile computing and context-aware. He is author of some research papers published at conference proceedings.

Guowei Wang is a PhD student in Computer Science and Technology at the University of Science and Technology of China. He received his bachelor degree in the University of Science and Technology of China in 2012. His research interests include Cloud computing, rule-based computing, mobile computing and distribute computing.

Jinghan Wang is a master student in Computer Science and Technology at the University of Science and Technology of China. He received his bachelor degree in Hefei University of Technology in 2011. His research interests include Cloud computing, rule-based computing, mobile computing and distribute computing.

Jing Li received his B.E. in Computer Science from University of Science and Technology of China (USTC) in 1987, and Ph.D. in Computer Science from USTC in 1993. Now he is a Professor in the School of Computer Science and Technology at USTC. His research interests include Distributed Systems, Cloud Computing and Mobile Computing. He is author of a great deal of research studies published at national and international journals, conference proceedings as well as book chapters.