International Journal of Software Engineering
and Knowledge Engineering
Vol. 18, No. 5 (2008) 573–596

World Scientific
www.worldscientific.com

INTEGRATING AGENT, SERVICE AND ORGANIZATIONAL COMPUTING*

LONGBING CAO
Faculty of Information Technology, University of Technology, Sydney, Australia
lbcao@it.uts.edu.au

Engineering open complex systems is challenging because of system complexities such as openness, the involvement of organizational factors and service delivery. It cannot be handled well by the single use of existing computing techniques such as agent-based computing and service-oriented computing. Due to the intrinsic organizational characteristics and the request of service delivery, an integrative computing paradigm combining agent, service, organizational and social computing can open complex systems more effectively engineering. In this paper, we briefly introduce an integrative computing approach named OASOC for system analysis and design. It combines and complements the strengths of agent, service and organizational computing to handle the complexities of open complex systems. OASOC provides facilities for organization-oriented analysis and agent service-oriented design. It also supports transition between analysis and design. Compared with the existing approaches, our approach can (1) support service and organization that are either rarely or weakly covered by single computing methods, (2) provide effective mechanisms to integrate agent, service and organizational computing, and (3) complement the strengths of various methods. Experiences in engineering an online trading support system have further shown the workable capability of integrating agent, service and organizational computing for engineering open complex systems.

Keywords: Open complex systems; agent-oriented software engineering; organizational modeling; service-oriented computing; integrative computing; OASOC.

1. Introduction

Open complex systems [5] are middle-size or large-scale [27] open autonomous, adaptive or human-involved [4, 17] organizations. Among them, open complex agent systems (OCAS) are critical because they can be built as the problem-solving system of open complex systems [25]. Besides the large-scale feature, OCAS also presents intrinsic system complexities such as variant services, openness [12], human involvement [25, 4] and socialization [23, 11]. For instance, OCAS goals may be multiform and/or hierarchical. Environment is often open, networked, heterogeneous or dynamic. Interactions, activities and behaviors may take place

*This research was supported by the Australian Research Council Discovery and Linkage Grants (DP0667060, DP0773412, LP0775041).

temporally, spatially or spatial-temporally with human involvement. The rules and constraints may involve multiple aspects. Therefore, it is important to handle such complexities in engineering OCAS.

There are a few agent-oriented software engineering (AOSE) approaches available, eg., Formal TROPOS [18], GAIA [34], MASE [32], MESSAGE [3], OperA [24], ROADMAP [19], SODA [23], KAoS [1], OAA [12], etc. To varying extents they provide good support like *role modeling* for engineering complex agent systems. However, there are some major issues in these approaches [4, 35] that indicate they cannot tackle OCAS very well. For example, some fundamental system members such as system dynamics and services are not supported.

To tackle the intrinsic complexities in engineering OCAS, it is useful to combine organizational theory [10, 30], artificial social systems [10, 11, 4], the science of complexity [31] and complex system theories. This leads to an *integrative computing*: combining the respective strengths of agent-based computing (ABC) [33], service-oriented computing (SOC) [28], organizational modeling and social computing into engineering OCAS. This paper briefs our work toward such integrative computing.

In developing a macro-economy decision support system[a] [5], a hybrid telecom business intelligence system[b] [7], and an online financial trading and mining support system F-Trade[c] (Financial Trading Rules Automated Development and Evaluation) [8], we bring ABC, SOC, *organizational modeling* [30] and *metasynthetic engineering* together to engineer OCAS [9]. We proposed an *integrative computing* approach, referred to as *organization, agent and service oriented computing* (OASOC). OASOC synthesizes organization, agent and service computing, and develops appropriate techniques for the interaction among OOM, ABC and SOC in engineering OCAS. In particular, it consists of *organization-oriented analysis* (OOA) for system analysis, and *agent service-oriented design* (ASOD) for system design.

This paper briefs the concept and major components of OASOC for OCAS abstraction, analysis and design. The main contributions of the OASOC include the following aspects.

(i) It supports the interaction among OOM, ABC and SOC. This not only supports key concepts such as service and organization that are either rarely or weakly covered by single computing methods, but also aggregates OOM abstraction based system analysis with agent service based system design.
(ii) It proposes a systematic abstraction framework named ORGANIZED and corresponding building blocks for explicitly modeling almost all major members in an OCAS. Some of them such as system dynamics have not been covered by other approaches.

[a]Large Grant of National Science Foundation of China (79990580).
[b]China Innovation Fund for Small Technology Based Firms (04C26211100957).
[c]www.f-trade.info.

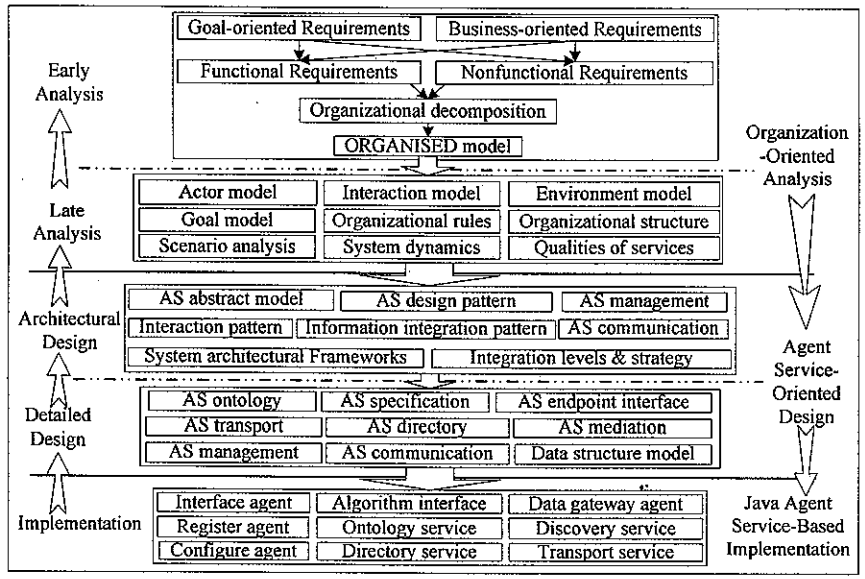

Fig. 1. The OASOC methodology.

(iii) It supports the full lifecycle and transition from system abstraction to analysis and design. As reported in [9], OASOC has been used for engineering F-Trade successfully.

2. The OASOC Overview

2.1. *Basic OASOC phases*

As shown in Fig. 1, OASOC consists of software engineering mechanisms for all major AOSE phases. It includes organization-oriented analysis (OOA) supporting early analysis and late analysis, agent service-oriented design (ASOD) for system architectural and detailed design, and Java agent service-based implementation (JASBI) for system implementation. The methodology of OASOC is as follows.

— The early analysis investigates goal-oriented and business-based requirements, extracts major system members in an OCAS in terms of the ORGANIZED framework;
— The late analysis models each member in detail, and analyze their relations;
— The architectural design provides major system architecture and its constituents, and arranges data, service and work flow and its internal connections;
— In the detailed design, architectural models are instantiated into programmable components, and connected to represent organizational behavior; and

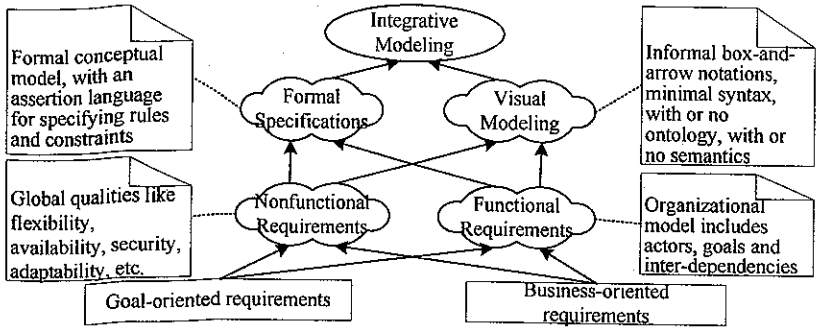

Fig. 2. Framework of integrative modeling.

— An OCAS is implemented in Java agents on the basis of Web service structure.

2.2. *OASOC philosophy*

The OASOC philosophy for the engineering of OCAS is based on the theory of systematology [25, 26], which combines top-down reductionism and bottom-up holism.

— The theoretical foundations of the OASOC include the science of complexity, computational organizations, organizational behaviors, and social intelligence, etc.;
— OCAS decomposition follows a top-down reductionism: high-level members such as goals and structures are extracted first. Further work goes step-by-step toward system members such as rules, interaction, dynamics and actors;
— Its modeling follows integrative modeling (as shown in Fig. 2), which combines (i) functional and nonfunctional requirements, (ii) goal-oriented and technical requirements in business fashion, and (iii) formal specifications and informal diagrammatic notations systematically;
— The aggregation follows a bottom-up holism: based on the decomposition results, constructing a unified overview of an organization; understanding system-level and subsystem-level characteristics by aggregating components into subsystems, architectural frameworks or design patterns.
— The synthesis of analysis, design and their refinement follows the theory of systematology: the above decomposition and aggregation and their refinement are undertaken progressively and iteratively with back-and-forth in the process of system analysis and design.

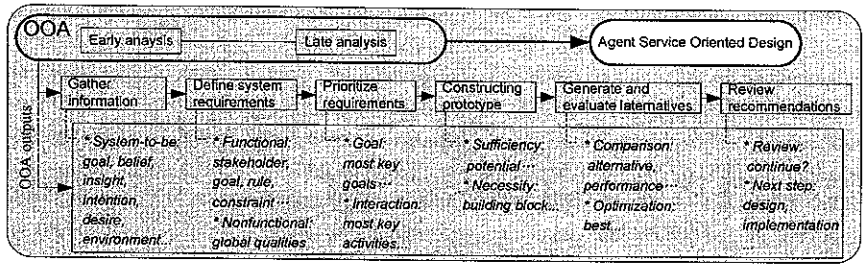

Fig. 3. OOA tasks and process.

3. Organization Oriented Analysis

3.1. *OOA tasks and process*

The tasks and process of OOA also follows the general ones in software engineering. However, those general tasks of system analysis are specialized in terms of organizational metaphor. In particular, the following OOA actions are taken to extract and model system complexities of an OCAS.

— Extract members like goals, constraints, beliefs, environment etc. in an OCAS-to-be;
— Defining functional and nonfunctional requirements in the system-to-be via conducting organizational abstraction and decomposition;
— Specifying major requirements such as stakeholders, goals and interaction activities by diagrammatic notations and/or formal specifications;
— Scrutinizing all building blocks of the system-to-be; this can be through developing scenarios and analysis, capturing state transition of stakeholders, listing state chain of each stakeholder to grasp the state transition in the organization;
— Evaluating and fine-tuning the model building technology and building blocks; and
— Determining how to transit to system design and implementation.

Figure 3 shows the major OOA processes and their tasks in analyzing OCAS. The outcome of OOA is a set of sufficient and optimal organizational members and their building blocks of models in the system-to-be.

3.2. *Organizational abstraction*

With the organizational metaphor, an OCAS is viewed as consisting of an agent organization and its environment. An agent organization is represented in terms of the following key organizational members: organizational *R*ules, *G*oals, *A*ctors, *N*orms, *I*nteractions, *S*tructures and *D*ynamics, while the *E*nvironment refers to the totality surrounding an agent organization. Some other elements such as Beliefs, Intentions, Policies, Laws, Self-Organization, Emergence, Plan and the like may also

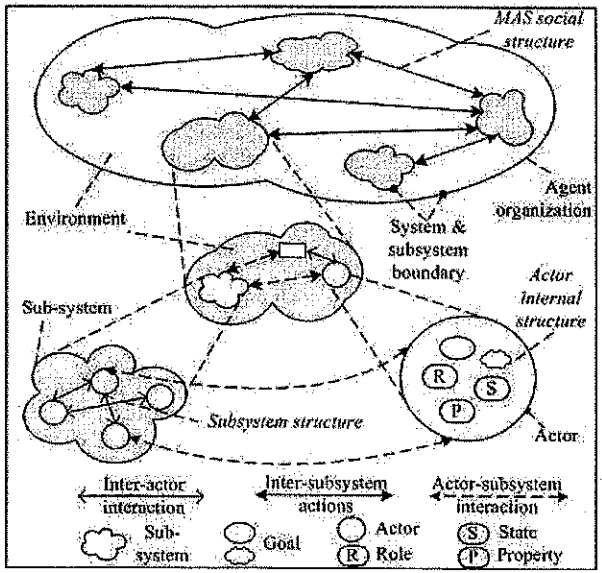

Fig. 4. OCAS as an agent organization.

be considered in different agent-based scenarios. In OASOC, they are modeled in terms of either attributes or properties of an agent organization.

Definition 1 (Agent Organization). An OCAS is an open complex agent-based software system, which is a function of an agent organization o and its environment e; while o is embodied through its attributes and system members r, g, a, n, i, s, d.

$$OCAS ::= \{(o,e)|o \in \{(r,g,a,n,i,s,d)|r \in R, g \in G, a \in A, n \in N, i \in I, s \in S, d \in D\}, e \in E\} \quad (1)$$

We propose an organization-centric framework named ORGANIZED (the capitalized letters form the name ORGANIZED) to capture all of the above members in an OCAS. Figure 4 illustrates an organizational view of OCAS-like agent systems.

In [6], we have presented model-building blocks to represent the system members R, G, A, I, S, E and D. This section further discusses the following issues: (i) the extraction and management of the ORGANIZED members in an OCAS, and (ii) the understanding of OCAS environment. These aspects have not been covered very well before.

3.2.1. *Member extraction and management*

In extracting organizational members, a fundamental job is to classify and manage carefully the categories and corresponding namespace of the ORGANIZED

members. This substantially involves both structural and semantic relationships between the ORGANIZED members in an OCAS. In fact, a member often has a family of populations. Among them some have further sub-populations underneath, and forms a hierarchical relation. For instance, in modeling organizational rules, four relations Control, Peer, Ownership and Dependency are identified to manage internal and external relations existing among relevant ORGANIZED members. We further develop ontological engineering techniques (see Secs. 4.3.1 and 4.3.5) to represent and manage the extracted members and their relations in terms of ontological items and relationships.

Definition 2 (Ontological Predicate Formula). An ontological predicate formula is a semantic annotation which links two or more ontological items (o_n) through some form of ontological structural or semantic relations. Here *predicate* refers to (i) the ORGANIZED members, and (ii) specific structural and semantic relationships.

$$f(o_1, o_2, \ldots) ::= predicate(o_n|n \in N) \quad (2)$$

For instance, we distinguish and classify actors in terms of entities, functions and roles in an open agent organization. As a result, the actor set A consists of four types of actors — *Human actor, Workspace actor, Autonomous actor* and *Service actor*. Table 1 lists such actors and examples.

Table 1. Actor model.

Actor Name	Symbol	Informal Definition	Instances
Human		Human Actor is a family of those social actors — human beings — who will interact with the system in different ways and permissions; these could be customers, users, administrator, and the like	Customer
Workspace		Workspace Actor is the place where all actors, interaction and activities will be lodged and undertaken; a common Workspace is the proposed system	F-Trade
Autonomous (Agent)		Artificial Actor includes system components or agents which fulfill some responsibilities or decision control by themselves; these are usually agents or components	Interface Agent
Service		Service Actor are actors who can perform some activities or some functions which are associated with certain autonomous actors; these could be services associated with agents	Ontology Service
Resource		Refers to various databases, knowledge bases, configuration files, etc.	Knowledge base

In this case, we have the following examples of actors A_H and A_A.

Example 1 (Actor). Let actor a_1 belong to Human A_H, a_2 belong to Agent A_A, we have the ontological relationships *disjoint_to* to associate the two actors a_1 and a_2 in terms of $disjoint\_to(a_1, a_2)$, where $a_1 \in A_H$, $a_2 \in A_A$, and $A_H \supset A$, $A_A \supset A$.

Following the above method, we can decompose and identify the whole populations of ORGANIZED members in an OCAS. We then are able to obtain a space of organizational domain including all building blocks of an OCAS. Further, as discussed in Sec. 4.3.5, an ontological space can be developed to map and present organizational elements in an appropriate ontological space. As a result, we can deal with these converted ontological elements in more formal and accurate manner in the period of system design. The ontological engineering techniques are helpful for the formalization and model-checking in the software engineering.

3.2.2. *Environment*

The OCAS environment can be largely varying in different systems. Basic characteristics [22] for agent environment model include Accessibility, Observability, Determinism, Uncertainty, Diversity, Controllability, Volatility, Temporality (Continuity), Locality, Spatiality, and so forth. With respect to OCAS-like systems, their environments present special complexities such as openness, adaptability, human involvement and dynamics.

To view from one of the above aspects, an agent environment may take the forms of:

— either fully, partially or not observable or accessible;
— static or dynamic;
— controllable or uncontrollable;
— the next state of the environment could be deterministic, stochastic, or adaptive;
— the task for the environment agent could be episodic or sequential;
— the environment, perceptions and actions could be discrete or continuous; and
— the environment may include single or multiple agents or services.

In the real-world systems, the environment usually takes the forms of a synthetic agency with a collection of potential features such as inaccessible, stochastic, sequential, dynamic and continuous, etc. For instance, the artificial ant colony exists in an accessible, discrete grid space with continuously changing hormone and stochastic state transfer.

Modeling OCAS environment involves the specification of all actors, resources, principles, processes, forces, interactions and system (or subsystem) boundaries surrounding an agent organization. The environment encompasses resources that an organization can exploit, control or consume when it goes toward the achievement of goals. To summarize all the main features of environment in variant agent systems, an abstract but comprehensive agent environment model can be built. We

can collect the main members and their features from agent environment and list all of them together on an abstract agent environment model. They can also be embodied through the modeling of agent organization such as organizational interaction model, structure model and interfaces. For instance, OCAS environment can be modeled in terms of Partially Observable Markov Dynamic Process [20].

3.3. *Modeling OCAS system members*

Following the ORGANISED framework, the OOA builds individual models for each member in an OCAS. For instance, [6] presents Organizational Rule Model, Goal Model, Actor Model, Interaction Model, Organizational Structure Model, Environment Model and Organizational Dynamics Model. In the following sections, we first illustrate the construction of model-building blocks in terms of Norm Model, and then demonstrate visual and descriptive modeling used in the OASOC for OCAS modeling.

3.3.1. *Norm model*

The concept of norm [13] is important in organization-oriented metaphor of OCAS. Norm refers to organizational constraints including permissions, obligations and prohibitions. They may be presented as social patterns governing perceptual, denotative, evaluative, cognitive or behavioral aspects. Based on our application situations, we focus on constraints and regulations on an object α such as an actor, a cause or an action from domain, data, interestingness, and deployment aspects.

$$
\begin{aligned}
&/^{*}\textit{Norm grammar}^{*}/\\
&\langle \textit{norms} \rangle ::= \langle \textit{norm\_type} \rangle \langle \textit{object} \rangle^{+} \langle \textit{norm\_formula} \rangle^{+}\\
&\langle \textit{norm\_type} \rangle ::= (\text{Domain} \mid \text{Data} \mid \text{Interestingness} \mid \text{Deployment})\\
&\langle \textit{object} \rangle ::= [\textit{actor}]\ [\textit{cause}]\ [\textit{action}]
\end{aligned}
$$

To represent such constraints, we define some key ontologies, for instance. Prohibition, Permission, Conditional Permission. The following formulas illustrate the representation and aggregation of some particular norms.

— Permission: $N(\alpha)$ indicating that α is permitted to be true.
— Prohibition: $N(\neg\alpha)$ representing it is prohibited that α is true.
— Conditional-Prohibition: $N(\neg\alpha|\beta)$ indicating that α is prohibited to be true if β is permitted to be true.
— Reverse-Prohibition: $N(\neg\alpha) \wedge N(\alpha|\beta)$ indicating that α is prohibited to be true, but α is permitted to be true if it is permitted that β is true.

3.3.2. *Visual models of ORGANIZED members*

The visual modeling of members in an agent organization is to develop diagrammatic notations and specifications to capture and represent ORGANIZED members, their attributes and properties, and relationships between members.

Table 2. Iteration rules.

Iteration	Symbol	Annotation	Informal Statement
While-Loop	⊕	**while(condition)*	Reiterate the subgoal while the "condition" is satisfied
For-Loop	⊗	**for(variable, listOfValues)*	List of values for the variable in the subgoal will be held iteratively
Interrupt	⊘	**whenever(variableList, condition)*	Values for the variables in the subgoal will be held whenever the condition is satisfied
If	⊖	*!if(condition)*	The subgoal will be operationalized if the condition is satisfied
Pick	⊖	*!pick(variableList, condition)*	Values for the variables in the subgoal will be picked non-deterministically that satisfy the condition

There are some AOSE methodologies focusing on developing visual modeling techniques, for instance, MASE.

Typically, static objects in an agent organization can be easily modeled. For instance, as shown in Table 1, actors Agent and Resource are represented by symbols ⓐ and □, respectively. In fact, we can define notations to denote static objects, dynamic objects, and semantic and structural relations among objects in an agent organization. For instance, in an OCAS, a subgoal may be executed iteratively. Iteration links describe under what conditions a subgoal is performed and whether it is done once or repeatedly. The relationship from a super goal to a subgoal is called *Parent-Child Relationship*. There are five types of iterations existing in varying agent organizations: the While-Loop, the For-Loop, the Interrupt, the If, and the Pick. Parent-child relationships exist in many situations. Notations are specified in Table 2 to describe such parent-child relationships. The first three are for iterating the subgoals, and the last two are for non-deterministically picking-up values for variables in the subgoal that satisfy the condition. Moreover, iteration can be notated by a generic formula, so that we can label them on the diagrams.

Definition 3 (Visual Model Building Block). A visual model building block represents an ORGANIZED member, which is described in terms of the following specification:

$$<member> \cdot <member\text{-}class>^* \cdot (<element\text{-}keyword> \cdot <element\text{-}symbol> \cdot <element\text{-}annotation> \cdot <element\text{-}statement>)^+$$

For instance, the iteration rule While-Loop discussed in Table 2 is fully represented as: $R.Iteration.While\text{-}Loop.\oplus.{}^*while(condition)$.[d]

[d]Reiterates the subgoal while the "condition" is satisfied.

OASOC develops systematic visual building blocks to represent the ORGANIZED members in an OCAS. In [6], we present visual models and examples for modeling actors, the roles of actors, interactions between actors, organizational structures, organizational rules, and the environment of an OCAS. Based on these developed visual building blocks, we can model an agent system.

3.3.3. *Descriptive models of ORGANIZED members*

Compared with visual modeling, descriptive or formal modeling usually can depict an object in a rational manner. In the OASOC, besides diagrammatic notations, descriptive specifications are also developed to capture and represent members and their relations in an agent organization. Descriptive specifications are based on temporal logic.

For instance, in an OCAS, every actor has some associated attributes and properties. Some actors take on roles. Agent and Service actors may hold mental states such as *belief*, *desire* and *intention*. Every role is responsible for taking some actions on some target actors. The activity undertaken by a role would be guarded by some properties. The following specification represents an actor.

*/*Actor grammar*/*
<*actor*> ::= <*actor_type*><*role*>$^+$<*attribute*>$^+$<*property*>*
<*actor_type*> ::= (Human | Workspace | Agent | Service | Resource | ...)
<*role*> ::= <dender>[*protocols*][*activities*]<dependee>[*properties*]
<*attribute*> ::= (Constant | [*activities*] | [*state*])
<*activity*> ::= (Permission | Responsibility | [*self-defined*])
<*property*> ::= <*property_type*><*property_mode*>
<*property_type*> ::= (Creation | Inner | Fulfillment | [*self-defined*])
<*property_mode*> ::= Mode (achieve | cease | maintain | avoid | optimize | ...)
<*state*> ::= (Belief | Desire | Intention | [*self-defined*])

In this section, we briefly discuss how ORGANIZED members in an OCAS can be modeled in terms of both visual and descriptive perspectives. We do not present a detailed description of all members. Some of them can be found from [6], their deployment is available from [9].

4. Agent Service Oriented Design

4.1. *ASOD tasks and process*

Agent service-oriented design is an integrated paradigm based on the dialogue between ABC and SOC. Figure 5 shows the major tasks and processes of ASOD-based system design of an OCAS.

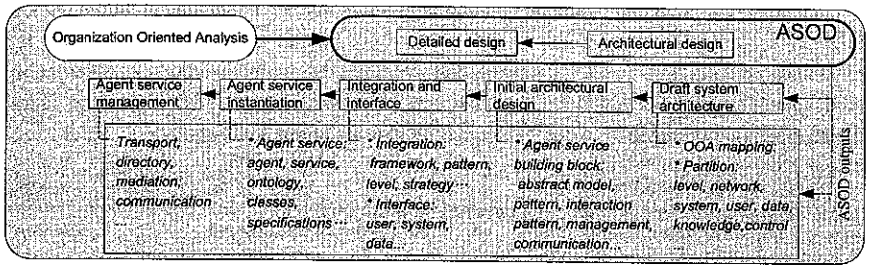

Fig. 5. ASOD tasks and process.

— Drafting system architecture based on understanding of system members and relations gained in OOA.
— Conducting initial architectural design by building up agent service abstract models, design patterns, interaction patterns, agent service management and communication.
— Defining agent service based system frameworks, information integration pattern and integration levels and strategies in terms of extracted agent services.
— Instantiating models built in architectural design into detailed and programmable agent service ontology, classes, specifications and endpoint interfaces.
— Specifying the transport, directory, mediation and communication of agent services.

4.2. *Agent service-oriented architectural design*

Agent service-oriented architectural design develops high-level abstract models, architectural frameworks and design patterns by analyzing and aggregating models developed in the OOA phase to construct agent service based architectural frameworks.

4.2.1. *Agent service abstract model*

Agent service abstract model defines the basic structure of an agent service. It specifies attributes, roles, protocols, activities and properties of an agent service, which describe its intra-agent service activities and inter-agent service communications. Agent service protocols and activities involve inter-agent service communications and interoperability.

An agent service is represented by attributes such as name, type, locator, owner, roles, activity, protocol, address, message, input variables, preconditions, output variables, post conditions and exception handling.

$$<agent\ service> ::= f(Name, Type, Activity, Protocol, Locators, Owner, Roles, Address, Message, InputVariables, Preconditions, \ldots$$

Table 3. Agent service functional patterns.

Agent service	Definition	Context scope
Brokerage	It facilitates the advertising and discovery of agent services.	Usually acting as matchmaking or mediation in SOA.
Negotiation	Two or more parties competitively jointly search a space of possible solutions with the goal of reaching a consensus.	With auction; distributive or integrative negotiation; contract net.
Auction	It specifies bidding or other forms of conventions for negotiating prices and offers based on a mediator.	Auction or auction house; combinatorial auctions.

4.2.2. *Agent service design patterns*

Agent service design patterns provide templates [14, 21] for constructing agent service architectures and functionalities. The architecture specifies a kind of system structures, working mechanisms and classes of agent services. It defines some specific commitments about internal structure and intra-agent service activities of one or a set of agent services.

There are mainly four categories of agent architectures. They are deductive agent architecture, reactive agent architecture, belief-desire-intention agent architecture, and hybrid architecture [33]. They can be abstracted and described in terms of certain architectural frameworks or some symbolic logics. For instance, for a reactive agent, decision-making can be modeled in the following form with a direct mapping from perception P to action A: $\wp(P) \rightarrow \wp(A)$.

In addition, agent services can also be classified in terms of system functionalities. For instance, the following agent service patterns are developed for middlewaring or enterprise application integration: Proxy Service, Wrapper Service, Adapter Service. While agent services such as Matchmaking, Brokerage, Gateway, Negotiation, Auction and Orchestration Engine are used for communications. Table 3 lists the definitions and scopes of three functional agent services.

4.2.3. *Agent-based service-oriented integration*

Agent-based service-oriented integration combines agent-based computing [33] and service-oriented computing [15]. In agent service-based integration, agents consist of main building blocks in the system. While services play significant roles undertaking action-to-be in inter-application communications and integration, etc. In agent service-based integration, applications are packed as agent services, while inter-application communication is based on agent service mediation.

Agent service-based integration can be instantiated into various integration architectures. For instance, in legacy integration, every legacy or enterprise application can be wrapped as a wrapper agent service, or an agent service-based mediator

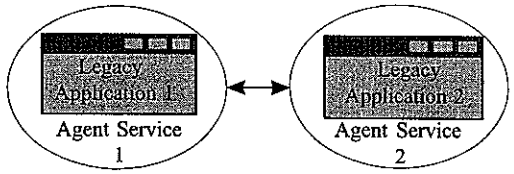

Fig. 6. Agent service-based wrapping.

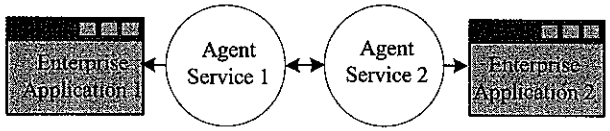

Fig. 7. Agent service-based middleware integration.

can be built between legacy applications. Figures 6 and 7 illustrate two integration architectures.

In addition, in implementing agent service-oriented enterprise application integration, the following issues need to be considered.

— Integration level: integration can be based on presentation, data, method or process.
— Integration type: integration can be based on wrapping, middlewaring, and so on.
— Interface design: agent services may automatically generate interfaces, mirror legacy applications, implement application APIs and specifications programmatically.
— Message passing: defining the communication protocol and message transport way for enterprise application integration. Communication protocol can be HTTP, SOAP, and so on. Agent service communication languages can be KQML, FIPA ACL, or other specific languages. Data can be exchanged as databases or XML-based.
— Event or transaction management: the dispatching, directory, transport, location, mobility, security and transaction support of agent services.
— Exception handling: capturing various exceptions about agent service communications and transaction processing; corresponding solutions should be triggered automatically.

4.3. *Agent service-oriented detailed design*

In the detailed design, models built in architectural design are instantiated into agent service ontologies, specifications, interfaces, the mechanisms for transport, directory, mediation, management and communication of agent services, data model, etc.

4.3.1. *Developing problem-solving ontology*

Agent service problem-solving ontologies specify and manage the attributes and relationships of agent services in a problem-solving system. This involves two steps [7]: (i) extracting problem-solving ontologies from the problem domain, (ii) further developing agent service ontologies for the detailed design.

Problem-solving ontologies can be extracted through the analysis and construction of the conceptual model, extended entity relationship model or the class diagram. These models assist us in capturing attributes and relationships of agent services. Semantic relationships between ontological items can be classified into seven types: Similar_to, Relate_to, Disjoin_with and Overlap_to, Part_of, etc. [7]. Further work is to represent, transform and discover agent service ontologies (see Sec. 4.3.5).

Furthermore, we can develop agent service ontology for the problem-solver. Some basic steps for developing agent service ontologies are as follows:

(i) Extracting actors of agent services and their goals from the problem-solving ontology diagram;
(ii) specifying roles, activities and type of agent services;
(iii) analyzing relevant environment objects of an agent service, and relationships to other agent services;
(iv) designing agent service ontological diagrams to specify and describe all agent services;
(v) refining the developed agent services through reverse re-engineering and scenarios-based analysis; and
(vi) presenting agent service ontology.

4.3.2. *Representation of agent services*

The representation of an agent service is based on agent service ontologies. Each ontological item specifies an attribute or a property of an agent service. There are the following ontological items for agent service abstract model.

— *Activity* (SA) describes the functionality of an agent service;
— *Locator* (SL, RL) of either a sender or receiver indicates the location of an agent service;
— *Owner* (SO) is the one holds an agent service;
— *Role* (SR) is held by an agent service;
— Attributes related to service transportation are *Type* (TT), *Address* (TA) and *Message* (TM);
— *InputVariables* (I) and *OutputVariables* (O) are in/out parameters;
— *Preconditions* (IC) and *Postconditions* (OC) define constraints on executing an agent service;
— *Cardinality* (IO) defines cardinality property on each attribute. They are expressed in the form as key-value pair (KVP); and

— *Exception* (E) defines varied unexpected events, messages, system operations etc.

To present an agent service with the above item atoms, some constraint properties, for instance cardinality, need to be considered. The following illustrates the cardinality constraints on each of the above item atoms.

$$\{\{SA, MO\}, \{SL, MO\}, \{RL, MO\}, \{SO, MO\}, \{TT, MO\}, \{TA, MO\}, \{TM, OM\}, \{I, OM\}, \{IC, OM\}, \{O, OM\}, \{OC, OM\}, \{E, OM\}\} \quad (4)$$

where MO and OM mean one mandatory or multiple optional elements, respectively.

4.3.3. *Agent service interface design*

The quality characteristics (e.g. usability) of an agent service framework and of an agent service-oriented infrastructure to some extent rely on endpoint interface design. Enterprise-wide generic and consistent interfaces further depend on establishing:

(i) responsibilities for designing an agent service interface;
(ii) generic, consistent naming conventions and interface design standards;
(iii) an integration layer providing standard interfaces to disparate applications.

For instance, typical responsibilities for designing an agent service interface include making:

(i) interface standard and naming conventions;
(ii) interface implementation documents;
(iii) message standard and specifications;
(iv) message documents; and
(v) interface consistency, clarity and extensibility.

4.3.4. *Directory and transport of agent services*

Agent service directory provides a location through which agent services register directory entries, and a consistent means by which agent services can discover services. It involves agent directory service [16] and service directory service [16]. An agent registering a directory service is passed with a directory entry. A service registers its service description as service directory entry.

The definition, presentation, location and discovery are some basic problems of agent service directory. The definition is closely related to the representation of directory services. A directory namespace and specifications for defining such directory service are required. The specification of directory services specifies what attributes to be brought into the directory entry for positioning agents and services, and how to represent these attributes in the directory entry. The location of agent services is undertaken through directory services by utilizing the representation and

organization of agent directory service and service directory service. For instance, the following specifies the interface for an agent locator.

```
public interface AgentLocator{
        int hashCode();
        String getType();
        String getAddress();
        setType(String type);
        void setAddress(String address);
}
```

Agent services discovery can be undertaken through querying an agent service with which to communicate or searching for a message. An agent can access agent directory services to discover and locate target agents and services with which the source agents want to communicate. In the process of searching for a destination agent, the agent will technically retrieve the agent/service ontology library and directory services for an agent/service directory entry. This agent/service directory entry encloses transport information which can be parsed to locate the target agents and services according to business rules defined in the agent service directories. Message-based searching can be through matching the queried value with the enclosed property value in the target entry item. However, message content is usually encoded into another format, or packed into another entity. This means the query of message may be based on original message, or encoded message in the whole agent space. The original message may be typed in as a user personification according to particular user profile. While encoded messages may be in the form of another syntactic pattern or semantic specification.

For simplicity, it is possible to only search original messages. To this end, it is required to design a message store structure, so that messages can be stored and managed in terms of original format and business ontology according to user personification. It also supports semantic transformation from user profiles to encoded message ontology.

Agent service transport defines what attributes can be used to describe the transport service, how to represent transport services of agent services, and what transport protocol can be used for the delivery of messages. We need to specify transport interfaces, deliver and process messages, and handle transport exceptions.

4.3.5. *Agent service ontology mapping and transformation*

The ontological items developed and accumulated in the software engineering of OCAS present rich syntactic and semantic information between business requirements and problem solving systems. They can support agent service oriented system design, in particular the representation, directory and transport of agent services.

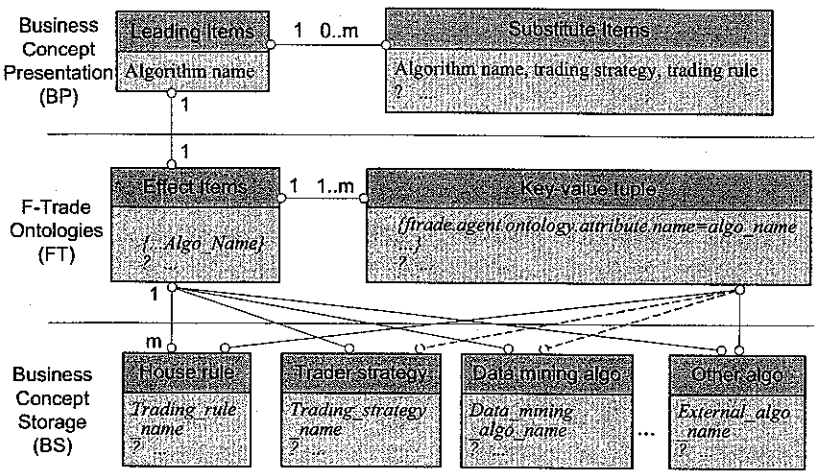

Fig. 8. Ontological mapping between business concepts and F-Trade ontologies.

It is essential to create the mapping relationship between business concepts and corresponding implementation terms. As shown in Fig. 8, in organization-oriented world, there may be more than one term used for expressing the same or similar concept. Such terms, referred to as business concept presentation (BP), for instance, trader's trading strategy, trading house trading rule, data mining algorithm, are original depiction of an agent organization and its functions. To manage the variety of business terms, we classify same or similar concepts into two types of business terms [7]. One is Leading Items that show in user interfaces, the other is Substitute Items can be customized by specific users. For instance, in constructing a problem-solving system like the F-Trade [9], it is necessary to set up a uniform namespace system, namely KVT-based F-Trade ontology (FT), to semantically aggregate same or similar terms of agent services, and map each other. The F-Trade ontology is further mapped to items in a transformed concept system, where various business terms are stored in terms of particular business problem domains.

To support the above mapping, a fundamental work is to develop mapping rules for (i) aggregating ontological items, and (ii) transforming an ontology item to another involved in either one ontological domain or multiple domains. Developing formal rules for ontological aggregation and transformation is vital towards the well-founded and automated interoperability of ontologies across domains. It is necessary to focus on developing formal rules to support transitivity, additivity and antisymmetry in transforming and reducing semantic relationships [29,7] and ontological items. By contrast, the aggregation of semantic relationships is necessary for further transformation of ontological items. It can simplify the combination of semantic relationships, and find the final reduced semantic relationship.

Relevant ontological rules are developed to [7] support the semantic aggregation of the above semantic relationships. For instance, Rule 1 shows an excerpt for some cases.

Rule 1. Let ontological items A, B and C be associated by the Generalization relationship *is_a*, then there are rules:

$$\forall\, (is\_a(A,B) \wedge is\_a(B,C)) \Rightarrow is\_a(A,C) \tag{5}$$

$$\forall\, (is\_a(A,B) \wedge is\_a(A,C)) \Rightarrow is\_a(A,(B \text{ AND } C))\,. \tag{6}$$

The operator *is_a* is defined as follows.

Definition 4. $is\_a(O_1,O_2)$ relationship: the relationship between O_2 and O_1 is *subtype/supertype* or as *subsumption*, i.e. O_2 *is_a* O_1, or O_2 is a kind of O_1. The *is_a* sometimes is also called *subclass_of*.

These sample rules show that rule-based aggregation and transformation make it easier to understand the relations between multiple ontological items and integrate semantic relationships. Furthermore, we need to develop semantic rules for aggregating and transforming ontological items in one or across multiple namespaces. For instance, Rule 2 shows an aggregation of ontological items, Rule 3 presents an example of transformation of ontological items.

Rule 2.

$$\forall (A \text{ AND } B), B ::= is\_a(A,B) \Rightarrow B, \text{ the resulting output is } B \tag{7}$$

Rule 3.

$$\forall C_i, C_j, (i \neq j), \exists : (is\_a(O,C_i) \wedge is\_a(O,C_j)) \Rightarrow O\,. \tag{8}$$

4.3.6. *Mediation of agent services*

The mediation and management [16] of agent services is very important for proper work of large agent service systems. With the organizational metaphor, an agent space can be viewed as consisting of multiple functional centers at certain granularities in the system, where relevant agent services work together to fulfill actions or support services both within and across these centers.

Mediation of agent services in an agent society can be at two levels. One is to support local mediation in a sub-system; another is to provide global mediation in the agent society. For these objectives, multi-tier mediation strategies may be useful to deal with the management at different levels; mediation protocols must be considered in the negotiation, mediation logic must be clarified from the ask request to the reply response.

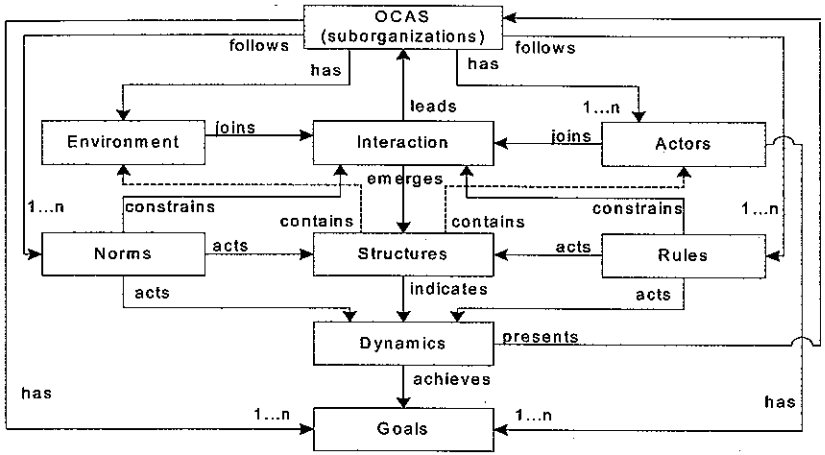

Fig. 9. Relations between ORGANIZED members.

5. Connections and Transitions in OASOC

Connections and transitions in the OASOC consist of those between phases and between models. An additional issue is the transition from OOA to ASOD. They actually involve (i) inherent connections between the ORGANIZED members, and (ii) connections between OASOC phases.

5.1. *Connections in ORGANIZED members*

There are internal links between system members in an OCAS, which reflect the system complexities of OCAS. Correspondingly, such links are embodied in terms of relationships between the ORGANIZED members.

For instance, an OCAS organization obeys certain norms and organizational rules in forming organizational structures based on interaction between actors and between actors and their environment. Figure 9 presents the relationships between the ORGANIZED members in an OCAS system.

5.2. *Connections in OASOC phases*

There are two levels of connections in the OASOC. One is inter-phase connections between major AOSE phases. The other is intra-phase connections between modules in a specific phase. Figure 1 shows top-down processes from OOA to ASOD, which instantiate and detail system analysis and design process step by step up to a coding system. The decomposition starts from goal and business understanding, objective and behavior definition, and is then followed by organizational members and their relationships. Further, agent-based service-oriented architecture, design

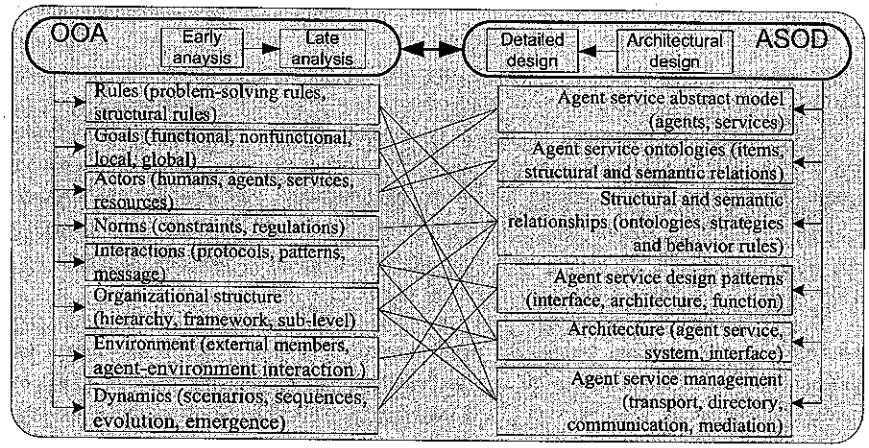

Fig. 10. Transition between OOA and ASOD in the OASOC.

patterns, activity and sequence relationships are built to manage organizational goals and behavior. Finally, sets of agent services are developed to implement the system-to-be.

In addition, due to system complexities of the OCAS, bottom-up retrospect, refinement and tuning of relevant models, components and their relations from implementation to ASOD and OOA are necessary. They can check the logics and potential conflict between layers and models. The refinement and update may be conducted by tuning architecture design and detailed design based on the rethinking of early and late requirements. In particular, organizational members may be updated and re-modelled on demand.

In particular, the connections between OOA and ASOD are directly embodied through the mapping between the outputs from OOA and ASOD. Figure 10 illustrates such dialogue between individual findings in the OOA and their counterparts in the ASOD. For instance, the problem-solving and structural rules developed in the OOA are converted and instantiated in terms of structural and semantic relationships in developing agent service ontologies, resource and application integration strategies, and behavior rules of agent services. In addition, they are also embodied in terms of the mechanisms for transport, directory, communication and mediation of agent services.

6. Related Work and Discussions

There are a few of AOSE approaches available for dealing with complex autonomous, adaptive or human-agent-cooperated agent systems. GAIA explicitly presents organizational abstraction, analysis and design, and basically focuses on late analysis and conceptual modeling. MaSE utilizes Use Case for implicit structure

analysis. MESSAGE exploits organizational metaphor, and defines an organization as a structure with topological relations. ROADMAP extends GAIA with environment, role hierarchies, social structure and relationship and dynamic changes. TROPOS was proposed in terms of goal-oriented analysis, but is currently converted towards the OOM and formal Tropos. It covers full phases of AOSE with nice building blocks. SODA addresses some of the shortcomings of Gaia, explicitly takes the agent environment into account, and provides engineers with specific abstractions and procedures for the design of agent infrastructures. In OperA, a language for contract representation, LCR, based on deontic temporal logic, is proposed as a formal theory for describing interaction in agent systems.

With regard to system design, the above-listed AOSE approaches except SODA present varying building blocks for architectural and related detailed design. On the other hand, service and SOC are increasingly involved in agent and ABC, for instance integrating Web Service (WS) into Agentcities [36] and developing DAML-S for autonomous semantic Web service. Some of existing work purely relies on WS to construct agent systems, which lose some intrinsic characteristics and strengths of agents such as autonomy and user interaction.

For the particular job of engineering OCAS-like systems, the existing approaches to varying degree provide nice supports. However, none of them are powerful enough to handle the system complexities in OCAS. For instance, there is no support available from them to model system dynamics, the concept of service is not strongly embodied in system design. As a trial of integrating OOM, ABC and SOC, OASOC aims to cover all major system members in OCAS through the ORGANIZED framework. In fact, not all of them are supported in existing approaches.

Based on our knowledge and empirical evaluation, the following presents one of comparative views of the above approaches in terms of: (i) which software engineering phases are supported, (ii) what ORGANIZED members are modeled. Figure 11 illustrates the comparison of the above mentioned approaches in terms of main software engineering (SE) phases: EA (early analysis), LA (late analysis), AD (architectural design), DD (detailed design), and I (implementation). The bars indicate that more or less the corresponding aspects can be supported in the respective AOSE approaches.

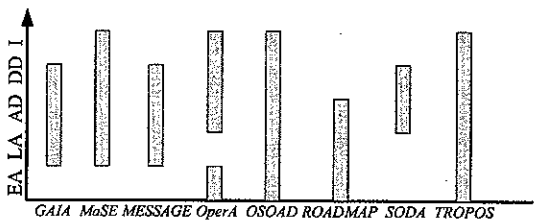

Fig. 11. Comparison with existing AOSE approaches in terms of engineering lifecycle.

7. Conclusion and Future Work

The engineering of open complex systems is important and challenging. An effective manner is to integrate the strengths of existing computing paradigms such as agent-based computing, service-oriented computing and organizational modeling. With such thoughts in mind, we have introduced an integrative computing approach, referred to as *Organization, Agent, and Service Oriented Computing* (OASOC), for the engineering of open complex systems. Compared with the existing approaches, OASOC has integrated and complemented the strengths of organizational modeling, ABC with SOC toward an integrative paradigm. It provides the ORGANIZED framework and corresponding analysis and design techniques for capturing and modeling all major complexities in an open complex system. The integration of ABC and SOC is helpful for designing large-scale interoperable intelligent software.

OASOC has been successfully used for engineering F-Trade. It is an online testbed supporting the development of trading agents, trading strategies for financial trading support, and exceptional behavior analysis for market surveillance. Our further work is on introducing social computing into the problem-solving of open complex systems.

Acknowledgments

Thanks are due to Prof. Chengqi Zhang at UTS, and reviewers and editors for their comments on this paper.

References

1. J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley, KAoS: Toward an industrial-strength open agent architecture, *Software Agents*, 1997.
2. E. Broek, C. M. Jonker, A. Sharpanskykh, J. Treur, and P. Yolum, Formal modeling and analysis of organizations, *OOOP05*, 2005.
3. G. Caire *et al.*, Agent-oriented analysis using message/UML, Lecture Notes in Computer Science, Vol. 2222, Springer Verlag, New York, 2002, pp. 119–135.
4. L. B. Cao, Issues in agent-based open giant intelligent systems, PhD thesis, Chinese Academy of Science, 2002.
5. L. B. Cao and R. W. Dai, *Open Complex Intelligent Systems* (Post & Telcom Press, 2008).
6. L. B. Cao, C. Q. Zhang, and R. W. Dai, Organization-oriented analysis of open complex agent systems, *Int. J. Intelligent Control and Systems* **10**(2) (2005) 114–122.
7. L. B. Cao *et al.*, Ontology-based integration of business intelligence, *Int. J. Web Intelligence and Agent Systems* **4**(3) (2006) 313–325.
8. L. B. Cao and C. Q. Zhang, F-Trade: An agent-mining symbiont for financial services, *AAMAS2007*, 2007.
9. L. B. Cao, C. Q. Zhang, and M. Zhou, Engineering open complex agent systems: A case study, *IEEE Trans. SMC*, Part C.
10. K. M. Carley, Computational and mathematical organization theory: Perspective and directions, *Computational and Mathematical Organization Theory* **1**(1) (1995) 39–56.
11. K. M. Carley, Artificial Social Agents. http://www.hss.cmu.edu/departments/sds/faculty/carley/publications.htm.

12. A. Cheyer and D. Martin, The Open Agent Architecture, *Autonomous Agents and Multi-Agent Systems* (Springer, 2001).
13. L. Cholvy and C. Garion, Desires, norms and constraints, *AAMAS04*, 2004, pp. 722–729.
14. T. T. Do, M. Kolp, and A. Pirotte, Social patterns for designing multi-agent systems, *SEKE03*, 2003.
15. T. Erl, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services* (Pearson Education, 2004).
16. FIPA, Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/bysubject.html.
17. A. Garcia *et al.* (eds.), *Software Engineering for Large-Scale Multi-Agent Systems* (Springer-Verlag, 2003).
18. F. Giunchiglia, J. Mylopoulos, and A. Perini, *The Tropos Software Development Methodology: Processes, Models and Diagrams*, Technical Report No. 0111-20, ITC–IRST, 2001.
19. T. Juan *et al.*, ROADMAP: Extending the GAIA methodology for complex open systems, *AAMAS02*, ACM, 2002, pp. 3–10.
20. H. Jung and M. Tambe, Performance models for large scale multiagent systems: Using distributed POMDP building blocks, *AAMAS2003*, 2003.
21. J. Lind, Patterns in agent-oriented software engineering, *AOSE2002*, Springer, 2002.
22. J. J. Odell, H. V. D. Parunak, M. Fleischer, and S. Brueckner, Modeling agents and their environment. The physical environment, in *Journal of Object Technology* **2**(2) (2003) 43–51.
23. A. Omicini, SODA: Societies and infrastructures in the analysis and design of agent-based systems, *AOSE'2000*, 2000, pp. 185–193.
24. OperA: http://igitur-archive.library.uu.nl/dissertations/2003-1218-115420/inhoud.htm.
25. X. S. Qian, J. Yu, and R. Dai, A new field of science — open complex giant systems and their methodology, *Nature Magazine* **13**(1) (1991) 3–10.
26. X. S. Qian, *Building Systemtology* (Shanxi Science and Technology Press, 2004).
27. Selmas Workshop, 2003–2007.
28. M. P. Singh and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes and Agents* (John Wiley & Sons, 2005).
29. V. C. Storey, Understanding semantic relationships, *The Very Large Data Bases Journal* **2**(4) (1993) 455–488.
30. G. Tidhar, *Organization-Oriented Systems: Theory and Practice*, PhD thesis, University of Melbourne, Australia.
31. M. M. Waldrop, *Complexity: The Emerging Science at the Edge of Order and Chaos* (Simon & Schuster, 1992).
32. M. Wood, S. A. Deloach, and C. Sparkman, Multiagent system engineering, *Int. J. Softw. Eng. Knowl. Eng.* **11**(3) (2001) 231–258.
33. M. Wooldridge, *An Introduction to Multiagent Systems* (John Wiley & Sons, 2002).
34. F. Zambonelli, N. R. Jennings, and M. Wooldridge, Developing multiagent systems: The GAIA Methodology, *ACM Trans. on Software Engineering and Methodology* **12**(3) (2003) 317–370.
35. F. Zambonelli and A. Omicini, Challenges and research directions in agent-oriented software engineering, *J. Autonomous Agents and Multi-Agent Systems* (Springer, 2004).
36. Agentcities: http://www.agentcities.net/.