Requirements Engineering for Context-Aware Systems

Andrei Cojocariu, andrei.cojocariu@uzh.ch

Institute for Informatics, University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland

Abstract. The goal of this paper is to offer a walkthrough of two main aspects of requirements engineering for context-aware systems. At first – the context, handling the context data and how it is the context actually influencing the behavior of a context-aware application. I am looking at how the user is being involved into the requirement engineering process while trying to sketch myself an elicitation method based on simple tables where the analyst can map requirements to contextual data. Second part covers uncertainty into requirements, how is uncertainty being build, why is it hard to detect and how can the analysts cope with it by using current state of the art.

1. Introduction

The term of Context-Aware System (CAS) has first been introduced by Schilit, Adams and Want [1] and ever since there is a growing community of scholars orbiting around it. Inside this group and to some extend in the outside the purpose of context-aware system could not be clearer; a better interaction between the device and its environment it's needed. It is envisioned that by tackling and trying to fill this gap the use of devices both in the industry and personal environments will bring a big plus to productivity in general. By being aware of its context an application can offer the user a set of services that the user can immediately access or the application can even change its operations set so that it matches the current context of the user. Though the benefits of working with such a system are not hard to notice, context-aware systems are not yet very common. Sitou and Spanfelner [2] point out that this could be because there is still a big gap between the system behavior and user expectations. This problem is also being found in regular systems, however along the way we have created good methods to cope with it. It is actually the requirements engineering part of a system design process that takes care that the user gets what he actually wants from the new system.

When it comes to context-aware systems, however, research is still being done in order to find the best ways of engineering requirements of the system to be. Today's standard methods are not suitable one hundred percent for this type of systems. This is mostly because there are several differences between a regular system where the operational context is rather static and a context-aware system which is designed to operate in a fast changing environment. Nevertheless requirements engineering for context-aware systems is as important as for all the other systems and in this paper I will

try to identify the current state of the art while pointing out the aspects and properties that make context aware applications special from a requirements engineering point of view. I will start with a general overview of context aware systems in section 2 and then go into the requirements engineering process itself in section 3. Section 3 covers also in more detail what context may refer exactly to, what methods we can use to catch and model it while also actively involving the user into this process. Towards the end of section 3 I am addressing uncertainty in requirements which another key challenge in requirements engineering for context-aware systems.

2. Context and Context-Aware Systems

2.1 Context Aware

Trying to define a context aware system is not an easy task. But when trying to do so one should first understand what *context-aware* actually means. Anind *K.* Dey [3] defines it with:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

where *context* is any information that can be used to categorize the current situation of a user.

2.2 Context Properties

A given context is made up of several elements, out of which we can easily identify:

- Location: position, orientation, velocity, etc.;
- User Identity: Profile, preferences, biometrics, social information etc.;
- Time: current date and time or future events, duration etc.;
- Activity: walking, sleeping, sitting, etc.;
- Current Task: work or social meeting, fitness, studying, etc.;
- Environment: temperature, humidity, light and noise levels;
- Hardware: current device information, network and surrounding devices;

Context aware systems can be then described as applications that are able to read certain contextual elements, reason about them and then adjust their behavior so that it meets in the best possible way the user needs in the current operational context. Further coverage of context and what exactly means for a context-aware application is discussed in 3.1.

3. Requirements Engineering

Requirements engineering is special in a context-aware system because in contrast to a regular system the CAS is expected to perform in a versatile environment with properties that might change or even be unknown at design time. As such, a context-aware system must continuously monitor changes in its context and react accordingly.

3.1 Using Context in Requirements Engineering

The term context is not new for requirement engineers. Contextual inquire for example is a method of requirement elicitation where the analyst creates a master-apprentice relationship with the future users of the future system. Meaning that the analyst (apprentice) tries to observe and understand how the user (master) behaves in his environment. In this setup the analyst and user have an active relationship where the analyst interacts with the user generally by asking questions about the current task. After the observations have been done the analyst tries to design a series of requirements for the system to be. Based on these requirements the future system will support the user by partially or fully automating parts of the work process.

In contextual inquire the context only defines the work environment of the user and it does not necessarily represent an input for the future system. The context here is rather associated with the user and not with application itself. So in this way we can talk about context-aware requirement engineering but can we say that the system based on these requirements is also context aware? No, the system itself is still not aware of its context. Thus when engineering requirements for a context aware system we have to consider context as an input for the application. The application should be able to reason about this input and then generate a context based output.

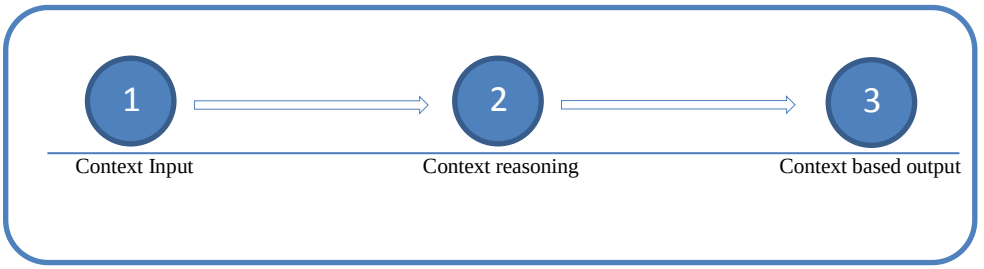

Fig 1, Flow of context information

3.2 Context as an Input

Our application behavior is highly dependent on input, mostly user input. Just imagine how our daily uses of computers and software will be if there will be no keyboard or mouse. We use these peripherals to tell our applications what we want and the applications than react to our input. In a similar way if we will like to have applications behave accordingly to the context, we should give them the context as an input. However as users we cannot do this, it will be just a plus of burden for us and probably also rather expensive in an enterprise setup. Therefore, the application should be designed to read the context information by itself from the surrounding environment. The analyst must also keep in mind that the application environment is not fully known, so we have to work with a high level of uncertainty.

Another key factor to be considered when doing context based requirement engineering is the quality of context (QoC). Even for a context-aware system the behavior should not be fully dependent on the context. The application should be fitted with a default operational mode that can be activated when the context information is

not sufficient to generate a valid context based output. According to Sheikh et al. [4] QoC can be used to describe the precision or freshness of the context. The system will have to rely on low level devices like sensors for reading context information. This means that in some situations the *image* of current context will have to be constructed using information from more than just one source. For example, if the mobile calendar specifies that the user is now in a meeting so the ring volume should be set on silent, the system, here a smartphone, cannot rely just on the calendar information to decide if the user is really attending the meeting. Rather it can combine this information with the location data provided by the cell-id or GPS, the location is can then be compared against the event location in the calendar. In this example we have the context made up of two elements, if one of them is not available, what should the system do? If the volume is set low and the user is not really attending the meeting we then have a conflict between what the user expectation are and the system behavior. This is of course a basic example but it is nevertheless showing a challenge that the analyst will face when trying to identify requirements which will reflect completely the user wishes for the system to be. As well, it becomes obvious that there is a need for a good mechanism that can be used to reason about the contextual data and the level of adaptation the system can undertake given these data.

3.3 Context based Output

In the previous chapter I have tried to explain how context is similar to just any other input an application requires in order to fulfill the user requirements. One aspect that is maybe not that obvious when thinking about context-aware system is that the output of the application should ideally also be context based. In fact, two different types of outputs can be identified with a context-aware system when context data is changing:

- Execution of services in an automatic way;
- Presentation of information or services;

In addition to this, the system can associate and record current user actions given the current context. We are talking about a simple learning module where the system is capable of reasoning and learning of one's user actions when a change occurs in the context.

The first behavior refers to the capacity of the system to execute services with no user intervention whenever the context is right. For example, in today's mobile devices the update and sync services are automatically started when the device has access to a data connection.

The presentation of information and services is similar to what most of the smart phones today do with the weather information. They detect changes in location and then update the information that is presented to the user. A bit more though must however be given to this point. If for example we are designing a mobile application that is supposed to inform its user about the current local events; Movies, presentations or exhibitions, whatever the user specified as interesting to him. In this example I suggest that the context to be used in a double way. Location information can be used to build a list of events in the area and then the current activity and future task of the user can be used as an input to sort out irrelevant information. If for example the user's calendar

specifies already a dentist appointment for Tuesday evening, presenting him a local theater play for that evening could be irrelevant. So it is maybe wise when designing and application to not consider only the current context but also future context information that can be accessed. The following steps describe the example above, notice that a two level reasoning over context is being proposed:

1. Get current location information; (City, Town)
2. Build a list of local events (Movies, presentations, exhibitions) ;
3. Get future context information; (future calendar events, business trips, appointments)
4. Sort out irrelevant events; (events that overlap with events at 3)
5. Present information to the user;

3.4 User Involvement and Context Models

In the previous section I have shown how context can be used when engineering a context aware system. However, this was more from an analyst point of view. The scope of requirements is to also get an idea of what the user wants and finally what he can expect from the finished system. Making the user understand what context is and how is it influencing the system is in the end another challenge for the requirements engineer.

In most of the requirement elicitations methods the users have an active role but in many cases they have a difficult time expressing their needs or expectations. Context-aware systems make the situation even more complex. As seen in the previous chapter, a context-aware system can sometimes automatically execute services. This is nevertheless a nice feature of this type of application but it has also a not so bright side. That is, the user might feel that he is not having full control over the application. It is therefore critical for the requirements engineer to understand not only the needs of the user but also his limits. One user might want his phone to go automatically on silent when in a meeting but he may not want to miss a call from his pregnant wife for example. This can for sure be a real situation and is the proof that when engineering automatic system behavior much though must be given in order to identify all of these situations. On the other hand, not much thinking has to be done to realize that covering the entire range of situation it is actually impossible. The only thing that we can do at the moment is to involve the user even more than we do with traditional applications.

By enlarging our elicitation vocabulary with keywords like *when, where* or *how* we can maybe map the traditional used *what* to places and certain situations. We can start by facing the user with the question: ***Where*** *are you, most probable, going to use the system?* The answers can then be placed in a simple table, like this, making it simple also for the user to understand and follow the process. The table below, pictures a way of capture requirements and mapping them to location, for a context-aware automatic vacuum cleaner.

Where	What shall the system do	What shall the system NOT do
Living room	operate at night when energy is cheaper	operate when lights are off (in order to avoid tripping)
Kitchen	operate at night when energy is cheaper	operate when wet floor is detected

Table 1, Example mapping of context to requirements where the used context is location.

The user has first identified the places where he will like the device to operate and then for each of these places he identifies what the system shall do.

The need for *What shall the system not do* is maybe not clear; but as such a system does not rely directly on user input for a correct operation or control it is important to allow the user specify what he will not like the device to do. Of course, this *not* wishes can also be expresses in the *What shall the system do* but I believe that users tend to think more about what they want than the opposite so this is why I think is important to have it on its own column; just to help the user think and visualize easier in the direction of *shall not.* Once the places have been identified by answering the *where* the Analyst and the user can then move to *when* and try to identify activities and situations.

When	What shall the system do	What shall the system not do
In a meeting	set volume on silent block incoming calls	block calls from family members.
…	…	…

Table 2, Example mapping of context to requirements where the used context property is current activity.

Such tables can be created for all the context properties presented in 2.3; combinations of several properties can also be made if required but addressing them individually is recommended in order to help the user focus specifically on a context property rather than having him thinking about the whole range of context changes in the same time.

Once the requirements are done, they need to be organized in models. Models are used to write down and organize the requirements in such a way that they are easy understandable both for the user and for the development and test teams. Of course one might argue that once you have them in tables they are organized enough and already quite easy to understand; However, here, the work of Desmet et al. [5] is worth mentioning as he had created a context diagram that enforces analyst to first think of a system in the classical way, un-ware of its context and then refine it by adding context dependent adaptations triggered by changes at certain variation points. The example in Fig. 2 shows that an incoming call triggers a different behavior of the cell phone function of the battery status, location or current time.

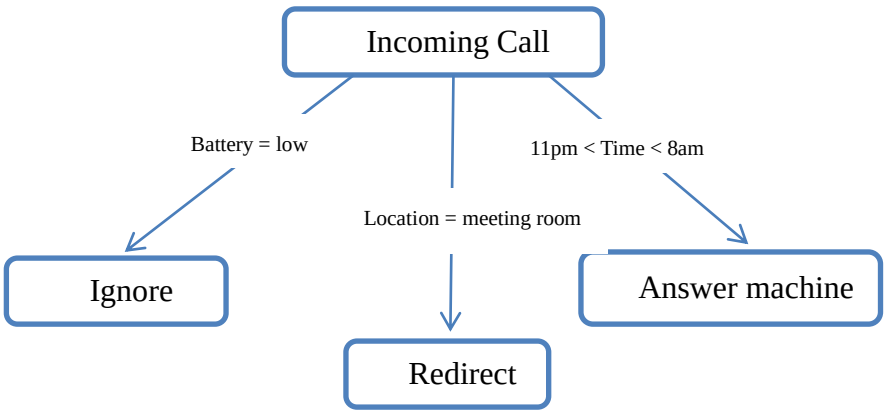

Fig. 2. Context diagram for a context-aware cell phone incoming call function.

For this particular example the diagram seems to indeed model a very good image of both adaptations and contextual constrains, but it is well known that diagrams have a problem with scalability so the diagram for a more complex system might be hard to understand.

3.5 Environmental uncertainty

In the previous chapters I have tried to define context and how is the context data being used in context-aware systems. While now it can be said that a context-aware application is dependent on its context as an input; the application environment is also very important for a context-aware system. Though maybe not so obvious, application context it is different from application environment. If context can be shortly defined as "*every piece of information which is computationally accessible"* [5]; the environment can be defined as the surroundings of the application out of which the context information is being extracted. In fact, a context-aware system relies on its environment for the quality of the contextual data it has access to. The weather service on smartphones for example, relies on the presence of a nearby antenna in order to determine the location of the phone and based on that, update the weather information. But if there is no antenna in the nearby environment no update can be made, therefore it can be said that the phone relies on its environment to provide the means for accessing contextual data. If earlier it was said that it is hard to envision at design time the entire range of contextual situation; the environment brings even more uncertainty in the picture. It is impossible to know where the application will be used and if the application will have access to sufficient contextual information in order to perform adaptation and eventually meet its goals.

Researchers had recently looked for ways to deal with this uncertainty. One interesting proposed solution is the RELAX requirements language.

3.6 The RELAX language

Going a bit back, to the context-aware vacuum cleaner, the user specifies that the vacuum should operate only at night when energy is cheaper, but on the other hand suggest that the vacuum should not operate when the lights are off in order to avoid accidents. These two requirements can make the operation of the vacuum uncertain, meaning that at a given point in time by trying to satisfy its requirements the vacuum cannot actually operate. If we assume that for example another requirement is introduced, that is:

The vacuum shall clean the floor every second day

we can then imagine under this conditions that the owners of the house go for a four day skiing trip during the winter holidays and they forget to turn off the vacuum. We can also assume that the price of energy is lower starting with 8 pm and the vacuum is aware of this. This means that the second day after its last operation the vacuum will try to clean shortly after 8pm, but given that in December at that time is already dark, the vacuum cannot really operate. One requirement specifies that it should not operate when there is no light. Thus, the vacuum had reached a requirements conflict. This

situation was obviously not envisioned in the design processed and because the vacuum does not know how to behave in uncertain situations will just do nothing.

The Relax language has been designed to allow the requirements engineer go around the strong meaning of the traditional *shall* and allow uncertainty in the requirements of a system. By using Relax the analyst can identify which goals of the system are critical and which can be relaxed. The relaxation is done by following the process diagram in Fig 3 [6]:

1. The analyst starts by defining a set of requirements, in a traditional SHALL way;
2. For each of the requirements defined at step 1, the analyst should consider whether it is really mandatory for the requirement to be satisfied. If it is critical for it to be satisfied then it should not be changed and the analyst can move to the next requirement. If on the other hand meeting the requirement it is not a must, then the *SHALL* should be replaced with a RELAX operator, thus relaxing the requirement.
3. For each relaxed requirement, the analyst shall then try to identify what are the environmental aspects that are prone to change and that can trigger the relaxation of this requirement. These aspects should be noted with the ENV keyword.
4. At this step, for each relaxed requirement, the analyst should look for properties of the environment that can be observed. These properties should be noted with the MON keyword and they will in most cases be identical with the properties at step 3.

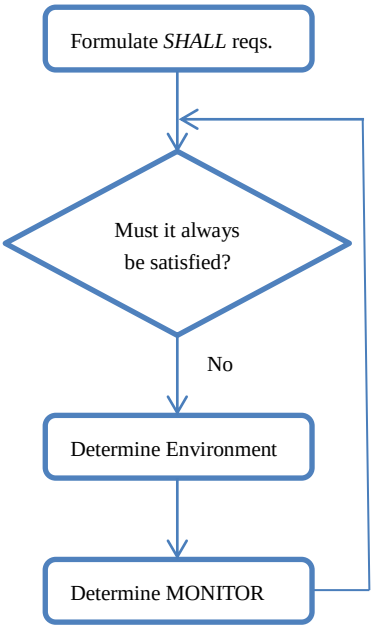

Fig. 3. Relax Process

In our example we can assume that cleanliness is more important that costs so the users and analyst will choose to change (RELAX) the

"Cleaning SHALL be done only during the night when the energy price is cheaper"

To:

"Cleaning SHALL be done AS MANY TIMES AS POSSIBLE only during the night when the energy price is cheaper."

Notice the use of *AS MANY TIMES AS POSSIBLE*, this is a RELAX operator that specifies that a requirement shall be satisfied always when it is possible. By relaxing our energy requirement, the vacuum will still try to operate during the night but in the same time it knows that this requirement is not as important as *The vacuum shall clean the*

floor every second day. Therefore whenever needed the system can now trade between requirements and eventually avoid deadlocks as the one shown earlier.

The full RELAX requirement must also specify the environmental properties that can trigger a relaxation but also which are the properties that the system should monitor in order to stay aware of these environmental properties.

"Cleaning SHALL be done AS MANY TIMES AS POSSIBLE only during the night when the energy price is cheaper."

***ENV**: time passed since last operation; level of light in the environment;*
***MON**: internal clock; light sensors;*
***REL**: internal clock is used to compute the time passed from the last cleaning; light sensors provide whether the there is enough light to operate"*

Where according with the RELAX documentation [9]:

***ENV**: defines a set of properties that define the system's environment;*
***MON**: defines a set of properties that can be monitored by the system;*
***REL**: defines the relationship between the ENV and MON properties;*

Covering fully the semantics and specifications of RELAX is outside the scope of this paper, an extended coverage of the language being offered in [6], [7], [8], [9]. My intention here was to merely cover briefly one of the solutions proposed for dealing with uncertainty in requirement engineering.

4. Conclusion and Summary

A context-aware system comes with a complexity that the user may or may not understand. It is therefore the job of the requirements engineers to somehow hide this complexity while trying in the same time to capture the needs of the user in the most pragmatic and efficient way. I have talked in the previous chapters about what context-aware systems are and by using many different examples I tried to create a good picture of the many shapes such a system can take. As far as requirements engineering is concerned, I have tried to show that yes, there is more complexity involved but this challenge can be addressed in some cases just by re-thinking traditional requirement engineering methods.

At the moment there is no agreement on what is the right way to go, and even though new methods have been suggested there are very few real world systems that make use of these methods and it is therefore hard to understand their advantages or disadvantages, for instance, the RELAX language is for sure an interesting idea but the examples given by the authors are maybe a bit too far-sighted. This could be why at the moment; most requirement engineers that do work with context-aware systems prefer to actually improvise their own methods.

5. References

1. B. Schilit, N. Adams, and R. Want. (1994). Context-aware computing applications (PDF). IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94), Santa Cruz, CA, US. pp. 89–101.
2. W. Sitou and B. Spanfelner, Towards requirements engineering for context adaptive systems, Computer Software and Applications Conference, 2007. COMP- SAC 2007 - Vol. 2. 31st Annual International, vol. 2, pp. 593 - 600, 2007.
3. A. Dey, G. Abowd, P. Brown, N. Davies, M. Smith, and P. Steggles, Towards a better understanding of context and context-awareness," in Proc. HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, ser. Lecture Notes in Computer Science, vol. 1707. London, UK: Springer-Verlag, 1999, pp. 304-307
4. K. Sheikh, M. Wegdam, and M. Sinderen, Mid-dleware support for quality of context in pervasive context-aware systems, in Proc. PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops. Washington, DC, USA: IEEE Computer Society, 2007, pp. 461-466.
5. B. Desmet, J. Vallejos, P. Costanza, W. De Meuter, and T. D'Hondt, Context-oriented domain analysis, in Proc. Modeling and Using Content, ser. Lecture Notes in Computer Science, vol. 4635, 2007, pp. 178-191.
6. J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, J. Bruel, RELAX: Requirements-Aware System. IEEE Computer Society 2010.
7. Jon W., Pete S., Nelly B., B.H.C. Cheng and Jean-Michael B. RELAX: Incorporating Uncertainty into Specification of Self-Adaptive systems.
8. J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, RELAX: A language for self-adaptive requirements. In Service-Oriented Computing: Consequences for Engineering Requirements, pg. 24, 2008. SOCCER '08
9. J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, J. Bruel, RELAX: a language to address uncertainty in self-adaptive systems requirement. In Requirements Engineering (2010) pg. 177-196 Springer-Verlang, London 2010