# Using Rules for the Integration of Heterogeneous and Autonomous Context-Aware Systems

**David Mosén**[1] and **Arantza Illarramendi**[2] and **Mohand-Said Hacid**[3]

**Abstract.** In this paper we introduce NUBIA, a middleware that combines, through business rules, information generated by heterogeneous and autonomous systems. Communication between NUBIA and systems is loosely-coupled and highly compatible, as Web Services and other standards are used. The main component of NUBIA is a rule engine, sensible to temporal knowledge and with integrated functions (e.g. frequencies, percentages). A user friendly interface allows entering, using a script language, customized rules, whose conditions depend on the information sent from the systems. For each rule, settings such as the maximum firing frequency and the activation within a group may be defined. Moreover, NUBIA also supports a rule editor role, which allows a more realistic viewpoint of context-aware rules customization. Finally, automatic rule translation to the user's language and a role-oriented interface facilitate the interaction with NUBIA.

## 1 INTRODUCTION

The term context awareness has evolved through time, but has always maintained a sense of somehow gathering data from the environment. In the last years, the definition of context by Dey et al. [1] seems to be the most widely embraced. It states that context is "any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects". Notice that this definition implies that context is not necessarily physical, so it can be sensed through virtual sensors (e.g. Web Services).

Nowadays, context-aware systems work in general in an autonomous way and do not interact among them. However, connecting them can, in many situations, increase the advantages that they provide separately. Let us take the example of two systems, in charge of monitoring a house and vital signs of a person, respectively. A connection between them can boost alertness in a home for elderly people scenario, where a single unit reacts to information about both the house and the owner's health. Of course, each system should still be able to be in control of its own domain.

For a tight coupling among systems, a complex manual process may be needed. With this in mind, we present in this paper NUBIA, a middleware that integrates, through loosely-coupled connections, any kind of context-aware systems, independently of the domain that they consider, while preserving the autonomy of each system. Thus, internal management in each domain remains within the respective system, whereas a combined management across domains is handled through NUBIA.

To test the middleware and demonstrate its usefulness, we constructed a typical homecare scenario, oriented towards a single inhabitant (i.e. the user), although visits pose no actual problem. Abundant previous work about homecare can be found [2] [3], and we refer to it for a more thorough study on the subject. However, our aim is to show how an integral homecare system can be build adding up independent, separately developed, components. Thus, the personal healthcare application, provided by SaludNova [4], and the domotics system we integrated were already developed, in order to test out the integration issue in a real scenario.

Benefits from integration through rules of the mentioned systems can be seen in several cases. For example, the healthcare system lacks fall detection, so if domotics determines using its sensors that the user fell, a warning is sent to NUBIA, which in turn warns the healthcare system. In the same way, the domotics system can be commanded to turn off the oven and stove if the user suffers a heart failure. Further, if a risk situation requires both systems to detect certain alarms, then a third system, in charge of handling joint alarms, is notified. As a final example, this additional alarm system can also be notified of errors in the main systems (see Subsection 3.2.1).

Summing up, the main features of NUBIA are the following ones:

1. Because sensed context is information sent from connected systems through Web Services, it is a context-aware application too.
2. It allows combining context-aware systems information through customized business rules sensible to knowledge extracted from a stored history.
3. Finally, it provides a user friendly graphical interface, which supports different user roles (final, rules editors and administrators).

## 2 RELATED WORK

Concerning related works, we can observe that trends in software development move toward generic design patterns. In that direction, there seems to be an agreement on the need for finding a context-aware standardized architecture [5]. This area has been widely studied, with several proposals of context-aware frameworks [6] [7]. We also present a proposal of a standardized architecture; however, our goal in this paper is to focus on the use of rules for getting the integration of context-aware systems.

On context integration, the work by Tao Gu et al. on SOCAM [8], an architecture for developing context-aware services, is the closest

[1] University of the Basque Country, Spain, email: david@mosen.es
[2] University of the Basque Country, Spain, email: a.illarramendi@ehu.es
[3] Univ. Claude Bernard Lyon, France, email: mshacid@liris.univ-lyon1.fr

to our proposal. We basically differentiate on what is integrated, as we aim at autonomous context-aware systems integration, while SOCAM directly integrates physical context. Anyhow, both proposals carry out the idea of reasoning with the context to offer some kind of service to external applications.

Similarly to NUBIA, the system proposed by Agarwal et al. on electronic medical records [9] uses business rules to combine gathered history information, namely, from RFID sensors and patient monitoring. Thus, our proposal is similar in the sense that we use gathered information in business rules. However, in the case of NUBIA, information used is context which comes from any kind of context-aware system, thus not restricted to the medical field.

Finally, our approach towards personalization, based on customized rules managed by a rule editor, can be situated among those that appear on the existing literature, neatly explained by Henricksen & Indulska [10]. They classify personalization approaches on context-aware systems into three categories: end user programming, machine learning and preference-based. The one used in NUBIA is similar to the end user programming, but it provides as a novelty a scheme where the user relies on a rule editor to make the adjustments he/she requires.

# 3 NUBIA ARCHITECTURE

The middleware core is divided into three main units. First, the *context handling module*, which transforms context to a standardized representation and deals with history knowledge. The *rule engine module*, which manages the combined information using business rules. Finally, the *communication module*, which provides the context handling module with the context coming from systems, and allows the rule engine to communicate about actions to execute in the systems. Hence, notice that communication flows both ways between NUBIA and systems.

In this section, we briefly explain the main features of the mentioned modules, which can be seen in Figure 1.

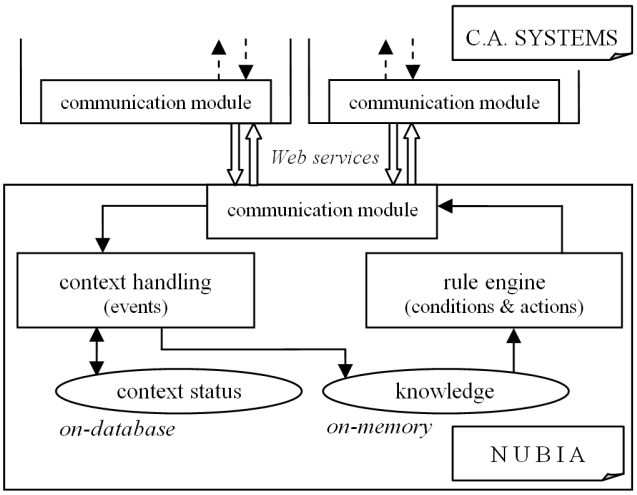


**Figure 1.** NUBIA's architecture

## 3.1 Context handling module

Taking into account that systems can communicate heterogeneous context, there is a need for its classification into predefined groups. This process is called categorization.

### *3.1.1 Categorization*

Two categorization viewpoints exist [11]: *conceptual*, based on what the context symbolizes (e.g. person, network, physical environment), and *measurement*, based on how the context is represented (e.g. a value, a predicate). There exists a bigger tendency to follow the conceptual categorization [12] [13]. However, we chose a measurement categorization in order to explore its possibilities in relation to the management of history knowledge. Hence, context is classified into the following four categories:

1. Single. Simple events, which happen at a very precise moment. Alerts and punctual detections fit into this context category.
2. Discrete. The context can only take one value once at a time from a finite set of values, each of which represents a state. Examples in this category are device's power state, which toggle between on and off, and generally any context whose possible states are well-defined.
3. Continuous. In this case, the value representing the context is a real number, so we can make comparisons to see whether it is within a certain range. Uncountable data belong to this category.
4. Descriptive. Compound content cannot be represented by any of the three categories above. This category is based on the description statement and uses the notion of predicate. A person's location, for example, is represented as location(person,place).

Furthermore, context reported by systems may correspond to different levels of reasoning, ranging from raw data to thoroughly reasoned knowledge. Figure 2 shows a layered scheme [14] where context information flows to the middleware from any of the two first layers of a context-aware system.

Nevertheless, the category to which context belongs is the only relevant distinction and context is manipulated based on it. For example, a light switch, the energy level of an electron and a storm emergency level are all considered as discrete context, regardless the complexity of the process to obtain them.

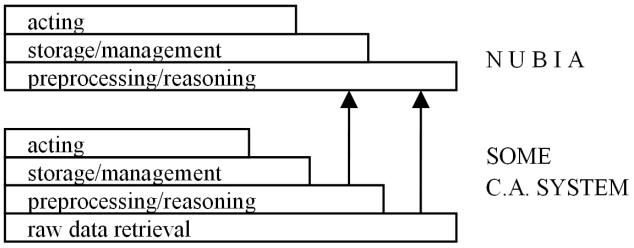


**Figure 2.** Information flow in a layered scheme

### *3.1.2 The summarizing log*

NUBIA manages a special type of history, called a *summarizing log*, which gets updated every time some context information is received from a system. A typical history logs everything, for an

eventual use, without any further modification than the addition of new records. In a summarizing log, instead, a logging action causes a modification that updates key information about the current situation. This logging method helps to extract knowledge from the history, that we refer as temporal knowledge.

Stored information such as number of times in a state or elapsed time within a range, together with interesting timestamps, are enough to infer the above mentioned temporal knowledge (e.g. frequencies, time since last state shift).

Let us suppose that we want to know the frequency of a given simple event. Two fields are required in the summarizing log: the first time the event is registered, and the times count. Thus, the frequency can be calculated:

$frequency = (now - first\_time)/times\_count$

There is also information directly extracted from the summarizing log, as it is useful without any further operation. Examples are: within which ranges from a defined set is some continuous value; and the number of times in a certain state. For these two cases, the following information could be part of the log:

*(name:corporalTemp(at)temp(at)biometrics, currentValue:36.6)*
*(range:[36.0,36.9], lastBegan:1206000000*[4]*, lastEnded:-)*
*(range:[36.0,36.6],*
*lastBegan:1206000800, lastEnded:1206000900)*
*(range:[36.8,max], lastBegan:1206001000, lastEnded:-)*

*(name:faintRisk(at)alarms(at)biometrics, currentState:low,*
*lastShift:1206010000, shifts:7)*
*(state:low, times:5, last:1206010000)*
*(state:average, times:2, last:12060000300)*
*(state:high, times:0, last:-)*

Conditions in the rules (see Section 3.2) are checked against all this knowledge, temporal and non-temporal.

Finally, notice that summarizing logs are not aimed at applications with infrequent or full history reasoning [16] [17], as they are not powerful enough. Context-aware systems, however, have a strong requirement on time, and a summarizing log helps to get quick response times. Moreover, it is compatible with a full-fledged history, so that the best of both worlds is available.

## 3.2 Rule engine module

This module evaluates rules that trigger depending on the context knowledge extracted from the summarizing log. The rule engine is independent from the communication process. As a result, systems can continue to report context information even if the rule engine is not active.

### *3.2.1 Rule structure*

Two classes of business rule engines exist. First, and the one used in our proposal, a production rule engine deals with rules with "IF condition THEN action" semantics. Usually, an external agent invokes the engine, so in the scenario of a context-aware system with production rules, the system typically invokes the engine whenever some context is sensed. If, given the new situation, the conditions of a rule are true, it fires.

Second, reactive rule engines. In this case rules fire when their conditions are true as well, but they need some event to happen in order to get evaluated. Hence their name, Event Condition Action (ECA). This class of rule engine is suited for most context-aware systems, because sensed context adjusts well to the concept of event. Thus, the system does not need to explicitly invoke the rule engine, as it is already aware of generated (context) events.

In NUBIA, many defined conditions depend on time, so they cannot be evaluated only when some context information arrives, because temporal knowledge must also be taken into consideration. There are two possible mechanisms to deal with this situation. Let us take the following as an example condition:

*The light has been in state off for 5 minutes*

The first option (continuous evaluation) is to constantly check whether 5 minutes have elapsed since the last shift to off. The alternative (evaluation scheduling) is to schedule the system to invoke the rule engine 5 minutes after each time the light changes to the off state. This alternative is more efficient, as it uses processing resources more wisely. However, it is also non-trivial, because depending on the condition semantics, evaluations should be scheduled in different ways (e.g. with a certain frequency for a limited time, when the event is detected). In either case (continuous or scheduled evaluation), the middleware is in charge of telling its rule engine when to evaluate the rules (i.e. events do not directly trigger the rules) so we chose to implement them as production rules. In particular, rules are implemented using JBoss Rules [19], following a forward-chaining production structure.

The right hand side of the rules comprises two kinds of actions: internal and external.

External actions are not executed by the middleware, but in a connected system. In addition to sending context information, systems may expose actions to NUBIA through Web Services, so that they can be ordered to execute the actions.

Internal actions control NUBIA itself and gathered information. This includes error count resetting. NUBIA detects both incoming communication errors (i.e. context reported by a system is invalid or the message is corrupt) and outgoing communication errors (i.e. the system to which to connect is unreachable). Data can also be reset if, for example, the information about a certain context should be initialized. Finally, a system may refuse to execute an ordered action, so this can be used in the condition part too.

### *3.2.2 Settings*

Some of the incorporated settings in the rule engine include:

Maximum firing frequency. Controls repetition of rule firing. Even if a rule is evaluated to true, it will not be fired unless the defined time has elapsed. In that case, it will only fire if it is still true. For example, the user may want to be notified of new mail after at least 2 hours since the last notification, even if mail arrives in the meanwhile.

Activation group. This setting has been extended from the JBoss Rules option with the same name. The rule within an activation group with the highest priority is executed; the rest, albeit evaluated as true, do not get a chance to be executed until the time defined by the group's maximum firing frequency goes by (all rules in a group have the same maximum firing frequency). For example, if the user has a tumble, the system should call a relative, but if, additionally, the user suffers a heart-attack, this action may be overridden by a call to the

[4] Timestamps are described in Unix time [15] (seconds elapsed since 01 Jan 1970, 00:00:00 UTC).

emergency number.

Other settings, such as firing count limit and expiry dates, may eventually be included as well. The existance of some of these settings in JBoss Rules might facilitate their implementation.

### *3.2.3 NIRE language*

As a final point concerning the rule engine module, we designed a script language to facilitate the definition of rules and check their validity, so that they do not cause errors during execution. The NUBIA Input Rule Editing (NIRE) language provides the following benefits:

1. Transparency and independence from the underlying rule implementation engine.
2. A compact syntax, with no unneeded verbosity.
3. Translation extensibility through XML, allowing the definition of new rule translations to other user languages without recompiling the application.

The following is an example of a rule in NIRE. Notice that settings are not defined in the language, as they are introduced through the graphical interface.

```
if
  is-true
    presence@locator@wear $somebody
  last-time-in-range
    temperature@temp01@domotics 15 27 > 3600
  time > 18:00:00
then
  turn-heater (using heater@domotics) "on"
  display (phone(at)aux-phone)
    "$somebody is home, turning heater on."
```

The rule has a typical ”IF condition THEN action” structure. Each of the first two conditions are stated over a certain context, while the third is an internal NUBIA condition which controls the time of the day. Each action is defined by a name and its corresponding device and system. In this rule, both actions require one parameter each, delimited by double quotation marks. Concerning symbols, ”@” (i.e. at) denotes in which device and system a context is sensed or an action is executed, whereas the dollar symbol, denotes variables. Thus, the first condition states that somebody must be detected, and saves his/her name in the ”somebody” variable.

If the chosen translation language is English, the user would see this resulting text:

*If presence is true for a certain person 'somebody', temperature has not been between 15 and 27 °C in the last hour and it is more than 6 in the afternoon, then turn heater on and display in the phone "'somebody' is home, turning heater on.".*

## 3.3 Communication

Information flows between systems and NUBIA in both ways: context information is reported to the middleware and orders to execute actions are sent back. In either case, Web Services are used. Thus, to make communication possible, developers who wish to have their systems integrated need to: make methods to be used by NUBIA available through a Web Service; and report desired context to NUBIA's Service.

Usually, applications use a fixed set of Web Services, but sometimes they may require to call beforehand unknown Web Services. Dynamic invocation allows client applications to invoke Web Services whose descriptions are unknown until the application is used. As an example implementation, the Dynamic Invocation Interface (DII) [18] for CORBA supports this functionality. NUBIA needs the dynamic invocation, given that it is aimed at working with beforehand unknown systems.

For a higher decoupling from inner operation, serialized XML are sent, so if more communication-related functionalities are added to the middleware, only the XML representation would change, whilst Web Services in connected systems and the middleware would remain the same. The following XMLs are examples of incoming and outgoing messages, respectively:

*<event time=”1206000000” xmlns=”http://www.tempuri.org”>*
*<signal name=”smoke” device=”smk” system=”domotics”/>*
*<continuousInfo>0.32</continuousInfo>*
*</event>*

*<request time='1206000000' xmlns='http://www.tempuri.org'>*
*<action name='switch' device='light01' system='domotics'/>*
*<parameter>off</parameter>*
*</request>*

Despite Web Services are the best communication option because of its wide de facto standardization, there are systems that do not fully accept them, such as smartphones, most of which cannot host a web server. To cope with this difficulty, socket communication stands in NUBIA as an alternative to Web Services.

# 4 INTERFACE

The task of integrating autonomous context-aware systems is not easy, so NUBIA provides a graphical interface that focuses on the separation of the different types of users (i.e. roles) for a more specific interaction with each of them. Therefore, before we show the main features of the GUI, we present the user roles considered by NUBIA.

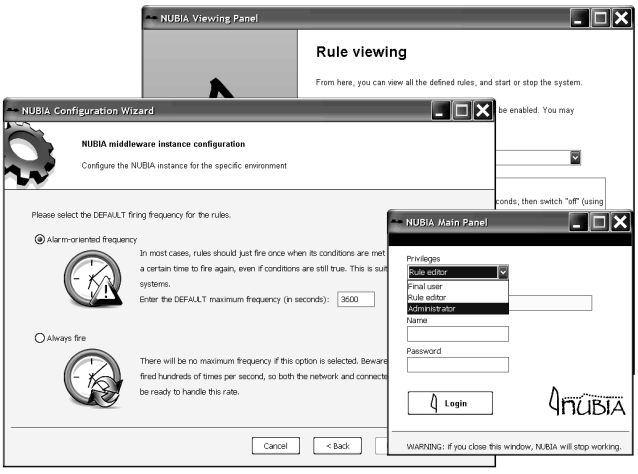


**Figure 3.** Some windows from NUBIA's interface

### 4.1 Roles

Relation with end users is a thoroughly studied issue in context-aware systems. Giving users control over a system they use implies they need to learn, in some degree, how to interact with it. Different approaches towards this interaction exist [10], but they lack either power of control or simplicity, so we define a role that fills the gap between the end user and the administrator, to keep the user somehow in control and yet increase simplicity in their interaction with the system. Thus, for our middleware NUBIA, we establish a three tiered role division to better focus on each role requirements:

1. Administrator. There only exists one, and defines which systems to connect to NUBIA and their specifications, probably handed by other administrators or developers. Also, he/she configures settings such as working mode (dedicated server or shared machine) and application defaults. Finally, the administrator is in charge of account management.
2. Rule editor. Manages rules without the need to know about application programming or technical details. A user with a little bit of technical knowledge may manage rules and act as a rule editor too.
3. End user. Can only see the defined rules and decide whether to start or stop NUBIA.

In this scheme, roles are incremental, with the administrator having privileges as a rule editor and user as well.

### 4.2 Ease of interaction

A configuration wizard à la MySQL [20] (see figure 3) guides the administrator to easily configure, for example, system defaults and working mode.

Systems specifications are defined in XML, so they can be checked through an XML-schema and translated to internal representation through an XSL transformation. This way, portability, independence and easy handling are achieved and hence, the administrator selects the file with the definitions and NUBIA does the rest.

Maximum firing frequency, validity expiration or activation policies within a group are settings that allow a more refined execution control. Nevertheless, defined defaults (by the administrator or NUBIA itself) make simple rule creation an easier task. A rule preview, an auxiliary panel with available data and syntax documentation, and an accurate error checking facilitate rule creation even more.

Automatic translation of rules to the user's language allows less technical users to easily understand them, whilst not giving extra work to the rule editor.

## 5 CONCLUSIONS

We have presented a middleware that successfully connects, with loosely coupled Web Services, autonomous context-aware systems, combining and making use of their information to trigger business rules. Interaction is made through an easy to use interface, designed taking roles into account.

Using the implemented prototype, we observed the following behavior: a rule may be triggered by incoming context information or because some defined time has elapsed. Taking both possible cases into account, the average response time starts at 10 milliseconds with a few defined rules, each of which adds a 4 microseconds overhead.

This last fact confers the middleware scalability concerning rules. Finally, the fast processing of Web Service messages supports heavy communication between NUBIA and the context-aware systems, so a great scalability is also achieved in the amount of integrated systems and communication with them.

## ACKNOWLEDGEMENTS

This work is supported by the Spanish Ministry of Education and Science (TIN2007-68091-C02-01) and the Basque Government (IT-427-07).

## REFERENCES

[1] A. Dey, D. Salber, and G. Abowd, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*, Human-Computer Interaction, vol. 16, pp. 97-166, 2001.

[2] A. Cesta & F. Pecora, *Integrating Intelligent Systems for Elder Care in RoboCare*, W.C. Mann & A. Helal (Eds): Promoting Independence for Older Persons with Disabilities, IOS Press, pp. 65-73, 2006.

[3] G. Virone et al., *An Advanced Wireless Sensor Network for Health Monitoring*, D2H2, Arlington, Virginia, 2006.

[4] SaludNova Sociedad Cooperativa, website: http://www.saludnova.com/ (as of April 2008).

[5] M. Baldauf & S. Dustdar, *A Survey on Context-Aware Systems*, Technical Report Number TUV-1841-2004-24, November 2004.

[6] J.E. Bardram, *The Java Context Awareness Framework (JCAF)*, Tech. Report CfPC 2004-PB-61, Centre for Pervasive Computing, Aarhus, Denmark, 2003.

[7] K. Henricksen & J. Indulska, *A Software Engineering Framework for Context-Aware Pervasive Computing*, Second IEEE International Conference on Pervasive Computing and Comms. (PERCOM 2004), March 2004.

[8] T. Gu, H. Pung , D. Zhang *A service-oriented middleware for building context-aware services*, Journal of Network and Computer Applications 28(1): 1-18, 2005.

[9] S. Agarwal, *Context-Aware System to Create Electronic Medical Encounter Records*, Technical Report Number TR-CS-06-05, 2006.

[10] K. Henricksen & J. Indulska, *Personalising Context-Aware Applications*, in OTM Workshop on Context-Aware Mobile Systems, Springer-Verlag, pages 122–131, 2005.

[11] M.A. Razzaque, *Categorization and Modeling of Quality in Context Information*, in Proceedings of the IJCAI Workshop on AI and Autonomic Communications, 2005.

[12] Eleftheria Katsiri & Alan Mycroft, *A first-order logic model for context-awareness in distributed sensor-driven systems*, RSPSI Workshop, 2006.

[13] Anjum Shehzad, Hung Q. Ngo, Kim Anh Pham and Sungyoung Lee, "Formal Modeling in Context Aware Systems", KI-Workshop Modeling and Retrieval of Context (MRC2004), 2004.

[14] S.W. Loke, *Context aware pervasive systems : the architecture of a new breed of applications* (ISBN: 0849372550), Abstract layered architecture (page 25) . Boca Raton, FL : Auerbach Publications, 2006.

[15] *Unix time* article from Wikipedia, available at http://en.wikipedia.org/wiki/Unix_time.

[16] G.M. Youngblood et al., *Automation Intelligence for the Smart Environment*, IJCAI-05, page 1513, 2005.

[17] V. Jakkula & D. Cook, *Anomaly detection using temporal data mining in a smart home environment*, Methods of Information in Medicine, 2008.

[18] BEA Documentation on *the Dynamic Invocation Interface*, available at http://e-docs.bea.com/tuxedo/tux80/creclien/dii.htm (as of April 2008).

[19] JBoss Rules, http://www.jboss.com/products/rules (as of April 2008).

[20] MySQL 5.0 Server Configuration Wizard, documentation available at http://dev.mysql.com/doc/refman/5.0/en/mysql-config-wizard.html.