SWRL-based context awareness for application servers hosting digital services

Yves-Gael Billet, Christophe Gravier, Jacques Fayolle

SWRL-based context awareness for application servers hosting digital services

Yves-Gaël Billet, Christophe Gravier, Jacques Fayolle

Université de Lyon, F-42023, Saint-Etienne, France;
Université de Saint-Etienne, Jean Monnet, F-42000, Saint-Etienne, France;
Télécom Saint-Etienne, école associée de l'Institut Télécom, F-42000, Saint-Etienne, France;
LAvoratoire Télécom Claude Chappe (LT2C), F-42000, Saint-Etienne, France.
{yves-gael.billet, christophe.gravier, jacques.fayolle}
@telecom-st-etienne.fr

Abstract. As the number of context-aware applications increases in the real world, it can be quite difficult to deploy such applications in traditional application servers, which are context-agnostics systems. To address this challenge, we propose a novel approach for easing the deployment of context-aware applications into application serversContext is encoded within an OWL-driven knowledge base. We couple this knowledge base with SWRL rules to encode context-awareness thresholds. SWRL rules are not predefined in the application server. They are instead embedded inside the application bundle built by the developer, next to the business logic of the application. At the application deployment time, SWRL rules are extracted to the knowledge base in order to monitor the relevant context for the application to be deployed. At runtime, the context of each session of the application is monitored in the knowledge base. When a rule is triggered (a context-awareness threshold is reached), a broker inside the application server notifies the application so that it adapts its behavior by switching to a more relevant modality. We show how our approach eases the work of developers for building context-aware application by using our context-aware framework.

Keywords. Rules, SWRL, Semantic, context-awareness computing, middleware, digital services, application server.

1 Needs for a Semantic CAS

We want to provide context awareness for digital service. The idea is to provide a context-aware framework for software architects in order to implement context awareness in their digital services. These services usually run on Applications Servers (henceforth AS) and are consumed through a network using the client-server paradigm. In this situation, context is composed of information about the network and features of the terminal. Context is session specific since applications are hosted on an AS, context-awareness could be externalized as a framework on them.

The objective is to provide a system that supports sensing, perception and reasoning over context. In this way, software developers could only focus on the

business logic and adaptation behavior of their application. Behaviors are specific for each application. Software developers must describe and model adaptation behavior for their digital services and communicate it to the context-aware framework.

Semantic Web technologies allow describing context and are comprehensible by both humans and computers and offer a loosely-coupling with the programming language.

Ontologies are used to model domain by describing concepts of the domain and relationships between those concepts [12]. The Web Ontology Language (OWL) was developed to provide a way to represent knowledge understandable by machines using a semantic formalization, in order to facilite computers to interpret human knowledge.

The Semantic Web Rule Language (SWRL) allows us to encode context rules on Horn clauses and designed for OWL. As presented in [13] rules are of the form of an implication between an antecedent (body) and consequent (head). SWRL are logical expression encoded in Conjunctive Normal Forms. It used to classify individuals according conditions. As stated in [14], SPARQL can be used to interrogate an ontology but it is RDF centric so not efficient when using OWL. O'Connor et al. proposes a Semantic Query-enhanced Web Rule Language (SQWRL) based on SWRL.

We propose a semantic context-aware system using OWL, SWRL and SQWRL to describe context and model the adaptation behavior for applications.

2 Related works

2.1 Context-awareness

Among the existing context-aware definitions [1-3], we base our definition from [2]. It defines context as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

We apply this definition in the case of digital services: a context-aware multimedia application is an application (entity), which can use the elements of context (any information that can be used to characterize the situation) to adapt his behavior in order to provide the corresponding interaction to the terminal.

2.2 Architectures of context-aware applications

J. Coutaz et al. present in [4] a global architecture for context-aware applications based on levels of abstraction for a general-purpose context-aware system. They define four levels:

- **Sensing layer** provides numeric observables.
- **Perception layer** provides symbolic observables, which are interpretations from the numeric observable (i.e. transform GPS information in location name).

- **Situation and context identification layer** identifies context and propose adaptation. It is the reasoning core.
- **Exploitation layer**. It is an adapter between the application and the context infrastructure.

Baldauf et al. in [5] also use a similar layered architecture to compare middleware and frameworks for context-aware systems.

2.3 Existing works on context-aware systems

Current approaches in context-aware frameworks make a clear separation between context acquisition, context processing and use. Thus, to create a separation of concerns as proposed by Dey [2]. He proposes a conceptual framework for supporting context-aware applications. The implementation of this framework is known as the Context-Toolkit [2]. It aims to ease development and evolution of context-aware applications using an object-based approach. It must be seen as an API for context-aware application. Each object from the API, which can be use to construct an application, has a role (i.e.: collect, transform, aggregate and serve context). The API approach creates a tight coupling between context processing and application.

More recent works, such as CoBrA or SOCAM are infrastructure-based approach rather than API. Hong et al. point out advantages of infrastructure for providing context abstraction [6].

CoBrA project from Chen et al. [7] is agent-oriented. A central unit, so called, a context broker, maintains and manages context on behalf of agents. An agent could be an application running on mobile devices, a service provided by a room, a web service, etc). The broker collects information about context and shares it with agents. This design addresses the issue for providing context-awareness to resource-limited computing devices.

The Service-Oriented Context-Aware Middleware (SOCAM) project introduces by Gu et al. [8] exists as middleware. In SOCAM, different services working together acquire, process, reason and deliver context to agents. The main contribution of this work is the context model, which uses ontology through OWL. The use of ontology allows them to describe context in a semantic way that is independent of programming language.

In context-awareness, the usage is a key element. Works presented in this section are related to the physical world. The main use is to provide services fitted to users activity (e.g.: forward calls to voice mails when users is currently sleeping in the bedroom, switch off light if the room is empty, etc).

3 Rule driven context-awareness

Our motivations to use a middleware are driven by the will to be unobtrusive (1) and to create an abstraction for context-awareness (2). It aims at deploying regular applications and context-aware applications (related to (1)) and evolving the middleware without impacting the already deployed applications (related to (2)).

We use rules for providing context-awareness thresholds in application through what we call a context signature. It contains rules that define behaviors according to

context, for each application. In order to create these rules, the software engineer must define a set of situations. A situation propose a running state for a corresponding environment, as for example, to stream a video through a 4 Mbit/s connection to a terminal we must code the video in 480p.

Each situation is made of observations of the computing environment and a corresponding reaction. Observations describe the computing environment for an application; upon relevant characteristic chosen by the software engineer. Indeed, in our reference scenario, bandwidth is a relevant characteristic. An observation is formatted as a triple (key, comparison operator and value). Values are strings or numeric; in case of a numeric value, a default unit is used. Key is the name of the characteristic, basically, the name of the data gathered by the sensors (e.g.: screen height for an observation about the height of a screen). The operator fills the gap between key and value.

Observations can be as simple as the reference scenario or more complex with additional elements. For example, a situation can be based on observations on bandwidth and screen size. But there can be only one reaction. So, a situation is composed by at least one observable and exactly one consequence.

Each reaction must be a running mode implemented. Rules allow choosing the most appropriate running mode for an application. For each relevant using case of a multimedia application, there must be a situation expressed as a rule. These rules are used against the knowledge base. Once the developer provides the context-aware application as a package which contains the binary and the context signature, the middleware unpacks these elements as shown in Fig. 1.

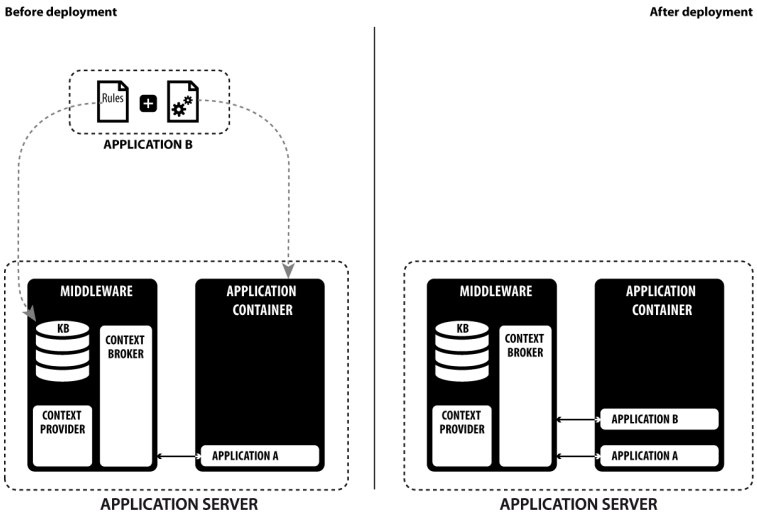

Fig. 1. Deployment of app in CAAS

The binary is deployed like a standard application and the signature is parsed in order to find rules. As stated before, a rule (situation) is composed with an antecedent (observation) and a consequence (reaction). Each one is then injected as knowledge in

the KB. Rules are consider as knowledge because they present running modes and associated necessary conditions for using them, of the newly deployed application.

Context-logic signature defines requirements, in term of context information, for changing application's behavior. We define behavior as a business logic modulation. The main functionality does not change, but can be realized under different forms.

3.1 Storing context in a domain ontology

A context aware application server (CAAS) hosts multiple digital services and provides context-awareness for them. Each application provides features under multiple running modes. A user is typified as a session, which consumes features offered by an application. Each session delivers its features through its environment called context. Sensors through context providers can grab it. Each one is able to gather only some kind of information (battery, screen size, localization, etc.) but not all data from the computing environment. In other words, a context-aware application relies on context providers to sense the environment, in order to choose the right business logic modality (or running mode) for a session.

These observations are the foundation for our resulting ontology. It uses two main classes (Context and Session) as shown in Fig. 2.

Context stores all necessary context information about the user computing environment. It is divided into subsets that represent a context provider.

Session represents all active connections to the CAAS. Each user, which consumes an application, is a member of the class session. Like the Session set, this class is divided into subclasses for representing applications.

Relation between a session class and a context class is hasContextInformation. This property is antisymetric, its domain is Session and its range is Context.

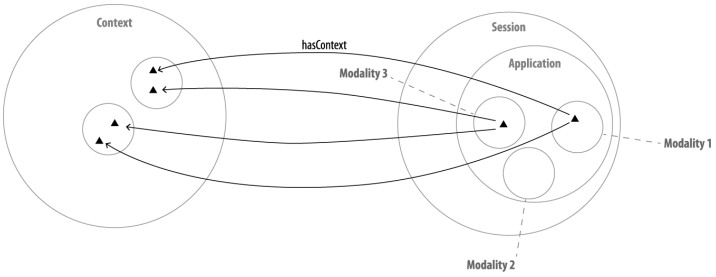

Fig. 2. General representation of our ontology

3.2 Reference scenario

We will use as reference scenario, a video on demand (VOD) application. The digital service has 3 behaviors (i.e.: 480p, 720p or 1080p) for streaming a video to a user. The context information is the available bandwidth. The application changes the resolution of the video (behavior) according to the user's available bandwidth (context) as shown in Table 1. The context-signature provides semantic formalization of these context configurations that is a set of observations. A context signature maps

each context configuration to a specific business logic behavior. This is a loose-coupling solution between business logic and context logic.

We run the above-mentioned ontology against our reference scenario. An application server with our middleware has a context provider called “User Terminal Context” that provide information’s about the terminal like bandwidth, screen size and CPU load. The AS hosts a context aware digital service called “Adaptable Video-on-Demand”, which provide adaptable video according user’s bandwidth.

Table 2**.** Classes of the ontology under our reference scenario

Name	Parent	Description
Context Provider	Thing	All context providers
Session	Thing	All sessions using middleware
UserTerminalCtx	Context Provider	Context provider that gather information about bandwidth, screen size and CPU load
AdaptableVoD	Session	All sessions for the Adaptable Video-on-Demand digital service
C480p	AdaptableVoD	Sessions for AdaptableVoD using 480p resolution
C720p	AdaptableVoD	Sessions for AdaptableVoD using 720p resolution
C1080p	AdaptableVoD	Sessions for AdaptableVoD using 1080p resolution

The data gathered by the context provider are represented as datatype relations. In this scenario the context provider gather information about bandwidth, screen size and CPU.

Table 3**.** Relations in the ontology under our reference scenario

Name	Domain	Range	Description
hasContext	Session	Context	Link a session with a context provider
hasBandwidth	UserTerminalCtx	int	Available bandwidth gathered by context provider
hasScreenSize	UserTerminalCtx	string	Screen size gathered by context provider
hasCPULoad	UserTerminalCtx	float	CPU Load gathered by context provider

As stated before, individuals from the Session context are sorted according to rules, which are injected in the KB. We use SWRL to model the context logic dynamic using this feature. In our reference scenario, the Video-On-Demand service must adapt the video coding according to bandwith. From the OWL point of view, individuals from the set AdaptableVoD must be move either in the c480p, c720p or c1080p subset, each one represents a video coding (480p, 720p and 1080p). This behavior uses the following SWRL:

For more readability, the common part of equations is represents as (0)

$$AdaptableVoD(?s) \wedge hasContext(?s,?c) \wedge hasBandwidth(?c,?b) \quad (0)$$

$$(0) \wedge swrlb:greaterThanOrEqual(?b,3000) \wedge swrlb:lessThan(?b,6000) \rightarrow c480p(?s) \quad (1)$$

$$(0) \wedge swrlb:greaterThanOrEqual(?b,6000) \wedge swrlb:lessThan(?b,9000) \rightarrow c720p(?s) \quad (2)$$

$$(0) \wedge swrlb:greaterThanOrEqual(?b,9000) \rightarrow c1080p(?s) \quad (3)$$

In our model, the reasoner and KB, use OWL and SWRL to choose business logic modality according to context. Each time a new connection is setup, the middleware injects information about the session and the context in the KB. In order to switch modality, the application must be aware of the KB's classification. We use SQWRL to interrogate the ontology in order to notify applications about modality to use for each connection. A typical SQWRL query for this is: *Modality(?s) -> sqwrl:select(?s)*. In our reference scenario:

$$c480p(?s) \rightarrow sqwrl:select(?s) \quad (4)$$

$$c720p(?s) \rightarrow sqwrl:select(?s) \quad (5)$$

$$c1080p(?s) \rightarrow sqwrl:select(?s) \quad (6)$$

The first one provides all sessions that must use a 480p resolution, the second one for sessions use a 720p and the last one for session uses a 1080p.

4 Conclusion

We have proposed and implemented a novel architecture that makes AS taking into account the context-awareness of the digital service they host. The architecture employs domain ontology in order to monitor the applications' context. Moreover, when a new context-aware application is deployed on the application server, its context-logic (a set of SWRL rules which rely the domain ontology) is extracted from the application bundle. Each SWRL rule encodes a context-aware threshold for the application. When a rule is triggered at runtime, the application server notifies the application, so that the application change its service delivery modality, as the current context favors another service delivery modality different than the current one.

The primary goal is to help developer of context-aware applications to quickly encode context-aware thresholds. It helps them to develop the context logic of the application for it to adapt its behavior when a significant context change is detected. Developers can seamlessly encode those thresholds by writing the SWRL rules corresponding to the conditions under which the application follows each service delivery modality and then ship the SWRL file into their application bundle.

Previously, this task was not a service offered by the application server, unlike logging, database mapping, authentication, etc., but actually embedded in each application business logics as an ad hoc encoded algorithm. Therefore, this approach is also a framework, as developers of context-aware applications no longer have to write source code for handling context changes.

We are currently developing additional algorithms that will enhance the context assertions in the knowledge base. Especially, we will add to the context logic parameters for gathering context (e.g. context sampling frequency) as each application may have different temporal needs regarding context updates. In future work, we are planning to consider cluster of context-aware application servers.

5 References

1. A. Zimmermann, A. Lorenz, R. Oppermann, An operational definition of context, *Modeling and Using Context*, Springer Berlin, 2007, pp. 558-578.
2. AK. Dey, Understanding and using context, personal and ubiquitous computing, 2001, pp. 4-5.
3. J. Strassner, Y. Liu, M. Jiang, J. Zhang, S. van der Meer, M. Ó Foghlú, C. Fahy, W. Donnelly, Modelling Context for Autonomic Networking, *5th IEEE International Workshop on Management of Ubiquitous Communications and Services (MUCS)*, April 11, 2008, Brazil
4. J. Coutaz, J. L. Crowley, S. Dobson, D. Garlan, Context is key, *Communication of the ACM*, 2005, 48:49-53
5. M. Baldauf, S.Dustdar, F. Rosenberg, A survey on context-aware systems, *International Journal of Ad Hoc and Ubiquitous Computing*, 2007, pp. 263-277
6. JI. Hong, JA Landay, An infrastructure approach to context-aware computing. *Human computer-interaction*, 2001, vol. 16.
7. H. Chen, An Intelligent Broker Architecture for Pervasive Context-Aware Systems. *PhD thesis, University of Maryland*, Baltimore County, 2004.
8. T. Gu, H. K. Pung, D.Q. Zhang, A middleware for building context-aware mobile services, *In Proceedings of IEEE Vehicular Technology Conference (VTC)*, Milan, Italy, 2004.
9. A. Outtagarts and O. Martinot, iSSEE: IMS Sensors Search Engine Enabler for Sensors Mashups Convergent Application, *International Journal of Computer Science Issues, IJCSI*, November 2009, Volume 6, pp1-7.
10. M.J. O'Connor, A.K. Das, A Method for Representing and Querying Temporal Information in OWL, Biomedical Engineering Systems and Technologies, *Communications in Computer and Information Science*, Springer, pp. 97-110.
11. ITU-T Recommendation, Y.2000-Y.2999, Next Generation Networks, Y-Series: Global Information Infrastructure, Internet Protocol aspects and Next-Generation Networks.
12. M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.2. Technical report, The University Of Manchester, March 2009.
13. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (May 2004). *SWRL: A semantic web rule language combiningOWL and RuleML*. Available from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.
14. M. J. O'Connor and A. K. Das. SQWRL: A query language for OWL. In R. Hoekstra and P. F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.