GCO: A Generic Collaboration Ontology

Germán Sancho, Thierry Villemur, Saïd Tazi
CNRS; LAAS; 7 avenue du colonel Roche, F-31077 Toulouse, France
Université de Toulouse; UT1, UT2, UPS, INSA, INP, ISAE; LAAS; F-31077 Toulouse, France
{*sancho,villemur,tazi*}*@laas.fr*

***Abstract*—Collaborative systems provide support for users that work together for achieving a common goal. In the past years, several ad-hoc models have been proposed in order to model collaborative activities in such systems. This paper proposes a shareable model for collaboration, the Generic Collaboration Ontology, that can be used by systems in run-time in order to implement session management and component deployment services. This model is an OWL ontology containing SWRL rules, and therefore it can be processed with standard Semantic Web tools in order to perform inference. This ontology is generic because it does not contain domain-specific knowledge, and it can be extended for specific domains.**

***Keywords*-ontology, collaboration, OWL, SWRL, session, inference**

I. INTRODUCTION

Collaborative applications are distributed systems especially designed to provide support to groups of users that act in a coordinated way in order to achieve a common goal. Such applications have been studied since the 1990s in the domain called Computer-Supported Collaborative Work (CSCW). These studies include concepts from very different domains such as Social Sciences, Cognitive Sciences, Human-Machine Interfaces and Distributed Computing in order to maximize the efficiency and ergonomics of CSCW systems.

In the past years, a variety of models and techniques have been developed in the CSCW domain. Applications using these models rely on them in order to represent the possible collaboration schemas and the current system configuration at a given time. These models have been used with more or less success in the implemented systems. However, although many of the modeled elements are common, very often these models are ad-hoc or application-specific, thus limiting their reusability and extensibility. Moreover, each model is described with a different formalism or language, or even worse, they are hard-coded inside the application. This results in a limited interoperability of the systems based in such models. It would be preferable to have common models shared between several applications. These models should be described in standard languages allowing them to be processed with standard tools.

Another disadvantage of existing collaboration models is that they are not well suited for enabling a model-based deployment service. The function of such a service is to deploy (i.e., download, install and configure) the application components necessary on each user device in order to implement the collaboration schema indicated by the model. For example, if, at a given time, the model indicates that an audio connexion must exist between two users, then audio components must be deployed on both users' devices in order to manage that connexion. In old systems, where collaborative software was monolithic, this function was performed statically, and therefore a deployment service was not needed. However, as the systems become more and more dynamic (e.g., in the context of Ubiquitous and Pervasive Computing), deployment needs to be adaptive at run-time, so a dynamic deployment service is needed.

The goal of this paper is to provide a shareable model enabling the development of collaborative applications requiring session management and dynamic component deployment. This model is represented in the OWL ontology language. As far as we know, a common ontology for modeling collaborative sessions has not been proposed yet.

Ontologies have received great attention in the recent years, due to their use for knowledge representation in the Semantic Web domain. The Semantic Web was proposed by Tim Berners-Lee [1] in order to enrich data contained in the World Wide Web. The main idea is to add metadata describing regular Web data (which is only human-readable) in order to make it *understandable* by machines, thus enabling the automation of distributed processing over the Web. Metadata describing the semantics of contents is expressed in several languages such as RDFS (Resource Description Framework Schema, based on RDF) and OWL. OWL (Web Ontology Language) is the Semantic Web standard for describing ontologies [2], which are common vocabularies allowing to model and represent knowledge. The main elements of ontologies are concepts, relations (between two concepts), individuals and axioms. All these elements are based on well-known formalisms such as Description Logics [3] in the case of OWL. Thus, knowledge can be automatically deduced by *inference engines* or *reasoners* (for example, Pellet [4]). These software elements can process an ontology in order to make explicit the implicit knowledge contained in them. Also, rules (expressed in SWRL, the Semantic Web Rule Language [5]) may be included in ontologies and processed by reasoners. Rules add some expressivity to OWL constructs.

 ISBN: 978-1-61208-104-5

For these reasons, OWL seems a good choice for the representation of a shareable collaboration model. Standard tools and frameworks are available and can be used for building and querying model instances. It also enables the sharing of collaboration concepts between several applications. Moreover, the use of reasoning and rules is very useful. For example, they allow deducing, at run-time, the deployment schema that corresponds to a given collaboration configuration.

The contents of this paper are organized as follows. Section II provides an overview of existing collaboration models in CSCW domain in order to analyze the elements to retain in our collaboration ontology. Section III details the elements of the GCO and explains the principles that have guided its design. Section IV presents some guidelines for using an ontology as the core model of a run-time system and provides some examples of systems using the GCO. Finally, Section V concludes and provides some perspectives for future work.

II. Existing collaboration models

This section provides a brief overview of existing models in the CSCW domain and the elements that have to be present in a model for collaborative activities enabling session management and component deployment.

The main models published in the literature are based on set formalisms [6] or first-order logic [7] in order to describe unstructured sessions. In the case of structured sessions, i.e., sessions where relations between members are clearly detailed (e.g., group coordination), models are based on graphs. The modeled elements take part in the definition of group activities. Some of the main elements found in these models are, users, hardware devices, and software tools.

Baudin et al. [8] propose a model capturing the most common elements found in previous systems. The goal of this model is to explicitly represent relationships of information exchange between users in order to keep a tight coupling between communication and network layers. Therefore, this model, which is graph-based, enables the construction of session management services. In this model, a collaborative session is composed of a set of three elements to be managed: users, tools and data flows. Such elements are represented in a unified graph-based model. Vertices represent users, and edges define the relationships between them. An edge going from user U_1 to user U_2 means that U_1 transmits data through a selected tool (e.g., a videoconference tool) to U_2. The type of data and the tools that handle data sent through a flow are defined by edge labels.

The proposed collaboration model is based on data producer/consumer relationships, to represent and process data exchanges for synchronous and interactive work sessions, that. Such sessions handle interactive data flows (e.g. video, real time audio).

This model is simple enough to be easily handled by session designers for various collaborative configurations. Moreover, instances of this model can be automatically taken into account by services or platforms that can be configured by the model. The sessions explicitly designed are managed by model-based platforms.

This model also considers the dynamics of the session: the current session configuration evolves whenever entries or exits of members occur. In the same way, role and function changes of the members already present in the collaborative session introduce modifications of the current graph (for instance a passive user becomes active by making an action and therefore new flows have to be set up). At any time, the current session configuration corresponds to a valid graph.

III. The Generic Collaboration Ontology

This section presents the Generic Collaboration Ontology (GCO)[1]. First, the design principles used for the design of the GCO are presented. Then, the elements present in the GCO are explained in detail.

A. Design Rationale

1) Ontology language: the GCO is expressed in OWL, which is the current web standard for ontology description. Since the expressivity of OWL is not enough for some of the required relations, rules are used. Rules are expressed in SWRL. Standard, open-source tools are available for processing OWL ontologies and SWRL rules.

2) Genericness: the GCO has been designed in order to be as generic as possible. This means that it may be used to model collaboration in any application, regardless of the domain. In this aspect, the GCO can be viewed as an upper ontology that can be extended by domain ontologies in order to model domain-specific concepts and relations. The simplest way of extending this ontology is to use *inhreritance* by defining sub-conceps and sub-relations of the concepts and relations present in the GCO (*is-a* relation).

3) Multi-Layered Architectures: the genericness of the GCO means that it can be used inside a multi-layered architecture. In such case, the GCO may be the core model of the layer that handles collaborative sessions. Domain-specific data may be handled in upper layers, while low-level data, such as network connexions, can be handled in lower layers.

4) Ontology contents: Since the main goal of the GCO is to support collaboration in run-time systems, the concepts and relations present in this ontology have been chosen among those that have been used in collaboration models until today (i.e., those presented in the previous section). For example, it contains concepts representing tools, flows, roles, etc. In order to enable dynamic deployment services based on the GCO, some other elements such as components,

[1]The GCO is a available online at http://homepages.laas.fr/gsancho/ontologies/sessions.owl

 ISBN: 978-1-61208-104-5

nodes hosting devices, etc. have been added to this ontology. The rules associated to the GCO are also designed in order to enable a simpler deployment process by making explicit the deployment schema that must support the collaborative activity described by the ontology.

5) Simplicity: the contents of the GCO have been chosen to enable a complete modeling of collaborative sessions. However, only basic elements have been retained. Therefore, this ontology is lightweight and reasoning and rule processing may be performed at run-time without heavy overhead. Moreover, this simplicity eases the task of designers willing to use or extend this ontology for domain-specific applications.

B. Description of the GCO

The main elements of the Generic Collaboration Ontology are represented in Figure 1. Concepts are represented as round-cornered rectangles, while relations between concepts are represented as arrows going from one concept (the domain of the relation) to another concept (the range of the relation). Individuals are represented as dash-line rectangles.

The basic concept of this ontology is `Node`. A node represents a communicating entity which takes part in a collaborative activity. Nodes may represent human users (i.e. human-controlled software components) but also autonomous software components, agents, etc. The nature of entities is not represented in this generic ontology.

Nodes play a role in the collaborative activity which determines the position of the entity within the collaborative group. Depending on their role, entities will have different functions in the group and they will need to communicate with different group members in order to better achieve the collaboration's goal. This is captured by the concept `Role`. Therefore a relation called `hasRole` links the `Node` and `Role` concepts.

Whether a node is an autonomous software component or it is a human-controlled component, it has to be executed on a physical machine. Such machines are represented by the concept `Device` (`Node` is linked to `Device` by the property `hasHostingDevice`). The execution context of the node will depend on the resources of the device that hosts it. At the present time, a minimal set of device properties is considered, containing IP addresses (`hasIpAddress`), operating system (`hasOS`), available memory (`hasAvailableMemory`), CPU load (`hasAvailableMem`) and battery level (`hasBatteryLevel`). Additional properties could complete this initial list in order to better capture and reason about the execution context.

Entities take part in the collaborative activity by sending and receiving data to/from other entities. The concept `Flow` represents a communication link between two entities. Therefore, `Flow` is linked to `Node` by two properties: `hasSource` and `hasDestination`. In this ontology, flows are considered as being unidirectional, and thus if a bi-directional communication between two nodes is required, it will be represented by two instances of `Flow` with two opposite directions. The `hasSource` property is functional, while `hasDestination` is not functional, i.e., a flow has a single source node, but it may have several destination nodes (thus representing multicast links).

In order to represent the nature of data exchanged through a flow, the `Flow` concept has a functional property called `hasDataType` that relates it to the `DataType` concept. Possible values of data types are captured through the `DataType` individuals `audio`, `text` and `video` (additional data types could be considered). The subconcepts of `Flow` differ in the value of their data type: `AudioFlow`, `TextFlow` and `VideoFlow` (not represented in the figure).

In order to handle data flows, nodes use external software components that are deployed on the same device as them. This enables the separation between business code (implemented in entities' components) and collaboration code (implemented in such external components). These external components are represented by the `Tool` concept. Tools are composed of several components, e.g., a sender component and a receiver component. Therefore the `Tool` concept is related to a concept called `Component` through the property `hasComponent`. Since components handle flows, a property called `managesFlow` links `Component` and `Flow`. Components have a data type (the same as the data type of the flow that they manage) and are deployed on a single device (`isDeployedOn` property which links `Component` and `Device`). The `Component` concept has several subconcepts that represent components depending on the handled data type (`AudioComponent`, `TextComponent` and `VideoComponent`, not represented in the figure) and on the direction of the handled flows (`SenderComponent` and `ReceiverComponent`). `SenderComponent` and `ReceiverComponent` are linked to `Flow` by two sub-relations of `managesFlow`: `sendsFlow` and `receivesFlow`, respectively.

Finally, the `Session` concept represents a set of flows belonging to the same collaborative activity. The `hasFlow` property relates a session to a flow. The inverse property, `belongsToSession`, is functional, i.e., a flow belongs to a single session. Since flows are related to nodes, nodes are indirectly related to one or more sessions depending on the flows that connect them to other entities.

C. Generic collaboration rules

A set of 6 SWRL rules is associated to the GCO in order to express some additional knowledge and to enable deployment-related inference. This section provides a description of the main rules associated to the GCO.

Let us consider the first rule:

 ISBN: 978-1-61208-104-5

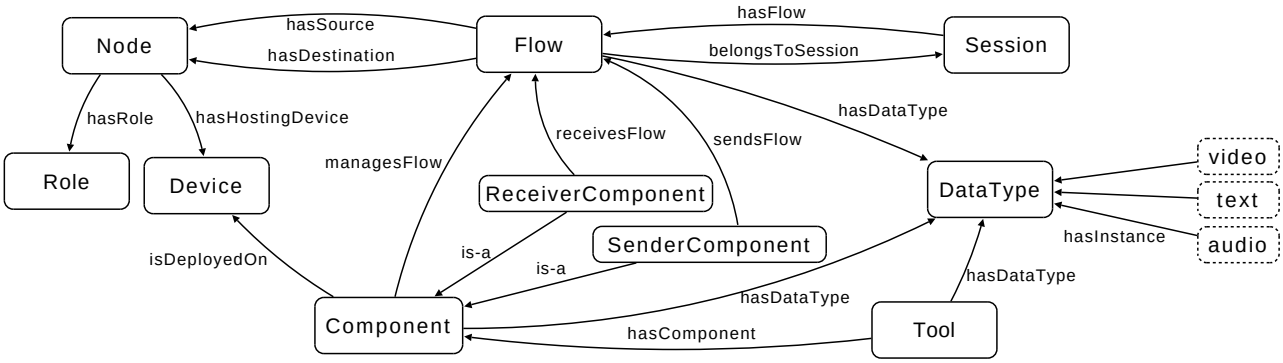

Figure 1. Main elements of the Generic Collaboration Ontology.

```
AudioFlow(?af) => hasDataType(?f,audio)
```

This rule allows expressing that the data type of `AudioFlows` is `audio`. Similar rules exist for video an text flows. The second rule is:

```
Flow(?f) & belongsToSession(?f,?s)
& hasDataType(?f,?dt)
& hasSource(?f,?src)
& hasDestination(?f,?dst)
& swrlx:createOWLThing(?sc,?src,?s)
& swrlx:createOWLThing(?rc,?dst,?f)
=> SenderComponent(?sc)
& hasDataType(?sc,?dt)
& isDeployedOn(?sc,?src)
& sendsFlow(?sc,?f)
& ReceiverComponent(?rc)
& hasDataType(?rc,?dt)
& isDeployedOn(?rc,?dst)
& receivesFlow(?rc,?f)
```

This rule states that, whenever a flow belonging to a session is found between two nodes, a `SenderComponent` has to be present in the source node and a `ReceiverComponent` has to be present on the destination node. These components send and receive, respectively, the flow, and they have the same data type as the flow. This rule uses the SWRL built-in `createOWLThing` that allows creating new individuals. Please note that the first `createOWLThing` matches the source node and the session, while the second matches the destination node and the flow. This choice enables multicast flows where a single sender component sends several flows to several receiver components.

These rules are generic w.r.t. collaboration, because they do not depend on the particular domain of the collaborative application.

IV. Use of the GCO in Runtime Systems

As explained before, the GCO has been designed to be used at run-time as the core model of systems providing support for collaborative activities. This section details this use and gives some examples of collaborative systems that use the GCO.

A. Using ontologies in run-time systems

An ontology may be considered as a meta-model which describes the possible concepts and relations of a given domain. Actual instances of this meta-model are represented by individuals of the concepts available in the ontology. Such individuals (and the relations between them) may be used in order to represent the state of the application at a given time. Relations and concepts are fixed at design-time, while individuals representing the state are created at run-time. In order to use an ontology as the core model in a run-time system, the system must be able to perform the following tasks:

- read the concepts and relations existing in the ontology;
- read/modify the individuals existing in the ontology and the values of their properties;
- create new individuals and set the values of properties;
- perform reasoning and rule processing over the ontology and its individuals.

The tools made available in the context of the Semantic Web enable the execution of these tasks. Implementations of APIs like OWL API [9] or Protégé-OWL API [10] allow performing the four first tasks programmatically. Reasoning can be performed by OWL reasoners such as Pellet [4]. Most reasoners are also capable of processing SWRL rules; however, SWRL *built-ins* are not fully supported yet. Therefore, it may be necessary to use rule engines such as Jess[2] in order to process rules containing such built-ins. Both reasoners and rule engines can be executed programmatically in order to process in-memory OWL models.

[2]http://www.jessrules.com/

 ISBN: 978-1-61208-104-5

The presented tools enable the creation of programs that modify the individuals of an ontology in order to represent the current state of of the system at every time. However, if reasoning and rules are used to deduce knowledge from individuals, the monotonic nature of OWL inference may represent a problem. Indeed, OWL does not support non-monotonic inference [2], [11]. This means that reasoning and rules can not modify (addition or removal) the information contained in an ontology. They only allow finding implicit knowledge contained in the ontology and making it explicit. For example, if the processing of a rule in the GCO results in the creation of an individual of the class `Flow` whose source is $node_A$ and whose destination is $node_B$, this information will always remain in the ontology. No other rule can remove it afterwards. If the application needs to remove this individual in order to reflect a new state, it can do it programmatically, but it can be very tricky and unpractical (or even impossible) to keep a track of which information has been inferred and to decide what has to be deleted at every moment.

The solution to this problem is to use the inference capabilities of OWL in a capture-inference-results loop such as the one depicted in Figure 2. The first step if to capture the *state of the world* that is modeled by the ontology. This is done by the code of the application using the ontology. Then, this state is introduced in the ontology by creating individuals of the available concepts and by establishing relations between these individuals through object properties. The result is a set of ontology individuals related between them reflecting the state of the modeled world. For example, if a system using the GCO has, at a given time, three users connected, there will be three individuals of the class `Node` representing these users, each one related to one individual of the class `Role` representing the role of the user in the group. Once this model has been built, the resulting ontology can be processed by the inference and rule engines. The result of this step is a new version of the ontology where new individuals and relations may have been introduced. In the example of the GCO, this step results, e.g., in the creation of several individuals of the class `Flow` having the existing `Node` individuals as values for the properties `hasSource` and `hasDestination`. This new model contains the new state of the world that has to be achieved. Therefore, the application code can read this model and perform the actions necessary in order to achieve this state. In the GCO example, the new `Flow` instances will be found and therefore the application will effectively set up this new flows between the users' devices. Whenever the state of the world is changed (e.g., when one of the users leaves), the whole loop has to be repeated in order adapt the response of the application to the new state.

The presented loop is discrete; the results of a step are valid until the next change in the state of the world. Whenever a change occurs, the whole loop is executed again in order to get the new results. Because of the monotonicity of OWL inference, the new state can not be represented by directly modifying the resulting ontology individuals; it would be necessary to delete all the inferred knowledge. Otherwise, the next inference process will result in an incoherent (inconsistent) ontology.

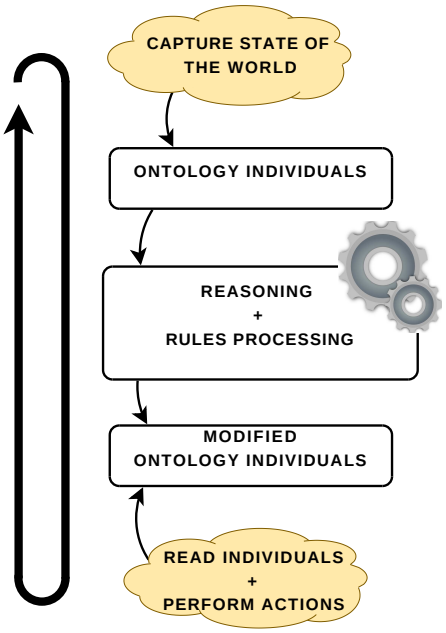

Figure 2. Capture-inference-results loop for run-time systems using ontology reasoning.

B. Systems using the GCO

In a recent work [12], [13] we have proposed a modeling approach and a framework enabling the design and implementation of collaborative applications for ubiquitous computing environments. Ubiquitous Computing provides a new range of challenges and opportunities for collaborative applications. Indeed, the fact that mobile users carrying smart devices are immersed in intelligent environments and the availability of contextual data may greatly enhance the possibilities of collaborative applications. The presented framework uses a multi-layered approach for enabling applications that can be adapted to both high-level requirements and low-level constraints. The proposed layers are called application layer, collaboration layer and middleware layer. The core model of the application layer is the GCO. The models of the application layer are domain-specific ontologies and rules that extend the GCO in order to capture the specific collaboration knowledge of the considered domains. This ontology is processed in a capture-inference-results loop as explained in the previous section. The results of this process are translated into a graph model, which is the core model of the middleware layer, and then it is processed with graph-transformation techniques. This allows

 ISBN: 978-1-61208-104-5

taking into account high-level requirements at the application and collaboration layers, while low-level requirements are handled at the middleware layer. The resulting graph is a low-level, detailed deployment descriptor that is used by the deployment service in order to carry out the deployment of components needed for supporting collaborative activities.

Bouassida Rodriguez et al. [14] propose an Emergency Response and Crisis Management System (ERCMS) that uses the GCO as the core collaboration model. ERCMSs support collaboration of policemen, firemen and physicians in order to better handle critical situations such as fires, earthquakes, terrorist attacks, etc. The proposed system is adaptive and takes into account the evolution of communication and processing resources in order to guarantee the required QoS properties. Non-functional properties are modeled in an OWL ontology that extends the GCO by relating `QualityAttributes` to the `Component` and `Device` concepts of the GCO. A domain-specific OWL ontology is used in order to describe ERCMS-specific collaborative knowledge. This ontology extends the GCO by providing sub-concepts of the `Flow` and `Node` concepts of the GCO. Several SWRL rules are provided for implementing adaptation transformations that handle context changes. However, the authors do not explain how their system uses the proposed ontologies and rules at run-time, neither they explain how the problem of the monotonic nature of OWL inference is handled.

V. Conclusion and Future Work

This paper has presented the GCO, a generic collaboration ontology that represents knowledge about session-oriented collaboration. This ontology is generic because it can be extended in order to model domain-specific collaboration knowledge. Rules associated to the GCO allow implementing ontology-driven systems using the GCO as their core collaboration model for implementing session management and deployment services. Explanations of how this usage of the GCO in run-time systems have also been provided.

Perspectives for future work include designing domain-specific ontologies that extend the GCO for several domains and building systems that use the GCO as their model for collaboration activities. The framework described in Section IV-B is currently being implemented, as well as proof-of-concept applications that use it for modeling and implementing collaborative activities.

Acknowledgment

The authors would like to thank ITEA2 USENET project for supporting this work.

References

[1] T. Berners-Lee, “A Roadmap to the Semantic Web,” W3C, Sep. 1998, http://www.w3.org/DesignIssues/Semantic, last access date: July 2010.

[2] M. K. Smith, C. Welty, and D. L. McGuinness, “OWL Web Ontology Language Guide,” W3C Recommendation, Feb. 2004, http://www.w3.org/TR/owl-guide/, last access date: July 2010.

[3] D. F. Baader, D. Calvanese, D. L. McGuinness, P. Patel-Schneider, and D. Nardi, *The Description Logic Handbook. Theory, Implementation, and Applications*.

[4] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, “Pellet: A practical OWL-DL reasoner,” *Web Semant.*, vol. 5, no. 2, pp. 51–53, 2007.

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

[6] W. K. Edwards, “Session management for collaborative applications,” in *CSCW ’94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM, 1994, pp. 323–330.

[7] M. Rusinkiewicz, W. Klas, T. Tesch, J. Wäsch, and P. Muth, “Towards a Cooperative Transaction Model - The Cooperative Activity Model,” in *Proceedings of the 21th International Conference on Very Large Data Bases*. San Francisco, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 194–205.

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

[9] M. Horridge and S. Bechhofer, “The OWL API: A Java API for Working with OWL 2 Ontologies,” in *OWLED*, ser. CEUR Workshop Proceedings, R. Hoekstra and P. F. Patel-Schneider, Eds., vol. 529. CEUR-WS.org, 2008.

[10] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, “The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications,” *The Semantic Web - ISWC 2004*, pp. 229–243, 2004.

[11] J. McCarthy, “Generality in artificial intelligence,” *Commun. ACM*, vol. 30, no. 12, pp. 1030–1035, 1987.

[12] I. Bouassida Rodriguez, G. Sancho, T. Villemur, S. Tazi, and K. Drira, “A model-driven adaptive approach for collaborative ubiquitous systems,” in *AUPC 09: Proceedings of the 3rd workshop on Agent-oriented software engineering challenges for ubiquitous and pervasive computing*. London, United Kingdom: ACM, 2009, pp. 15–20.

[13] G. Sancho, I. Bouassida, T.Villemur, S. Tazi, and K. Drira, “A model-driven adaptive framework for collaborative ubiquitous systems,” in *Proceedings of the 9th Annual International Conference on New Technologies of Distributed Systems, NOTERE 2009*, Montreal (Canada), July 2009, pp. 233–244.

[14] I. Bouassida, J. Lacouture, and K. Drira, “Semantic Driven Self-Adaptation of Communications Applied to ERCMS,” in *The 24th IEEE International Conference on Advanced Information Networking and Applications*, Perth (Australia), Apr. 2010.

 ISBN: 978-1-61208-104-5