Journal of Web Engineering, Vol. 10, No.3 (2011) 175-196

EXCEPTION HANDLING IN PERVASIVE SERVICE COMPOSITION USING NORMATIVE AGENTS

J. OCTAVIO GUTIERREZ-GARCIA

Gwangju Institute of Science and Technology, Republic of Korea
joseogg@gmail.com

FELIX F. RAMOS-CORCHADO

CINVESTAV-IPN, Unidad Guadalajara, Mexico
framos@gdl.cinvestav.mx

Received February 20, 2011
Revised August 30, 2011

The full integration of pervasive computing services into daily life leads to smart spaces with a wide range of intelligent devices that must be dynamically composed to provide a transparent service to users. To achieve this, pervasive services have to coordinate among themselves in both an automated and autonomous manner, with the aim of satisfying complex user requirements that no single service can fulfill. However, existing pervasive environments are normally ad-hoc and isolated systems incapable of: 1) reacting to dynamic and unforeseen situations that may raise exceptions, and 2) collaborating with pervasive services beyond their physical limits. The contributions of this work are: 1) Proposing a normative agent-based service composition method capable of handling exceptions in open pervasive systems. 2) Using the Web as the underlying infrastructure where pervasive services provided by either smart devices or web services can coexist and interact with each other. 3) Providing a modular and hierarchical agent coordination method based on virtual organizations to sustain coordination among agents belonging to multiple organizations (smart spaces). 4) Integrating exception handling mechanisms based on social norms and formalized by event calculus predicates into virtual organizations to support and guide exception handling in both a dynamic and autonomous manner.

Key words: Exception handling, multiagent systems, normative systems, pervasive computing, service composition
Communicated by: D. Schwabe & A. Ginige

1 Introduction

Pervasive environments are complex systems by nature where many self-interested and independent entities must dynamically coordinate among themselves to provide pervasive and transparent composite services to users. In addition, users are nomads that move from place to place performing a broad range of daily activities that may require interacting with various smart devices, which may belong to several smart places. However, current pervasive systems are static and ad-hoc systems deployed in closed and isolated environments where all possible entities are known beforehand and

fixed interaction protocols are defined. This precludes pervasive systems from: 1) reacting to dynamic and unforeseen situations that may raise exceptions, e.g., loss of messages or unavailable services, and 2) collaborating with pervasive services beyond their physical limits.

The inherent complexity, dynamicity, and interactivity of pervasive systems emphasize the need for the agent paradigm [42]. Agents as self-interested and autonomous entities can dynamically coordinate (see [10]) and react in the presence of unexpected situations, e.g., exceptions [24], while achieving a given objective (e.g., pervasive service composition).

Exceptions [24] are abnormal situations presented during the enactment of an interaction protocol. Exceptions are common in complex and dynamic scenarios like those provided by pervasive systems. Examples of exceptions are: loss of messages, unwillingness of message receivers to reply, and unavailable services among others.

In closed agent-based pervasive systems [7, 41], the semantics of interaction rules is commonly missing or depends on agents' private states that cannot be publicly verified and require homogeneous agents in order to have standard agents' beliefs (c.f., the belief-desire-intention approach [6]). This increases the amount of a priori knowledge required to establish agent interaction, and limits its validity to closed environments with homogeneous agents. In this regard, social norms [3, 4, 5, 8], namely obligations [14], can be publicly verified and supervised [34], which allows interaction among heterogeneous pervasive services provided by different vendors.

In this work, we propose a normative agent-based service composition method capable of handling exceptions in open pervasive systems. The Web is used as the underlying infrastructure where pervasive services provided by either smart devices or web services can coexist and interact with each other. Pervasive services provided by both smart devices and web services are wrapped and represented by agents. Furthermore, a hierarchical and reactive agent coordination method supported by virtual organizations (VOs) is used to coordinate agents belonging to multiple smart spaces. The control system of VOs [1, 5, 12] is defined by means of obligations, which are formalized by event calculus [25] predicates. Finally, exception handling mechanisms based on social norms are integrated into VOs to guide and handle exceptions in both a dynamic and autonomous manner.

This paper is structured as follows: Section 2 defines obligation-based agent interaction. Section 3 defines pervasive services in terms of obligations. Section 4 contains the definition of pervasive VOs. Section 5 presents an agent-based pervasive service composition algorithm. Section 6 describes exception handlers that are integrated into the pervasive service composition method. Section 7 presents a comparison with related works. Finally, Section 8 gives some concluding remarks and future research directions.

2 Interaction via Obligations

Agents should follow interaction protocols to achieve a certain objective. Moreover, agent interaction protocols should be endowed with semantics that provides a standard meaning for each message. Based on this, we have defined obligation-based operational semantics that gives meaning to agent interaction at the same time that leads the interaction.

Obligations, as the main element of the interaction semantics, are impositions to oneself demanded by a social force [14], i.e., agents acquire obligations to comply with society's standards.

2.1 Formalizing Obligations

The logical framework that supports the management of obligations is provided by the event calculus [25], which is an action formalism that defines events/actions and their effects in a temporal basis. In this case, events represent either the emission and reception of messages, or sensor signals captured by smart devices. The effects of events are reflected in changes over propositional properties of a given domain called fluents. In this work, fluents are Boolean, i.e., fluents can have *true* or *false* values.

Fluents may represent: 1) a state of affairs denoted by *f*, 2) an ordinary obligation of an agent *a* to achieve a state of affairs *f* denoted by *O(a, f)*, or 3) a conditional obligation denoted by *CO(a, f, g)*, which indicates that agent *a* is obliged to achieve *f*, but only when fluent *g* holds, i.e., fluent *g* acts as a switch for activating/deactivating the obligation in question.

The manipulation of obligation fluents, either ordinary or conditional, is defined by means of event calculus predicates. Thus, a set of operations that creates, releases, and cancels agents' obligations is defined. The event calculus predicates utilized to formalize obligations are described in Table 1 (see [33] for a detailed definition).

Predicate	**Meaning**
HoldsAt(f, t)	Fluent *f* holds at time *t*.
Initiates(m(), f, t)	Fluent *f* holds after the execution of event *m()* at time *t*.
Terminates(m(), f, t)	Fluent *f* does not hold after the execution of event *m()* at time *t*.
Happens(m(), t)	Event *m()* is executed at time *t*.
InitiallyP(f)	Fluent *f* holds from time *t=0*.
InitiallyN(f)	Fluent *f* does not hold from time *t=0*.
Trajectory(f, t, g, inc)	Fluent *g* holds at time *t+inc*, if fluent *f* is initiated at time *t*.
Declipped(t_1, f, t_2)	Fluent *f* is initiated between t_1 and t_2, $t_1 < t_2$.

Table 1 Event calculus predicates

We selected the event calculus formalism because it provides an intuitive management of actions and their effects, supporting either abductive or deductive reasoning [28]. In addition, the event calculus temporal framework allows defining one-shot obligations (e.g., an obligation to print a file) as well as continuous obligations (e.g., an obligation to grant access to a database from 8 a.m. to 7 p.m.). With respect to time management, in the event calculus, time is assumed to be linear and in correspondence to natural numbers.

2.2 Management of Obligations

Obligations can be created, released, and canceled. The definitions for each group of operations are as follows:

2.2.1 Creating Obligations

An ordinary obligation *O(a, f)* over an agent *a* is created when a message *m()* is sent by the same agent *a*, in view of the fact that obligations are self-acquired objects.

CreateO(m(), O(a, f), t):
{Happens(m(), t) ^ Initiates(m(),O(a, f),t)}

A conditional obligation *CO(b, f, g)* over an agent *b* is initiated when an agent *a* sends a message *m()* to agent *b* attempting to induce an obligation, but unlike ordinary obligations, a fluent *g* must hold. Fluent *g* indicates whether agent *b* accepts or rejects the social induction of the obligation.

CreateCO(m(), CO(b, f, g), t):
{Happens(m(),t) ^ Initiates(m(),CO(b, f, g),t) ^ HoldsAt(g,t)} where g=AllowedBy(b,O(b, f))

2.2.2 Releasing Obligations

An ordinary obligation *O(a, f)* of agent *a* to achieve a state of affairs *f* is released when *f* is achieved. This involves the transmission of a message *m()* that terminates the obligation fluent *O(a, f)*, and initiates fluent *f* to indicate that the state of affairs has been achieved.

ReleaseO(m(), O(a, f), t):
{Happens(m(), t) ^ Initiates(m(), f, t) ^ Terminates(m(), O(a, f), t)}

A similar definition is used for releasing conditional obligations, but adding a fluent *g* that indicates the previous acceptance of the obligation.

ReleaseCO(m(), CO(a, f, g), t):
{Happens(m(),t) ^ Initiates(m(), f, t) ^ Terminates(m(),CO(a, f, g), t) ^ HoldsAt(g, t)}
where g=AllowedBy(a, O(a, f))

2.2.3 Canceling Obligations

When an obligation *O(a, f)* is abruptly terminated, a set of compensatory obligations (Φ) must be created to compensate for its cancelation [4], in addition to terminating strongly linked obligations (Γ) to keep a consistent interaction.

CancelO(m(), O(a, f), Φ, Γ,t):
{Happens(m(), t) ^ Terminates(m(), O(a, f), t) ^
Initiates(m(), g, t) | g∈Φ ^ Terminates(m(), h, t) | h∈Γ}

In case of conditional obligations, no side effects are required, given that conditional obligations only depend on agent autonomy.

CancelCO(m(), CO(a, f, g), t):
{Happens(m(), t) ^ Terminates(m(), CO(a, f, g), t)}

2.3 Obligation-based Agent Communication Language

We use obligations as the cornerstone of an agent communication language (ACL) to provide operational semantics to agent interaction. In doing so, meaning and purpose are given to agent interaction.

The obligation-based ACL is founded on speech act theory, which considers speech acts as actions [31]. The types of obligation-based illocutionary acts are based on the taxonomy presented in [32], which classifies acts as: assertives, directives, commissives, declarations, and expressives. However, expressive acts are discarded due to their link with psychological states, which are detached from social norms.

2.3.1 Assertive Acts

An obligation-based assertive act aims to convince the receiver (agent *b*) of a message, that an obligation has been achieved by the sender (agent *a*).

Inform(a, b, f, data) ≡ ReleaseO(Inform(a, b, f, data), O(a, f), t)

Where *f* stands for state of affairs, and *data* for set of application domain elements associated to the releasing of the obligation.

2.3.2 Directive Acts

An agent *a* attempts to induce an obligation to achieve a state of affairs *f* to an agent *b* by means of conditional obligations, i.e., an agent *a* creates a conditional obligation over an agent *b*, but condition *g* (which is necessary to activate the obligation) is left to agent *b*'s autonomy denoted by predicate *AllowedBy(b, O(b, f))*.

Request(a, b, f) ≡ CreateCO(Request(a, b, f), CO(b, f, AllowedBy(b, O(b, f))),t)

2.3.3 Commissive Acts

A commissive act is a two-phase procedure: 1) an agent *b* receives a *Request* message to accept an obligation, and 2) agent *b* is entitled to decide whether to accept or reject the obligation. In case of

acceptance denoted by *AllowedBy(b, O(b, f)) = true*, agent *b* sends an *Agree* message that cancels the conditional obligation used as an intermediary obligation and creates the obligation. In case of rejection, the intermediary obligation is simply canceled.

Agree(b, a, f) ≡
CreateO(Agree(b, a, f),O(b, f),t) ^
CancelCO(Agree(b, a, f),CO(b, f, g),t) ^ HoldsAt(AllowedBy(b, O(b, f))=true, t)

Reject(b, a, f) ≡
CancelCO(Reject(b, a, f),CO(b, f, g),t) ^ HoldsAt(AllowedBy(b, O(b, f))=false, t)

2.3.4 Declaration Acts

The aim of obligation-based declaration acts is to cause changes in agents' obligations, but only on those self-acquired obligations, given that an agent may acquire, by itself, obligations according to its convenience without any external induction. For declaration acts, two primitive acts are defined: *Confirm* and *Disconfirm*.

Confirm(a, b, f) ≡ CreateO(Confirm(a, b, f), O(a, f),t)

Disconfirm(a, b, f) ≡ ReleaseO(Disconfirm(a, b, f), O(a, f),t)

A *Confirm* message creates an obligation towards agent *a* on both sides: sender (agent *a*) and receiver (agent *b*). In doing so, agent *b* can rely on the fact that agent *a* is committed to achieve a given obligation, e.g., agent *a* may declare its obligation to provide a new service, which can be accessed by agent *b*. On the other hand, a *Disconfirm* message releases an obligation of agent *a* also on both sides: sender and receiver.

3 Pervasive Computing Services

We model pervasive services as agent roles defined in terms of obligations.

A pervasive service *S* has the next five elements:

1) Interaction context (IC). The IC operates as an interface and description of services. The IC contains all the fluents managed by agents. A fluent listed in the IC is tagged according to its properties:
   - In-out induction (↑) and Out-in induction (↓). These properties indicate that the agent may induce the fluent as an obligation to an external agent, and that the agent may accept the induction of the obligation by other agents, respectively.

- Self-induction (¤). It indicates that the agent may self-acquire the fluent as an obligation by itself.
- Final (*). It indicates that the fluent, assigned as an obligation, can be directly released by means of agent's actions.

2) Application knowledge parameters (DK). Fluents may contain application domain parameters. In addition, illocutionary acts also may contain domain parameters, e.g., parameter *data* included in *Inform* acts.

3) Initial obligations (IO). Agents implementing services may be initially committed to a set of obligations (e.g., in an e-commerce context, an agent may be obliged to provide quotes). This is formally defined as follows:

$$IO = \{f_1, f_2, \ldots f_M\}: \text{InitiallyP}(O(a, f)) \; \forall f \in IO \; \& \; IO \subseteq IC$$

4) Interaction state (IS). The IS is grouped by agent conversation, given that agents may be involved in several conversions at the same time. The elements that define the IS correspond to current ordinary and conditional obligations, as well as achieved obligations. The IS is formally defined as follows:

$$IS_{Ci} = \{\text{HoldsAt}(O(a, f), t) \cup \text{HoldsAt}(CO(a, g, h), t) \cup \text{HoldsAt}(i, t)\}$$
$$\forall f, \forall g, \forall h, \forall i \in IC \text{ and } a \in \{\text{Partners} \cup \text{Itself}\}$$

Where *Itself* stands for agent from whose perspective is computed the IS, and *Partners* stands for agents, who maintain a conversation with agent *Itself*. Additionally, the initial interaction state (IS_0) of an agent *a* is defined as follows:

$$IS_0 = \{\text{InitiallyP}(O(a, f)) \cup \text{InitiallyN}(g)\} \; \forall f \in IO \text{ and } \forall g \in (IC/IO)$$

5) Actions (AC). An action is a 3-tuple <P, M, E> where *P* stands for preconditions, *M* stands for message, and *E* stands for effects.

- Preconditions are conjunctions and/or disjunctions of *HoldsAt(h, t)* predicates where fluent *h* can be an ordinary obligation *O(a, f)*, a conditional obligation *CO(a, f, g)*, or a simple fluent *f*. In addition, preconditions can include application domain parameters in the form $(\varepsilon_p \lozenge \varepsilon_q) \mid \varepsilon_p \neq \varepsilon_q$ where $\varepsilon_i \in DK$, and $\lozenge \in \{=, <, >, \leq, \geq\}$.
- Messages can be either external or internal. An external message is instantiated by the obligation-based primitive illocutionary acts previously defined in Section 2.3, such messages have standard effects. On the other hand, an internal message may be activated by the evolution of the interaction, and it can have customized effects.

- Effects are represented by means of creating, releasing and/or canceling obligations.

4 Pervasive Virtual Organizations

A VO is an association of agents with a shared objective [5, 11] that is regulated by control mechanisms. Based on this, a pervasive VO is defined as the association of obligation-based pervasive services.

Pervasive VOs have five elements:

1) Members. The members of a pervasive VO $\{S_1, S_2, \ldots, S_N\}$ are all the pervasive computing services, either web services or smart devices, contained in a specific smart space.

2) Interaction context. The interaction context provides a description of the functionalities of a given VO as well as an interface to couple with other VOs. The interaction context of a VO is defined as the union of the interaction contexts of its members:

$$InteractionContext = \bigcup_{i=1}^{|Members|} S_i.IC$$

The properties of fluents are affected by the union, e.g., $f{\downarrow} \cup f{\uparrow} = f{\downarrow}{\uparrow}$.

3) Interaction state. In a similar manner, the interaction state of a VO is the union of all the interaction states of its members:

$$InteractionState = \bigcup_{i=1}^{|Members|} S_i.IS$$

4) Triggers. A trigger τ is an event that activates pervasive computing services. A trigger can be seen as a sensor, which can be activated by external events, as well as by the emission or reception of messages.

A set of triggers $\{\tau(x_a)_1, \tau(x_b)_2, \ldots, \tau(x_x)_K\}$ may have a list x_i of application domain parameters, returned by sensors, and passed to agents' obligations. Moreover, application domain parameters can be also used as triggers' conditions, e.g., a person entering a room can activate a trigger only if the room's temperature is lower than 23° C.

A VO, then, has a set of rules to accept obligations induced by triggers:

$$\text{CreateO}(\tau(x), \text{O}(\text{VO}, f(x)), t) \leftarrow \prod \text{Happens}(\tau(x), t) \prod (x \Diamond value) \mid$$
$$\tau(x) \in \text{Triggers}, x \in \text{DK}, \text{and } \Diamond \in \{=, <, >, \leq, \geq\}$$

5) Actions. Pervasive VOs operate as interfaces among member agents and the Web. For this reason, its coordinating role requires a specific service structure provided by three groups of actions:

- When two pervasive computing services are linked, means that one of them requires the service of the other. Then, an induction of obligations between the services may be performed. This is achieved through the pervasive VO, given that not all smart devices could support coordination processes. Then, when a service requires inducing an obligation, the service induces the obligation to its VO, which must have *Agree* and *Reject* messages to handle such requests:

P_h: {HoldsAt(CO(VO, f, AllowedBy(VO, O(VO, f))), t) ^
HoldsAt(AllowedBy(VO, O(VO, f))=true, t)}
M_h: Agree(VO, b, f)
E_h: {∅} f∈InteractionContext | f has↓

P_i: {HoldsAt(CO(VO, f, AllowedBy(VO, O(VO, f))), t) ^
HoldsAt(AllowedBy(VO, O(VO, f))=false, t)}
M_i: Reject(VO, b, f)
E_i: {∅} f∈InteractionContext | f has ↓

- Once the obligation is accepted, the VO requires a set of *Request* messages to induce the obligation.

P_j: HoldsAt(O(VO, f), t)
M_j: Request(VO, b, f)
E_j: {∅} f∈InteractionContext | f has ↑

- Finally, actions in charge of updating the interaction state of the VO's members should be provided. Then, whenever an obligation is released, the VO informs the releasing of the obligation to its members.

P_k: HoldsAt(f, t)
M_k: Inform(VO, b, f, data)
E_k: {∅} f∈InteractionContext | f has *

All the groups of actions have no effects other than those defined in the primitive acts' operational semantics.

5 A Pervasive Service-Oriented Architecture

In the proposed framework, three kinds of service providers coexist: smart devices, web services, and VOs. Fig. 1 shows the general architecture.

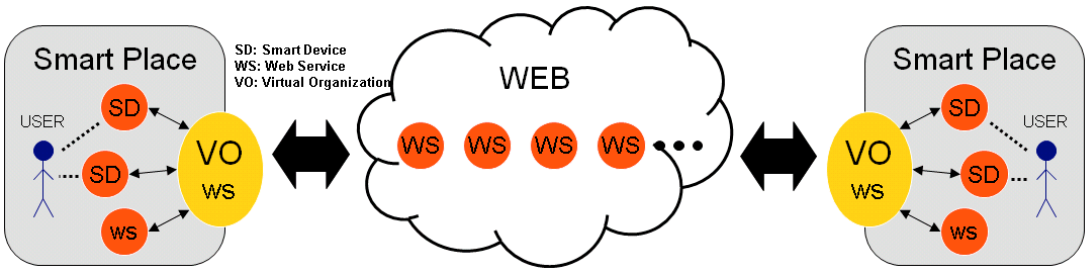

Figure 1 Pervasive computing service architecture

A VO is a service composer agent, whenever a trigger is executed, a set of obligations is induced to the VO. The VO, in return, induces the obligations to its members, if no member of the VO can release the obligation, the VO searches for external obligation-based web services that may accept and release such obligation. In addition, a VO may interact with other VOs.

The algorithm followed by VOs to compose services is the following:

1. INPUT: Firstly, some triggers should be fired according to sensors activated by user events denoted by Happens($\tau_i(x)$, t) | $\tau_i(x) \in$ Triggers.
2. Then, obligations are induced to the VO by means of the triggering rules:

   $$\text{CreateO}(\tau(x), \text{O}(\text{VO}, f_i(x)), t) \leftarrow \prod \text{Happens}(\tau(x), t) \prod (x \Diamond value)$$

   Afterwards, a set of goal obligations (i.e., obligations that have to be fulfilled by the VO) is created: Goal-Obligations = {O(VO, $f_1(x)$), O(VO, $f_2(x)$), …, O(VO, $f_N(x)$)}
3. Now, for each O(VO, $f_i(x)$) $\in$ Goal-Obligations, the VO checks the ICs of its members for a fluent $f_i(x)$ that contains {↓}, i.e., the VO searches for members that could accept the obligation.
   a. If a member *S* is found,
      i. The VO sends a *Request(VO, S, $f_i(x)$)* message
      ii. If an *Agree(S, VO, $f_i(x)$)* message is received, then

          Goal-Obligations / O(VO, $f_i(x)$)

          Else
      iii. If a *Reject(S, VO, $f_i(x)$)* message is received, then

          Keep searching - Go back to step 3.

   Else
   b. The VO searches for a web service that could accept the obligation

i. If an appropriate service *S* is found,

The VO sends a *Request(VO, S, $f_i(x)$)* message

If an *Agree(S, VO, $f_i(x)$)* message is received, the VO adds the service *S* as a member by performing:

Members = Members $\cup$ S

InteractionContext = InteractionContext $\cup$ IC_S

InteractionState = InteractionState $\cup$ IS_S

Else

If a *Reject(S, VO, $f_i(x)$)* message is received, then

Keep searching - Go back to step 3.b

Else

Abort service composition.

4. Once goal obligations are induced, some members of the VO may require interacting with other members to induce obligations, this is done through the VO. The interaction is mainly based on induction and releasing of obligations, therefore:
   a. A member *m* could send a *Request(m, VO, f(x))* message

      Then, the VO sends an *Agree(VO, m, f(x))* message, and

      Goal-Obligations $\cup$ O(VO, f(x)).
   b. Go back to step 3
5. The composition ends when Goal-Obligations = {$\varnothing$}.

OUTPUT: An obligation-based service composition.

A web service *S* could represent another VO that can compose the services of its members. When a VO is added to another VO, triggers are discarded because they are normally associated to physical sensors, which should be only locally addressed. A flow diagram of this algorithm is depicted in Fig. 2.

5.1 Example Scenario: The Meeting Room

Joseph is going to give a presentation at his university, he already has his presentation slides in his wireless personal digital assistant (PDA), when he arrives to the meeting room, the PDA transfers the slides to a projection computer. Then, the projection computer starts a projector device and requests an adjustment in the room's light intensity to proceed with the presentation. In addition, the slides are labeled as public, meaning that the presentation should be publicly transmitted, then a microphone and video cameras are activated, and an external video broadcasting service is requested.

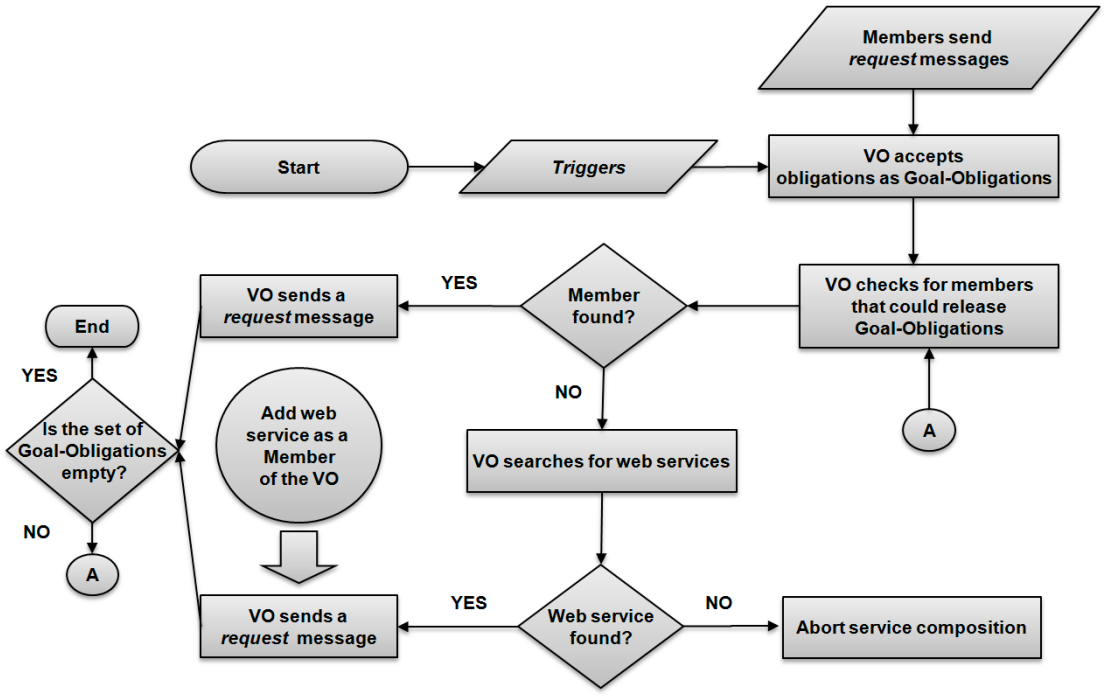

Figure 2 Algorithm for service composition

The meeting room scenario has six services: a PDA, a projection computer (PC), a projector (PR), an electronic light system (EL), a videoconference equipment (VE), and a video streaming service (VS). The PDA is a mobile smart device that looks for VOs to extend its service. The projection computer, the projector, the electronic light system, and the videoconference equipment are members of the meeting room (MR), a VO, which provides appropriate interfaces to contact its specific services.

The meeting room scenario has three VOs: PDA, MR, and VS. Although PDA and VS provide only one service, each one of them constitutes a VO. For explanation purposes, a partial definition of the VOs is presented in Fig. 3. The models presented are PDA, VS, and the general structure of MR. The definition of the remaining pervasive computing services is similar, and thus omitted. In addition, the preconditions and effects are also omitted, given that they can be derived from the inspection of the operational semantics of each obligation-based communicative act.

Although the definitions of services (see Fig. 3) contain only actions to interact with other pervasive computing services, there could be internal actions with parameterized preconditions and effects. Internal actions could evolve the internal state of pervasive computing services. Moreover, the evolution of the internal state of the services is finally reflected in their external interaction, given that internal actions could enable or disable external features by means of releasing or adopting internal obligations. In doing so, internal actions define the executable specification of services.

1: ---------- Personal Digital Assistant (PDA) – Pervasive computing service ----------
2:
3: IC_{PDA}= {DisplaySlides(file, mode) ↓↑, …}
4: DK_{PDA} = {file, mode, …}
5: AC_{PDA} = {
6: A_1 : Request(PDA, b, DisplaySlides(file, mode))
7: … }
8:
9: ---------- Video Streaming Service (VS) – Pervasive computing service ----------
10:
11: IC_{VS}= {DisplayVideoStreaming(source) ↓* , … }
12: DK_{VS} = {source, videoPublicIP}
13: AC_{VS}= {
14: A_1 : Agree(VS, b, DisplayVideoStreaming(source))
15: A_2 : Reject(VS, b, DisplayVideoStreaming(source))
16: A_3 : Inform (VS, b, DisplayVideoStreaming(source), {videoPublicIP})
17: …}
18:
19: ---------- Meeting Room (MR) – Virtual organization ----------
20:
21: $Services_{MR}$ = {PC, PR, EL, VE}
22: IC_{MR} = {AdjustLight(lightIntensity)↓↑*, DisplayVideoStreaming(source) ↓↑,
23: DisplaySlides(file, mode) ↓↑*, … }
24:
25: Triggers = {Agree(MR, b, DisplaySlides(file, mode))}
26:
27: CreateO(Agree(MR, b, DisplaySlides(file, mode)), O(MR, AdjustLight(lightIntensity)), t)
28: ← Happens(Agree(MR, b, DisplaySlides(file, mode)), t)
29:
30: CreateO(Agree(MR, b, DisplaySlides(file, mode)), O(MR, DisplayVideoStreaming(source)), t)
31: ← Happens(Agree(MR, b, DisplaySlides(file, mode)), t) ^ (mode='public')
32: …
33: AC_{MR} = {
34: A_1 : Request(MR, b, AdjustLight(lightIntensity))
35: A_2 : Request(MR, b, DisplayVideoStreaming(source))
36: A_3 : Agree(MR, b, DisplaySlides(file, mode))
37: A_4 : Reject(MR, b, DisplaySlides(file, mode))
38: … }
39:
40: AC_{EL} = {
41: A_1 : Agree(EL, b, AdjustLight(lightIntensity))
42: A_2 : Reject(EL, b, AdjustLight(lightIntensity))
43: A_3 : Inform(EL, b, AdjustLight(lightIntensity), {∅})
44: … }
45: AC_{PC} = { … } AC_{PR} = { … } AC_{VE} = { … }

Figure 3 Specification of pervasive services of the meeting room scenario

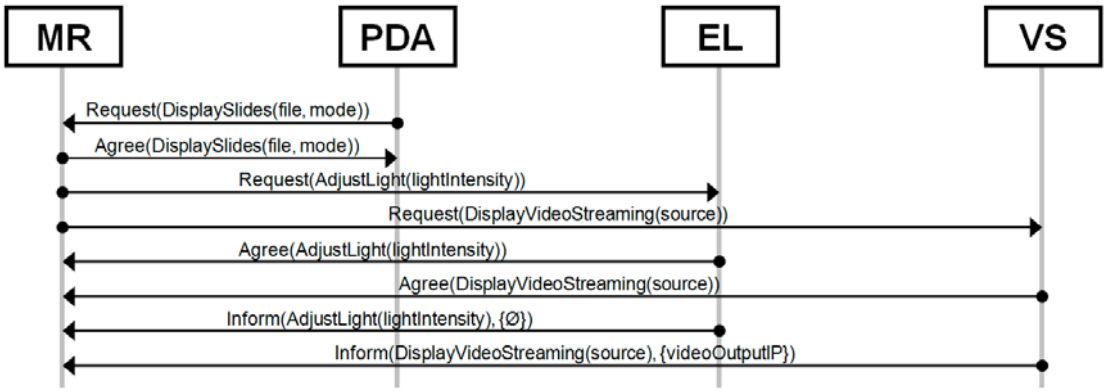

Figure 4 Agent conversation for a service composition in the meeting room scenario

Fig. 4 shows an agent conversation that results from the composition algorithm, which coordinates all the pervasive computing services of the meeting room scenario. Firstly, the PDA requests to MR to display some slides activating the MR's triggers and inducing obligations *AdjustLight(lightIntensity)* and *DisplayVideoStreaming(source)* (see lines 27 and 30 of Fig. 3). Afterwards, obligations *AdjustLight(lightIntensity)* and *DisplayVideoStreaming(source)* are induced to EL (a member of MR) and to VS (a web service), respectively. Obligation *DisplayVideoStreaming(source)* is induced to VS because no member of MR could release such obligation. Then, VS was adopted as a member of MR.

The coordinating role of MR (defined according to the methodology given in Section 4) allows the composition of the pervasive services in a reactive manner without planning features. Furthermore, the pervasive services are endowed with interaction contexts, which indicate the obligations that the services may release, accept, and/or induce to others. Moreover, the proposed specification formalism supports both reactive and intelligent agents. On the one hand, reactive agents may be provided of detailed service definitions where highly customized preconditions and effects are established according to its environmental sensors (triggers). On the other hand, intelligent agents with planning skills (capable of selecting different courses of action) can be also beneficiated, given that the interaction expressed in terms of obligations is state-based, i.e., there is not a predefined ordered sequence of messages. In this regard, the obligation-based framework provides agents with the support to plan their conversations by either abductive or deductive reasoning [28].

6 Exception Handling in Pervasive Service Composition

The service composition process is based on induction of obligations that requires the enactment of a small request-accept protocol. However, due to the dynamicity of pervasive computing environments, exceptions such as loss of messages, unwillingness of message receivers to reply, and unavailable services among others may occur. To cope with these situations, exception handlers should be included.

For each *Request* message to be sent by an agent *a* to an agent *b*, the next event calculus rule should be included:

Trajectory(CO(b,f,g), t_1,TimeException(a, CO(b,f,g)), p) ← Happens(Request(a,b,f), t_1)

The *Request* message is used to trigger a timeout exception handler, and according to its operational semantics (see Section 2.3.2), it creates a conditional obligation *CO(b, f, g)*. Then, the *Trajectory* predicate indicates that if fluent *CO(b, f, g)* holds at time t_1, fluent *TimeException* will hold at time t_1+p. This activates a time counter *p*, whose value should be previously defined. If the timeout is reached, an exception is thrown, this is formalized as follows:

Happens(ExceptionHandling(a, CO(b, f, g)), t_2) ←
HoldsAt(TimeException(a, CO(b, f, g)), t_2) ^
¬Declipped(t_1, NormalCourse(a, CO(b, f, g)), t_2) ^ (t_2= t_1+p)

If fluent *TimeException* holds, and fluent *NormalCourse* is not initiated between t_1 and t_2, then agent *a* throws an exception, expressed by the execution of action *ExceptionHandling*. Now, in order to handle the exception, a consistent interaction state should be recovered. Given that agent *a* created a conditional obligation to agent *b*, two measures have to be taken: (1) to cancel the conditional obligation, and (2) to terminate fluent *TimeException*, which corresponds to the exception thrown. This is defined as follows:

CancelCO(ExceptionHandling(a, CO(b, f, g)), CO(b, f, g), t) ^
Terminates(ExceptionHandling(a, CO(b, f, g)), TimeException(a,CO(b, f, g)), t) ←
Happens(ExceptionHandling(a, CO(b, f, g)) , t)

If no exception is raised, then the timer should be interrupted. This is done by initiating fluent *NormalCourse* whenever *Agree* or *Reject* messages are received:

Initiates(Agree(b, a, f), NormalCourse(a,CO(b,f,g)),t) ← Happens(Agree(b, a, f), t)

Initiates(Reject(b, a, f), NormalCourse(a,CO(b,f,g)),t) ← Happens(Reject(b, a, f), t)

To exemplify the exception management mechanism, let's take the first message exchanged between MR and PDA presented in Fig. 4, as follows:

Request(PDA, MR, DisplaySlides(file, mode))

When the obligation-based *Request* message is sent, a timeout counter is started:

Trajectory (CO(MR, DisplaySlides(file, mode), g), t_1, TimeException, p=5) ←
Happens(Request(PDA, MR, DisplaySlides(file, mode)), t_1)
where g = AllowedBy(MR, O(MR, DisplaySlides(file, mode)))

The value of time counter *p* is set to 5. Now, if before t_1+5 time units MR replies, the time counter is disabled by means of activating fluent *NormalCourse*:

Initiates(Agree(MR, PDA, DisplaySlides(file, mode)), NormalCourse, t) ←
Happens(Agree(MR, PDA, DisplaySlides(file, mode)), t)
where $t_1 < t < t_1+5$

This rule indicates that a normal course of interaction has been followed. On the other hand, if MR does not reply on time, an exception handler is triggered and the intermediary conditional obligation *DisplaySlides(file, mode)* is terminated to keep a consistent interaction state. This is stated as follows:

CancelCO(ExceptionHandling, CO(MR, DisplaySlides(file, mode), g), t_1+5) ^
Terminates(ExceptionHandling, TimeException, t_1+5) ←
Happens(ExceptionHandling, t_1+5)
where g = AllowedBy(MR, O(MR, DisplaySlides(file, mode)))

In this exemplification, fluents *TimeException* and *NormalCourse* as well as event *ExceptionHandling* were presented without parameters for clarity reasons.

The exception handling mechanism is mainly focused on exceptions caused by broken communication among agents. However, what would happen when agents can communicate, but they are incapable of releasing previously acquired obligations due to the presence of abnormal situations, for instance, MR may be unable to display some slides due to unavailable video streaming services. Thus, whenever this kind of exceptions occurs in obligation-based agent interaction, it gives as a result that an obligation cannot be released [15]. Then, a controlled manner to manage the exception is canceling the obligation and keeping a consistent interaction. To do this, we have defined another obligation-based communicative act:

Failure(a, b, f, Φ, Γ) ≡ CancelO(Failure(a, b, f), O(a, f), Φ, Γ, t)

Symbols *a* and *b* represent sender and receiver of the message, respectively, fluent *f* refers to the state of affairs that agent *a* is obliged, symbol Φ denotes compensatory obligations, and symbol Γ denotes strongly linked obligations to be canceled.

The *Failure* communicative act requires the definition of two lists of obligations: compensatory obligations (Φ) and strongly linked obligations (Γ).

$$\Phi = \{O(a, f_1), O(a, f_2), \ldots, O(a, f_m)\} \qquad \Gamma = \{O(b, f_n), O(c, f_o), \ldots, O(d, f_z)\}$$

A set of these lists, creates a new component in the service definition, called *Expected Exceptions* (*EX*), to which agents refer whenever a common exception occurs:

$$EX = \{f_1:(\Phi_1, \Gamma_1), f_2: (\Phi_2, \Gamma_2), \ldots, f_N: (\Phi_N, \Gamma_N)\}$$

For each fluent f_i (to which exceptions want to be caught), a set of compensatory obligations, and a set of strongly linked obligations should be defined.

An agent sends a *Failure* message whenever the agent has attempted to release an obligation and has failed. In addition, *Failure* messages inform other involved agents of exceptional situations.

Since the obligation that causes a given exception must hold, the pattern for a *Failure* message has the next definition:

P : HoldsAt(O(a, f), t)
M : Failure(a, b, f, Φ, Γ)
E : {$\varnothing$} $\quad f \in EX$

The effects are not others than those defined in the *Failure* communicative act, i.e., even though the set of effects of the *Failure* message is empty, it has effects, the ones provided by operation *CancelO()*, which are the cancelation of strongly linked obligations and the creation of compensatory obligations. This was left unspecified for clarity reasons and for indicating that additional effects can be added to evolve the internal state of the sender agent.

Whenever an agent sends a *Failure* message and creates a set of compensatory obligations, the agent should release the newly acquired compensatory obligations. To do so, the agent can release the compensatory obligations 1) by means of the agent's actions, for this, the agent should be endowed with *Inform* messages, or 2) by means of inducing the obligation to another agent, using a *Request* message.

P: HoldsAt(O(a, f), t)
M : Inform(a, b, f, data)
E : {$\varnothing$} $\quad f \in EX \mid f$ has {*}

P: HoldsAt(O(a, f), t)
M : Request(a, b, f)
E : {∅} f ∈ EX | f has {↑}

Both message patterns should be instantiated according to the properties of the fluents of a given pervasive service in order to release compensatory obligations.

The management of exceptions in such a manner applies for expected exceptions, however unexpected exceptions may require additional support, e.g., repositories of exception handlers [24], where agents may select appropriate handlers according to the situation presented.

7 Related Work

The use of the agent paradigm in the pervasive computing domain is vast [7, 39, 41], however standard agent communication semantics is normally ad-hoc or missing, and when present, it is based on belief-desire-intention semantics [6], which cannot be publicly verified and require standard beliefs' concepts, only possible in closed environments, that is the case of [26], [36], and [37], which present FIPA-complaint agent platforms, which based their communication semantics on agents' beliefs, desires, and intentions [13].

With regard to the intersection of pervasive computing and web-deployed services relevant research efforts have been done, such as: [22], [23], [27], and [35]. Moreover, not only isolated services are considered, but their combination to create new services, that is the case of the service composition methods presented in [9], [21], [29], and [30]. Nonetheless, exception management during the composition process is omitted, even though pervasive computing environments are highly dynamic and therefore the presence of exceptions is common. In this aspect, we extend the state-of-the-art by providing mechanisms that support exception handling in agent communication and as a consequence a robust service composition method based on obligation-based agent conversations in pervasive environments.

In [18] and [19], a policy-based language based on deontic concepts to define the behavior of pervasive services is proposed. References [18] and [19] make use of obligations, rights (permissions), dispensation (releasing of obligations), and prohibitions. In our framework, prohibitions are obligations to not achieve a state of affairs, and everything that is not prohibited is permitted, following the approach given in [17] and [20]. Our interaction framework is exclusively based on obligations adding simplicity and avoiding redundancy with additional social norms.

In addition, in [18] and [19], definitions for services' actions and speech acts are also proposed. Preconditions and effects of actions are not defined in terms of deontic concepts, as we do, but in terms of Prolog predicates over application domain parameters. With respect to speech acts, [18] and [19] provide Prolog-based definitions of four acts: delegation, request, cancel, and revocation speech acts. *Delegation* speech acts are used to confer rights to agents/entities. *Request* speech acts are used to ask for actions and/or rights. *Cancel* speech acts annul previous requests. *Revocation* acts remove rights previously granted. In a broader sense, the communication language proposed by Kagal *et al*. [18, 19]

assumes that some kind of hierarchical structure of pervasive entities is defined, so agents can delegate and revoke rights, but in an open pervasive system with heterogeneous entities such hierarchical relation cannot be assumed. In this regard, our approach assumes that all agents have the same level of power, and agents' autonomy directs the acceptance or rejection of obligations.

It is acknowledged that this present work is a considerably and significantly extended version of the works presented in [14, 15]. In [14], a preliminary version of the obligation-based ACL and a service composition algorithm that only composes services in one VO were defined. In [15], exceptions handlers based on static and ad-hoc compensation rules that do not take into account time parameters were presented. The enhancements and improvements of the present work are: 1) Identifying (Section 1) and discussing (Section 7) issues of pervasive computing environments (Section 5.1) where normative agents are proposed to support flexible and dynamic collaboration among pervasive services even in the presence of exceptions. 2) Defining a failure communicative act that provides standard operational semantics to failures in obligation-based agent interaction (Section 6). 3) Designing and integrating exception handlers that support generic and time-based exceptions into pervasive VOs (Section 6). 4) Formalizing events in pervasive systems by means of triggers (Section 4). 5) Composing services taking into account multiple VOs consisting of heterogeneous pervasive services (Section 5).

8 Conclusion and Future Work

The novelty and significance of this research is that, to the best of authors' knowledge, it is the first effort in providing a normative agent-based approach for dealing with service composition taking into account: 1) open pervasive computing environments, e.g., coping with heterogeneous pervasive services, and 2) dynamic exception handling in both a dynamic and reactive manner.

In this paper, both web-deployed services and services provided by smart devices were wrapped and represented by agents. Agent interaction as well as pervasive VOs were formally defined by using obligations that endowed agents with standard operational semantics, which sustains service composition in open systems. A modular and hierarchical service composition method supported by the coupling mechanism of pervasive VOs was designed. Finally, exception handling mechanisms for managing generic and time-based exceptions during service composition were defined and integrated into the normative agent-based service composition method.

The obligation-based ACL as well as the exception handlers integrated into the pervasive service composition algorithm endow agents with mechanisms 1) to dynamically interact among each other in open pervasive computing environments, 2) to compose pervasive computing services that may be distributed in multiple pervasive computing environments in both an automated and autonomous manner, and 3) to handle exceptions during the composition process in both a dynamic and automated manner.

Our future work is addressed to: 1) develop an argumentation layer where intelligent devices may take benefit from arguing about the acceptance of a given obligation, similar to [38], and to 2) achieve semantic interoperability among agents as well as providing context-awareness by means of adopting semantic web services [2, 40], similar to [16].

Acknowledgements

This research was partially supported by CoECyT-Jal Project No. 2008-05-97094, whilst author J. Octavio Gutierrez-Garcia was supported by CONACYT grant No. 191493. The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

1. Artikis, A., Pitt, J., Sergot, M., Animated specifications of computational societies. In Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent systems: Part 3, Italy, 2002, pp. 1053-1061.
2. Berners-Lee, T., Hendler, J., Lassila, O., The semantic web. Scientific American, 284(5), (2001), 34-43.
3. Boella, G., Van Der Torre, L., Regulative and constitutive norms in normative multiagent systems. In Proceedings of the 10th International Conference on the Principles of Knowledge Representation and Reasoning, Menlo Park, 2004, pp. 255-265.
4. Boella, G., Van Der Torre, L., Fulfilling or violating obligations in normative multiagent systems. In Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Beijing, 2004, pp. 483-486.
5. Boella, G., Hulstijn, J., Van Der Torre, L., Virtual organizations as normative multiagent systems. In Proceedings of the 38th Annual Hawaii international Conference on System Sciences - Volume 07, Hawaii, 2005, pp. 192-202.
6. Bratman, M.E., Israel, D.J., Pollack, M.E., Plans and resource-bounded practical reasoning. Computational Intelligence 4(4), (1988), 349-355.
7. Campo, C., Service discovery in pervasive multi-agent systems. In Proceedings of AAMAS Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Agents, Bologna, 2002, pp. 133-146.
8. Carabelea, C., Boissier, O., Coordinating agents in organizations using social commitments. Electronic Notes in Theoretical Computer Science, 150(3), (2006), 73-91.
9. Chakraborty, D., Perich, F., Joshi, A., Finin, T.W., Yesha, Y., A reactive service composition architecture for pervasive computing environments. In Proceedings of the 7th Conference on Personal Wireless Communications, Singapore, 2002, pp. 53-62.
10. Christoffel, M., Wojke, G., Werner, S., Rezek, R., Xu, S., An agent-oriented approach to the integration of information sources. Journal of Web Engineering 4(3), (2005), 224-243.
11. Dignum, V., Meyer, J.-J., Weigand, H., Dignum, F., An organization-oriented model for agent societies. In Proceedings of AAMAS Workshop on Regulated Agent-Based Social Systems: Theories and Applications, Bologna, 2002, pp. 31-50.
12. Ferber, J., Gutknecht, O., Michel F., From agents to organizations: an organizational view of multi-agent systems. In Proceedings of Agent Oriented Software Engineering, Melbourne, 2003, pp. 214-230.
13. FIPA communicative act library specification. Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00037/.
14. Gutierrez-Garcia, J.O., Ramos-Corchado, F.F., Koning, J.-L., Obligation-based agent conversations for semantic web service composition. In Proceedings of the 2009 IEEE/WIC/ACM international Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01, Milan, 2009, pp. 411-417.
15. Gutierrez-Garcia, J.O., Koning, J.-L., Ramos-Corchado, F.F., An obligation approach for exception handling in interaction protocols. In Proceedings of the 2009 IEEE/WIC/ACM

international Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03, Milan, 2009, pp. 497-500.
16. Ha, Y., Park, C., Kang S., A semantic web services-based infrastructure for ubiquitous service systems. Journal of Web Engineering 8(2), (2009), 182-210.
17. Huang, Z., Masuch, M., The logic of permission and obligation in the framework of ALX3: how to avoid the paradoxes of deontic logic, Logique et Analyse, 149, (1997), 55-74.
18. Kagal, L., Rei : A policy language for the me-centric project. HP Labs Technical Report, 2002.
19. Kagal, L., Finin, T., Joshi, A., A policy language for a pervasive computing environment. In Proceedings of the 4th IEEE international Workshop on Policies For Distributed Systems and Networks, Lake Como, 2003, pp. 63-74.
20. Kagal, L., Finin, T., Modeling conversation policies using permissions and obligations. Autonomous Agents and Multi-Agent Systems 14(2), (2007), 187-206.
21. Kalasapur, S., Kumar, M., Shirazi, B.A., Dynamic service composition in pervasive computing. IEEE Trans. Parallel Distrib. Syst. 18(7), (2007), 907-918.
22. Kindberg, T., Barton, J., A web-based nomadic computing system. Comput. Netw. 35(4), (2001), 443-456.
23. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., Spasojevic, M., People, places, things: web presence for the real world. Mob. Netw. Appl. 7(5), (2002), 365-376.
24. Klein, M., Dellarocas, C., Exception handling in agent systems. In Proceedings of the 3rd Annual Conference on Autonomous Agents, Seattle, 1999, pp. 62-68.
25. Kowalski, R., Sergot, M., A logic-based calculus of events. New Generation Computing, 4(1), (1986), 67-95.
26. Lech, T.C., Wienhofen, L.W., AmbieAgents: a scalable infrastructure for mobile and context-aware information services. In Proceedings of the 4th international Joint Conference on Autonomous Agents and Multiagent Systems, The Netherlands, 2005, pp. 625-631.
27. Masuoka, R., Parsia, B., Labrou, Y., Task computing - the semantic web meets pervasive computing. In Proceedings of the 2nd International Semantic Web Conference, Florida, 2003, pp. 866-881.
28. Miller, R., Deductive and abductive planning in the event calculus. In Proceedings of the 2nd AISB Workshop on Practical Reasoning and Rationality, Manchester, 1997, pp. 1-12.
29. Mokhtar, S.B., Fournier, D., Georgantas, N., Issarny, V., Context-aware service composition in pervasive computing environments. In Proceedings of the 2nd International Workshop on Rapid Integration of Software Engineering Techniques, Heraklion, 2005, pp. 129-144.
30. Robinson, J., Wakeman, I., Owen, T., Scooby: middleware for service composition in pervasive computing. In Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, Toronto, 2004, pp. 161-166.
31. Searle, J.R., Speech acts: an essay in the philosophy of language. Cambridge University Press, Cambridge, UK, 1969.
32. Searle, J.R., A taxonomy of illocutionary acts. Language, mind, and knowledge. Minnesota studies in the philosophy of science, vol. 7, K. Gunderson, Ed. University of Minnesota Press, USA, 1975, 344-369.
33. Shanahan, M.P., The event calculus explained, in Wooldridge, M.J., and Veloso, M. (eds.), Artificial Intelligence Today, LNAI 1600, Springer, 1999, pp. 409-430.
34. Singh, M.P., A social semantics for agent communication languages. Technical Report. UMI Order Number: TR-99-03., North Carolina State University at Raleigh, 1999.
35. Singh, S., Puradkar, S., Lee, Y, Ubiquitous computing: connecting pervasive computing through semantic web. Information Systems and E-Business Management, 4(4), (2006), 421-439.

36. Soldatos, J., Pandis, I., Stamatis, K., Polymenakos, L., Crowley, J.L., Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services. Comput. Commun. 30(3), (2007), 577-591.
37. Spanoudakis N., Moraitis, P., Agent based architecture in an ambient intelligence context. In Proceedings of the 4th European Workshop on Multi-Agent Systems, Lisbon, 2006, pp. 163-174.
38. Tzagarakis, M., Karousos, N., Karacapilidis, N., On the development of web-based argumentative collaboration support systems. In Lytras M. et al. (eds.), Visioning and Engineering the Knowledge Society: A Web Science Perspective, LNAI 5736, Springer, 2009, pp. 306-315.
39. Vallee, M., Ramparany, F., Vercouter, L., A multi-agent system for dynamic service composition in ambient intelligence environments. In Proceedings of the 3rd International Conference on Pervasive Computing, Munich, 2005, pp. 175-182.
40. W3C Semantic Web Activity. World Wide Web Consortium (W3C). http://www.w3.org/2001/sw/.
41. Wohltorf, J., Cissée, R., Rieger, A., Scheunemann, H., An agent-based serviceware framework for ubiquitous context-aware services. In Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent systems, New York, 2004, pp. 1-6.
42. Wooldridge, M., An introduction to multiagent systems, second ed., John Wiley & Sons Ltd, Chichester, England, 2009.