Semantic Driven Dynamic Redeployment of Components in Pervasive systems

Ismael Bouassida Rodriguez, Aymen Kamoun, Saïd Tazi, Thierry Villemur, Khalil Drira

Semantic Driven Dynamic Redeployment of Components in Pervasive systems

Ismael Bouassida Rodriguez[a,b], Aymen Kamoun[a,b], Said Tazi[a,c], Thierry Villemur[a,d], Khalil Drira[a,b]

[a] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
[b] *Univ de Toulouse, LAAS, F-31400 Toulouse, France*
[c] *Univ de Toulouse, UT1, LAAS, F-31000 Toulouse, France*
[d] *Univ de Toulouse, UTM, LAAS, F-31100 Toulouse, France*

Abstract

The emergence of pervasive computing interweaving different devices and sensors introduces new challenges to ensure the continuity of the communication and collaboration between humans in distributed systems. The communication may be interrupted due to the change of context, such as a change of a users role or an energy level diminution, thus leading to lose the cooperation or shared data. This paper describes a framework that aims at ensuring the continuity of communication services even if changes happen. This framework is based on a multi-level modeling approach. Dynamic redeployment algorithms based on semantic description of collaboration with a scenario for validation are presented.

Keywords: pervasive computing, context, semantic, dynamic redeployment

1. Introduction

Collaboration between human beings should gain from the development of pervasive computing environments. These environments should encourage people to communicate and cooperate everywhere at any time, because they are immersed, willingly or not, into environments containing large sets of devices, sensors, actuators and services that are available in a pervasive and automatic manner [1, 2, 3]. Several issues of research arise from the use of pervasive computing for collaboration. For example the devices used in pervasive computing are heterogeneous in terms of communication and processing capabilities, mecanisms to ensure interoperability are needed [4, 5] .

Fluid communication between collaborators is needed when a change takes place, regardless of the level change occurs at, application level or otherwise. Indeed at the applicative level, the user's needs are dynamic because of complex situations of cooperation, such as where there is user mobility or the change of roles that necessitates new software components. Changes may also happen at a lower level, such as machine resources becoming limited due to a decreased level of energy, or a limitation of hardware resources such as main memory or CPU. In pervasive computing environments, the sessions of collaborative activities should be spontaneously initiated and managed (i.e. without user input), thus exploiting implicit collaboration situations. Adaptation such as dynamic deployment of components is needed for establishing multimedia sessions. Therefore, the main adaptation actions in a collaborative pervasive system involve component (re)deployments and/or flow (re)configurations. The goal of the research presented here is to enable continuous communication between collaborators no matter what may happen at different levels by providing adaptation mechanisms of architectures of components at runtime [6, 7]. In our approach, this is made possible through model driven architecture transformations, from abstractions of collaboration to concrete redeployment of software components applied on collaborative sessions. For instance, if a battery level drops during a session, the framework adapts the architecture by redeploying new components in the network to save the session and to ensure continuous communication and collaboration between the nodes (devices) of the session. The events that trigger adaptation actions are the changes in the context of the application. Context is divided into external context (e.g. user preferences, user presence and position, priority of communications, etc.) and execution resource context (e.g. battery level, CPU load and available memory of user devices, etc.) [8, 9]. Semantic computing technologies [10] and graph models [11] have been used as the foundation for the algorithms we have implemented for adaptation concerns. We consider an intermediary level in which the semantics of collaboration are presented at a high conceptual level. At the application and collaboration levels, the semantics of the application and of the collaboration are represented as ontologies and as a set of rules. At the resource level, the set of components are represented as graphs where vertices represent software services and components, and edges represent the communication relationships between such components. Adaption algorithms use these models to detect the change of the context at runtime and to find the best component to deploy dynamically, in manner that ensure the continuity of communica-

tion [12, 13]. In this paper, we focus on architectural adaptation using the framework we have developed. The remainder of this paper is organized into seven sections. Related work is presented in Section 2. Section 3 introduces a scenario and an example of an application that illustrates several aspects of the identified problem. Section 4 presents the generic approach proposed in order to solve such a problem and a proposition of implementation. Section 5 details how our approach can be applied to the proposed scenario. Section 6 presents an evaluation of our approach being applied to the scenario. Finally, Section 7 provides conclusions and possible directions for future work.

2. Related work

We have classified related works into three subsections in accordance with the features highlighted in this section: model driven approaches, architecture oriented, and platform or existing frameworks.

2.1. Model driven approaches

Becker and Giese [14] present an approach based on graph transformation techniques coupled with UML stereotypes in order to model self-adaptive systems. Adaptation, which is performed at run-time, is decomposed into three levels: goal management, change management and component control. However, in this approach, context and collaboration are not explicitly modeled. Edwards [15] presents a system named Intermezzo that enables the construction of applications making use of rich, layered interpretations of context. It provides a context data store and an integrated notification service. This system is collaboration-oriented; contextual information is structured through *activities*, that represent the use of resources (e.g., documents). The underlying context model is very rich and deals with problems such as *ambiguity*, *identity*, *evolution* and *equality* of contextual data, providing solutions for each one of them. However, the case of low-level resources context is not considered.

Pad Ovitz et al. [16] present a context model and reasoning approach developed with concepts from the state-space model, which describes context and situations as geometrical structures in a multidimensional space. A context algebra based on this model is also presented. This work shows how merging different points of view over context enhances the global context reasoning process. The authors provide a model (named Context Spaces) that

unifies the context views of different entities into a single context representation. The cooperative aspect of this work focuses on migration, modeling and reasoning, partitioning, and merging context descriptions between agents for attaining optimal reasoning and context-awareness.

2.2. Architectural adaptation

Zhang et al. [17] propose an adaptive model infrastructure for pervasive computing environments. They propose three adaptive layers: adaptive collaboration layer, adaptive middleware layer and adaptive services layer. The adaptive collaboration layer provides a service cooperation platform in dynamic environments. The adaptive middleware layer is a self-reconfiguring layer that provides an optimized uniform high-level interface for implementation of distributed applications. The adaptive services layer provides an adaptive contents service and adaptive user interfaces. Using this model, the approach makes environmental changes invisible to collaboration among applications and users. From the architectural point of view, this work is similar to ours in that it defines three layers. However, collaboration in this work is limited to the collaboration between services to adapt contents to the user interface.

2.3. Platform oriented research

Ejigu et al. [18] propose a collaborative context-aware service platform named CoCA. This platform is data-independent and it may be used for context-aware application development in pervasive computing. It performs reasoning and decisions based on context data and domain-based policies using ontologies. The platform introduces a neighborhood collaboration mechanism to facilitate peer collaboration between pervasive devices in order to share their resources. The generic context management modeling deals with the way the context data are collected, organized, represented, stored and presented. In this work, collaboration is used between devices in order to better achieve context data acquisition.

Lee et al. [1] present the project Celadon in order to establish an infrastructure enabling on-demand collaboration between heterogeneous mobile devices and environmental devices. The Celadon project is a middleware architecture for ubiquitous device collaboration. Collaborative environments are organized into Celadon zones, which are public areas equipped with wireless access points for technologies. Collaboration in this work is limited to

sharing hardware or software resources. Only the user context is considered in this work.

The Conami middleware [19] is a Collaboration-based Content Adaptation Middleware for dynamic pervasive computing environments. The authors consider the case of Mobile Ad hoc Networks (MANETs). The middleware allows devices in MANETs to collaborate with each other to perform content adaptation. Content adaptation is derived from the context of the user and the user environment and is done by composing available nearby services. In this work, collaboration is considered as a technique to adapt the content to the devices' capabilities. In this middleware, component deployment is not considered.

Perich et al. [2] present the design and implementation of a Collaborative Query Processing protocol that enables devices in pervasive computing environments to locate data sources and obtain data matching their queries. The features of the protocol allow devices to collaborate with other devices in order to obtain an answer for their queries regardless of their limited computing, memory, and battery resources. The presented approach deals mainly with low level context parameters.

3. Introductive Scenario (Collaborative warships game)

The scenario presented in this section illustrates the main features needed by collaborative applications in pervasive environments, such as context-awareness, spontaneous implicit sessions and automatic component deployment.

This scenario has been produced in the context of ITEA2's A2NETS (Autonomic Services in M2M Networks) project [1]. In our case, multi-media games are distributed over a fleet of buses and are available to passengers. Whenever a user boards a bus, his/her personal device notices the presence of the bus network and connects to it.

We consider a game called *Collaborative Warships*, which is a multi-player extension of the classic warships board game. Each player uses his/her mobile device to play and to interact with other players.

[1] `https://a2nets.erve.vtt.fi/`

3.1. Ship Types

We distinguish three ship types: *Battle Ships*, *Repair Ships* and *Spy Ships*. Each ship has two main attributes: life points and speed.

Battle Ships (BS) can shoot at the ships of the opposite team. Any opposing ships that receive damage loose life points.

Repair Ships (RS) restores life points to the companion ships.

Spy Ships (SS) can detect the position of the enemy ships.

One of the *Battle Ships* of a team is the leader of the BSs and SSs. This ship is called *Battle Ship Leader* (BSL). One of the *Repair Ships* of a team is the leader of the RSs. This ship is called *Repair Ship Leader* (RSL).

3.2. Game Rules

Two teams are required for the game to begin. Each team must have at least 1 BS. During the play, the positions of the enemy warships are unknown (except to SSs).

When a ship loses all its life points, it sinks, and leaves the game. The game ends when a team loses all its BS, and the team with ships remaining is declared the winner.

Two main requirements of pervasive collaborative applications are highlighted and instantiated in this example:

- *Context awareness* is necessary in order to detect changes in both external context (e.g. user presence) and execution context (e.g. battery level, available memory).

- *Adaptation* and especially *adaptive deployment* are necessary in order to respond to context changes. If, for instance, the device of a BSL has not enough energy to host several software components handling the exchanged flows, then a set of those components has to be moved and deployed on another mobile device in order to ensure a continuous communication between the team members.

3.3. Team Coordination

Players need to collaborate during the game. Therefore, data flows must be set up between the members of a team. Depending on the type of warship used by each player, different communication flows for cooperating with the other members of his/her team are needed. For example, when a BS reaches an enemy with its fire, it can inform the BSL of the position of the enemy.

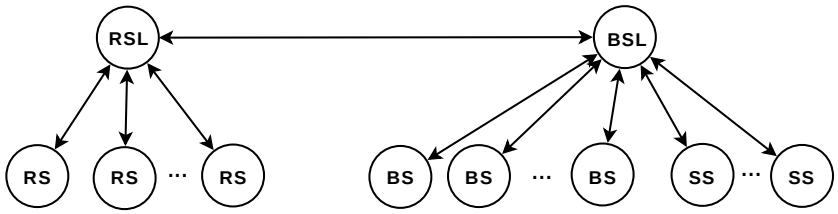


Figure 1: Team coordination description.

The BSL can then transmit that position to the other BSs of his/her team having more destruction points or having a more strategic position in order to shoot at the discovered enemy.

According to this description, several types of communication links are needed between the members of a team. Those links are represented in figure 1. Each RS is linked to the RSL of his/her team. BSs and SSs collaborate with the BSL of their team. The two leaders collaborate.

In the following sections, we propose a generic approach that may be used by developers in order to build collaborative pervasive applications.

4. Proposed approach

The main idea of our approach is to provide a generic framework which enables modeling, session management, multi-level adaptation, and automatic component deployment for pervasive applications. This enables code modularization and reutilization.

4.1. Multi-level Architecture Modeling

In order to clearly separate different concerns in our approach, a multi-level architecture is proposed.

A configuration is denoted as $A_{n,i}$, where n is the considered abstraction level and i is the sequence number (i.e. an architecture $A_{n,i}$ evolves to $A_{n,i+1}$ when it is reconfigured). For a given configuration $A_{n,i}$ at level n, multiple configurations $(A_{n-1,1}, \ldots, A_{n-1,p})$ may be implemented at level $n-1$. Adapting the architecture to constraint changes at level $n-1$ by switching among these multiple configurations allows maintaining unchanged the n-level configuration.

4.2. Multi-level modeling of collaborative pervasive architectures

This subsection presents how we apply the multi-level approach presented above to collaborative pervasive systems in order to build a comprehensive generic framework for such systems.

Adaptation at the highest level should be guided by the evolution of activity requirements. Adaptation at the lowest levels should be driven by execution context constraint changes.

4.2.1. Application level

The application level represents applications needing collaboration among groups of users and/or devices. It contains software elements that are relevant to collaboration,and is represented as the abstraction level $A_{3,i}$.

Application designers also have to implement the refinement procedure which obtains the set $\mathbb{A}_2^i$ of collaboration level models that implement a given $A_{3,i}$ model.

4.2.2. Collaboration level

The main issue addressed by the collaboration level is the determination of a high-level collaboration schema that responds to the application's collaboration needs. Hence, collaborative sessions may be managed from this level. The elements needed to implement such sessions are determined from this level. This model is inspired by classic graph-based session description formalisms such as dynamic coordination diagrams [20].

4.2.3. Middleware level

The middleware level provides a communication model that masks low-level details in order to simplify the representation of communication channels. The architectural model produced by this level, $A_{1,i}$ represents a detailed deployment descriptor containing the elements needed in order to implement the sessions defined by the collaboration level.

4.2.4. Context Capture and Representation

We target the adaptation of cooperative applications to different context changes. These changes may concern resource constraints (e.g. connectivity, energy level, available memory, etc.) or the evolution of the collaborative activity and environment (where participants can arrive and leave, change roles, active a "do not disturb" mode in their devices, etc.). We respectively call

these two sets of parameters *resources context* and *external context*. Context parameters are captured by external modules.

Relevant events that are produced in the external context are taken into account by the application level (i.e. they are detected and *translated* into modifications on the $A_{3,i}$ model, thus producing a new model $A_{3,i+1}$).

Relevant events that are produced related to run-time resource changes are taken into account by the middleware level. If it is possible, changes are handled by reconfiguring the $A_{1,i+1}$ model (i.e. by selecting a different model among the possible refinements of the collaboration model).

4.3. Generic Refinement and Selection Procedures

For a given architectural configuration $A_{n,i}$ in level n, a procedure, called `Refine()` computes the set $\mathbb{A}^i_{n-1}$. This set represents all possible architectural configurations in level $n-1$ that implement $A_{n,i}$.

Given a set of possible architectures, it is necessary to choose an architecture to be effectively deployed. We present here a procedure, `Select()` (table 2), that allows for the choosing of an architecture depending on several parameters.

This procedure uses the resources context (e.g. variations of communication networks and processing resources) to eliminate the architectural configurations that cannot be deployed within the current resources levels. Among the set of selected architectures, the best configuration w.r.t architectural characteristics is selected.

The choice of an architecture must first take into account the resources context. The `Context_Adaptation()` function (table 2, line 5) is a generic function that depends on the resources context. For example, it can express the availability level of a given resource (bandwidth, memory, energy level, etc.). This function is used for two purposes: it allows discarding architectures that cannot be deployed within the current resources context, and it allows for the selection of the architectures best adapted to that context.

The function `Context_Adaptation()` associates a given architecture to a value that reflects its degree of adaptation to the current resources context. Our criterion for this function is as follows: well adapted architectures are those which have fewer nodes in a *critical situation*. A node is in a critical situation when its level for a certain resource is close to the threshold defined for that resource. This function is detailed in table 1. In this function, a set of resources $Resource_1 \dots Resource_R$ (e.g. $Resource_1$=energy, $Resource_2$=CPU load and $Resource_3$=available RAM) is considered. L^i_r

Table 1: Context Aware Function

```
1  Context_Adaptation()
2  {
3  Let A_{1,q} be an architectural configuration at level 1
4  Let N = card(A_{1,q})
5  Let Resource_1 ... Resource_R the set of considered resources
6  Let R = card({Resource_1 ... Resource_R})
7  Let node_i^q be deployment node i of A_{1,q}
8  Let L_r^i the level of the resource r for node_i^q
9  Let T_r be the threshold associated with the resource r
10 Let α_r^i  r ∈ [1..R] be weights associated with each resource r for node_i^q
11 Let β_r  r ∈ [1..R] be weights associated with each resource r for A_{1,q}
12 Let cadapt=0
13 for each i ∈ 1..N
14     for each r ∈ 1..R
15          P_r^i = α_r^i L_r^i − T_r
16          if P_r ≤ 0 then return -1
17     end for
18 end for
19 for each r ∈ 1..R
20     cadapt=cadapt+β_r min_i(P_r^i)
21 end for
22 return cadapt
23 }
```

Table 2: Generic selection procedure.

```
1 Select(Policy)
2 {
3 Let A_{n,p} ∈ 𝔸_n , p ∈ ℕ
4 Let C denote the context attributes
5 Select S_1 = {A_{n-1,k} ∈ 𝔸^p_{n-1}, k ∈ ℕ such that:
     Context_Adaptation(A_{n-1,k}, C) ≥ Context_Adaptation(X, C), ∀X ∈ 𝔸^p_{n-1}}
6 if card(S_1) ≠ 1
7   if Policy == Dispersion
8      Select S_2 = {A_{n-1,k} ∈ S_1 , k ∈ ℕ such that:
       Dispersion(A_{n-1,k}) ≥ Dispersion(X), ∀X ∈ S_1}
9   if Policy == Distance
10     Let A_{n,p} and A_{n,q} ∈ 𝔸_n , p, q ∈ ℕ
11     Let A_{n-1,p} the current mapping at level n − 1 of A_{n,p}
12     Select S_2 = {A_{n-1,k} ∈ 𝔸^q_{n-1}, k ∈ ℕ such that:
       Relative_Cost(A_{n-1,p}, A_{n-1,k}) ≤ Relative_Cost(A_{n-1,p}, X), ∀X ∈ S_1}
13  if card(S_2) ≠ 1
14     Select any configuration from S_2
15 }
```

represents the available level of the resource $Resource_r$ for the node $node_i$. L_r^i, expressed as a percentage, is calculated as the resource's level before

deployment (given by the resources context module) minus the amount of resource consumed by each deployed component. The value T_r is a threshold that indicates the critical percentage of the resource r, for a given node, under which the deployment is not possible. The coefficients α_r^i represent the importance assigned to each level L_r^i with respect to the characteristics of $node_i$. For instance, the CPU load is more critical for a smartphone than for a laptop, because the smartphone needs CPU processing for other important functions (such as answering calls, etc.). Therefore, α_{CPU}^i is 1 for a laptop nodes and 0.5 for a smartphone (i.e. a smartphone will be considered as critical when its CPU level is lower than $2T_{CPU}$). P_r^i is calculated for every node as the difference between the level of the $Resource_r$ and the threshold T_r. If this difference is negative for a node, this means that the node is in a critical state with respect to $Resource_r$, and hence the considered architecture cannot be deployed. Therefore, -1 is returned and the considered architecture will not be selected. If no node was found to be in a critical situation, then the *degree of adaptation to the context* (`cadapt`) of the considered architecture is calculated, as shown in table 1, line 20. First, for every resource, the minimum value of P_r^i found on any deployment node of the architecture is retained. Second, `cadapt` is calculated as an average of these minima (weighted by β_r coefficients). The β_r coefficients represent the global importance degree given to each resource ($\sum_1^R \beta_r = 1$). Resources with higher β_r are considered more important than other resources. These coefficients can be defined by the administrator or the business logic. This definition of the function `Context_Adaptation` leads to the selection of the architectures having the highest values of resources for their more critical nodes.

When several architectures have the same value of `Context_Adaptation()`, a policy (indicated by the parameter `Policy`) is used by the `Select()` procedure in order to retain the optimal configuration. If the chosen policy is `Weight`, the selection is based on minimizing the function `Dispersion()` (table 2, line 8). This generic function corresponds to the cost or the efficiency/performance of an architecture. For instance, it may be defined as the number of software components deployed per node. If the chosen policy is `Distance`, the selection minimizes the distance between two architectural configurations at level $n-1$, both implementing the corresponding n-level architectural configuration. This is performed using the function `Relative_Cost()` (table 2, line 12).

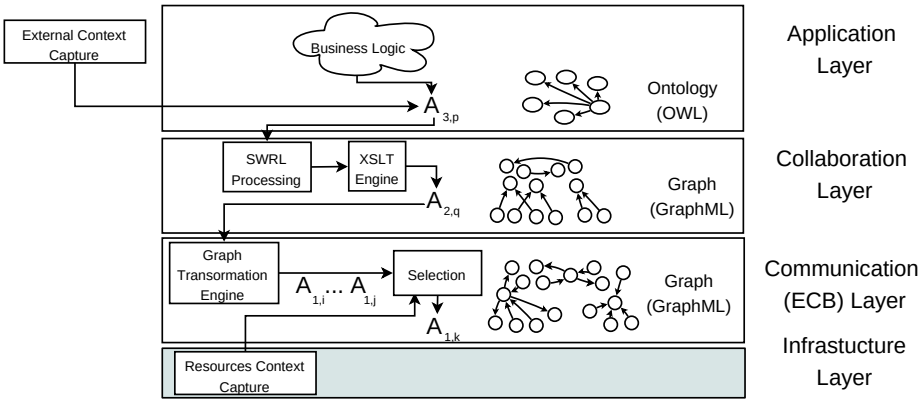


Figure 2: Model-oriented leveled framework.

4.4. A Multi-level Collaborative Framework for pervasive Systems

The proposed multi-level modeling approach for collaborative pervasive systems remains generic with respect to implementation. As we have separated the approach and its implementation, this generic approach may be implemented in different ways by different designers. In this subsection we propose an implementation that can be used by application designers as a collaboration framework for pervasive systems, with our choices illustrated in figure 2. We first present the models used in each level, followed by the refinement and selection procedures that enable the transitions between levels.

4.4.1. Application level Model – $A_{3,i}$

We have chosen an ontology based model because it constitutes a standard knowledge representation system, allowing reasoning and inference. We have chosen to describe these models in OWL [21], the Semantic Web standard for metadata and ontologies.

In general, ontologies are separated in two levels: a generic ontology and a specific ontology. The former is a domain-wide ontology, but independent of applications. The latter ontology extends the generic one with terms specific to an application category. We have followed the same pattern in our implementation: there is a generic collaboration ontology (that describes sessions, users, roles, data flows, nodes, etc.) and an application ontology that extends the collaboration ontology with business-specific concepts and relations.

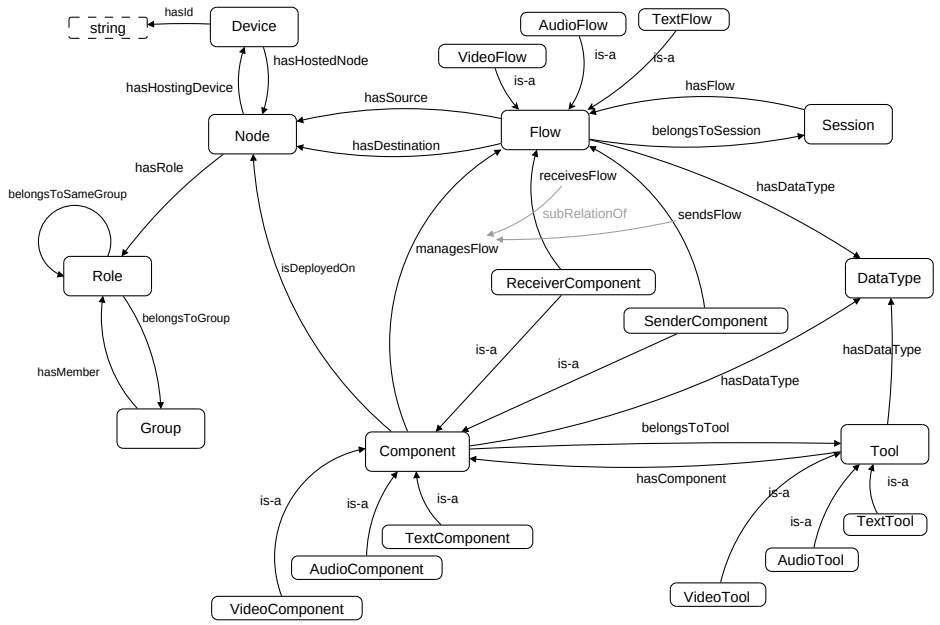


Figure 3: Generic collaboration ontology.

4.4.2. Collaboration level Model – $A_{2,i}$

The collaboration level model is a graph, inspired by dynamic collaboration diagrams [20]. This model is a generic collaboration ontology that is common to all applications, and therefore it is provided within the framework. This ontology[2] is represented in figure 3. The main concept in this ontology is `Session`. A session contains one or more `Flows`, which have a source `Node` and a destination `Node`. `Nodes` are hosted on `Devices`. Each `Node` has one or more associated `Roles`. `Flows` are processed with `Tools`, which are composed of several `Components` (e.g. `SenderComponents` and `ReceiverComponents`). Related `Flows`, `Tools` and `Components` share the same `DataType` (e.g. `Audio`, `Text` or `Video`). This graph is expressed in the GraphML language (an XML dialect for representing graphs [11]). Further explanations about this ontology and the associated choices can be found in [22].

[2] `http://homepages.laas.fr/tazi/ontologies/sessions.owl`

4.4.3. Middleware level Model – $A_{1,i}$

For the middleware level, we have retained the Event Based Communication (EBC) paradigm [23]. The EBC model represents a well established paradigm for interconnecting loosely coupled components and it provides a one-to-many or a many-to-many communication pattern.

EBC entities are represented in the middleware level model. This model is a detailed graph containing a set of *event producers* (EP), *event consumers* (EC) and *channel managers* (CM) connected with *push* and *push* links. Multiple producers and consumers may be associated through the same CM. Since this model is also a graph, it is also expressed in the GraphML language.

4.4.4. Application–Collaboration Refinement and Selection

When a reconfiguration event related to the *external context* is detected, it is captured by the application level and translated into changes in the application level ontology, thus producing a new instance, $A_{3,1}$. As the application level model is represented in OWL, we use SWRL rules [24] in order to implement its refinement to a collaboration architecture.

Some rules have been included along with the generic collaboration ontology. For example, let us consider the rule shown in table 3.

Table 3: A SWRL Rule for AudioFlows

```
AudioFlow(?af) ∧ hasSource(?af,?src) ∧ hasDestination(?af,?dst) ∧
swrlx:createOWLThing(?asc,?src) ∧ swrlx:createOWLThing(?arc,?dst)
 → AudioSenderComponent(?asc) ∧ isDeployedOn(?asc,?src) ∧
AudioReceiverComponent(?arc) ∧ isDeployedOn(?arc,?dst)
```

This rule states that, whenever an instance of `AudioFlow` is found in the ontology, two components have to be instantiated[3]: an `AudioReceiverComponent` having the flow's destination node as its deployment node, and an `AudioSenderComponent` having the source node as its deployment node. Similar rules are used for text and video flows, thus generating text and video sender and receiver components.

[3]The SWRL *built-in* `CreateOWLThing()` creates new instances of existing concepts within a SWRL rule.

The rules implementing the transition from the application-specific ontology to the generic collaboration ontology are application-dependant, and therefore they have to be specified by the application designers along with the application ontology.

The processing[4] of the SWRL rules along with the application ontology produces a new ontology instance ($A_{2,1}$ that describes the collaboration level graph in OWL language). This graph is translated into GraphML (by means of a simple XSLT transformation) in order to be shared with the middleware level. If the new model $A_{2,1}$ is equal to the previous $A_{2,0}$, then the process ends, as no architecture reconfiguration is required in lower levels. Each application level model corresponds to a unique collaboration level model. Therefore the selection procedure at this level is straightforward.

4.4.5. Collaboration–Middleware Refinement and Selection

We use graph grammars [25] for the Collaboration-Middleware Refinement.

In order to refine a given collaboration architecture into a set of EBC architectures, the graph grammar $GG_{COLLAB \rightarrow EBC}$,is used[5]. In this graph grammar, non-terminal nodes are collaboration entities while terminals nodes are EBC entities. For clarity's sake, here only the case of audio sessions is considered. Therefore, the productions of this graph grammar refine `AudioReceiverComponent` and `AudioSenderComponent` (AR and AS) into EPs, ECs and CMs. Similar grammar productions have been developed for text and video components.

In order to select the optimal architecture among those built by the refinement process, the generic selection procedure presented in table 2 is used. We detail here our choices for the functions `Dispersion()` and `Relative_Cost()`.

The function `Dispersion()` is used to select architectures having fewer CMs deployed in the same device. The goal is to efficiently balance resources consumption and to be more robust. It associates an architecture $A_{1,q}$ with the number of nodes containing at least one CM. This definition gives higher values to architectures having CMs dispersed in more nodes.

[4]This processing is done with a rules engine such as Jess or a SWRL-enabled reasoning engine such as Pellet.

[5]This implementation is done with a Graph Matching and Transformation Engine (GMTE), available at `http://www.homepages.fr/khalil/GMTE`.

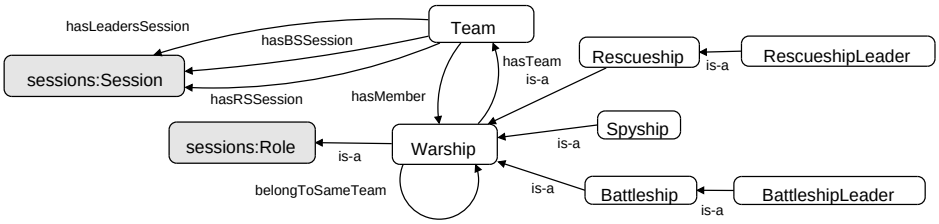


Figure 4: Collaborative Warships application ontology.

The function `Relative_Cost()` is used to select the *closest* architecture to the currently deployed architecture. We choose as criterion of selection the number of redeployments needed to switch from a given architecture to another.

If the selection process is not able to find a valid architectural configuration, then the high-level requirements cannot be implemented within the current resources context. Therefore, the middleware level sends a reconfiguration event to the application level. This level takes this into account in order to change the application level model into a model $A_{3,i}$ that can be implemented.

4.4.6. Deployment Service

The $A_{1,i}$ model produced by the middleware level is the detailed deployment descriptor that implements the low-level elements of the required architecture: producers, consumers, channel managers, and links. In order to effectively deploy such elements into real devices, a *Deployment Service* is needed. This service takes a deployment descriptor $A_{1,i}$ as input and then it downloads, installs and starts the required components on each device. The implementation of this deployment service is based on the OSGi technology [26].

5. Application to the Scenario

Designers of collaborative pervasive applications only have to create i) the application level model, and ii) the SWRL rules that enable the refinement of application level models into collaboration level models. In this section, these design tasks are applied to the *Collaborative Warships* application.

5.1. Application Ontology for the Collaborative Warships Application

The domain ontology[6] modeling the business concepts and relations of the *Collaborative Warships* application is illustrated in figure 4. The main concept of this ontology is `Warship`. The different types of warships (BS, RS, SS) are modeled as sub-concepts of `Warship`. Warships belong to a `Team`. This domain ontology is related to the generic collaboration ontology.

5.2. SWRL Rules for the Collaborative Warships Application

Application-dependent SWRL rules are needed in order to process instances of the application ontology presented in the previous subsection, and thus generate a collaboration level model.

In the case of the *Collaborative Warships* application, we consider, for instance, the rule presented in table 4. Here, we have only considered the case of audio flows. Text and video flows are handled in a similar way.

Table 4: `BS_BSL_audio_flows` Rule

```
BattleShip(?bs) ∧ BattleShipLeader(?bsl) ∧
belongsToSameTeam(?bs, ?bsl) ∧ differentFrom(?bs, ?bsl) ∧
swrlx:createOWLThing(?af1, ?bs) ∧ swrlx:createOWLThing(?af2, ?bs)
→ sessions:AudioFlow(?af1) ∧ sessions:hasFlow(sessionBS,?af1)∧
sessions:hasSource(?af1, ?bsl) ∧ sessions:hasDestination(?af1, ?bs) ∧
sessions:AudioFlow(?af2) ∧ sessions:hasFlow(sessionBS,?af2) ∧
sessions:hasSource(?af2, ?bs) ∧ sessions:hasDestination(?af2, ?bsl)
```

The `BS_BSL_audio_flows` rule, presented in table 4, states that whenever a BS and a BSL belonging to the same team are found, two audio flows are created between them. This flow belongs to a session called `sessionBS`, which is an individual of the class `sessions:session` already existing in the ontology Both audio flows will be captured by the collaboration level rules and corresponding audio senders and receivers will be created on each node in the collaboration level graph.

[6] `http://homepages.laas.fr/tazi/warships.owl`

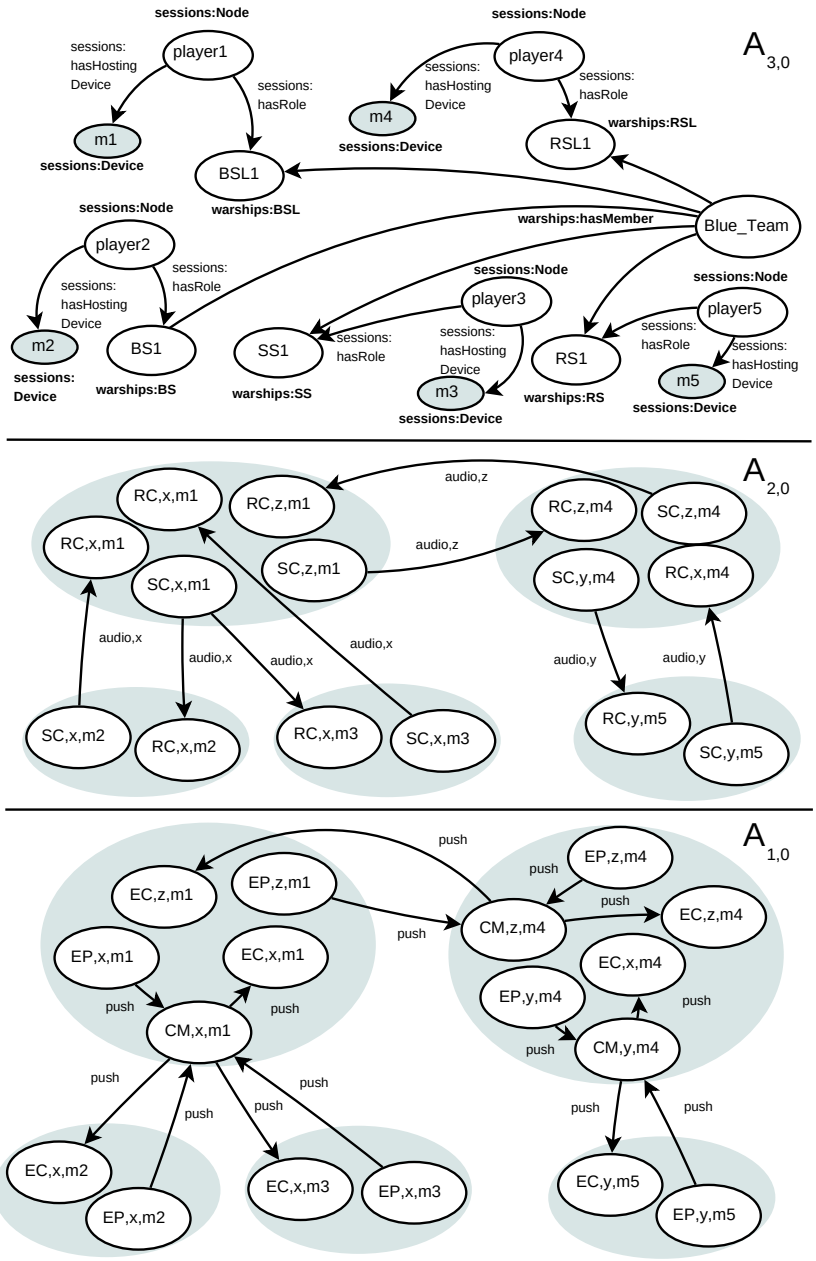


Figure 5: Top-down refinement example.

5.3. Initial Refinement and Adaptation Process Examples

This subsection presents a top-down example showing the architecture models for the three considered levels as well as the refinement processes between levels. The adaptation process is also illustrated.

In this example, we focus only on one team in order to illustrate the initial refinement and the adaptation processes. The team has five players (`player1` to `player5`). Each player represents a node in the application. Each node has an associated role: `player1` is a BSL, `player2` is a regular BS, `player3` is a SS, `player4` is a RSL and `player5` is a regular RS. Each node has an associated device (`m1` to `m5`).

This business level architectural configuration is captured by the game application and represented in the game ontology $A_{3,0}$. The resulting ontology instances are represented in figure 5, application level. Concepts are drawn as ellipses, while relations are drawn as arrows going from one concept to another concept. As the figure shows, concepts and relations from the game-specific ontology (`warships:` prefix) and from the generic collaboration ontology (`sessions:` prefix) are instantiated together.

In order to refine this application model, SWRL rules are processed over these ontology instances. The `BS_BSL_audio_flows` rule creates two audio flows between `player1` and `player2`, the `BSL_RSL_audio_flows` rule creates two audio flows between `player1` and `player4`, etc. The rule presented in table 3 is processed for each instance of `AudioFlow` found, thus creating the corresponding `AudioSenderComponent` and `AudioReceiverComponent` at the endpoints of the audio flow.

The resulting collaboration graph is represented in figure 5, collaboration level. This graph contains 7 audio senders (AS) and 7 audio receivers (AR). The graph edges correspond to data flows, and are labeled by data type (*audio*) and by the session to which they belong. Each component has three attributes: the identifier, the type (sender or receiver) and the deploying machine identifier.

In order to refine this collaboration level graph, the $GG_{COLLAB \rightarrow EBC}$ graph grammar, is used. This produces a set of valid configurations $\mathbb{A}_1^0$. The procedure `Select()` is used in order to find the optimal configuration. The retained configuration, $A_{1,0}$, is presented in figure 5, middleware level. This configuration contains only terminal nodes (i.e. nodes belonging to the EBC level).This refinement creates a detailed deployment descriptor that is used by the deployment service in order to deploy the indicated components on each device, thus implementing the required application level session.

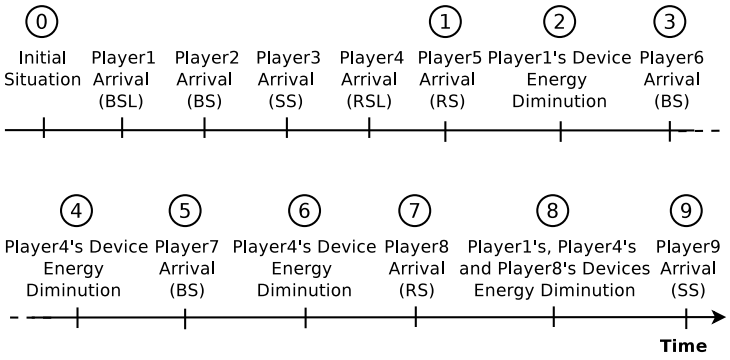


Figure 6: Experimentation scenario.

6. Evaluation

We conducted evaluation experiments using our engine GMTE for executing the graph grammar transformations, and the rule engine Jess for executing the SWRL rules.

In this example, we present results related to one team composed of 9 players (`player1` to `player9`). Each node has an associated role: `player1` is a BSL; `player2`; `player6` and `player7` are regular BSs; `player3` is a SS; `player4` is a RSL ;and `player5` and `player8` are regular RSs. Each node has an associated device (`m1` to `m9`).

The scenario shown in the figure 6 consists of adding a new player every 2 minutes combined with important diminutions of the energy level of the players' devices as shown in Table 5. The underlined values in Table 5 represent degradations that trigger a new reconfiguration. Each time a player joins the game, a refinement and a selection process is triggered. Each time an energy level diminution of a player's device is detected, the selection process is triggered. In the case of this game we consider only three collaboration sessions as mentioned above and shown in figure 1. Since we deploy a channel manager for each session, 1, 2 or 3 channel managers can be redeployed after each selection process. In the worst case, 3 channel managers are redeployed.

6.1. Threshold calculation

Figure 7(a) illustrates the elapsed time for the generation of the collaboration ontology, the collaboration graph and the middleware graphs during each phase.

Table 5: Devices Energy Levels

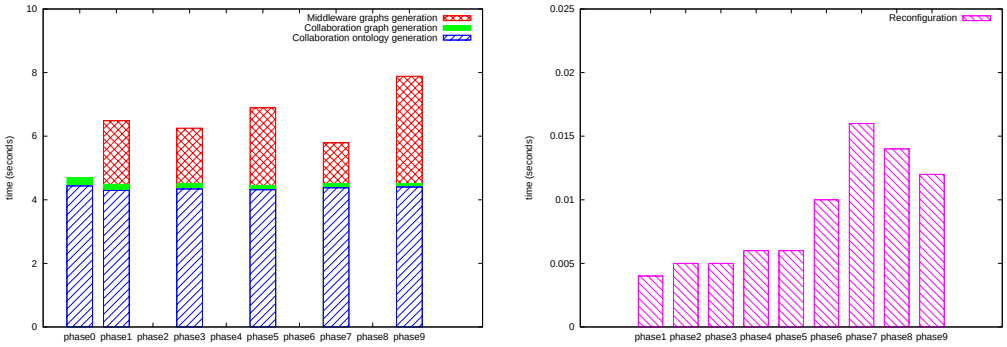
Device \ Phase	1	2	3	4	5	6	7	8	9
player1's device	95%	***82%***	82%	82%	81%	81%	81%	***62%***	62%
player2's device	90%	88%	87%	87%	86%	85%	***70%***	70%	70%
player3's device	90%	88%	87%	87%	86%	85%	85%	85%	84%
player4's device	90%	88%	87%	***80%***	86%	***50%***	50%	49%	49%
player5's device	91%	88%	87%	87%	86%	85%	85%	84%	82%
player6's device	92%	88%	87%	87%	86%	85%	85%	83%	83%
player7's device	92%	88%	88%	86%	86%	85%	85%	***70%***	70%
player8's device	93%	89%	89%	87%	86%	85%	83%	***50%***	48%
player9's device	90%	88%	87%	87%	86%	85%	85%	83%	82%

Before starting the game, the framework needs to be initialized, at this time no player is connected to the game so no middleware graph is generated. Figure 7(a) (phase0) shows the time necessary for the initialization. The average initialization time is about 5 seconds. With respect to the initialization process that occurs only once at the beginning of the session, this duration is acceptable to the game.

This business level architectural configuration of phase 1 is captured by the game application and represented in the game ontology $A_{3,0}$. The resulting ontology instance is represented in figure 5. We represent the time elapsed to generate the ontology instance and the corresponding collaboration and ECB graphs as shown in figure 7(a) (phase1) when `player5` joins the game. The generation time (about 6 seconds) is still acceptable for this kind of application.

We now encounter an important diminution of the energy level of the device `m1`, and as a result the ECB level runs the `Select()` procedure. This procedure chooses a new configuration (e.g. $A_{1,5}$) more adapted to the new context parameters. Within this new configuration, `m1` has fewer components deployed on it. The deployment service uses $A_{1,5}$ for effective redeployment. To study the time consumption of the `Select()` procedure, the scenario illustrates further diminutions of energy levels and the arrival of new players.

The figure 7(b) shows that the selection time is still acceptable within the range of 0.005 to 0.017 seconds. This duration indicates that the framework



(a) Time elapsed ontology and graphs generation.

(b) Time elapsed for reconfiguration.

Figure 7: Evaluation experiments

can quickly adapt the middleware to energy level diminutions and can process frequent middleware reconfigurations. These results are in conformance of the characteristics of mobile devices handled by the users connected to the framework.

7. Conclusion

This paper has presented a multi-level architecture modeling approach for collaborative pervasive systems. Architectural models for application, collaboration and middleware levels have been detailed. Ontologies, SWRL rules, graphs and graph grammars have been used for implementing a rule-based refinement process. These rules handle both transforming a given architecture within the same level and architectural mappings between different levels. This implementation constitutes a framework for building collaborative pervasive systems. The performances of the framework have being evaluated, especially the computing time of adaptation actions and the time of component (re)deployment. Our approach has been illustrated by a multi-media game distributed over a bus fleet. This game is an example of a collaborative pervasive application developed in the context of the European project A2Nets, on real field experiments using pervasive networks. A deployment service is being developed based on the OSGi technology. Also, the possibility of implementing a centralized and a P2P version of the deploy-

ment service for different applications is under investigation. Future research actions include defining further adaptation rules that consider more context parameters (e.g. by extending one of the cited context ontologies), extending refinement procedures and extending graph grammars to handle middleware level abstractions other than Event Based Communications. Finally, the design and implementation of mechanisms enabling the spontaneous setup of implicit sessions will be considered.

Acknowledgement

This article was funded partially by the IST IP's IMAGINE project and the ITEA2's A2NETS. We thank all the LAAS' members who have contributed to achieve earlier versions on top of which this work has been built. Our thanks are addressed in particular German SANCHO who has contributed to the development of FACUS and special thanks to David Allison for proofreading.

References

[1] M. Lee, H. Jang, Y. Paik, S. Jin, S. Lee, Ubiquitous device collaboration infrastructure: Celadon, in: Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop on, 2006, pp. 141–146.

[2] F. Perich, A. Joshi, Y. Yesha, T. Finin, Collaborative joins in a pervasive computing environment, The VLDB Journal 14 (2) (2005) 182–196.

[3] L. Touseau, How to guarantee service cooperation in dynamic environments?, in: L. du Bousquet, J.-L. Richier (Eds.), Feature Interactions in Software and Communication Systems IX, International Co nference on Feature Interactions in Software and Communication Systems, ICFI 2007, 3-5 September 2007, Grenoble, France, IOS Press, 2007, pp. 207–210.

[4] D. Saha, A. Mukherjee, Pervasive computing: A paradigm for the 21st century, Computer 36 (2003) 25–31.

[5] M. Weiser, The computer for the 21st century, Scientific American 265 (3) (1991) 94–104.

[6] J. Mäntyjärvi, T. Seppänen, Adapting applications in handheld devices using fuzz context information, Interacting with Computers 15 (4) (2003) 521–538.

[7] J. Cámara, C. Canal, G. Salaün, Multiple concern adaptation for runtime composition in context-aware systems, Electr. Notes Theor. Comput. Sci. 215 (2008) 111–130.

[8] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems, Int. J. Ad Hoc Ubiquitous Comput. 2 (4) (2007) 263–277.

[9] P. Bellavista, A. Corradi, M. Fanelli, L. Foschini, A survey of context data distribution for mobile ubiquitous systems, ACM Comput. Surv. 44 (4) (2012) 24:1–24:45.

[10] Z. Ma, Soft Computing in Ontologies and Semantic Web, Studies in Fuzziness and Soft Computing, Springer, 2006.

[11] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, M. S. Marshall, GraphML Progress Report, in: Graph Drawing, 2001, pp. 501–512.

[12] S. Maoz, Using model-based traces as runtime models, Computer 42 (2009) 28–36.

[13] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, A. Solberg, Models@ run.time to support dynamic adaptation, Computer 42 (10) (2009) 44–51.

[14] B. Becker, H. Giese, Modeling of correct self-adaptive systems: a graph transformation system based approach, in: CSTST '08: Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology, ACM, New York, NY, USA, 2008, pp. 508–516.

[15] W. K. Edwards, Putting computing in context: An infrastructure to support extensible context-enhanced collaborative applications, ACM Trans. Comput.-Hum. Interact. 12 (4) (2005) 446–474.

[16] A. Padovitz, S. W. Loke, A. Zaslavsky, Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems, Trans. Sys. Man Cyber. Part A 38 (4) (2008) 729–742.

[17] X. Zhang, Z. Li, J. Liu, An Adaptive Infrastructure Concept Model based on CORBA in Pervasive Computing, in: Pervasive Computing and Applications, 2007. ICPCA 2007. 2nd International Conference on, 2007, pp. 490–495. doi:10.1109/ICPCA.2007.4365493.

[18] D. Ejigu, M. Scuturici, L. Brunie, Coca: A collaborative context-aware service platform for pervasive computing, in: Information Technology, 2007. ITNG '07. Fourth International Conference on, 2007, pp. 297–302.

[19] Y. Fawaz, A. Negash, L. Brunie, V.-M. Scuturici, Conami: Collaboration based content adaptation middleware for pervasive computing environment, in: Pervasive Services, IEEE International Conference on, 2007, pp. 189–192. doi:10.1109/PERSER.2007.4283915.

[20] V. Baudin, K. Drira, T. Villemur, S. Tazi, E-Education applications: human factors and innovative approaches, Ed. C.Ghaoui, Information Science Publishing, ISBN 1-59140-292-1, 2004, Ch. A model-driven approach for synchronous dynamic collaborative e-learning, pp. 44–65.

[21] M. K. Smith, C. Welty, D. L. McGuinness, OWL Web ontology Langage Guide, W3C Recommendation, url : `http://www.w3.org/TR/owl-guide/` (Feb. 2004).

[22] G. Sancho, S. Tazi, T. Villemur, A Semantic-driven Auto-adaptive Architecture for Collaborative Ubiquitous Systems, in: 5th International Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST'2008), Cergy Pontoise (France), 2008, pp. 650–655.

[23] R. Meier, V. Cahill, Taxonomy of distributed event-based programming systems, in: ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2002, pp. 585–588.

[24] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission 21 May 2004 (2004).

[25] N. Chomsky, Three models for the description of language, Information Theory, IEEE Transactions on 2 (3) (1956) 113–124.

[26] OSGi Alliance, OSGi Service Platform Release 4 (May 2007).