Situation-Awareness for Adaptive Coordination in Service-based Systems

S. S. Yau, D. Huang, H. Gong, and H. Davulcu
Arizona State University,
Tempe, AZ 85287-8809, USA
{yau, dazhi.huang, haishan.gong, hasan.davulcu}@asu.edu

Abstract

Service-based systems have many applications, including collaborative research and development, e-business, health care, environmental control, military applications, and homeland security. Service coordination is required for these systems to coordinate distributed activities. To achieve adaptive service coordination under changing environment and workload, situation-awareness is needed. In this paper, a model is presented for situation-awareness (SAW) requirements in service-based systems. Based on this model, SAW agents are developed to incorporate situation-awareness and adaptive coordination in service-based systems.

Keywords: Situation-awareness requirements, service-oriented architecture, adaptive service coordination, situation-awareness agents, service-based systems.

1. Introduction

Service-based systems have the major advantage of enabling rapid composition of distributed applications, regardless of the programming languages and platforms used in developing and running the applications. Service-Oriented Architecture [1] has been adopted in many distributed systems, such as Grid and Global Information Grid (GIG) [2], in various application domains, including collaborative research and development, e-business, health care, environmental control, military applications and homeland security. In these systems, various capabilities are provided by different organizations as *services* and interconnected by various types of networks. We consider a *service* as a software/hardware entity with well-defined interfaces to provide certain capability over wired or wireless networks using standard protocols, such as HTTP and SOAP (Simple Object Access Protocol). The *services* can be integrated following specific *workflows*, which are series of cooperating and coordinated activities designed to achieve users' goals. *Service coordination* is required to ensure the correctness of workflow execution. S*ervice coordination* is a process of locating participant services, monitoring their status, invoking proper services, and propagating necessary information among them to ensure the correct results obtained from the coordinated participant services. In service-based systems, service coordination needs to be *adaptive* because (1) services may be unavailable or cannot provide desirable QoS due to distributed denial-of-service attacks, system failures or system overload, (2) workflows may need to be adapted when the situation changes in order to satisfy the requirements, and (3) new workflows may be generated in runtime to fulfill users' new requirements.

To achieve *adaptive* service coordination, *situation-awareness (SAW)*, which is the capability of being aware of situations and adapting the system's behavior accordingly [3, 4], is needed for checking whether a service can be invoked and adapting new workflows. A *situation* is a set of contexts in the application over a period of time that affects future system behavior [3, 4]. A *context* is any instantaneous, detectable, and relevant property of the environment, the system, or users, such as location, available bandwidth and a user's schedule.

In this paper, we will present a model for SAW in service-based systems. Based on this model, we will develop SAW agents to incorporate SAW and adaptive coordination in service-based systems.

2. Current State of the Art

Situation-awareness has been studied in artificial intelligence [5], human-computer interactions [6] and data fusion community [7]. Situation Calculus and its extensions [8-11] were developed for describing and reasoning how actions and other events affecting the world. A situation is considered as a complete state of the world and cannot be fully described, which leads to the well-known frame problem and ramification problem [9]. A core SAW ontology [12, 13] refers a situation as a collection of situation objects, including objects, relations and other situations. However, it does not address how to verify the specification and perform situation analysis.

Substantial work has been done on service coordination [14-18]. Industrial standards, such as WS-Coordination [14] and WS-CF [15], aim at providing standard and extensible coordination frameworks to support coordinated workflows on web services, but do not provide techniques for achieving adaptive service coordination. To coordinate distributed systems, a formal specification framework [16] was developed for modeling dynamically changing contexts and rules in reactive systems. MARS [17] promotes context dependent coordination by incorporating programmable coordination media in distributed systems. EgoSpaces [18] introduced a coordination model and a middleware for specifying and managing agent-centered contexts to facilitate easy application development in mobile ad hoc environments. These approaches only use current contexts in service coordination and do not consider variations of contexts over a period of time, which are important information for service coordination.

3. Our Approach

In this section, we will present our approach to incorporating SAW for adaptive coordination in service-based systems. It consists of two major parts:

(1) Modeling and specifying SAW for adaptive service coordination.
(2) Developing SAW agents for adaptive service coordination.

It is assumed that there is a mission planner (MP) [20, 21] or its equivalent in a service-based system, which accepts goals specified by users and generates execution plans based on available services and current situation. The generated execution plan is a series of service compositions to be executed in order to fulfill the overall goal. A step (service invocation) in the execution plan may have certain dependencies on situations, i.e., a step can be executed only when a certain situation is detected. The execution plan can be decomposed [19] and delivered to SAW agents for execution. Figure 1 depicts the interactions among services, SAW agents, and the MP.

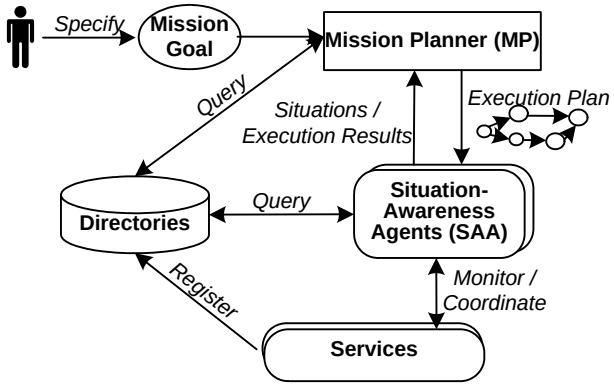

Figure 1. SAW agents for adaptive service coordination

An MP must be able to perform planning based on partial domain information, i.e. the MP does not know all the information related to the execution environment and services at the time of planning. This requirement for MP is needed because there is usually no central control on adding/removing services in service-based systems and services may be unavailable without notifying users. This requirement makes traditional planners, such as TAL planner [20] unsuitable. Instead, planners with CTR-S [21] or other adaptive workflow synthesis techniques [22] should be used. However, a consequence of planning with partial domain information is that although the generated workflows are always sound at the time of planning, these workflows might be non-executable during their execution due to dependency violations in a service invocation caused by situation changes by uncontrollable external agents. Due to this difficulty, we develop SAW agents to coordinate the services in the execution plan. In the following subsections, we will present our model for SAW and the design of SAW agents.

3.1. Modeling and Specifying SAW for Adaptive Service Coordination

We consider a *service* as a process, which can accept inputs from other processes and produce outputs. Hence, a *service-based system* can be considered as a collection of parallel processes, each of which can send/retrieve data to/from other processes. Consequently, the service coordination in such a system becomes the coordination of these parallel processes.

Before we present the SAW agents, we need to model SAW for adaptive service coordination, which includes

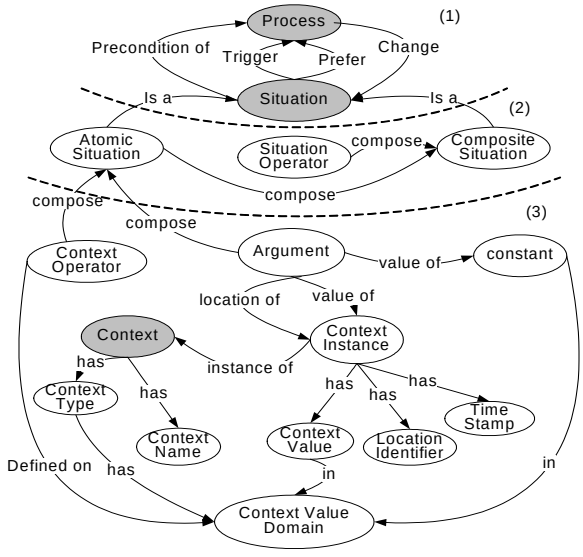

Figure 2. A conceptual view of our model for SAW for adaptive service coordination

two aspects: (1) modeling situations, and (2) modeling the relations between situations and processes.

Figure 2 shows a conceptual view of our model. Since context acquisition and operations on contexts are highly domain-specific and often involve low-level system processes, our model will not include the ways contexts are collected and the semantics of operations on contexts. Instead, we assume that each context is collected periodically by invoking at least one service in a service-based system, and that a service that can collect a context also implement operations for preprocessing this context.

Def. 1: A *context* is a measurable property of the environment, the system or users:

- c_i is the unique name of a context
- $\tau_i = contextType(c_i)$, the context type of c_i.
- $D_i = domainOf(\tau_i)$, the value domain of τ_i.

Def. 2: A *context instance* I_{ix} of the context c_i is a quintuple $(c_i, \tau_j, v_k, t_x, l_m)$, where $\tau_j = contextType(c_i)$, $v_k \in domainOf(\tau_j)$, t_x is a timestamp, and l_m is a location identifier. Given $I_{ix} = (c_i, \tau_j, v_k, t_x, l_m)$, the following two functions are defined: $valOfIns(c_i, t_x) = v_k$, which returns the value of a context at a particular time, and $locOfIns(c_i, t_x) = l_m$, which returns where the context is measured.

Def. 3: An *argument arg* is one of the following:

- a constant value in a context value domain D_i
- a variable ranging over a context value domain D_i
- $valOfIns(c_i, t)$, in which t is a time variable
- $locOfIns(c_i, t)$, in which t is a time variable

An *arg* is *bounded* if it is a constant, a variable with a value v_x ($\in D_i$) assigned to it, or the return value of $valOfIns(c_i, t)$ or $locOfIns(c_i, t)$ at a given time stamp t.

Def. 4: Given a set of arguments, $\{arg_1, \ldots, arg_n\} \subseteq D_i$, two types of *context operators* are defined as follows:

- Boolean operators: $op_i(arg_1, \ldots, arg_n) = true \mid false$
- Value operators: $op_j(arg_1, \ldots, arg_n) = v \in D_i$

Def. 5: A *term* is either an application of a context operator, $op(arg_1, \ldots, arg_n)$, or a nested application of context operators, $op(term_1 \mid arg_1, \ldots, term_n \mid arg_n)$.

Def. 6: An *atomic situation*, aS_i is a term which returns boolean values. It can be expressed as follows: $aS_i(x_1, \ldots, x_m) \equiv op(arg_1, \ldots, arg_n) \mid op(term_1 \mid arg_1, \ldots, term_n \mid arg_n)$, where *op* is a boolean operator, and $x_1, \ldots, x_m$ are all unbounded arguments of *op*.

Def. 7: A *composite situation*, cS_i, is defined as follows:
1) $cS_i \equiv aS_x$; 2) $cS_i \equiv \neg cS_x \mid cS_x \wedge cS_y \mid cS_x \vee cS_y$; 3) $cS_i \equiv P(cS_x, \omega, \varepsilon)$: cS_x was true sometime within $[now-\omega, now-\omega+\varepsilon]$; 4) $cS_i \equiv H(cS_x, \omega, \varepsilon)$: cS_x was always true within $[now-\omega, now-\omega+\varepsilon]$; 5) $cS_i \equiv Know(cS_x, \varphi)$: The process φ knows that cS_x is true; 6) All composite situations are defined by recursively applying (1) – (5). $\neg$, $\wedge$, $\vee$, *P*, *H*, and *Know* are *situation operators*.

Def. 8: Let s_0 and s_1 be situations, and σ and φ processes. Five basic relations between situations and processes are defined as follows:
1) $precondition(\varphi, s_0)$: s_0 must be true when φ can be executed. 2) $do(\varphi, s_0, s_1)$: The execution of φ makes s_1 true when s_0 is true. 3) $trigger(\sigma, \varphi, s_0)$: When σ knows that s_0 is true, σ triggers φ. This relation models the *reactive behaviors* of processes. 4) $tell(\sigma, \varphi, s_0)$: σ sends φ the information about s_0. This relation models the *knowledge sharing* between processes. 5) $prefer(s_0, \sigma, \varphi)$: When s_0 is true, it is preferable to use σ instead of φ. This relation models the *preferences* on the usage of processes.

Definition 9: A model M for SAW in a service-based system is a tuple $(C, T, L, S, \Phi, R, CH)$, where *C* is the set of definitions of the contexts in the system, *T* is the set of timestamps appeared in the system since the system started to run, *L* is the set of possible location identifiers in the system, *S* is the set of definitions of the situations in the system, Φ is the set of processes in the system, *R* is the set of relations between situations and processes defined in the system, and *CH* is the context history, which consists of instances of the contexts in the system.

M is an abstract model since the actual representation of *T* and *L*, and the size of *CH* depends on the system to be modeled.

Our model for SAW in service-based systems has strong expressive power because of the following reasons:

- ❖ Based on Defs 4 and 7, our model can capture temporal relations among instances of contexts.
- ❖ Based on Def. 8, our model allows service providers and developers to define the situations that trigger, allow or prohibit the execution of processes in the service-based system.
- ❖ Based on Def. 8, our model allows users to express their preferences on the usage of services.
- ❖ Using the five basic relations in Def. 8, our model allows modeling control structures, which are commonly used in service coordination.
- ❖ Our model can be used to express the situation that timestamped common knowledge [23], which is very important for the coordination of distributed processes, is attained among distributed processes.

The following are the properties of our model *M*, which are useful in developing the SAW agents:

P1) Given *M* for SAW in a service-based system, the definition of a situation s_0, and a timestamp t_0 in *T*, the question "Does *M* satisfy s_0 at t_0?" is decidable.

P2) Given *M* for SAW in a service-based system, and the definitions of situations s_0 and s_1, the question "Does s_0 implies s_1?" is decidable.

P3) Given *M* for SAW in a service-based system, the definition of a situation s_0, and the fact that M has not satisfied s_0 since the system started to run, the question "Is it possible that *M* will satisfy s_0 sometime

in the future?" is decidable if the contexts involved in the definition of s_0 have finite context value domains.

Based on our model, a formal specification language can be derived to specify SAW in service-based systems. The specifications will be used by the MP to generate suitable execution plans. As a proof-of-concept, we will show an example in Section 4 using Transaction F-Logic [24] to specify SAW requirements.

3.2 Design of SAW Agents

SAW agents are distributed autonomous software entities, which should have the following capabilities to support situation analysis and service coordination:

C1) **Participant service management**. SAW agents should be able to accept "join" or "leave" requests from participant services, monitor their status, select and invoke appropriate services when needed, and report their status to other agents or the MP.

C2) **Agent discovery**. An SAW agent should advertise its existence and allow other agents or MP to query its configuration, including participant services managed by the agent, and context and situation information provided by the agent. This is necessary for the cooperation of multiple SAW agents and the MP to support situation analysis and service coordination.

C3) **Context acquisition and situation analysis**. An SAW agent should collect contexts from its participant services and analyze situations continuously based on its configuration.

With these capabilities, SAW agents can adaptively coordinate services in execution plans as follows:

(1) In each step of execution, SAW agents check whether all the dependencies on situations are satisfied.

(2) If the dependency on a situation is not satisfied, SAW agents will check whether the step can be undone.

(3) If the step is undoable, SAW agents will first undo the step and then search for an alternative service.

(4) If an alternative service is found, SAW agents will resume the execution using the alternative service. Otherwise, SAW agents will notify MP to do the re-planning to find an alternative workflow.

Multiple SAW agents in a service-based system will form a hierarchy to process situation information and coordinate workflow execution. In this hierarchy, the agents in lower levels often do not have direct interactions with users, and perform some simple tasks, such as collecting contexts, recognizing atomic situations, or controlling the invocation of one or several services. The agents in higher levels usually have more interactions with users and need to perform more complex tasks, such as recognizing composite situations, discovering other agents and services for coordinating the workflow execution, and adapting workflows. Hence, agents at higher levels need to be reconfigured more often than the agents in lower levels. Based on this, two types of SAW agents are used in the system to achieve a balance between reconfigurability and performance:

(I) **SAW agents with internal knowledge bases**. An SAW agent with an internal knowledge base stores the definitions of contexts and situations, and the relations between situations and processes in its knowledge base. Such an SAW agent performs situation analysis, and determines processes to be triggered by reasoning with knowledge in its knowledge base. Depending on the formal language based on our model, a corresponding inference engine can be used to provide reasoning support. For example, if we choose Transaction F-Logic to specify SAW in a service-based system based on our model, Flora-2 [25] can be used.

This type of SAW agents can be easily reconfigured by updating its internal knowledge base, and can support the adaptation of workflows. However, such SAW agents are heavyweight and it is difficult to maintain consistency of the knowledge bases of multiple agents, especially when the number of agents is large. Hence, we only use this type of SAW agents in higher levels of the hierarchy, where only a few high level agents with more frequent reconfiguration are expected.

(II) **SAW agents as finite state machines (FSM)**. In [26], agents are modeled as FSMs, and they are used to monitor certain variables, change their states based on the update values of monitored variables, and output some variables monitored by other agents during the transition from one state to another. It can be easily shown that situations defined by our model can be detected by FSMs:

- An atomic situation can be detected by a two-state machine as shown in Figure 3.

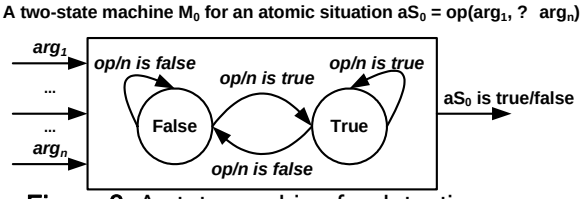

Figure 3. A state machine for detecting an atomic situation

- A composite situation can be detected by a more complex FSM generated by combining FSMs for detecting the atomic situations.

This type of SAW agents is lightweight, does not require maintenance of their internal knowledge bases, and can be automatically generated from the model of SAW. However, the reconfiguration of such agents is difficult. In order to improve the performance of the system, we use this type of SAW agents in lower levels of the hierarchy, where less reconfiguration is expected.

There is no clear boundary in the hierarchy of SAW agents indicating which types of agents should be used at what level. Further investigation is needed to analyze the

tradeoff between reconfigurability and performance, and develop design methods for generating optimal or sub-optimal designs of service-based systems using SAW agents for adaptive service coordination.

4. An Example

To illustrate our approach, consider the following "ship rescue" example: A passenger ship (*BS*) with 200 passengers on board has trouble and may be sinking. There are two nearby ports *P1* and *P2*, and each port has a hospital. There are two rescue ships, *A* and *B* at *P1* and *P2* respectively. Both ships have enough capacity to hold 200 passengers, but only *B* has a helicopter *H* on-board. A rescue center *RC* is responsible for coordinating rescue operations. Five services are involved in this example:

- $\mathbf{S_{BS}}$: the service on *BS* for sending out SOS signal with various context data, including the number of life-threatening injuries.
- $\mathbf{S_H}$: the rescue service which controls the helicopter *H* on *B* to perform rescue operations.
- $\mathbf{S_A}$: the rescue service which controls *A* to perform rescue operations.
- $\mathbf{S_B}$: the rescue service which controls *B* to perform rescue operations. $\mathbf{S_B}$ may invoke $\mathbf{S_H}$.
- $\mathbf{S_{RC}}$: the service at the rescue center *RC* which receives information from other services, analyzes the received information and plans rescue operations.

Upon detecting the situation "*An SOS signal is detected from ship BS*", $\mathbf{S_{RC}}$ automatically generates a plan to perform necessary rescue operations for *BS*. At first, there is no injury report about the passengers from *BS*. $\mathbf{S_{RC}}$ discovers the nearby rescue services ($\mathbf{S_A}$ and $\mathbf{S_B}$), and selects $\mathbf{S_A}$ to perform the rescue mission since *A* is closer to *BS*. However, after *A* is on the way to ***BS***, another report from *BS* comes indicating several life-threatening injuries caused by an explosion just happened on ***BS***. ***A*** is unable to reach *BS* and return to *P1* fast enough to save the injured passengers. Hence, $\mathbf{S_{RC}}$ finds $\mathbf{S_B}$ and $\mathbf{S_H}$ to rescue the injured passengers on *BS*.

(F1) rescued(passengerShip) ← rescueShip ∧ *Dependencies* ∧ *GOAL*

Depedencies : [detectSOS(passengerShip,Now)] → A ∨ B
∧ [lifeThreaten(passengerShip,Now)] → B
∧ [withinRange(rescueShip,passengerShip,port, Now)] → B.helicop

GOAL : [detectSOS(passengerShip,Now)] → rescueShip.load(passengerShip.passengers)
∨ [lifeThreaten(passengerShip,Now)] → (rescueShip.load(passengerShip.passengers)
∧ rescueShip.helicop.load(passengerShip.lifeThreaten_passengers))

(F2) wf_1 ← A.moveTo(BS.shipLoc) ⊗ ((¬[lifeThreathen(BS, Now)] ⊗ A.load(BS.passengers))
∨ ([lifeThreaten(BS, Now)] ⊗ rescueShip))

(F3) wf_{new} ← wf_1 ∧ ¬A

(F4) wf_2 ← B. moveTo(BS.shipLoc) | ([withinRange(B, BS, P1, Now)] → B. helicop.flyTo(BS.shipLoc))
⊗ B.helicop.load(BS.lifeThreaten_passengers) ⊗ B.load(BS.passengers)

Figure 4. CTR-S formulas for workflow generation (F1) and generated workflows (F2-F4)

We will use Transaction F-Logic to specify the SAW requirements in the example. Due to the limited space, only parts of the specifications are shown here:

```
/* ontology for context classes*/
ship[status(t)*=>string, shipLoc(t)*=>location].
passengerShip::ship[ life_threaten(t)=>integer,
                     passengers=>> passenger,
                     lifeThreaten_passsengers=>>passenger].
rescueShip::ship[helicop=>helicopter,
                     load(passengers)=>boolean, …].
… …
/* Rules for situations*/
/* BShip is at helicopter's reachable range*/
withinRange(RShip, PShip, Port, Time):-
        RShip:rescueShip, PShip:passengerShip,
        RShip[helicop→H], H[range→D],
        distance(RShip, PShip, Time, D1),
        distance(PShip, Port, Time, D2),  D1 + D2 < D.
… …
/* Facts */
BS:passengerShip [status(0)→"Normal", injury(0)→0,
                     life_threaten(0)→0, … ].
… …
BS:passengerShip [status(68)→"SOS ", injury(68)→38,
                     life_threaten(68)→5, … ].
… …
```

❖ **Situation-triggered mission planning**

Two SAW agents, a rescue center agent (RC agent) and a ship agent were developed to monitor S_{RC} and ships (*A*, *B, BS*) respectively. Each of them maintains its internal knowledge base storing collected contexts. The RC agent periodically checks the situation "*An SOS signal is detected from a passenger ship*" by querying *detectSOS(PShip, Now)* from its knowledge base. Whenever the result is true, the RC agent will send the situation and PShip's information to the MP to trigger a workflow planning process.

We assume that the MP uses CTR-S for workflow planning as shown in Figure 4. The CTR-S formula F1 states that a rescue ship service (*rescueShip*) that satisfies all dependencies (*Dependencies*) and the goal (*GOAL*) should be identified and scheduled. In this example, two rescue ship services (*A* and *B*) are available.

F1 is used for workflow generation. F2, F3 and F4 are the generated workflows. The *Dependencies* in F1 indicates that when SOS signal is detected, either *A* or *B* could be used to rescue the passenger ship. If there are passengers have life threatening juries, *B* should be used. Helicopter on-board could only be sent out

when *BS* is within *H's* reachable range. The *GOAL* indicates that use a rescue ship to load passengers if no life threatening injury is detected. Otherwise, use a rescue ship with helicopter on-board to load injured passengers.

F2 shows the initial workflow, in which *A* needs to move to the location of *BS* first, and load the passengers on *BS* if no passenger has life threatening injury. Otherwise, another rescue ship (with helicopter on-board) needs to be used.

❖ **Workflow execution**

Workflow defined by F2 is sent to the ship agent, which monitors the two rescue ship services (*A* and *B*). The ship agent finds *A* and invokes its *moveTo* action. At the same time, the ship agent periodically queries whether there are life threatening injuries detected in *BS*.

❖ **Workflow adaptation**

When a ship agent detects that several people have life threatening injuries, the ship agent will undo the *A's moveTo* action and try to adapt the workflow without the help of MP by finding an alterative service that can be used under current situation and resuming the execution of the remaining workflow. F3 will be used by the ship agent to find such an alternative service. F4 corresponds to such an adapted workflow: *B* will move to *BS*. When *B* gets into its *H's* reachable range, *H* will fly to *BS* and load the life-threatening passengers. *B* will load the remaining passengers after reaches *BS*. If such an alternative service cannot be found, the ship agent will notify MP with the situation violation, and MP will do re-planning from the current situation.

5. Conclusions and Future Work

In this paper we have presented an approach to incorporating SAW for adaptive coordination in service-based systems. A model for SAW and the design of SAW agents based on the model to enable adaptive coordination in service-based systems have been presented. Future work in this direction includes incorporation of other QoS, such as security and real-time, in the service coordination, and development of software tools for SAW agent specification, verification, generation and deployment, as well as support for agent mobility.

Acknowledgment

This work is supported by the DoD/ONR under the Multidisciplinary Research Program of the University Research Initiative, Contract No. N00014-04-1-0723.

References

[1] Web Services Architecture. Available at: *http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.*

[2] U.S. Department of Defense Directive (DODD) 8100.1: "Global Information Grid (GIG) Overarching Policy," The Pentagon, Washington D.C., 2002.

[3] S. S. Yau, Y. Wang and F. Karim, "Development of Situation-Aware Application Software for Ubiquitous Computing Environments", *Proc. 26th IEEE Int'l Computer Software and Applications Conf.*, 2002, pp. 233-238.

[4] S. S. Yau, et al, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing*, vol. 1(3), 2002, pp. 33-40.

[5] S. Russell, P. Norvig, *Artificial Intelligence: A modern Approach*, 2nd ed., Prentice Hall, 2003.

[6] S. Card, T. Moran, A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbraum Associates, 1983

[7] David L. Hall, James Llina, *Handbook of Multisensor Data Fusion*, CRC Press, 2001

[8] J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence", *Machine Intelligence 4*, 1969, pp. 463-502.

[9] J. A. Pinto, *Temporal Reasoning in the Situation Calculus*, PhD Thesis, University of Toronto, 1994.

[10] J. McCarthy., "Situation Calculus with Concurrent Events and Narrative", *http://wwwformal.stanford.edu/jmc/narrative/narrative.html*, 2000.

[11] D. Plaisted "A Hierarchical Situation Calculus", *J. Computing Research Repository (CoRR)*, 2003.

[12] C. J. Matheus, M. M. Kokar, and K. Baclawski, "A Core Ontology for Situation Awareness", *Proc. 6th Int'l Conf. on Information Fusion*, 2003, pp. 545 –552.

[13] C. J. Matheus, et al, "Constructing RuleML-Based Domain Theories on top of OWL Ontologies", *Proc. 2nd Int'l Workshop on Rules and Rule Markup Languages for the Semantic Web*, 2003, pp. 81–94.

[14] Web Services Coordination (WS-Coordination), Available at: *http://www-106.ibm.com/developerworks/library/ws-coor/*

[15] Web Services Coordination Framework (WS-CF), *http://www.oracle.com/technology/tech/webservices/htdocs/spec/WS-CF.pdf*

[16] P. Braioneand G. P. Picco, "On Calculi for Context-Aware Coordination", *Proc. 6th Coordination Conf.*, 2004, pp. 38-54

[17] G. Cabri, L. Leonardi and F. Zambonelli, "Engineering Mobile Agent Applications via Context-Dependent Coordination", *IEEE Tran. On Software Engineering,* vol. 28(11), 2002, pp. 1039-1055.

[18] C. Julien and G. Roman, "Egocentric context-aware programming in ad hoc mobile environments", *Proc. 10th Int'l. Symp. On the Foundations of Software Eng.*, 2002, pp. 21-30.

[19] M. P. Singh, *Distributed Scheduling of Workflow Computations*, Technical Report TR-96-18, North Carolina State University, 1996. *http://www.csc.ncsu.edu/ faculty/mpsingh/papers/databases/wfscheduling.ps*

[20] P. Doherty, J. Kvarnstrom, "TALplanner: A temporal logic-based planner", *AI Magazine*, vol. 22(3), 2001, pp.95-102.

[21] H. Davulcu, M. Kifer, I.V. Ramakrishnan, "CTR-S: A Logic for Specifying Contracts in Semantic Web Services", *Proc. 13th Int'l World Wide Web Conf.*, 2004, pp.144-153.

[22] S. S. Yau, et al, "Adaptable Situation-Aware Secure Service Based Systems", *Proc. 8th IEEE Int'l Symp. on Object-oriented Real-time distributed Computing,* to appear.

[23] J. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed enviroment," *J. ACM*, vol. 37(3), 1990, pp. 549-587.

[24] M. Kifer, "Deductive and object-oriented data languages: A quest for integration," *Proc. DOOD*, 1995, pp. 187-212.

[25] FLORA-2 website, *http://flora.sourceforge.net/*

[26] R. Bharadwaj, "A Framework for the Formal Analysis of Multi-Agent Systems," *Proc. Formal Approaches to Multi-Agent Systems*, 2003.