Towards Intelligent Distributed Computing: Cell-Oriented Computing

Ahmad Karawash, Hamid Mcheick and Mohamed Dbouk

Abstract Distributed computing systems are of huge importance in a number of recently established and future functions in computer science. For example, they are vital to banking applications, communication of electronic systems, air traffic control, manufacturing automation, biomedical operation works, space monitoring systems and robotics information systems. As the nature of computing comes to be increasingly directed towards intelligence and autonomy, intelligent computations will be the key for all future applications. Intelligent distributed computing will become the base for the growth of an innovative generation of intelligent distributed systems. Nowadays, research centres require the development of architectures of intelligent and collaborated systems; these systems must be capable of solving problems by themselves to save processing time and reduce costs. Building an intelligent style of distributed computing that controls the whole distributed system requires communications that must be based on a completely consistent system. The model of the ideal system to be adopted in building an intelligent distributed computing structure is the human body system, specifically the body's cells. As an artificial and virtual simulation of the high degree of intelligence that controls the body's cells, this chapter proposes a Cell-Oriented Computing model as a solution to accomplish the desired Intelligent Distributed Computing system.

Keywords Distributed computing · Intelligence · Cell theory

A. Karawash (✉) · H. Mcheick
Department of Computer Science, University of Quebec at Chicoutimi (UQAC),
555 Boulevard de l'Université Chicoutimi, Chicoutimi G7H2B1, Canada
e-mail: ahmad.karawash1@uqac.ca

H. Mcheick
e-mail: hamid_mcheick@uqac.ca

A. Karawash · M. Dbouk
Department of Computer Science, Ecole Doctorale des Sciences et de Technologie (EDST),
Université Libanaise, Hadath-Beirut, Lebanon
e-mail: mdbouk@ul.edu.lb

A.T. Azar and S. Vaidyanathan (eds.), *Computational Intelligence Applications in Modeling and Control*, Studies in Computational Intelligence 575,
DOI 10.1007/978-3-319-11017-2_2

1 Introduction

Distributed computing (DC) is the consequence of permanent learning, the improvement of experience and the progress of computing knowledge. It offers advantages in its potential for improving availability and reliability through replication; performance through parallelism; sharing and interoperability through interconnection; and flexibility and scalability through modularity. It aims to identify the distributable components and their mutual interactions that together fulfil the system's requirements. In order to achieve DC goals, the client/server model is undoubtedly the most consolidated and regularly applied paradigm. With the extensive deployment of DC, the management, interoperability and integration of these systems have become challenging problems. Investigators have researched and developed important technologies to cope with these problems. One of the results of the continuous evolution of DC in the last decade is the Service-Oriented Computing (SOC) paradigm, which offers an evolution of the internet-standards based DC model, an evolution in processes of architecting, design and implementation, as well as in deploying e-business and integration solutions. The other key result is the Mobile Agent Computing (MAC) paradigm, which provides an alternative computing paradigm to the traditional client-server paradigm. Moreover, the latest DC technology is expressed by cloud computing, which evolved from grid computing and provides on-demand resource provisioning. Grid computing connects disparate computers to form one large infrastructure, harnessing unused resources.

Trends in the future of the Web require building intelligence into DC; consequently the goal of future research is the Intelligent Distributed Computing (IDC). The emergent field of IDC focuses on the development of a new generation of intelligent distributed systems. *IDC* covers a combination of methods and techniques derived from classical artificial intelligence, computational intelligence and multi-agent systems. The field of DC predicts the development of methods and technology to build systems that are composed of collaborating components.

Building a smart distributed model that controls the whole of Web communications needs to be based on an extremely consistent system. The ideal system that can be adopted in building IDC is the model of the human body system, specifically the body cell. Based on the high degree of intelligence that controls body cells, this chapter proposes a Cell-Oriented Computing (COC) methodology. COC is an artificial simulation of human cell functions that is proposed as a solution to achieve the desired intelligent distributed computing.

All parts of the human body are made up of cells. There is no such thing as a typical cell. Our bodies are composed of different kinds of cells. The diverse types of cells have different, specialized jobs to do. Cell computing simulates the human cell functions in the distributed systems environment. In fact, there are approximately 10 trillion cells in the human body [1]. Cells are the basic structural and functional units of the human body. Each cell has a specialized function and works in collaboration with other cells to perform a job. The cell acts like a mini computer. It is composed of a decision centre (the nucleus), the protein industry (mitochondria), store of human

traits (genes) and a defence system (cell membrane). All cells in the body are connected to a giant computer called Intelligence that controls their tasks. The human intelligence works like a super-computer. Indeed, the human cell network is millions of times larger than the communication networks of the whole Web. Each cell has a great capacity to receive and transmit information to every cell in the body; each remembers the past for several generations, stores all the impressions of past and present human lives in its data banks and also evaluates and records possibilities for the future. It has an internal defence system to face intruders when an external attack occurs.

This chapter is arranged as follows: Sect. 2 discusses previous work on intelligent distributed computing, Sect. 3 introduces the Cell computing methodology, Sect. 4 discusses some definitions relating to the proposed model, Sect. 5 shows the Cell components, Sect. 6 describes the strategy of Cell-oriented computing, Sect. 7 discusses the characteristics of the proposed computing type and finally, Sect. 8 summarizes the over-arching ideas of the chapter.

2 Background

Nowadays, most computing procedures are directed towards intelligence and towards decrease processing time and cost, while the main research question today is about how to add intelligence to distributed computing. Service computing and software agent computing are the two dominant paradigms in distributed computing work within the area of Service-Oriented Architecture (SOA). Although the service computing paradigm constituted a revolution in World Wide Web, it is still regarded as a non-autonomous pattern. With the support of mobile agent's computing aptitude, the service computing model may be improved to be more efficient and dynamically prototyped. Furthermore, in the area of SOA, cloud and grid computing have become more popular paradigms and their role in developing distributed computing toward autonomy and intelligence is important. This section discusses several previous researches dealing with distributed computing intelligence from the service, agent, cloud and grid computing perspectives.

2.1 Service Paradigm

Service-oriented computing is the most cross-disciplinary paradigm for distributed computing that is changing the way software applications are designed, architected, delivered and consumed [2]. The paradigm is moving towards intelligent Web services, WSMO (Web Service Modelling Ontology) and towards semantically enhanced information processing empowered by logical inference that will eventually allow for the development of high quality techniques for the automated discovery, composition and execution of services on the Web [3]. On the other

hand, Suwanapong et al. [4] propose the Intelligent Web Service (IWS) system as a declarative approach to the construction of semantic Web applications. IWS utilizes a uniform representation of ontology axioms, ontology definitions and instances, as well as application constraints and rules in machine-processable form. In order to improve single service functions and meet complex business needs, Li et al. [5] introduced composite semantic Web services based on an agent. To discover better Web service, Rajendran and Balasubramanie [6] proposed an agent-based architecture that respected QoS constraints. To improve Web service composition, Sun et al. [7] proposed a context-aware Web service composition framework that was agent-based. Their framework brings context awareness and agent-based technology into the execution of Web service composition, thus improving the quality of service composition, while at the same time providing a more suitable service composition to users. Another approach towards better Web service composition was made by Tong et al. [8], who proposed a formal service agent model, DPAWSC, which integrates Web service and software agent technologies into one cohesive entity. DPAWSC is based on the distributed decision making of the autonomous service agents and addresses the distributed nature of Web service composition. From a different perspective, Yang [9] proposed a cloud information agent system with Web service techniques, one of the relevant results of which is the energy-saving multi-agent system.

2.2 Mobile Agent Paradigm

The mobile agent paradigm provides many benefits in developments of distributed application, while at the same time introducing new requirements for security issues in these systems [10]. To achieve a collaborative environment, Liu and Chen [11] proposed a role-based mobile agent architecture, in which agents are grouped into a specific group according to their roles. Furthermore, through agent communication, mobile agents of an agent group can collaborate with each other to contribute to group tasks. In accordance with this agent-oriented programming approach, Telecom Italia launched the Java Agent Development Framework (JADE), which supports the following features: (a) a completely distributed situation as an existential platform for the agents, (b) a very effective asynchronous message transport protocol that provides location transparency, (c) implementation of white and yellow pages, providing easy search mechanisms for agents and their services, (d) easy, but still effective agent lifecycle management while monitoring the uniqueness of agent's ID, (e) support for agent mobility that provides a mechanism for agent code transfer to other platforms (by storing the agent's state) and (f) a flexible core that allows programmers to add new features [12]. Elammari and Issa [13] propose using Model Driven Architecture for developing Multi-Agent Systems so as to increase their flexibility and avoid any previously imposed restrictions.

Brazier et al. [14] proposed a compositional multi-agent method, DESIRE, as a methodological perspective, which was based on the software engineering principles of process and knowledge abstraction, compositionality, reuse, specification and verification.

2.3 Cloud Paradigm

Cloud computing has recently emerged as a compel paradigm for managing and delivering services over the Internet. It has rapidly modified the information technology scene and eventually made the goals of utility computing into a reality [15]. In order to provide distributed IT resources and services to users based on context-aware information, Jang et al. designed a context model based on the ontology of mobile cloud computing [16]. In the same area of smart cloud research, Haase et al. [17] discussed intelligent information management in enterprise clouds and introduced eCloudManager ontology in order to describe concepts and relationships in enterprise cloud management . In an equivalent manner, the work of Block et al. [18] establishes an alignment between ontologies in a cloud computing architecture. However, this work did not rely on reasoning among the distributed ontologies. By contrast, a distributed reasoning architecture, DRAGO, has been designed, based on local semantics [19, 20]. It uses a distributed description logics outline [21] to represent multiple semantically connected ontologies. Unlike DRAGO, the model introduced in Schlicht and Stuckenschmidt [22, 23] creates a distributed, comprehensive and terminating algorithm that demonstrates consistency of logical terminologies and promises that the overall semantics will be preserved.

2.4 Grid Paradigm

The aim of grid computing is to enable coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [24]. Shi et al. proposed an intelligent grid computing architecture for transient stable constrains that reassign a potential evaluation of future smart grids. In their architecture, a model of generalized computing nodes with an ‘able person should do more work’ feature is introduced and installed to make full use of each node [25]. GridStat has been introduced as a middleware layer capable of keeping pace with the data collection capabilities of the equipment present in the power grid [26]. Liang and Rodrigues [27] proposed a service-oriented middleware for smart grids. Their solution is capable of tackling issues related to heterogeneous services, which are most common in the smart grid domain.

3 Cell Theory

Cell theory is the modular representation of human cell characteristics from the perspective of computer science. It is a flexible and scalable virtual processing unit that treats complex distributed computing smartly by organized and accurate decisions. A cell is a software object that:

- Is sited within a command/execution environment;
- Holds the following compulsory properties:
  - Collaborative: works in groups to finish a job;
  - Inheritance: serves clients according to their environmental profile if there is no specification in their requests;
  - Shares business processes: each cell business process represents a group of business processes of components with the same goal. However, every cell is open for collaboration with all other cells and can keep up best process quality via dynamic changes in process nodes. Thus, the cell has great processing power since all cells' business processes can be shared by one cell to serve the client;
  - Uniqueness: each cell deals with a specific type of job;
  - Reactive: cell senses modification in the environment and acts in accordance with those changes;
  - Autonomous: has control over its own actions;
  - Optimal: keeps to best functional and non-functional requirements;
  - Federative: each cell has its own information resources;
  - Self-error covering: monitors changes in the computing environment and applies improvements when errors are detected;
  - Dynamic decision making: applies decision alteration based on the change of context;
  - Learning: acclimatizes in accordance with previous experience;

In order to introduce the proposed cell theory, we discuss a new software design style: the cell architecture and then show how cell computing works. For simplicity, we identify the infrastructure of the Cell-Oriented Architecture (COA) and the functionality of Cell-Oriented Computing (COC) model with definability in the mathematical model.

3.1 Cell-Oriented Architecture

COA is a novel software design principle targeted generally at Web resource computing devices. The architecture allows users to engage in smart collaborations among devices during Web resource invocations. COA is based on a centre of intelligence, which collects cells in order to exchange data between participants and manage organized standard communication methods to obtain information.

The architecture is designed to achieve smart Web goals and overcome the limitations of existing Web infrastructures. The cell architecture presented here is device, network and provider independent. This means that COA works across most computing machines and ensures a novel methodology of computing.

COA is designed to cater to smart Web requirements and aims to achieve at last an ambient, intelligent Web environment. Cells in COA are internally secured, sustain autonomic analysis of communications and are able to support the mechanism of collaborations through the following requirements:

[R1] Management and Communication: to establish local and remote sessions, the underlying infrastructure provides the ability to find any other cells in the network and then to establish a session with that cell.
[R2] Context-based Security: to enable secure interactions in the communication spaces among all connected participants.
[R3] Analysis: supporting analysis of data exchange among cells, plus encompassing the interior analysis of cell process infrastructure.
[R4] Validation: to verify cell components and ensure consistent process combinations among cells.
[R5] Output Calculation: to evaluate the suitable output results with less cost and minimal use of resources.
[R6] Trait Maintenance: to avoid and deal spontaneously with all sources of weakness in cells' communications.

To realize these goals, we developed a complete command-execute architecture, designed from the ground up to work over existing Web standards and traditional networks. COA makes it possible to merge the material and digital worlds by incorporating physical and computing entities into *smart spaces*. Put simply, it facilitates the steps to achieving a pervasive form of computing. Figure 1 outlines the components of this COA, its functionality and the operation of the underlying protocols.

COA is composed of three main components: Cell Commander, Executive Cell and Cell Feeding Source. Cell theory is introduced to provide intelligence in distributed computing; however, it combines client/server and peer-to-peer models at once. This is a client/server representation because we have a client component (the

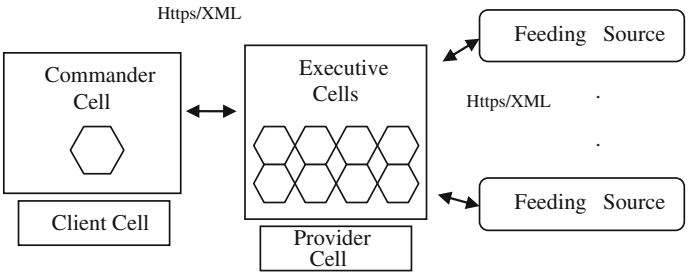

Fig. 1 Cell-oriented architecture

Commander Cell) invoking a server component (the Executer Cells) to solve a problem. On the other hand, virtually, it is an illustration of peer-to-peer applications because we have two types of cells communicating with each other.

Commander (Client) Cell: This is a commander component that looks for a procedural module to accomplish a required function. The commander can be an application, another service, or some other type of software module that needs the service. The Commander Cell works like a brain cell in the human body; it demands a solution for a definite problem and suggests a general view of the solution to be realized by a specific type of executive cells.

Executive (Provider) Cell: This is an intelligence centre consisting of a definite number of cells that are ready to serve commanders. Each cell is characterized by uniqueness of goal, self-governance, federated role, internal security and interoperability. Cell business processes, which are called genes, are built directly by the cell designer or else can be transformed by any type of service business processes. Genes use the ontology of an abstract business process and link different processes with the same purpose into a specific node. Similar to the gene in human body, each artificial gene serves a specific type of job in a different style and no other gene is capable of doing the same job. Based on gene characteristics, an Executive Cell is unique in delivering a specific type of service; for example, if a client cell requires a booking room service, there is only one, replicated, Executive book room cell to be invoked. Cell theory maintains diversity and competition between companies to serve clients; however, it hides complexity issues when selecting or composing Web processes. Solutions are prepared in an autonomic manner without any interference from the client; that is why there is no complex discovery and selection of cells or processes of composition or intervention.

Feeding Source: It represents a pre-built component that forms a base for building genes of Executive Cells. The Feeding Source can be a Web service provider, a company, or any third party that is capable to supplying a process design. A cell's internal system can use a pre-designed business process or demand the building of new designs by process designers, making it suitable to be a cell gene.

4 Structure of COA Components

The abovementioned main components of COA are discussed in detail in this section.

4.1 Commander Cell Structure

The Commander Cell represents the client side in COA and is the main requester of an output. This section discusses the structure of cells from the client side (Fig. 2).

Command Cell Manager (CCM): the client cell's 'head' that is responsible of any external collaboration with the Executive Cells. It receives a client as a list of

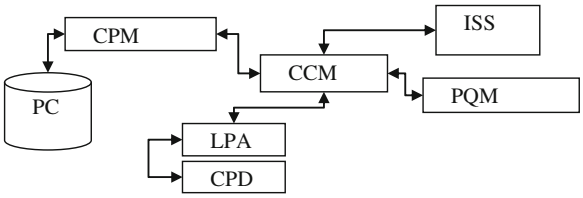

Fig. 2 Structure of client cell

four components: proposed cell input, interval of output of Executive Cell result, proposed Cell process's general design (if available) and the required cell process quality. Some of these components can be inherited from the client cell's environment. The Command Cell Manager monitors the context profile of the Commander Cell via the profile manager. It also manages the access to the client cell by specified rules of internal security.

Internal Security System (ISS): this is protection software that is responsible of giving tickets for Executive Cells to access the Command Cell manager. It depends mainly on the analysis of the outer cell's context profile to ascertain whether it can collaborate with the client Cell.

Process Quality Manager (PQM): software used by the Commander Cell to select the required quality of the cell process. For example, the client may need to specify some qualities such as performance, cost, response time, etc. If there is no selection of specific qualities, these qualities are inherited from the environment's qualities (as an employee may inherit a quality from his company).

Cell Process Designer (CPD): a graphical design interface that is used to build a general cell process flow graph or to select an option from the available process graphs. If there is no graph design or selection, the Executive Cell has the right to pick a suitable gene based on the commander profile.

Logic Process Analyser (LPA): after designing a general proposition for the executive gene design via the process designer, the job of the logic process analyser is to transform the graph design into a logical command to be sent to the executive side.

Context Profile Manager (CPM): this tool is responsible for collecting information about the Commander Cell profile, such as place, type of machine, user properties, etc. Since the commander profile is dynamic, several users may use the same Commander Cell; the profile information is instantaneously provided when needed.

Profile Core (PC): this storage is performed by a special database that stores information about the Commander Cell profile and allows the Executive Cell to tell whether there are several users utilizing the same Commander Cell.

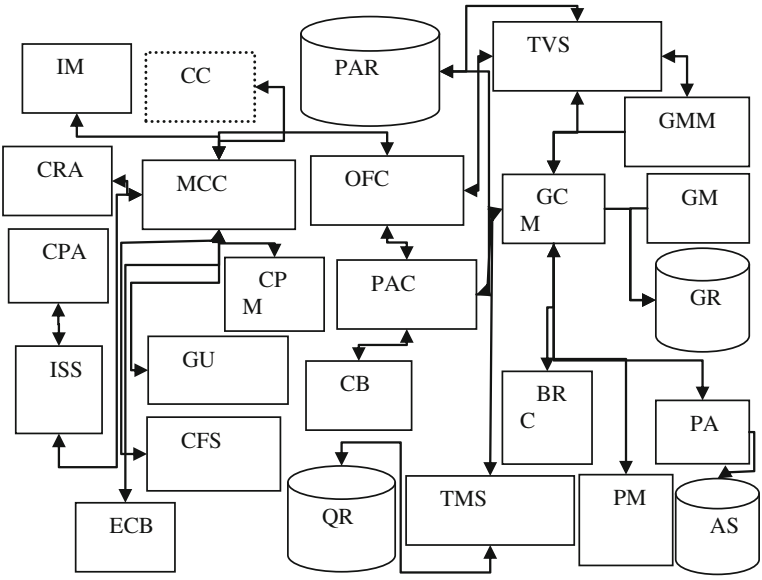

Fig. 3 Structure of cell provider

4.2 Cell Provider Infrastructure

The cell provider represents the supplier side in the COA, which is responsible for building suitable outputs for client invocation. This section discusses the structure of the cell provider shown in Fig. 3.

Management and Control Centre (MCC): Smart software work like an agent and is considered to be similar to the brain of the COA, in which it orchestrates the whole computing infrastructure. It is composed of a virtual processing unit that controls all the internal and external connections. So, Executive Cells are supported and managed according to well-defined cell level agreements. It monitors every connection among cells and prepares all decisions, such as update requirement, communication logics, maintenance facilities, access control management, repository stores and backups, etc. The COA management and control centre have stable jobs inside the cell provider. However, it cannot respond to an external job from other cells without security permission from the internal security system. Since one of the main principles of cell theory is availability, the management and control centre is replicated in order that collaboration can be carried out to serve cells. Each cell uses its Decision System to communicate with the COA management centre.

Testing and Validation System (TVS): the cell testing and validation system describes the testing of cells during the process composition phase of the Executive Cell. This will ensure that new or altered cells are fit for purpose (utility) and fit for use (warranty). Process validation is a vital point within cell theory and has often been the unseen underlying cause of what were in the past seen as inefficient cell

management processes. If cells are not tested and validated sufficiently, then their introduction into the operational environment will bring problems such as loops, deadlocks, errors, etc. In a previous book chapter [28] we have discussed a new model of how to validate the business processes of Web service; the concepts of the same validation method can be used to validate the cell business process (Gene). Cell validation and testing's goal means that the delivery of activities adds value in an agreed and expected manner.

Cell Traits Maintenance System (TMS): the challenge is to make cell technology work in a way that meets customer expectations of quality, such as availability, reliability, etc., while still offering Executive Cells the flexibility needed to adapt quickly to changes. Qualities of genes are stored in a *QoG* repository and the maintenance system has permission to access and monitor these qualities. *QoG* can be considered a combination of *QoS* with a set of Web services if the source of the cell is a Web service provider. *QoG* parameters are increasingly important as cell networks become interconnected and larger numbers of operators and providers interact to deliver business processes to Executive Cells.

Process Analyser Core (PAC): since a cell process map can be composed of a set of other components' business processes, there should be a method for selecting the best direction for the cell map. In addition to the context of environment dependency, cell theory uses a deep quality of service analysis to define a best process. This type of process map analysis is summarized by building a quality of process data warehouse to monitor changes in process map nodes. Every process component invokes a set of subcomponents, similar to sub services in a service model, in which all these subcomponents are categorized in groups according to goals. The process analyser core applies analysis to these subcomponents and communicates with the cell broker to achieve the best map of the Executive Cell process. In addition to analysing Executive Cell process, the process analyser core also analyses and maps the invocations from the Commander Cells. This type of dual analysis results in an organized store of collaboration data without the need to re-analyse connections and without major data problems.

Output Fabrication Centre (OFC): depending on the specific output goal, options may be available for executive cells to communicate with the output fabrication centre. This centre provides more control over the building of the executive cell process to serve the client cell. Based on the results of the process analyser core and the consequences of the test and validation system, executive cells, specifically their output builder systems, collaborate with the output fabrication centre to return a suitable output to the commander cell.

Cell Profile Manager (CPM): traditional styles of client/server communications suffer from a weakness: the dominance of the provider. Indeed, a server can request information about client profiles for security purposes, but power is limited in the converse direction. In cell theory, every ell must have a profile to contact other cells. The cell profile manager works to build suitable profiles for executive cells to help in constructing a trusted cell instruction tunnel.

Cell Federation System (CFS): the system coordinates sharing and exchange of information which is organized by the cells, describing common structure and

behaviour. The prototype emphasizes the controlled sharing and exchange of information among autonomous components by communicating via commands. The cell federation system ensures the highest possible autonomy for the different cooperating components.

Cells' Core (CC): this forms a centre of Executive Cells. A cell is an item of smart software that performs a specific type of job. All cells have the same structure but different processes. Thus, the executive cell is considered an example of a general cell component. Each executive cell is composed of seven sub-components, as follows: decision system, gene store system, trait maintenance system, output builder system, process validation system, process analyser system, defence system and gene storage. These sub-components communicate with the cell provider subsystems to carry out their jobs.

Inheritance Manager (IM): a client is observed as a Commander Cell so as to decide which types of cell inherit the properties of their environment. For example, if the commander is a professor, they can be seen a part of a university environment by Executive Cells. A commander can be part of more than one environment; and results in a hybrid profile of context. The inheritance manager maps the commander cell to its suitable environment. To serve a commander, the executive cell uses a quality of process compatible with its surroundings or follows the commander's requirements to build a suitable process.

Cell Request Analyser (CRA): cell theory is based on the concept of collaboration to serve the client. However, every client has a different request, so a computing component is needed to detect which cells will work in generating the answer. In general, the job of the cell request analyser is to map the Commander Cell to the appropriate Executer Cells to accomplish a job.

Cell Profile Analyser (CPA): this component is related to the security of cells. One of the main concepts of cell theory is its context-based property. There are sensors for profile context collecting information about the commander at the client side. The cell profile analyser verifies the commander profile by a specific method before allowing access to executer cells.

Internal Security System (ISS): since some commander cells can access sensitive data, stringent protection must be provided from the server side. The available security methods follow two types of protection: network and system protection. In network protection, the data among nodes is encrypted to hide the content from intruders. In system protection, a token (username and password), antivirus application and firewall are used. Cell theory proposes a new type of protection which is specific to the application itself. It is described as an internal system protection that verifies the profile of the user by several methods before allowing access.

Cell Process Modelling (CPM): a procedure for mapping out what the Executive Cell process does, both in terms of what various applications are expected to do and what the Commander Cells in the provider process are expected to do.

Enterprise Cell Bus (ECB): The enterprise cell bus is the interaction nerve core for cells in cell-oriented architecture. It has the propensity to be a controller of all relations, connecting to various types of middleware, repositories of metadata definitions and interfaces for every kind of communication.

Cell Broker (CB): analytical software that monitors changes in cell processes and evaluates quality of processes according to their modifications. The evaluation of quality of process is similar to that of quality of service in the service model. However, the new step can be summarized as the building of a data warehouse for quality of process that permits an advance online process analysis.

QoG Repository (QR): a data warehouse for the quality of cell process. It collects up-to-date information about process properties, such as performance, reliability, cost, response time, etc. This repository has an *OLAP* feature that support an online process analysis.

COA Governance Unit (GU): the COA governance unit is a component of overall IT governance and as such administers controls when it comes to policy, process and metadata management.

Process Analysis Repository (PAR): a data warehouse of all cells' process connections. It stores information about cell processes in the shape of a network graph, in which every sub unit of a process represents a node. The collected data summarizes analytical measures such as centrality.

Gene Core Manager (GCM): software responsible of gene storage, backups and archiving. It receives updates about business processes from sources and alters the gene ontology, backs up the gene when errors occur and archives unused genes.

Gene Mediator (GM): the problem of communication between the gene core manager and the sources of business processes may be complex, so GM defines an object that encapsulates how a set of objects interact. With the gene mediator, communication between cells and their sources is encapsulated by a mediator object. Business process sources and cells do not communicate directly, but instead communicate through the mediation level, ensuring a consistent mapping of different business process types onto the gene infrastructure.

Gene Meta-Data Manager (GMM): genes are complex components that are difficult to analyse, so for analysis and validation purposes, the gene meta-data manger invokes gene meta-data from the gene repository and supplies gene core data through this process.

Gene Repository (GR): ontologies are used as the data model throughout the gene repository, meaning that all resource descriptions, as well as all data interchanged during executive cell usage, are based on ontologies. Ontologies have been identified as the central enabling technology for the Semantic Web. The general use of ontologies allows semantically-enhanced information processing as well as support for interoperability. To facilitate the analysis of the gene map, meta-data about each gene is also stored in the gene repository.

Backup and Recovery Control (BRC): this refers to the different strategies and actions occupied in protecting cell repositories against data loss and reconstructing the database after any kind of such loss.

Process Archiving (PA): the archiving process helps to remove the cell process instances which have been completed and are no longer required by the business. All cell process instances which are marked for archiving will be taken out from the archive set database and archived to a location as configured by the administrator.

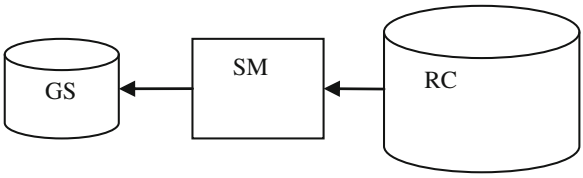

Fig. 4 Structure of cell source

The job of the process archiving component includes the process-, task- and business log-related content from the archive database.

Archive Set (AS): a database for unused genes that is accessed and managed by the process archiving component.

4.3 Cell Source

Cell source can be any kind of code that can be reused and follow specific composition rules. Generally, the first sources of cells are Web service business processes (such as BPEL and OWL-S) or reusable code (Java, C# etc.). This section discusses the structure of the sources that feed Executive Cells (Fig. 4).

Resource Code (RC): a store of cell sources, such as business processes or reusable code. If the cell source is a Web service provider, then its business process may be BPEL, OWL-S, or another. Further, the cell source may be a reusable programming code for a combination of objects (in Java, C#, etc.).

Source Mediator (SM): transformer software that maps the process of a cell's source into a gene. The mediator's job is similar to that of the BPEL parser in a Web service provider, which maps BPEL code into a WSDL code. In COA, every source business process is converted into OWL-S ontology. However, the obtained OWL-S ontology has a special property: the extension of OWL-S' business process.

Gene Store (GS): a store that is composed by mapping the source business process. This is an abstract of a source process in shape of an ontology, organized in a structure compatible with the cell's job.

5 Definitions and Notations

Definition 1 Let $W(P, Q, T)$ be a finite nonempty set that represents Web infrastructure, where: $P = \{p_1, p_2, \ldots, p_n\}$ represents the set of feeding sources of Web applications, $Q = \{q_1, q_2, \ldots, q_m\}$ represents the set of consumers of Web sources and $T = \{t_1, t_2, \ldots, t_k\}$ represents the set of tools that are used by Web providers to serve Web customer, where $n, m, k \in IN$.

Definition 2 Let set $J = \{\bigcup_m^z j_m / j_m is\ specific\ goal\ and\ j_m \neq j_n\}$ and set $S = \{\bigcup_m^z s_m / s_i\ is\ structure\ of\ component\}$.

As with most things in the business world, the size and scope of the business plan depend on specific practice. A specific practice is the description of an activity that is considered important in achieving the associated specific goal. Set J represents a group of components, each of which supports a specific computing goal based on a particular practice. However, the structure of the studied components is denoted by set S.

Proposition 1 *A set* $\sigma = \{\bigcup_i^n L_i / \sigma_i denote\ a\ Cell\} \complement T$, *is a finite and ordered set such that* $J_{\sigma_i} \cap J_{\sigma_j} = \emptyset$ *and* $S_{\sigma_i} = S_{\sigma_j}$, *where* $i, j, n \in IN$.

In all other computing models, different components may perform similar jobs. For example, two classes, in the object-oriented model, can utilize similar inputs and return the same type of output but using different coding structures. Furthermore, in the discovery phase of service-oriented computing, service consumers receive a set of services that do the same job before selecting which one of them to invoke. The main advantage of Web service theory is the possibility of creating value-added services by combining existing ones. Indeed, the variety involved in serving Web customers is useful in that it gives several aid choices to each one of them. However, this direction in computing failed since service customers found themselves facing a complex service selection process. One of the main properties of cell methodology is the avoidance of the 'service selection' problem. The cell is developed to provide highly focused functionality for solving specific computing problems. Every cell has its own functionality and goal to serve, so one cannot find two different cells which support the same type of job. However, all cells are similar in base and structure: they can sense, act, process data and communicate. That is to say, regarding cell structure there is only one component to deal with, while in function there are several internal components, each with a different computing method and resource.

Definition 3 Let φ be a property that expresses the collaboration relation such that $\alpha \varphi \beta$ where $\alpha, \beta \in \sigma$.

Business collaboration is increasingly taking place on smart phones, computers and servers. Cells in COC are intelligent components that are capable of collecting information, analysing results and taking decisions and identifying critical Web business considerations in a collaborative environment.

Proposition 2 *A collaboration relation* φ *defined on the set* σ *is transitive, in which, if* $\alpha \varphi \beta$ *and* $\beta \varphi \gamma = > \alpha \varphi \gamma$, *where* $\alpha, \beta, \gamma \in \sigma$.

Transitive structures are building blocks of more complex, cohesive structures, such as response-cliques, which facilitate the construction of knowledge by consensus [29]. The collaboration among cells follows a transitive mechanism to

provide consistency. Transitivity among cells can be summarized by this example: if we consider three cells X, Y, Z and if X collaborates with Y, Y collaborates with Z, then indirectly X collaborates with Z.

Proposition 3 $\forall c_i \in \sigma$ *and* $\forall q_e \in Q, \exists \sigma_i, \sigma_j, \ldots, \sigma_n$ *s.t.* $\bigcup_{i,j}(\sigma_i \varphi \sigma_j) = > q_e$, *where* $i, j, e, n \in IN$.

COC's goal is to be introduced to serving Web customers with minimal cost, lower resource consumption and optimal results. For every customer request (t_e), there exists a cell collaboration ($\bigcup_{i,j}(\sigma_i \varphi \sigma_j)$) to return the appropriate answer. Cell collaboration is dynamic; results are produced without delay. Any future error in the proposed results generated by COC is corrected by an automatic repairing mechanism.

Definition 4 (*cell subsystems*)

An Executive Cell system is an ordered set $C = (DS, GSS, TMS, OBS, PVS, PAS, DFS)$ such that:

DS set builds and manages cell decisions
GSS set is responsible for cell process storage
TMS set monitors the cell's characteristics
OBS set maintains best output results of cells
PVS set is responsible for cell process validation
PAS set analyses the cell's business process
DFS set is responsible for cell security

Proposition 4 *A relation between cell subsystems is managed according to a set of mathematical mappings M* $(\mu, \pi, \rho, \tau, \gamma, \delta, \varepsilon, \theta, \vartheta)$ *such that*:

$$\begin{aligned} \mu : DFS &\rightarrow DS \\ x &\rightarrow \mu(x) \end{aligned} \tag{F.1}$$

$$\begin{aligned} \pi : OBS &\rightarrow DS \\ x &\rightarrow \pi(x) \end{aligned} \tag{F.2}$$

$$\begin{aligned} \rho : TMS &\rightarrow OBS \\ x &\rightarrow \rho(x) \end{aligned} \tag{F.3}$$

$$\begin{aligned} \tau : PAS &\rightarrow OBS \\ x &\rightarrow \tau(x) \end{aligned} \tag{F.4}$$

$$\begin{aligned} \gamma : PVS &\rightarrow OBS \\ x &\rightarrow \gamma(x) \end{aligned} \tag{F.5}$$

$$\delta : GSS \rightarrow TMS \quad x \rightarrow \delta(x) \tag{F.6}$$

$$\varepsilon : GSS \rightarrow PAS \quad x \rightarrow \varepsilon(x) \tag{F.7}$$

$$\theta : GSS \rightarrow PVS \quad x \rightarrow \theta(x) \tag{F.8}$$

Theorem *If q denotes a commander cell request and x denotes a cell gene,* then:

$$\mu(q) \equiv \pi(\rho(\delta(x)) \cap \tau(\varepsilon(x)) \cap \gamma(\theta(x)).$$

The management of a cell's internal system is divided among its subsystems according to a definite number of roles. In order to invoke a cell, a client request (q) must pass the cell's security system (F.1). After ensuring a secure cell invocation, DS begins the response process. It demands building output by the OBS (F.2). OBS output is based on a deep cell process analysis (F.4), a precise cell process validation (F.5) and assessing relevant cell characteristics (F.6). Tests (analysis and validation) are applied to cell process storage through GSS (F.6–F.8).

6 Components of Executive Cell

The proposed executive cell in cell theory is composed of (Fig. 5): decision system (DS), gene store system (GSS), trait maintenance system (TMS), output builder system (OBS), process validation system (PVS), process analyser system (PAS), defence system (DFS) and gene storage.

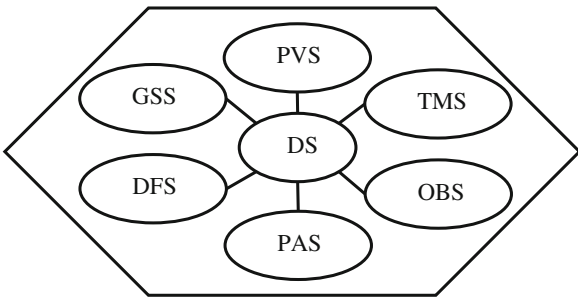

Fig. 5 Components of executive cell

6.1 Decision System (DS)

The decision system is the brain of the Executive cell in COC. It is controlled by the management and control centre and is responsible for taking decisions and directing other components of the cell. Cell inputs are received by the DS which study the client request and emit suitable outputs. Cell computing is characterized by two levels of collaboration that are managed through DSs. The first collaboration level is expressed by internal cooperation among cell subsystems, while the second level of collaboration is applied among cells to build a complete answer for cell customers. In the case of a customer request, the DS asks the defence system to verify the customer identity and request before starting the answer process. If the customer request is safe, DS sends the input to the OBS and waits for the answer. Sometimes, one cell is not sufficient to serve a customer. In this case, the DS asks for collaboration from other cells to produce an answer.

6.2 Defence System (DFS)

Cell computing aims to correct the problems of the service model. One of the main service-oriented computing problems is security. Security weakness is less of a danger in the case of Web service, but currently most cloud services are public and store sensitive data, so that any security fault may be fatal to some institutions. As a way of obtaining strict computing resource protection, COC introduces internal cell protection. As is well known, there are two main steps to protecting the Web. The first step is network protection via several encryption methods. However, the second step is characterized by server resources protection via user tokens and security tools. The proposed COC security technique ensures protection against any internal or external unauthorized access to a cell. In addition to network and system protection, the cell defence system aims to introduce a double verification method. This is a hidden type of cell protection that verifies, on one side, if a customer has the right to invoke a cell, while it also checks, on the other side, if a customer's machine is capable of receiving an output from such a cell. COC aims to make the distributed Web application as secure as possible.

6.3 Gene Store System (GSS)

There are several combinations of processes that return the same results in a distributed application. Some of these applications are Web services that are divided into a set of groups, such that in each group all the applications can do the same jobs. The problem for service theory is summed up by the question of how to select the best service from an ocean of similar job services? COC has indeed found a

solution to the service selection problem. Simply put, why not transform all the business processes into a new structure to be used by a novel model like COC? In order to obtain a successful COC model, we need to build a suitable business process (gene) for each cell. The first step in building cell genes is to transform the service business processes and their combinations into a graph (or map) of abstract business processes. The obtained graph has no abstract information about any service business process. For example, if several services make a *division* job, then all of their *abstract business processes* are linked to a *division* node of the gene graph. Each cell uses a specific part of the obtained abstract graph and is known as a cell business process or gene. The gene store system's job is to store the genes and classify them, shaped by logical rules in a database to be easily used by cell subsystems.

6.4 Process Analyser System (PAS)

Changes allow companies to improve processes, to expand in new directions and to remain up-to-date with the times and technology. A business process is a sequence of steps performed for a given purpose. Business process analysis is the activity of reviewing existing business practices and changing these practices so that they fit a new and improved process. The role of PAS is to keep up-to-date analysis of the cells' business processes. A cell's business process design is based on a composition of process combinations transformed from service business processes. In order to return the best cell output, PAS must select the best plan from these combinations. This job requires a permanent process design analysis. Since business process combinations are transformed into a graph of abstract business processes, the process design analysis is achieved by a multi-graph analysis. The multi-graph analysis is discussed in detail in our book chapter [30].

6.5 Process Validation System (PVS)

The cell business process, in COC, is built on a dynamic composition of a group of service business processes. If there are problems in one or more business applications that support a cell business process, then the consequences of disruption to the cell process can be serious. For example, some process compositions may result in infinite loops or deadlocks. The process validation system's job is to monitor and validate the changes in altered or new composition processes. The validation technique used by PVS is described in our chapter [28]. This validation technique is divided into two steps. First the business process is transformed into a graph. Then, a *depth first search* is applied on the obtained graph to detect deadlock errors.

6.6 Traits Maintenance System (TMS)

A cell business process is a dynamically coordinated set of collaborative and transactional activities that deliver value to customers. Cell process is complex, dynamic, automated and long running. One of the key characteristics of a good cell business process is continuous improvement. These improvements ensure a constant flow of ideal traits into the cell process. Cell computing is built upon achieving a group of architectural traits such as: performance, reliability, availability and security. These qualities require stable monitoring to maintain the supply of customers. Cells in COC apply internal and external efforts to maintain best traits. External efforts are achieved via cell collaboration, while internally the job is done by TMS. Indeed, TMS analyses the quality of gene (QoG) of a cell; these are combinations of the traditional quality of service (QoS) analysis. It uses a data warehouse of QoG to accomplish this type of analysis. The service analysis based on the QoS data warehouse is discussed in details in our book chapter [31].

6.7 Output Builder System (OBS)

Cells in COC are considered as intelligent modular applications that can be published, located and invoked across the Web. They are intended to give the client best results by composing their distributed business processes dynamically and automatically based on specific rules. Based on the service model, companies only implement their core business and outsource other application services over the Internet. However, no single Web service can satisfy the functionality required by the user; thus, companies try to combine services together in order to fulfil the request. Indeed, companies face a major problem: Web service composition is still a highly complex task and it is already beyond human capability to deal with the whole process manually. Therefore, building composite Web services with an automated or semi-automated tool is critical. As a solution to the service composition problem, cell theory proposes a cell that is capable of achieving an automated composition of its business process. In sum, after analysing, validating and ensuring the good characteristics of business process choices to be used by a cell by PAS, PVS and TMS, OBS selects and executes the best process plan based on the user's request. The role of OBS is to apply a dynamic and autonomic composition of the selected business processes of the collaborating cells.

7 Strategy of Cell Computing

Cell computing allows sharing of the process to reach a solution. This way of computing results, indirectly, in a shared resources environment similar to that of grid computing. Recursively, a client cell has access to all other executive cells as

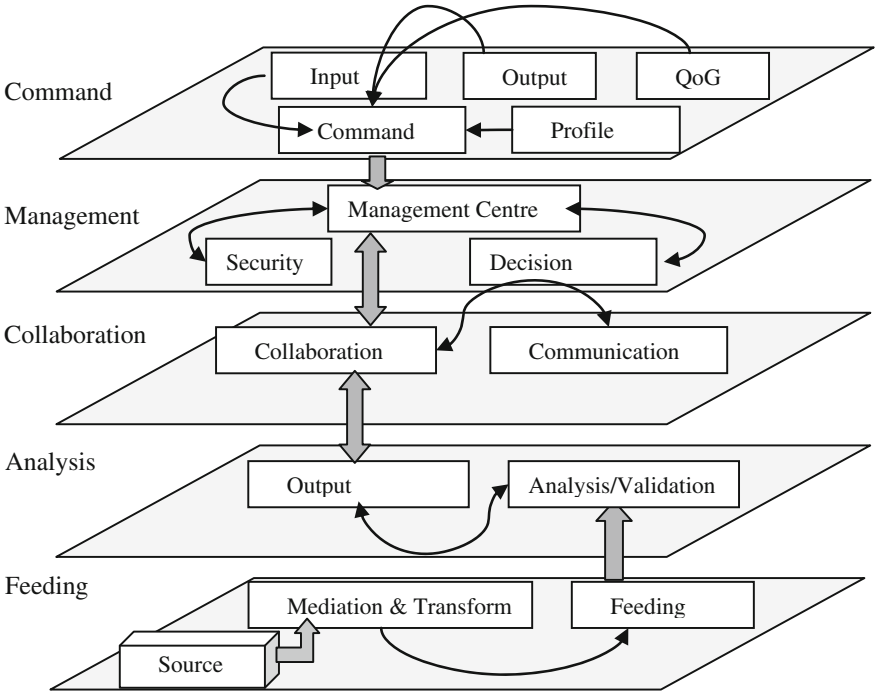

Fig. 6 Strategy of cell computing

they are running on one machine. The cell network is organized, secure, reliable, scalable and dynamic. Cell computing strategy, as shown in Fig. 6, is based on five main layers of computation: command layer, management layer, collaboration layer, analysis layer and feeding layer.

Command Layer: The command layer consists of solutions designed to make use of the smart selection of cells that can provide a specific service. It makes up the initial step of the exchange in cell architecture. An important role of the command layer is to allow for clear separation between available solutions and a logical methodology in producing a solution based on the client's command. The traditional Web service methodology gives clients the right to select one of the predesigned Web applications that will process their solution depending on several qualities and a complex selection process. However, cell methodology has improved the process by making clients give commands and creating the application according to these commands. This approach enables a slew of new applications to be created that make use of the COA's cooperative capabilities, without requiring in-depth knowledge of application processes, communication protocols, coding schemes or session management procedures; all these are handled by the upper layers of the cell strategy. This enables modular interfaces to incorporate new services via a set of commands composed of specifying inputs, output intervals, QoG requirements and the user profile.

Management Layer: this layer provides configurable controlling and reporting for client commands and server facilities at operational and services levels. It also provides visibility across physical, virtual-based layers, making it possible to govern the enforcement and migration of COA across the distributed enterprise. The management layer of the cell-based architecture not only reduces deployment, maintenance and operation costs but also allows for the provision of better performance, scalability and reliability. Its agent-based capabilities provide for comprehensive management of all cell collaboration procedures. The management layer controls the start-up and status of solutions, the logging of maintenance events, the generation and processing of Cells, the supervision of security and the management of application failures. The management layer provides centralized solution control and monitoring, displaying the real-time status of every configured solution object, as well as activating and deactivating solutions and single applications, including user-defined solutions. This layer additionally provides simple integration with a variety of enterprise-level business intelligence, reporting and event correlation tools for deeper analytics and insight. It automatically associates recovery with the solutions as active conditions in the system until they are removed by another maintenance event.

Collaboration Layer: in COA, cells work with each other to perform a task and to achieve a shared goal. They utilize recursive processing and a deep determination to reach the client's objective. Most collaboration requires leadership; in COA, each cell, by its decision system, can take the leading role. In particular, the collaborative property of the cells results in better processing power when facing competition for complex jobs. COA is based on specific rules of collaboration and manages the communications among cells. These rules characterize how a group moves through its activities. The desired cell collaboration aims to collect suitable sub-tasks that are composed to achieve a complete and efficient process in carrying out a specific job.

Analysis Layer: the analysis layer generates the statistical data used for interaction management and control centre reporting. It also enables solutions to communicate with various database management systems. Through the analysis layer, providers of processes can be seen as a store of dynamic, organized quality of process, generating new cell processes. In this layer, the collected data of ontologies that represent business processes are analysed, validated and tested before operational use by cells. Two types of analysis are used in cell methodology. The first type studies the qualities of source processes; while the other type studies the graph analysis measures of the selected sub-processes.

Feeding Layer: this layer aims to find sources of business processes and tries to handle the complexity and diversity transforming business processes through a special mediator. The feeding process starts by fetching sources about process designs and results in a semantic design, as ontology, compatible with cell requirements.

8 Discussion

The most dominant paradigms in distributed computing are Web service and software agent. Inserting intelligence into these paradigms is critical, since both paradigms depend on non-autonomous driven architecture (SOA) that prevents one-to-one concurrence with the needs of third party communication (i.e., the service registry). In addition to this, the Web service and agent paradigms suffer from negative complexity and migration effects, respectively. The complexity, in general, comes from the following sources. First, the number of services accessible over the Web has increased radically during recent years and a huge Web service repository to be searched is anticipated. Second, Web services can be formed and updated during normal processing; thus the composition system needs to detect this updating at runtime and make decisions based on the up-to-date information [32]. Third, Web services can be developed by different organizations, which use different conceptual models to describe the services; however, there is no unique business process to define and evaluate Web services. Therefore, building composite Web services with an automated or semi-automated tool is critical.

The migration of processes and the control of that migration, together with their effect on communication and security, was a problem for mobile agents. Indeed, the first problem of the Web is security; giving an application the ability to move among distributed systems and choose the place to make execution may cause critical problems. Agent methodology has several advantages; however, it can destroy human control if it is not covered by rules and limits.

How we can benefit from the wide use of service-oriented architecture in building intelligent architecture? How can we avoid the complex selection process of the Web service model? How can we achieve dynamic business process composition despite the variety of companies providing different types of service processing? How can we use the intelligence of multi-agent systems as a control mode from the client side? How can we reach the best non-functional properties of processes in an autonomic manner? How can we avoid the security weaknesses resulting from mobile agent communications? How can we prevent damage to service caused by internal and subservice fail? Why not separate software processes based on their purpose? How might we arrange procedures of distributed computing in a way that evades big data analysis problems resulting from random connections among distributed systems? How can globally consistent solutions be generated through the dynamic interaction of distributed intelligent entities that only have local information? How can heterogeneous entities share capabilities to carry out collaborative tasks? How can distributed intelligent systems learn, improving their performance over time, or recognize and manage faults or anomalous situations? Why not use dynamic online analysis centres that monitor the on-the-fly qualities of distributed software processes? How should we validate the processes of distributed software at the design phase? How should we accomplish the internal protection of distributed components based on the dual context-profile of both consumer and solution provider?

Cell methodology uses commands among smart components: neither an invocation of non-smart component nor a migration of processes. It is based on cells that can benefit from the variety of already built Web components to achieve intelligent distributed computing. They have brains, decision support systems that can do the same jobs as a mobile agent. This brain can communicate with the mobile agent on the client side by messages without process migration. Furthermore, it has its own strategy to analyse and organize connections based on communications with the management and control centre. Cell methodology requires no discovery or selection steps to use a cell because it uses a new model of the composition process to realize the user's request. It participates in solving the big data problem by making a real time analysis of communications. It is highly secure, since it uses a combination of context-aware and pervasive computing among cells.

Cells are smart components that combine a collection of characteristics from different environments. They apply autonomy and intelligence based on a mobile agent computational perspective. In addition, they map the human cell traits, such as inheritance and collaboration, into distributed computing. From the software engineering side, cells try to achieve best architecture properties such as security, availability and performance.

8.1 Autonomy

The cell approach proposes that the problem space should be decomposed into multiple autonomous components that can act and interact in a flexible way to achieve a processing goal. Cells are autonomous in the sense that they have total control over their encapsulated state and are capable of taking decisions about what to do based on this state, without a third party intervention.

8.2 Inheritance

The commander cell inherits the profile property from its environment (company, university, etc.). However, the executer cell can serve commanders according to their environmental profile (selection of suitable qualities of a process) or by special interference from the commander's side to specify more precisely the general design of a process and its qualities. This inheritance property in COA is similar to the inheritance among generations of human beings. For example, babies inherits traits of their parents such that cells combine traits from the father and the mother, but the parent can ask a doctor for specific trait in a baby different from their own traits (blue eyes, brown hair, etc.). In this case, they have given more specifications to the cell in order to select suitable genes.

8.3 Internal Security

When application logic is spread across multiple physical boundaries, implementing fundamental security measures such as authentication and authorization becomes more difficult. In the traditional client-server model, the server is most responsible for any protection requirements. Well-established techniques, such as secure socket layer (SSL), granted a so-called transport level of security. Service and agent models emphasize the emplacement of security logic at the messaging level. Cell methodology applies an internal level of security in cells. Thus, command and executive cells can communicate after the protection steps summarized by verifying the context profile of the cell that requests collaboration.

8.4 Availability

Availability of cells and data is an essential capability of cell systems; it is actually one of the core aspects giving rise to cell theory in the first instance. The novel methodology of cell theory decreases the redundancy of servers to ensure availability. Its strength lies in the ability to benefit from the redundancy of processes that serve similar goal, so failures can be masked transparently with less cost.

8.5 Collaboration

Collaborative components are need in today's primary resources to accomplish complex outcomes. Cell methodology depends on collaboration-by-command that enables coordination by one of the collaborative components. Collaboration allows cells to attain complex goals that are difficult for an individual cell to achieve. The cell collaborative process is recursive: the first collaborative agent makes a general command that is passed gradually through collaborative cells to more specific cells until reaching the desired results.

8.6 Performance

Distributed computational processes are disjointed; companies' coding is not ideal and it is difficult to monitor the complexity of every process. Thus, performance problems are widely spread among computational resources. Cell theory introduces a solution for performance problems in a distributed environment. The solution can be summarized as applying a permanent analysis of different processes aiming for the same goal, attached to a unified cell, then selecting the best process to do a job,

based on basic properties such as response time and code complexity. Furthermore, an increase in communication acquaintance can be a guide to an improvement in performance as it enables cells to communicate in a more efficient manner.

8.7 Federation

Cells are independent in their jobs and goals. However, all distributed processes that do same type of job are connected to a specific executive cell. Thus each executive cell is federated with respect to the commander cell's request. Cells map can be considered as a set of federated components that are capable of collaborating to achieve a solution.

8.8 Self-error Cover

There are two types of errors that can be handled by cell computing: structural and resource errors. The cell process is based on a combination of codes that are fabricated by different computational sides. These combinations may fail because of coding or system errors and fall in deadlock. The process validation system's job is to monitor changes in process and recover errors if detected. Resource errors are described as failure in providing a service from the computational resource. The solution to these types of error is to connect spare procedures in each cell process to achieve the same quality of job from different sources.

8.9 Interoperability

Cell interoperability comes from the ability to communicate with different feeding sources and transform their business processes into cell business processes. For example, in spite of differences among business processes, such as BPEL and OWL-S, every provider of service is seen as a source of genes and as useful in cell computing. Based on cell interoperability, all procedures and applications used by service providers can be unified under a unique type of process computing, the cell gene, with respect to cell provider.

9 Conclusion

With the extensive deployment of distributed systems, the management and integration of these systems have become challenging problems, especially after smart procedures are implemented. Researchers build new technologies to deal with these

problems. Trends of the future Web require inserting intelligence into the distributed computing model; thus, the goal of future research is intelligent distributed computing. At this time, the research introduces the cell computing theory to cover the distributed system problems through intelligent method of processing. Cell theory is the implementation of human cells' functions in a distributed computational environment. The cell is an intelligent, organized, secure and dynamic component that serves a specific type of job. Cell methodology divides the task between two types of components, the commander and the executer. The commander is a light cell that represents the client and can communicate smartly with its distributed environment to request solutions. The executive cell works as a smart supplier that depends on wide collaborations to fabricate a solution. Cell strategy is based on high-level communication among Cells, a permanent analysing process among collaborating components and context-based security among collaborating cells.

Acknowledgments This work has been supported by the University of Quebec at Chicoutimi and the Lebanese University (AZM Association).

References

1. Brain, M.: How cells work, howstuffworks? A Discovery Company (2013). http://science.howstuffworks.com/life/cellular-microscopic/cell.htm
2. Petrenko, A.I.: Service-oriented computing (SOC) in engineering design. In: Third International Conference on High Performance Computing HPC-UA (2013)
3. Feier, C., Polleres, A., Dumitru, R., Domingue, J., Stollberg, M., Fensel, D.: Towards intelligent web services: the web service modeling ontology (WSMO). In: 2005 International Conference on Intelligent Computing (ICIC'05), Hefei, 23–26 Aug 2005
4. Suwanapong, S., Anutariya, C., Wuwongse, V.: An intelligent web service system. Engineering Information Systems in the Internet Context, IFIP—The International Federation for Information Processing vol. 103 (2002), pp. 177–201 (2014)
5. Li, C., Zhu, Z., Li, Q., Yao, X.: Study on semantic web service automatic combination technology based on agent. In: Lecture Notes in Electrical Engineering, vol. 227, pp. 187–194. Springer, Berlin (2012)
6. Rajendran, T., Balasubramanie, P.: An optimal agent-based architecture for dynamic web service discovery with QoS. In: International Conference on Computing Communication and Networking Technologies (ICCCNT) (2010)
7. Sun, W., Zhang, X., Yuan, Y., Han, T.: Context-aware web service composition framework based on agent, information technology and applications (ITA). In: 2013 International Conference
8. Tong, H., Cao, J., Zhang, S., Li, M.: A distributed algorithm for web service composition based on service agent model. IEEE Trans. Parallel Distrib. Syst. **22**, 2008–2021 (2011)
9. Yang, S.Y.: A novel cloud information agent system with web service techniques: example of an energy-saving multi-agent system. Expert Syst. Appl. **40**, 1758–1785 (2013)
10. Maryam, M., Varnamkasti, M.M.: A secure communication in mobile agent system. Int. J. Eng. Trends Technol. (IJETT) **6**(4), 186–188 (2013)
11. Liu, C.H., Chen, J.J.: Role-based mobile agent for group task collaboration in pervasive environment. In Second International Conference, SUComS 2011, vol. 223, pp. 234–240 (2011)

12. Rogoza, W., Zabłocki, M.: Grid computing and cloud computing in scope of JADE and OWL based semantic agents—a survey, Westpomeranian Technological University in Szczecin (2014). doi:10.12915/pe.2014.02.25
13. Elammari, M., Issa, Z.: Using model driven architecture to develop multi-agent systems. Int. Arab J. Inf. Technol. **10**(4) (2013)
14. Brazier, F.M.T., Jonker, C.M., Treur, J.: Principles of component-based design of intelligent agents. Data Knowl. Eng. **41**, 1–27 (2002)
15. Shawish, A., Salama, M.: Cloud computing: paradigms and technologies. Stud. in Comput. Intell. **495**(2014), 39–67 (2014)
16. Jang, C., Choi, E.: Context model based on ontology in mobile cloud computing. Commun. Comput. Inf. Sci. **199**, 146–151 (2011)
17. Haase, P., Tobias, M., Schmidt, M.: Semantic technologies for enterprise cloud management. In Proceedings of the 9th International Semantic Web Conference (2010)
18. Block, J., Lenk, A., Carsten, D.: Ontology alignment in the cloud. In Proceedings of ontology matching workshop (2010)
19. Ghidini, C., Giunchiglia, F.: Local model semantics, or contextual reasoning = locality + compatibility. Artif. Intell. **127**(2), 221–259 (2001)
20. Serafini, L., Tamilin, A.: DRAGO: distributed reasoning architecture for the semantic web. In: Proceedings of the Second European Conference on the Semantic Web: Research and Applications (2005)
21. Borgida, A., Serafini, L.: Distributed description logics: assimilating information from peer sources. J. Data Semant. **2003**, 153–184 (2003)
22. Schlicht, A., Stuckenschmidt, H.: Distributed resolution for ALC. In: Proceedings of the 21th International Workshop on Description Logics (2008)
23. Schlicht, A., Stuckenschmidt, H.: Peer-peer reasoning for interlinked ontologies. Int. J. Semant. Comput. (2010)
24. Kahanwal, B., Singh, T.P.: The distributed computing paradigms: P2P, grid, cluster, cloud, and jungle. Int. J. Latest Res. Sci. **1**(2), 183–187 (2012). http://www.mnkjournals.com/ijlrst.htm
25. Shi, L., Shen, L., Ni, Y., Bazargan, M.: Implementation of an intelligent grid computing architecture for transient stability constrained TTC evaluation. Journal Electr Eng Technol **8** (1), 20–30 (2013)
26. Gjermundrod, H., Bakken, D.E., Hauser, C.H., Bose, A.: GridStat: a flexible QoS-managed data dissemination framework for the Power Grid. IEEE Trans. Power Deliv. **24**, 136–143 (2009)
27. Liang, Z., Rodrigues, J.J.P.C.: Service-oriented middleware for smart grid: principle, infrastructure, and application. IEEE Commun. Mag. **2013**(51), 84–89 (2013)
28. Karawash, A., Mcheick H., Dbouk, M.: Intelligent web based on mathematic theory, case study: service composition validation via distributed compiler and graph theory. Springer's Studies in Computation Intelligence (SCI) (2013)
29. Aviv, R.: Mechanisms of Internet-based collaborations: a network analysis approach. Learning in Technological Era, 15–25 (2006). Retrieved from http://telem-pub.openu.ac.il/users/chais/2006/04/pdf/d-chaisaviv.pdf
30. Karawash, A., Mcheick H., Dbouk, M.: Simultaneous analysis of multiple big data networks: mapping graphs into a data model. Springer's Studies in Computation Intelligence (SCI), (2014a)
31. Karawash, A., Mcheick H., Dbouk, M.: Quality-of-service data warehouse for the selection of cloud service: a recent trend. Springer's Studies in Computation Intelligence (SCI) (2014b)
32. Portchelvi, V., Venkatesan, V.P., Shanmugasundaram, G.: Achieving web services composition—a survey. Sci. Acad. Publ. **2**(5), 195–202 (2012)