2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing

Resolving Coordination Challenges in Cooperative Mobile Services

Ramón Alcarria, Tomás Robles, Augusto Morales Domínguez, Edwin Cedeño
ETSI Telecomunicación
Technical University of Madrid
Madrid, Spain
{ralcarria,trobles,amorales,edwinc}@dit.upm.es

***Abstract*—The Internet of Things enables environments where objects are fully interconnected, allowing the execution of smart services and the consumption of functionalities provided by surrounding Web objects. This loose-coupled object interconnection demands improvements in the control plane for an optimum coordination between distributed services in mobile devices. There are several coordination challenges in these environments, related to the interaction between services, the communication channels establishment across service fragments and the transmission of events at runtime. This paper defines a coordination model and proposes solutions to these challenges by developing a cooperative service execution model for mobile environments, using the publish-subscribe paradigm for communicating control events. Subsequently, we evaluate this model and analyze the improvements of the designed optimization mechanisms over the MQTT protocol and the NS-3 simulator.**

Keywords-service coordination; web of things; publish-subscribe; workflow patterns

I. INTRODUCTION

The Internet of Things envisions a world in which all objects are interconnected and interact. The emergence of the Web of Things (WoT) inspires these heterogeneous objects to be accessible in the digital world. The convergence at the network level should also be applied to the service level, where the infrastructure has to provide appropriate abstractions to describe objects by the functionality or the information they provide. This evolution reflects the current user behavior, which is primarily interested in real-world entities (things, places, and people) and their high-level states (empty, free, walking, etc.) rather than in individual sensors and their raw output data.

We consider a service model for the WoT based on a control-driven approach, which allows the development of more complex services and more control in the management of these services by the execution environment. In contrast, the control plane (which defines the execution sequence of activities invoking environment elements) becomes more complex than the data plane (which handles the content exchange between activities).

The communication between elements from the WoT is often delegated to orchestration processes using WS-BPEL for information control. However, to enable collaboration between various entities, a distributed model based on choreography is needed, which focuses more into complex coordination between entities or devices.

Physical items from the WoT execute and validate parts of business processes and enable inter-organizational collaboration and interoperability of heterogeneous hardware. The services described in this paper can be fragmented and executed in mobile terminals in a distributed way, since the ubiquitous access to the functionality of the WoT objects (which support standard application layer protocols and techniques such as HTTP or REST) is decoupled from the invocation control in workflow diagrams, as described by WS-BPEL or BPMN.

In this work we contribute to solve some coordination problems found in cooperative mobile services (CMS). To address these problems we define an event-based communication model and we use the publish-subscribe paradigm [1] to ensure functional decoupling of information producers and consumers. We propose solutions to the identification of each element participating in the service interaction, the correlation between execution instances and the communication between processes composing the distributed service.

The paper structure is as follows. Section II describes CMS model for the Web of Things. Section III describes the distributed architecture for mobile terminals and the interaction between different modules. Section IV contributes to resolve coordination challenges in distributed processes that are implemented and evaluated in Section V. Finally the paper concludes with related work and some conclusions of the proposed solution.

II. SERVICE MODEL

This section describes an example of a distributed mobile service that presents the coordination challenges discussed in the introduction and then defines the CMS model for the WoT.

A. Motivating example

The service, called *Item purchasing*, shows information about a selected product to a customer and, after checking in his profile if the product is suitable for him (detecting intolerances and food preferences), it manages the payment and update the logistical information of the store (automatic stock update).

The service execution logic, shown in Fig. 1 is represented as a workflow diagram, divided into three fragments or tasks (like the role-based task delegation supported by BPMN or BPEL4People/Human Tasks [2]). This execution logic accesses the so called Web objects, represented in blue color: an online shopping list, a store's

978-0-7695-4684-1/12 $26.00 © 2012 IEEE
DOI 10.1109/IMIS.2012.131

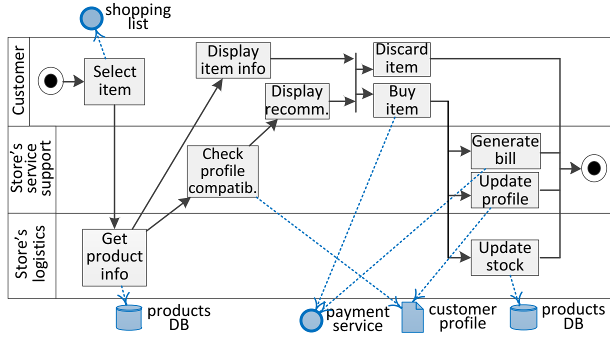

Figure 1. Motivating example

database with information about its products, the customer profile of this store and a payment service.

This example involves some requirements over the service model. In order to restrict access to Web objects depending on entities and favor a more decoupled service execution the services are distributed and executed from the terminals of different roles. This has several advantages. First, the service can easily access the capabilities of the mobile terminal (GPS, NFC) and the information is sent only to the entity that is authorized to manage it. In this motivating example information related to the store's logistics should not be managed in the client terminal. Second, it is easier to dynamically adapt the service coordination to include more participants that arrive during service execution.

Considering these requirements the service needs to be fragmented, distributed and executed in the participant terminals. In this work we propose solutions to the mobile coordination of service fragments, the main challenge of this decision.

B. CMS model

The coordination model defined in this paper meets the requirements proposed in the previous paragraph and relates the concepts of services, tasks and activities. A *Service* consists of a distributed workflow which represents the CMS. We define a *Service Fragment* as each one of these distributed workflows which were previously created and arranged in a service fragmentation/partitioning process [3]. The fragmentation process covers the actions of computing, initializing and distributing a set of fragments needed for carrying out a service. The service should consider other aspects such as user interaction, lifecycle management, security, etc., which are out of the scope of the paper. We also assume that all the fragments are successfully placed in the mobile devices and the information about fragment interaction is stored in the SDL (Service Description Language) document, which contains the necessary information to execute the service.

A *Task* is the instantiation of a fragment that performs a work. Tasks are arranged and initialized in the service bootstrapping process, which will be explained later. A task is composed by at least one Activity.

An *Activity* is an atomic unit of a task. It manages the communication with an object that can be physical or digital, to perform an operation. We classify operations according to

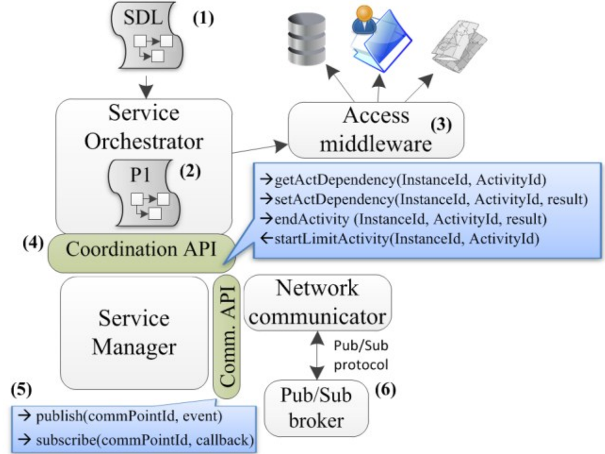

Figure 2. Distributed architecture

their ability to produce data (sensors), consume data (actuators) and process data (processors). Activities trigger data and control events that are consumed by other activities in their own task scope or external task scopes. In the motivating example, product information data is retrieved by the customer and shown in the *display item info* activity. The user, when buying the item, generates a control event consumed by the *generate bill* activity from the store's service support. In a distributed workflow scenario there are interactions between activities from different tasks. We define *Limit Activity* as any activity that communicates with other activity contained in a different fragment by using data or control events. Section IV solves various problems of communication between limit activities.

III. Architecture For CMS

The proposed system can be described by the terminal architecture depicted in Fig. 2, which is applied to each of the mobile devices participating in the service execution. We describe the process and the interaction between the main elements.

Once the system receives the SDL document (1), which contains information about control and data dependencies of all service fragments, the *Service Orchestrator* (SO) starts the service execution (2) by invoking the functionality of the activities in the workflow. The middleware is responsible for accessing (3) the functionality described by the running activity. It includes support libraries for local invocations to device capabilities (camera, contacts, etc.) and to remote objects (objects in the web of things, databases, web services, etc.). For more information about this module the authors refer to their previous work [4].

When the orchestrator detects a limit activity means that there exists a dependency with another fragment, either in the control plane (a neighbor limit activity must be executed) or in the data plane (an activity requires a data located in another fragment). To satisfy dependencies in the data plane the Service Orchestrator uses the *Coordination API* (4) to invoke the *getActDependency (InstanceId, ActivityId)* method to obtain the necessary data and the

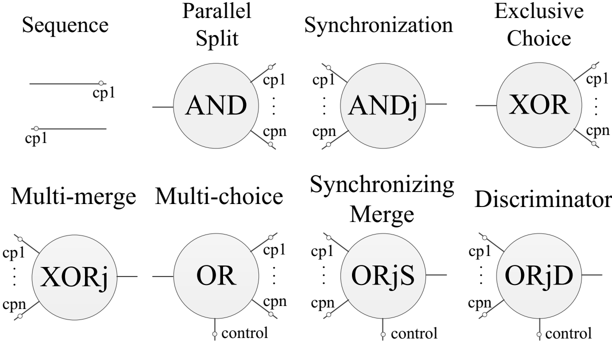

Figure 3. Logic gates and associated patterns

setActDependency (InstanceId, ActivityId) method to provide these data to other activities.

For the control plane, once the activity is executed, the *endActivity (InstanceId, ActivityId, result)* method is invoked so that the *Service Manager* (SM) carries out the process of communicating the limit activity with their neighbors. The service orchestrator waits until the service manager asks to execute a new limit activity with the *startLimitActivity (InstanceId, ActivityId)* method. To address these coordination challenges, the SM uses the *Network communicator* (NC), which initiates the exchange of events between mobile devices at the network level, following the publish-subscribe paradigm. Using this paradigm is justified by the need for time decoupling (wherein the sender and receiver of a message do not need to be involved in the interaction at the same time) and space decoupling (wherein the messages are directed to a particular symbolic address or channel and not directly to the address of an endpoint), which enable the publication of data and control events to an unknown number of nodes in an unknown location.

We define *Communication point* as each of the input and output information ports of each activity. The communication between the SM and the NC is via the *Communication API* (5), which includes the *publish (commPointId, event)* and *subscribe (commPointId, callback)* methods, used to publish (control or data) events that generates a given communication point with id commPointId and to receive events published by other devices in a callback method.

The NC solves the correlation problem, i.e. the unique identification of each service instance running on the terminal. To do this, it serializes the Service, Fragment, Task and Communication Point identifier (present at the service's SDL) and uses a resource identifier $R_{id} = S + F + T + A + \pi_p$, which univocally identifies messages that are generated from any communication point from a running service. The NC uses the R_{id} identifier as a topic for the pub/sub messages [5] and transmits them to the *Pub/sub Broker* (6), which is an external entity (not implemented in the mobile phone) that manages the subscription information necessary to deliver publish-subscribe messages.

IV. Managing Distributed Processes

The SM resolves three coordination challenges in CMS. The first one is the interaction between service fragments,

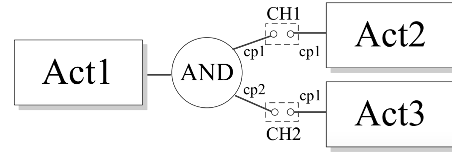

Figure 4. Coordination model

assuming in this paper that communication between activities in the same service fragment is resolved by the service orchestrator. To resolve the fragment interaction problem we define logic gates, corresponding to the most common workflow patterns [6]. The second challenge, related to the communication establishment between limit activities, is tackled by creating communication channels and its subsequent optimization. Finally, a contribution to runtime coordination is performed by defining interactions between logic gates.

A. Fragment interaction concepts and definitions

We use logic gates to enable communication between service fragments, which can be seen as structured workflows. These logic gates follow the **workflow patterns model**, defined by Van Der Aalst et al. [6], corresponding to basic control flow patterns and advanced branching and merging.

A logic gate LG is a tuple (Π, *type*, π_{ctrl}), where Π is a set of publication (Π_P) or subscription (Π_S) *communication points*, *type* $\in$ *{publication, subscription}* indicates whether the logic gate is publisher (its communication points send a message to other fragment) or subscriber (its communication points receive a message from other fragment) of data and π_{ctrl} represents a control point present only in some LG.

Fig. 3 shows the existing communication point in each logic gate. The *Sequence* (SEQ) pattern is modeled with a single communication point and an OR gate with *n* outputs is defined as $\Pi_{OR} \equiv \{\pi_1, \pi_2, ..., \pi_n, \pi_{ctrl}\}$.

We consider the *AND*, *XOR*, *OR* gates (activates all the branches, only one, or an empty or non-empty set of them respectively) as publication gates and *ANDj*, *XORj*, *ORjS* and *ORjD* as subscription gates. *ANDj* transmits the execution when all branches have been activated and *XORj* for any activated branch. We define the *ORjS* and *ORjD* logic gates with a control communication point connected to a previous *OR* gate to support the *Structured Synchronizing Merge* and the *Structured Discriminator* workflow patterns.

The structure between the *OR* and *ORjS/ORjD* gates is blocked until all the active branches are processed. *ORjS* transmits the execution when it receives the first branch activation and *ORjD* delays the transmission until all branches have been activated. The *SEQ* gate (transmits the branch activation) can be used for publication and subscription.

B. Channel creation

Let α_1 and α_2 be two producer and consumer limit activities respectively. We define the predecessor and successor functions such that α_1=*pre*(α_2) and α_2=*suc*(α_1). In order to connect these activities it is needed to introduce a

```
Channel creation: ∀α∈ A
1: ∀ π_j ∈ Π_S from LG(α)
2: commPointId = getId( π_j )
3: search in SDL associated π_j ∈ Π_P from LG(suc(α))
4: create new callback = f(π_j)
5: if: multiple Π_i = {π_i1, π_i2, …, π_in }
6:      set ch(Π_i, π_j)
7:      invoke subscribe(commPointId[], callback)
8: else:  set ch(π_i, π_j)
9:      invoke subscribe(commPointId, callback)
```

Figure 5. Channel creation and optimization pseudocode

publication logic gate after α_1 and a subscription gate before α_2, and, then, create channels between the communication points, as shown in Fig. 4. Thus, we associate each limit activity with a logic gate. We define channel as the tuple (π_p, π_s), where π_p and π_s belong to the communication point set from a publication and subscription gate respectively.

At this stage, channel creation occurs by following the process illustrated in the pseudocode of Fig. 5. Let $A_{OR} \equiv \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be the set of consumer limit activities of a service fragment. After scanning all the communication points of the subscriber logic gates of each activity (1) the identifiers of each communication point are retrieved (2) and used to look up the point of the assigned publication gate into the SDL document (3). After that, a callback address is created and bounded to the communication point (4), a channel is generated (8) and, finally, the subscribe method, from the communication API, is invoked (9).

C. Channel optimization

If the output events of some communication points of a logic gate are equal (they share trigger conditions), it is possible to integrate multiple communication points in the same channel, avoiding generating additional channels (see lines 5, 6 and 7 in Fig. 5). We define optimized channel as a tuple $ch(\Pi_i, \pi_j)$, where Π_i is a subset of the whole communication point set of a publisher logic gate and π_j is a communication point that belongs to the set of a subscriber logic gate. The degree of optimization of a channel *O(ch)* is given by card(Π_i), i.e. the number of communication points that compose the optimized channel. For example, for an AND = (Π_{AND}, *publicator*) gate, *O(ch)* is equal to the total number of communication points of the gate, as this gate replicates the same events in each output. This way, using a logic gate with an optimization level of *O(ch)* means that the number of publication messages is reduced by *O(ch)-1* (since all the communication points share the same pub/sub topic the network broker can use the multicast technique to forward a single publication packet to all subscribers).

D. Runtime coordination

At runtime, control events are transmitted through the created channels. Depending on the type of the logic gate involved in the channel formation the procedure varies:

For the subscription gates, in the case of *SEQ*, once the data is received from the established channel, the SM invokes the *startLimitActivity* method from the coordination API so that the Service Orchestrator executes the limit activity associated to the gate.

In the case of *ANDj*, the SC waits until all its branches receive events to contact the orchestrator. Regarding the *XORj* gate, the SC invokes *startLimitActivity* for each event received from the established channels. In the case of *ORjS*, to implement the *Structure Synchronizing Merge* pattern, the information from the π_{ctrl} of a previous *OR* gate is used to determine how many branches the *OR* gate has activated. The SM waits for the control events in all activated branches and, when the last event arrives, asks the orchestrator to start the execution. If the previous gate is an AND the SM knows that all branches are activated and waits for the arrival of the control event in all branches.

In the case of the *ORjD*, to implement the *Structured Discriminator*, the SM, using the information received from π_{ctrl}, routes the first control event and filters the events from the rest active branches.

For publication gates, in the case of *SEQ* and *AND*, the orchestrator invokes the *endActivity (InstanceId, ActivityId, result)* method from the coordination API when a limit activity completion event arrives; and the SM publishes the control event by all the communication points. In the case of *XOR* and *OR* gates, a decision is required to activate the branches, depending on the result values. Furthermore, the *OR* gate publishes the branch activation decision through the control port.

V. PROTOTYPE EVALUATION

We have implemented the described model and architecture using the MQTT (Message Queue Telemetry Transport) protocol, which is currently in process of standardization. Two different environments (a real simple environment and a simulated complex one) have been defined. The real environment consists of three Android mobile phones (one Samsung Galaxy Note and two Google Nexus S), a MQTT client for Android and an open source message broker called Mosquitto [7], installed on a server with Core i7 1.80 Ghz and 4Gb of RAM. The simulated environment uses the network simulator NS-3 with a MQTT support library that we have implemented [8]. We have used UDP as the transport level protocol.

Our goal is to deploy this architecture and the coordination model within the SmartAgriFood project (under the FI.ICT-2011.1.8 FP7 Work Programme) for the deployment of a service execution system for the Future Internet in grocery stores. We evaluate the performance of the *Item purchasing* service in a real basic test (a single client) and in a simulated environment, adapted to the requirements of the pilot in a grocery store (150 customers in rush hour).

We distribute the service execution timeline into 4 time ranges (T0, T1, T2 and T3) in order to facilitate the interpretation of the data. As shown in Table 1, in the T0-T1 time range the creation of all communication channels for

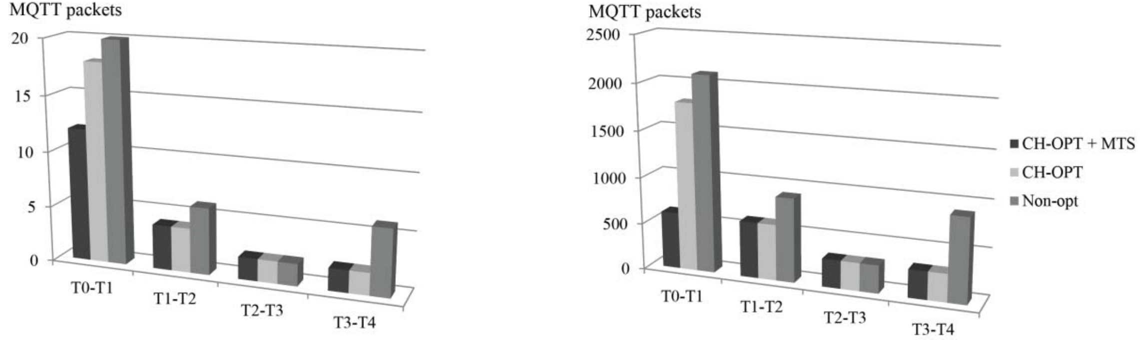

Figure 6. Optimization comparison between real scenario (left) and simulated scenario (right)

TABLE I. TIME RANGES FOR ACTIVITY EXECUTION

Time range	Executed activities
T0-T1	*Channel creation*: Subscriptions to 1,2,3 and 4
T1-T2	Select item (1), Get product info (2)
T2-T3	Check profile compatibility (3)
T3-T4	Buy item (4)

the service is produced and, in the following ranges, activities are executed and limit activity completion events are sent to subscribers.

We apply two levels of optimization. The first one, ***channel optimization***, is related to the publish-subscribe model and specifically to the broker's capability of using the multicast technique to send a single publish to multiple subscribers of the same topic, i.e. the same channel (we have explained this in Section IV.C). We perform the channel optimization technique over the *Item purchasing* service, reducing the total channels from 7 to 4 in the basic scenario and from 1050 to 600 in the simulated scenario.

The second optimization level is called ***multiple topic subscription*** (MTS) and enables the establishment of all channels associated to a service fragment with a single MQTT *Subscribe* message. MQTT makes possible the use of this technique.

As shown in Fig. 6, the optimization levels represent a considerable advantage from using the non-optimized model, in which, in the worst case, the broker has to manage 20 MQTT packets in the real scenario and 2106 packets in the simulated one. However, in the T2-T3 time range, the optimizations do not reduce the amount of packages handled by the broker. This is because the *Check profile compatibility* activity only publishes control events for the *Display recommendation* activity so that the multicast technique is not utilized.

After MTS is applied, a considerable reduction in the number of packets in the T0-T1 subscription period is achieved and, as explained above, this reduction does not affect other time ranges, as subscription message interchange does not exist in the rest of steps.

VI. RELATED WORK

Related work tries to solve the problem of communications between mobile workflows from the Internet of Things. Some works are related to the field of user/prosumer participation [9]. Generally, service communication is based on a data-driven approach, so that services can be created easily, with some composition or mashup tools. Although there are some studies that combine data-driven composition with control flow specification [10], we consider that the coordination between services based on the transmission control events (control-driven service composition) allows the execution of more complex cooperative services. The Presto framework [11] provides a service development platform for user participation in Smart workflows, based on business processes. Our work also relies on user interaction with elements of the Web of things through their mobile devices. However, we mainly focus on the problems of service fragment coordination. Thus, we find more similarities in the field of decentralized service orchestrations [12] or choreographies [13].

To manage this coordination some authors [14] propose the use of design patterns as reusable parts to compose services. In our work we base on workflow patterns, specifically in the patterns defined by van der Aalst et al. [6], to model the connections between service fragments. Van der Aalst et al. also point out the importance of unique identification of the elements of the process [13] and the correlation problem [15], which we described in Section III.

The interaction between service activities is often described in a SDL document, expressed in a standard language like BPEL or BPMN, or some other languages adapted to the service logic [16]. In our work we leave the door open to the possibility of using any service definition language compatible with the used workflow patterns for our SDL document.

The information exchange between coordinated service fragments has been less addressed in related work. However, some proposals related to workflow decentralization [17], task communication [18] and distributed orchestrations [19] have been found. Some authors [19] choose to solve the activity wiring using WSDL interfaces and SOAP messages. Other solutions use a tuple space [17] to manage the

execution of scientific workflow applications by subscription/notification methods. In other work [18], virtual channels are used between sending and receiving tasks to ensure data communication.

In our work we use the publish-subscribe communication paradigm [1] as alternative to de-synchronize producers and consumers of information, and ensure functional decoupling in time and space [20]. Pub-Sub based models can provide advantages [21] over classic *polling*, which can overuse services and networks' resources by continuously querying information.

VII. Conclusion and Future Work

This work has defined a cooperative service execution model for mobile environments in scenarios from the Web of Things. In this model, user mobile devices execute service fragments that access Web objects. The need to coordinate these elements at the data plane (transfer of information produced by users or Web objects to other terminals) and the control plane (synchronization and management of the execution flow of tasks and activities) has been detected. This paper contributes to solve three coordination challenges detected in such environments. The interaction between service fragments is resolved by introducing logic gates between limit activities, based on well-known workflow pattern. The channel creation and optimization contribute to the communication establishment between limit activities, and finally, the runtime coordination is described by the interactions between the different modules of the defined architecture. The validation of this work in both real and simulated environments allow us to check that the optimization mechanisms supported by the utilization of a Pub-Sub underlying communication model for event transmission and, particularly, the MQTT protocol, offers an improvement over the basic model without optimizations.

As future work, in the field of coordination of distributed services, we will investigate automatic workflow partitioning mechanisms and user participation in the design or personalization of the execution process of workflow activities, as an evolution of our work on the prosumer user [9]. In the communication layer, we will investigate on Pub-Sub broker federation protocols to support service deployment in real environments with higher performance requirements.

References

[1] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," ACM Computing Surveys, vol. 35, no. 2, pp. 114-131, June 2003.

[2] X. Wang, Y. Zhang, and H. Shi, "Access Control for Human Tasks in Service Oriented Architecture," IEEE International Conference on e-Business Engineering, Oct. 2008, pp. 455-460.

[3] W. Fdhila, M. Dumas, and C. Godart, "Optimized decentralization of composite web services," 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing, Oct. 2010, pp. 1-10.

[4] R. Alcarria, U. Aguilera, T. Robles, D. López-de-Ipiña, and A. Morales, "Ubiquitous Capability Access for Continuous Service Execution Mobile Environments," V International Symposium on Ubiquitous Computing and Ambient Intelligente, Dec. 2011.

[5] P. Eugster, "Type-based publish/subscribe: Concepts and experiences," ACM Trans. Program. Lang. Syst., vol. 29, no. 1, 6, Jan. 2007.

[6] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow Patterns," Distrib. Parallel Databases, vol 14, no. 1, pp. 5-51, July 2003.

[7] Moskitto, An Open Source MQTT v3.1 Broker. Web page: http://mosquitto.org

[8] MQTT for NS-3 SourceForge project Web page: https://sourceforge.net/projects/mqttforns3/

[9] R. Alcarria, T. Robles, A. Morales, and S. González-Miranda, "New Service Development Method for Prosumer Environments," Proc. of the Sixth International Conference on Digital Society, Jan.-Feb. 2012.

[10] F. Rosenberg, F. Curbera, M.J. Duftler, and R. Khalaf, "Composing RESTful Services and Collaborative Workflows: A Lightweight Approach," IEEE Internet Computing, vol. 12, no. 5, pp. 24-31, Sept.-Oct. 2008.

[11] P. Giner, C. Cetina, J. Fons, and V. Pelechano, "Developing Mobile Workflow Support in the Internet of Things," IEEE Pervasive Computing, vol. 9, no. 2, pp. 18-26, April 2010.

[12] M. Jayaprakash, M. Shanmugam, P. Manikandan, and S. Shivaraj, "Decentralized Service Orchestration by Continuous Message Passing," International Journal on Computer Science and Engineering, vol. 02, no. 05, pp. 1627-1632, 2010.

[13] D. Fahland, M. de Leoni, B. F. van Dongen, and W. M. P. van der Aalst, "Many-to-Many: Some Observations on Interactions in Artifact Choreographies," Proc. of the ZEUS conference, pp. 9-15, 2011.

[14] M. T. Tut and David Edmond, "The Use of Patterns in Service Composition," Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, pp. 28-40, 2002.

[15] D. Fahland, M. De Leoni, B. F. Van Dongen, and W. M. P. Van Der Aalst, "Conformance checking of interacting processes with overlapping instances," Proc. of the 9th International Conference on Business Process Management, pp 345-361, 2011.

[16] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: yet another workflow language," Information Systems, vol. 30, no. 4, pp. 245-275, June 2005.

[17] R. Ranjan, M. Rahman, and R. Buyya, "A Decentralized and Cooperative Workflow Scheduling Algorithm," Proc. of the 8th IEEE International Symposium on Cluster Computing and the Grid, pp. 1-8, May 2008.

[18] S. Narayanan, L. Devaux, D. Chillet, S. Pillement, and L. Sourdis, "Communication service for hardware tasks executed on dynamic and partial reconfigurable resources," 19th IEEE/IFIP International Conference on VLSI and System-on-Chip, pp. 196-199, Oct. 2011.

[19] U. Yildiz and C. Godart, "Centralized versus Decentralized Conversation-based Orchestrations," The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, pp. 289-296, July 2007.

[20] P. Costa, C. Mascolo, M. Musolesi, and G.P. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks," IEEE Journal on Selected Areas in Communications, vol. 26, no. 5, pp. 748-760, June 2008.

[21] L. Fiege, M. Cilia, G. Muhl, and A. Buchmann, "Publish-subscribe grows up: support for management, visibility control, and heterogeneity," IEEE Internet Computing, vol. 10, no. 1, pp. 48- 55, Jan.-Feb. 2006.