Context-awareness for Next-Generation Applications Servers

Yves-Gael Billet, Christophe Gravier, Jacques Fayolle

Context-awareness for Next-Generation Applications Servers

Yves-Gael 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. The design of context-aware applications calls for *ad hoc* software patterns. Meantime, most applications are deployed in an application server. An application server hosts several context-aware application which embeds in its business logic the ability to tune the service by taking into account significant context updates. As a consequence, such applications present their own implementation of context sensing, reasoning and decision-making. Therefore it is difficult to adapt them to new context information as well as making them revise their reasoning algorithms. This strong coupling between context awareness and the business logic of the application is an utmost issue for the development of context-aware applications. This paper describes how to generalize context-awareness as a service provided by the application server. This allows freeing the developer from the task of implementing collecting and reasoning with context.

1 Introduction

The era of fixed-mobile convergence is imminent, thanks to the Next Generation Network [1] (henceforth NGN) paradigm. Such approaches encompass converged networks where users can use any kind of terminals and network access technologies. These imply that users connected through a mobile broadband connection (e.g. EDGE, HSPDA) and users connected through a fixed broadband connection (e.g. xDSL, FTTx) use the same network. Until now, different networks were used for each access technology.

The fixed/mobile convergence of networks is not the only innovation brought by NGNs. A NGN is also designed to integrate digital services like multimedia telephony, push-to-talk over cellular, conferencing, through services providers [2]. Users must be able to consume multimedia telephony services through different terminals (smartphone, tablet computer or laptop). An utmost issue for NGN is to provide multimodal digital services, since the user's terminal or network access are expected to change over time during his own session. This means services must be able to shift from one form of service delivery to another according to context updates. A pragmatic solution is to create as many applications as network access

technologies and terminals. This approach obviously does not scale well. Other approaches encompass middleware for context-awareness in digital services as described in the following section.

1.1 Context-aware applications

Context-aware applications can self-adapt according to significant elements from their environment. These elements from the environment shape the *situation* of use, called *context*. Among the existing context-aware definitions [3-5], we base our definition from [4]. 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 its behavior in order to provide the corresponding display and interactions to the end-user terminal.

1.2 Architectures of context-aware applications

J. Coutaz et al. present in [6] a global architecture for context-aware applications based on levels of abstraction for a general-purpose context-aware system. They define four levels:

- The sensing layer provides numeric observables.
- The perception layer provides symbolic observables, which are interpretations from the numeric observables (i.e. transform GPS coordinates in location name).
- The situation and context identification layer identifies context and proposes adaptation. That is the reasoning core.
- The exploitation layer. It is an adapter between the application and the context infrastructure.

Abstraction levels are compulsory for providing loose-coupling between context handlings services and business logic of applications. Baldauf et al. in [7] also use a similar layered architecture for reporting a literature review on middleware and frameworks for context-aware systems.

1.3 Related works on context-aware frameworks

Current approaches in context-aware frameworks make a clear separation between context logic (i.e. acquisition and processing), and business logic of applications. This introduces a separation of concerns as proposed by Dey [4]. He proposes a conceptual framework for supporting context-aware applications and an implementation known as the Context-Toolkit [3]. It aims to ease development and evolution of context-aware applications using object-oriented programming. For this goal, he provides an API for context-aware applications. Each object from the API, which can be used to

construct an application, endorses a role (i.e.: collecting, transforming, aggregating, and serving context). Although this API ease the development of context-aware application, creates a tight coupling between context processing and the application business logic. Hong et al. [8] point out advantages of infrastructure for providing context abstraction. Then, recent works, such as CoBrA or SOCAM are infrastructure-based approaches rather than API.

CoBrA project from Chen et al. [9] is agent-oriented. A central unit, so-called, a "context broker", maintains and manages context on behalf of software 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 of providing context-awareness to resource-limited computing devices.

The Service-Oriented Context-Aware Middleware (SOCAM) project introduced by Gu et al. [8] exists as a middleware. In SOCAM, different services are working together to acquire, to process, to reason and to deliver context to software agents. The main contribution of this work is the context model description logics, which take part into the foundation of OWL[1]. The use of OWL ontology allows them to describe context in a semantic way that is independent of the programming language.

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 the lights when the room is empty, etc).

This paper presents a way to implement context-awareness for digital services in Next Generation Networks in order to adapt content to terminal features and network capacity. Section 2 describes the architecture of our middleware and the context-signature. Section 3 concludes.

2 Proposal

2.1 Context-awareness for digital services

We propose to implement context awareness for digital services in NGNs by providing context-awareness to application servers hosted in a NGN. These services running on applications servers (AS) are consumed through a network using the client-server paradigm. In this situation, the context is composed of information about the network and features of the end-user terminal. The context is therefore specific and describes machine-oriented information, unlike in the aforementioned related works. Furthermore, as applications are hosted on an application server, context-awareness is provided as a service from the application server to the hosted applications.

The objective is to design and implement a middleware that provides sensing, perception and reasoning over context for application running in a NGN environment. Because context processing is common to each application and externalize as a middleware, software developers only care about business logic and adaptation

[1] Web Ontology Language

behavior of the applications they create. Adaptation behaviors, which are specific for each application, are described and implemented with the business logic. That is to say, software developers describe each possible context they want to provide to their end-users, and communicate it to the context-aware middleware on the application server. The application server will adapt applications according to these context descriptions

Like the related works presented before, we propose a separation of concerns to split context-aware applications in two logics: business and context logic. We call business logic every line of code related to the core of applications (e.g. providing telephony, conferencing...). Whereas context logic is every line of code relative to context-awareness (e.g. collecting data, reasoning, brokering context updates, etc.). The coupling between the context and business logics must be loose for the leverage of context-aware architecture on application servers. In order to achieve this goal, context logic is externalized as a middleware inside the AS. Each service exploiting this inner middleware (i.e. taking advantage of the context logic) becomes a context-aware service. We suggest going further as we aim at freeing the software developers from the task of developing the interface between the business logic of his/her application and the context logic provided as a service by the application server.

We propose to use context-driven applications in NGN AS using our middleware. It musts help deploy applications, which are able to adapt their behavior according to context changes. Through this proposition, applications using the client/server paradigm choose the most suitable behavior for each connection (client). Furthermore we provide a solution to create context-aware applications for distributed environments, allowing moving them from one server to another to create context-aware applications that are agnostic to application servers. In a distributed system, applications are deployed in a cluster and migrate dynamically between AS instances. So, context-aware applications need to be easily and rapidly deployed. It is expected that applications migrate with their context-awareness. Such approaches for dealing with context-aware applications in AS must therefore provide:

- Context-driven applications: continuously adapt their behavior according to consumer's environment in order to provide the best user experience.
- Functional separation: application servers provide context as a service for applications. Software engineers do not code context-awareness in applications.
- Ease of deployment: deployment of these context-driven applications must be as simple as possible.
- Embedded context-awareness as a bundle: applications embed context-awareness. This is required in order to provide ease of deployment and functional separation.

The key idea is to provide architecture for context-awareness in applications which use the client/server paradigm. In this paper the term "service" refers to a functionality that an application offers to users.

2.2 Functional separation

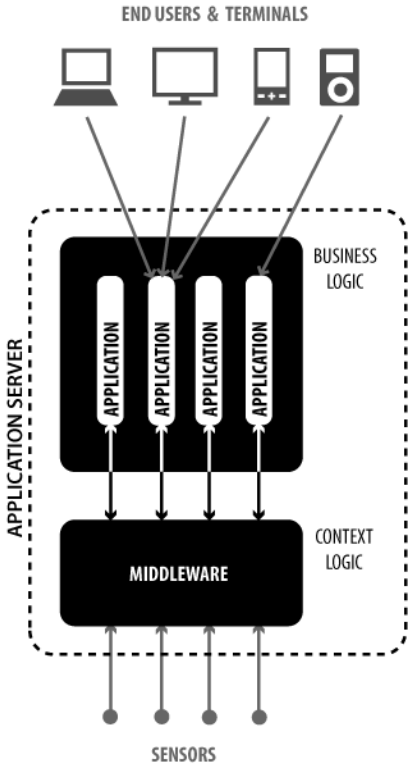

Fig. 1. Functional separation and environment

We use functional separation for logics (context logic and business logic) as follows: functions that relate to context are in the context logic whereas what refers to providing service to user is business-logic (e.g. multimedia telephony, conferencing...). We modularize context-aware applications by separating them in two parts. This approach lets developers focus only on business-logic because they do not have to worry about context implementation. Context logic is then shared among the hosted applications, as a service delivered by the AS.

Although they do not need to consider context implementation, developers need to manage application's behavior according to context. They define the adaptation conditions that represent the context awareness thresholds, and the subsequent behavior of the application. Adaptation conditions are thresholds in terms of context (e.g. low bandwidth and small screen size) for which an adaptation must occur. They are linked with a consequence that is the fulfillment of this adaptation (e.g. switching to a 240p encoding). We leave to the developer the task to write rules that formalize their context awareness thresholds and consequences. Application designers must not code any context-aware related logic in order to provide a generic and transferable context-aware application. This model provides an AS service for applications through a layer-based architecture.

In our case, business-logic is the application and context-logic exists as a middleware, as shown in . Besides it is an architecture scheme for distributed environments.

Although we implement context sensing in our middleware, we exclude sensors implementation. Context-awareness could not exist without sensors but implementing sensors to provide raw data is a different issue from using context to adapt applications in NGN application servers. Furthermore, there are already works on sensing context in NGN [9]. Our approach is to rely on monitoring solutions in order to retrieve data from sensors. In this way, we use them as context datasource in the application server.

As presented before, developers create applications and in the meantime define context adaptation conditions with possible behaviors. Adaptation conditions refer to elements of context that are used to characterize the situation, whereas consequences are related to the adaptation behavior for providing the corresponding interaction. We coined the term "context-logic signature" for conditions and consequences of adaptation, expecting that context signature is actually embedded in the application bundle, as part of its metadata. Developers define the signature, according to server's capabilities (i.e.: sensors reachable by the middleware) and ship it with the application. Context-awareness thresholds are therefore bound to the application, whereas the context logic, which is able to process it, is a service offered by the application server. The application server delivers context-awareness as a service, just like authentication, authorization, logging, mailing, data persistency, message broker, etc.

2.3 Context-logic signature

A core element for our model is the context-logic signature. Context-logic signature is an expression used to denote the aforementioned rules that encode context-awareness thresholds. It 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. Each form depends on context.

Table 1. Abstract example of context-logic signature

Context observable	Behavior
Bandwidth from 3 to 5 MBit/s	480p
Bandwidth from 7 to 13 MBit/s	720p
Bandwidth from 15 to 20 MBit/s	1080p

For example, a video on-demand (VOD) application may have 3 behaviors (i.e. 480p, 720p or 1080p) for streaming a video to a user. The behavior corresponding to the user's situation is chosen according to the available bandwidth (it is context

information). So, the application changes the resolution of the video (behavior) regarding the user's bandwidth (context) as shown in Table 1. The context-logic signature provides a semantic formalization of each context configurations that are a set of observations, in order to map them to a specific business logic behavior. Each adaptation decision for an application's session (i.e.: a user consuming the service) is based on such rules.

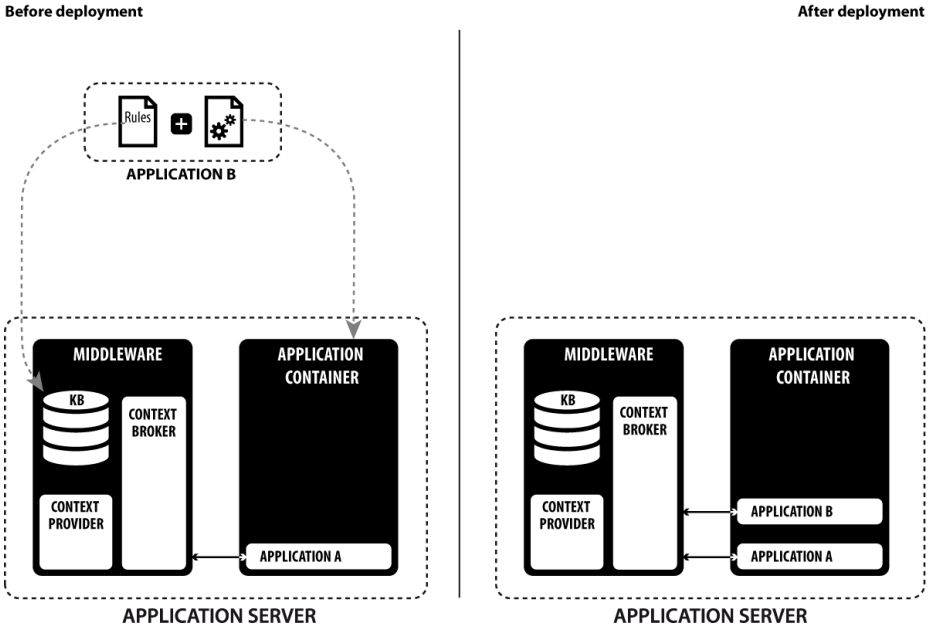

Fig. 2. Deployment process of an application

Applications embed their context-logic signature. At deployment time, we extract its context-awareness signature from the application bundle. The context-logic signature is injected in the context-logic middleware and the application in its container through a standard application server deployment as illustrated in **Fig. 2**. After deployment, if the set of rules needs to be changed, it can be flushed for a new one. Replacing SWRL rules without dropping previously imported and inferred knowledge was made possible through Protégé-OWL API thanks to previous works [10].

2.4 Context-logic

As stated before, applications rely on context-logic middleware for context processing, and middleware relies on monitoring software for context harvesting. We define monitoring software as an entity (with respect to the context definition presented at 1.1) that retrieves raw data from some sensors and is able to send them to another entity. So, the context middleware subscribes to sensors through monitoring

systems (context source) according to context data that must be monitored. In other words, raw data from sensors are published to the context middleware via the monitoring system. The middleware therefore subscribes to sensors via monitoring interfaces of this system. Data from sensors is sent to the middleware.

In order to customize the user's digital experience, raw data is harvested from sensors. Knowledge is afterwards inferred (e.g.: the user's location, role, activity, device, connection, etc.). As a response, the system may go as far as modifying the delivered user experience. Fig. 3 illustrates our general approach of "context processing".

This chain is the process of transforming raw data from sensors into context information that are understandable and computable by the machine, so that the business logic of an application could be seen as adaptable or intelligent from the user's point of view. In Fig. 3, the different processing blocks are:

- Harvester (H). This unit is usually proprietary. It aims at retrieving raw data from one, several or a network of sensors. In more elaborated systems, sensor networks may provide implicit description (basically: RDF[2] triples).
- Semantic Formalization (SF). The SF transforms raw data harvested into assertions in the Knowledge Representation domain.
- Knowledge Base (KB). The KB hosts the Knowledge Representation, which means:
  - Rules extracted at the deployment time (from the context-signature),
  - Context information harvested from context sensors.
- Context-Querying (CQ). The CQ is held responsible for:
  - Extracting all the rules introduced in the SCAS for context awareness. They are provided by the applications. It usually corresponds to the thresholds values that require an adaptation of the service modality. The context has been recognized as being significantly changed.
  - Querying at regular interval the KB in order to detect changes. An assertion has changed between two querying samples (i.e. the context has changed), the CQ notifies the Context-Broker. We plan to enhance the CQ module with temporal context information querying. Temporal information querying is an emerging concept in order to encode the temporal dimension of data in knowledge representation and querying [12]. In ongoing works, we are aiming at providing knowledge revisions for semantic context-aware systems based on temporal and OWL explanations [13], yet this is outside the scope of this paper.
- Context-Broker (CB). The CB provides middleware connectivity to the CQ in order to cast context changes to "adaptive services" (which are looking for possible changes).
- Application Server (AS). It holds adaptive services that are delivered to the end user. The signalization process between clients and the AS is taking place in a Next Generation Network (NGN) in our model [1].

[2] Resource Description Framework

2.5 Implementation

This work is related to context-awareness for applications in Next Generation Networks (NGN). We do not present NGN in this paper. We choose IP Multimedia Subsystem (IMS) [14] as an implementation of a NGN. In order to be IMS compliant, an application server with the Session Initiation Protocol (SIP) is compulsory. We looked for a J2EE application server that implements SIP servlets and choose JBoss Mobicents [3]

We implemented the deployment process of our middleware and the registration of an application. The user upload through a web interface is application container (.war, .sar, .ear…) with the context signature embedded. The signature is extracted, checked and injected. If no errors are encountered, the application is then deployed to the application server and registered to the middleware. As each application is written using a java interface, it enables the business logic to listen to significant context changes (a context rule was triggered) and to switch modality accordingly.

For this part we mainly use HTTP[4] Servlet, JSP[5], EJB[6] 3 Session and JMS[7].

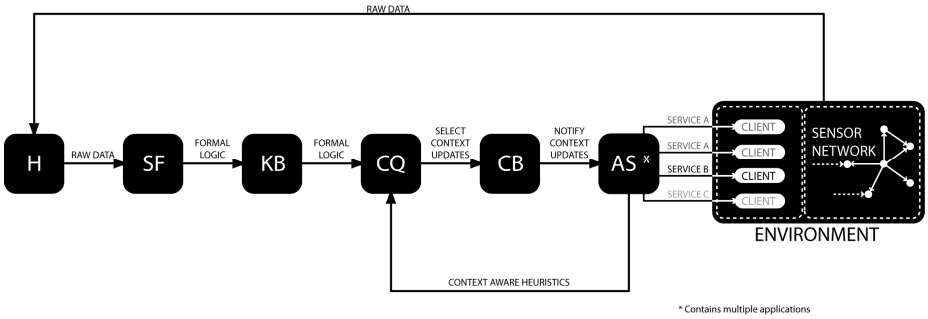

Fig. 3. Context processing chain in a next generation application server.

3 Conclusion

Our goal is to constantly adapt an application in client/server mode to provide the best experience to the user. In this context, we propose architecture for context-awareness in application servers to go one step further in development of context-aware applications. For this reason we created a loosely-coupled system which allows to easily enhance existing applications or future applications with context awareness.

The actual approach is to provide, in the same bundle, context-awareness as a signature which is bound to applications business-logic. These signatures are thresholds, which are conditions to adapt the business logic of this application, in order to deliver the best representation at the right time.

---

[3] Application servers for IMS : http://www.mobicents.org

[4] Hypertext Transfer Protocol – RFC 2616

[5] JavaServer Pages – Java technology that serves dynamically generated web pages.

[6] Enterprise JavaBeans – Server-side component architecture for Java J2EE applications.

[7] Java Message Service – An API for sending messages between Java J2EE applications.

We will continue to implement our model and focus on interaction between sensing and context logic and define our context model.

4 References

1. ITU-T Recommendation, Y.2000-Y.2999, Next Generation Networks, Y-Series: Global Information Infrastructure, Internet Protocol aspects and Next-Generation Networks.
2. C.J. Pavlovski, Service Delivery Platforms in Practice [IP Multimedia Systems (IMS) Infrastructure and Services], IEEE Communications Magazine, IEEE Communications Society, March 2007, vol. 45-3, pp. 141-121.
3. AK. Dey, Understanding and using context, personal and ubiquitous computing, 2001, pp. 4-5.
4. A. Zimmermann, A. Lorenz, R. Oppermann, An operational definition of context, *Modeling and Using Context*, Springer Berlin, 2007, pp. 558-578.
5. 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
6. J. Coutaz, J. L. Crowley, S. Dobson, D. Garlan, Context is key, *Communication of the ACM*, 2005, vol. 48 pp. 49-53
7. M. Baldauf, S.Dustdar, F. Rosenberg, A survey on context-aware systems, *International Journal of Ad Hoc and Ubiquitous Computing*, 2007, pp. 263-277.
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. Y.G. Billet, C. Gravier, J. Fayolle, Can We Streamline QoS in NGN: the half-filled glass, Proceedings of International Conference on Next Generation Networks & Services (NGNS'10), Marrakech, Maroc, July 2010.
10. C. Gravier, M. O'Connor, J. Fayolle, J. Lardon, Adaptive System for Collaborative Online Laboratories*, IEEE Intelligent Systems*, IEEE computer Society Digital Library, IEEE Computer Society, accepted 06 Jan 2011, In Press.
11. H. Chen, An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland*, Baltimore County, 2004.
12. 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.
13. M. Horridge, B. PArsia, U. Sattler, Laconic and Precise Justifications in OWL, *Internatioal Semantic Web Conference*, Karlsruhe, Germany, 2008.
14. M. Koukal, R. Bestak, Architecture of IP Multimedia Subsystem, *Multimedia Siganl Processing and Communications*, June 2006, pp. 323-326.s