Feedback in Context: Supporting the Evolution of IT-Ecosystems

Kurt Schneider, Sebastian Meyer, Maximilian Peters, Felix Schliephacke, Jonas Mörschbach, and Lukas Aguirre

Software Engineering Group, Leibniz Universität Hannover
Welfengarten 1, 30167 Hannover, Germany
{Kurt.Schneider,Sebastian.Meyer}@inf.uni-hannover.de
{maximilian.peters,felix.schliephacke,
jonas.moerschbach}@stud.uni-hannover.de,
lukasaguirre@gmail.com

Abstract. IT ecosystems consist of dynamically interacting subsystems, components, and services containing software. Companies provide parts of IT ecosystems, e.g. for airports, train stations, and shopping malls. Due to the complex interaction of subsystems, overall behaviour cannot be completely anticipated or engineered. IT ecosystems constantly evolve by adapting to new user requirements and to changes in their environment. On-going improvement requires feedback from users. However, feedback is not easy to get. This paper presents an approach facilitating feedback in context. It is gathered by mobile devices like Smartphones. Effective support for evolution needs to cover (1) identifying the component or subsystem a user wants to address, (2) the ability to send feedback at very low effort and cost, and (3) support for interpreting incoming feedback. We present an architecture, a framework, and an application example to put stakeholder feedback into context. Contextualized feedback supports providers in driving the IT ecosystem evolution.

Keywords: IT ecosystem, feedback, context, architecture, improvement cycle.

1 Introduction: Evolution in IT Ecosystems

Today, many technical systems and devices interact with each other. In public places like airports, train stations, or universities, citizens become stakeholders of tightly interwoven systems. Their mobile phones can be used to make reservations, pay tickets, and receive confirmations from banks, ticket counters, and train information systems. Software controls subsystems and services, all of which are developed independently. They often depend on each other to provide higher level services. Subsystems interact with other subsystems, and with users. We call a system an “IT ecosystem“, if those parts act and react autonomously or semi-autonomously [1]. This term is used as a metaphor. Bosch presents a taxonomy of ecosystems [2]. He starts from biological ecosystems and distinguishes human, social, and economic ecosystems. In particular, he is interested in software ecosystems: "A software ecosystem consists of the set of software solutions that enable, support and automate the activities and transactions by the actors

M. Ali Babar, M. Vierimaa, and M. Oivo (Eds.): PROFES 2010, LNCS 6156, pp. 191–205, 2010.

in the associated social or business ecosystem and the organizations that provide these solutions." Our notion of *IT ecosystems* emphasizes emergent behaviour similar to an ecosystem in nature, where different species and animals interact autonomously.

Stakeholders perceive resulting system behaviour as a complex “smart environment“. IT ecosystems are not designed as a whole; they rather evolve. Components and requirements change often, and new parts are added. Lentz and Bleizeffer state: "Modern IT ecosystems have evolved organically into complex systems of hardware, software, middleware components, applications, organizations, practices, application lifecycles, and job specializations." [3]. Changes in requirements are not easily recognized in IT ecosystems. Nevertheless, getting feedback on acceptance of their products or services is vital for providers to tune and improve their software. Traditional requirements engineering is not appropriate for software in an IT ecosystem for several reasons, as a comparison shows:

- ***In traditional, individual software projects***, requirements engineering has been a distinguishable project activity. Requirements could be elicited, analyzed, and validated before implementation started. Iterations and prototypes may be advisable, but requirements are still elicited before they are implemented. Usually, there is a defined user community. Interviews and workshops are the recommended approach for eliciting and validating requirements. There is a good chance to reproduce observations and problems. When a small system does not behave as expected or desired, users have a good chance to recognize the deviation. They will be able to identify the responsible software system and to contact the developers.
- ***In "IT ecosystems",*** new requirements and suggestions need to be identified while the system is already running. They result from on-going changes in environment, interaction, and user expectations. Stakeholder will recognize unforeseen or undesired system behaviour, but will often not be able to identify the responsible subsystem. Even if they could, it would take time to find provider contact information. In most cases, stakeholders may be angry or annoyed, but not provide feedback. From the provider perspective, opportunities for improvement are missed.

  Eliciting requirements proactively (instead of a reaction to a problem) is also difficult: Requirements of different user groups may diverge and will depend on context. Representative samples of users need to cover all groups and many contexts or conditions. It is expensive and ineffective to conduct interviews or workshops with such a large sample of users: Instead, companies use questionnaires, marketing studies and other tools to reach many (potential) customers [4], which is again a suboptimal solution:

  - Stakeholders are interrupted in their original tasks.
  - They will often not remember or report a problem - unless they encounter it just before they were asked.
  - Emergent effects may occur in an IT ecosystem with all its interacting subsystems. Some phenomena might not be reproducible in a laboratory setting or user test if the context and conditions differ.

- ***Our approach to "IT ecosystems":*** Focused feedback channels are made available to stakeholders *when and where* they encounter a situation or problem they consider worth reporting. Feedback can be given via Smartphone. By integrating this feedback into an improvement cycle, stakeholders engage in a community effort for improving their own environment – and for their own benefit.

The tools that are required to support this approach need to facilitate and guide the identification of appropriate addressee subsystems in the IT ecosystem. Whatever is "close" might be the responsible part. Therefore, we suggest using heuristics on geographical and logical proximity to identify addressees for feedback: Context can be captured implicitly and at minimal effort by locating users automatically. In addition, tailor-made mechanisms for pre-sorting feedback are proposed to facilitate analysis of feedback and fast improvement reaction.

Accordingly, this paper makes a three-fold contribution:

1. We present a technical concept that enables feedback in context. It contains an architecture and a framework to be used in different applications.
2. We argue how providers of subsystems can benefit from feedback in context. Empirical evaluation of this aspect (acceptance, benefit) is not covered in this paper.
3. We show the technical feasibility of our concept by implementing it using technology that is currently available and in wide use.

Section 2 presents the main assumptions and concepts that underlie our approach. Related work is described in Section 3. In Section 4, we explain the architecture of our distributed support framework. An implementation of our concepts is presented in Section 5. We illustrate our approach and framework by applying it to an example (Section 6). With these mechanisms at hand, user evaluation and optimization of heuristics can now be the next step. We discuss the current status and conclude.

2 Assumptions, Opportunities, and Concepts

A number of trends in modern technology and society have created a new situation. So far, the only way for unsatisfied customers of software and systems to complain was by calling or writing to providers. However, spending extra time and effort for identifying and contacting the responsible provider prevented most people from giving feedback. Why would someone spend time and money to report requirements? And why would providers care to collect complaints after they already sold a product?

Assumption: *Competitiveness in IT ecosystems depends on available feedback*
Providers of systems, software, or services compete with others in an IT ecosystem. For example, traffic providers like Deutsche Bahn (German railways) or airports will depend on embedded software that interacts with mobile phones, proprietary business systems, and the internet. Dissatisfied users may turn away from services and look for a similar service from a competitor. In order to keep customers, and to keep customers satisfied, providers will need to improve their processes and their software-based products while they are running. The ability to recognize problems and implement

suggestions for improvement quickly will be a key to success in IT ecosystems [4]. Stakeholders (i.e., subscribed and potential customers) could be encouraged to indicate their feedback and desires. Thus, an important new communication channel could be opened. Providers could play an active role in shaping their portion of the evolving IT ecosystem.

Opportunity: *Stakeholders are familiar with new and ubiquitous technology*
Our work tries to support the upcoming generation of stakeholders who are familiar with video-equipped mobile phones and multimedia handhelds. Two new developments encouraged us to explore ad-hoc video and light-weight feedback in context (as explained in more detail in [5]):

(1) The advent of inexpensive ubiquitous recording and sending devices. For example, digital cameras, Smartphones and flatrates are in wide use today.
(2) A generation of stakeholders who have grown up using Smartphones and PDAs voluntarily in their private life. Today's high school and university students represent that generation. They are current and future customers.

Many people recognize faults and weaknesses in public systems (e.g., airport displays, navigation or information services). We assume that many citizens will be *able* to use their familiar mobile devices for giving feedback. We further assume that many technology-affine citizens are *willing* to provide feedback if that causes no or marginal cost and effort. According to Davenport [6], incentives do not need to be financial. Stakeholders may be happy to provide short feedback to enhance the systems they use themselves. When stakeholders wait for a service or a system reaction (e.g. at an ATM), this is an opportunity to get feedback. There is a two-fold benefit: waiting time appears shorter, and stakeholders can express frustration.

Concept: *Context from object perspective*
Context is an important issue for feedback. Many context-aware systems consider the *context of users* and adapt system behaviour to the context observed. In Fig. 1, they would consider objects A and B to be in the context of stakeholder S (dashed line). Our perspective in this paper is slightly different: The geographical and logical position of a user in an IT ecosystem is identified along several dimensions (GPS, WLAN, Bluetooth etc.). When a user indicates the intention to give feedback, the multi-dimensional position of his or her Smartphone is used to identify those components of the IT ecosystem that are most likely to be the intended addressee of that feedback. Administrators can define under which circumstances a stakeholder is considered "close". There can be individual definitions for each object. Instead of asking: “What objects are in the context of S?” (dashed line in Fig. 1), we ask: “What objects would consider S to be close enough for giving useful feedback to them?”

In Fig. 1, S is in the context of B. Object A has defined a narrow context and would not consider S close enough. If S triggers feedback, the Smartphone of S displays only B as a potential addressee, since only B considers S close enough. Please note that Fig. 1 maps the multiple context dimensions to a geographical model of “closeness” for visual presentation.

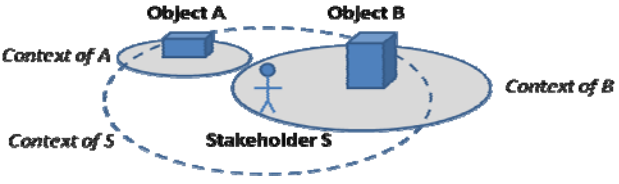

Fig. 1. Context depends on perspective: S is close enough to B, but not to A

Concept: *Context as a key to low-threshold feedback*
The main contribution of this paper is the use of multi-dimensional context of feedback for lowering effort and threshold to participate. As outlined in the introduction, we assume that proximity or closeness indicate appropriate addressees. Context is established by *physical* proximity to a real-world object (display, ticket machine, tree, car), by *logical* proximity to a Web site (as identified by the URL currently connected), or by being *in the range* of a bluetooth or Wi-Fi sender. We assume that "participating objects and systems" will be registered as potential feedback receivers. When a stakeholder wants to submit feedback, "close" objects are identified, and the stakeholder selects the best addressee for his or her feedback.

In [7] and [5], we suggest using ad-hoc video-clips for eliciting feedback and supporting requirements validation. In this paper, we will not discuss videos as a feedback medium, but focus on the framework and infrastructure for feedback.

3 Related Work

Non-traditional requirements engineering. In most traditional development environments, requirements are elicited early from stakeholders. Validating those requirements is an important prerequisite for good quality. Interviews and workshops are often used for bridging the gap between customers and software developers. Ethnographic approaches were recommended for observing and analyzing complex situation that are difficult to explore by asking stakeholders [8]. When a system is developed for an entire market rather than an individual customer, product management and market-driven requirements engineering [4] are more relevant than individual up-front interrogation. In that case, phases of building and phases of analysis and validation must take turns. Karlsson et al. [4] point to the drastically growing importance of feedback when an operational version of a system is supposed to be improved.

Fickas and Feather [9] state that requirements change over time. Requirements monitoring helps to automatically detect mismatches between the system functions and desired goals. Goals that have not been made explicit cannot be observed automatically. Our approach utilizes the users of the system directly to detect mismatches. User feedback can uncover mismatches between the system and their own personal goals - which may contribute to making tacit goals explicit.

Video clips are a straight-forward extension to the concepts presented in this paper. As discussed in [5], some researchers have investigated high-effort approaches in using videos [10]. For getting contextualized feedback from everyday situations, we advocate ad-hoc videos recorded on mobile devices by normal citizens. A video

shows a concrete situation in context, which is an advantage over textual feedback. Audio explanations can provide the intention of recording this situation. In text, however, context must be described explicitly, or it will be ignored.

Zachos and Maiden [11], [12] used their ART SCENE system on mobile devices to guide stakeholders through scenarios. By following those scenarios in concrete and contextualized situations, misunderstandings and invalid assumptions can be detected. Again, this approach is directed towards requirements engineers and stakeholders who are willing to spend a considerable amount of time on requirements validation. Our approach is complementary in nature. It enables ordinary citizens to send a short, but contextualized feedback within a minute. If many people participate, even small pieces may help to get a picture.

Sitou and Spanfelner [13] propose a set of possible models to capture the usage context and users expectations. They argue for the **multi-dimensionality** of the context in order to be helpful for requirements engineering. Sitou and Spanfelner observe users while they interact with the system in order to elicit new or changed requirements. We do not observe users, but enable them to provide feedback in context.

Dey et.al. suggest a software infrastructure for **smart environments** [14]. They gather context data from sensors and use it as a substitute for user input to trigger defined processes. Again, we do not substitute users and their activity but enable them to give feedback actively. Context is used to identify possible addressees. Sutcliffe, Fickas and Sohlberg propose a method for requirements engineering that takes into account the context of a person as parameter for requirements [15]. Rather than eliciting requirements from feedback directly, we encourage and gather light-weight feedback. It indicates where more in-depth requirements engineering is needed.

The concept of **derivating implicit feedback** is a common task in information retrieval to obtain better results by adapting to the user's needs. Fox et al. described how implicit ratings like the number of returned result sets or the duration of a session can be used to get an indication of user satisfaction [16]. Another class of sources for implicit feedback is analyzing clicktrough data from web logs as described by Dupret and Liao [17]. These approaches rely on observing the user during the interaction with a system. Contrary to this, our approach is dedicated to an environment where observing the user is not an option, although recognizing the surroundings of the user is possible.

4 Architecture and Process of Feedback in IT Ecosystems

An IT ecosystem consists of several subsystems, components, and services. Not all parts of such a system will participate in soliciting feedback. For example, only the passenger-related parts of an airport may be in focus, while baggage handling, human resources, or runway services may not be taking feedback. Therefore, elements must register in order to qualify for feedback. We call registered elements "SmartObjects".

There will be many registered elements. Their representatives, SmartObjects, are managed by a central feedback management unit. Although that unit can be distributed for increased efficiency or robustness, we consider it *one central unit* from a logical perspective. It contains software to represent and manage all SmartObjects. Management includes features for defining new SmartObjects and for specifying the

circumstances under which a given stakeholder will be considered "in the current context" of that SmartObject.

We postulate stakeholders to use a mobile device with several sensors for different dimensions of context, such as Bluetooth, Wi-Fi, GPS, or the URL of a website. When stakeholders want to provide feedback, they connect to the central unit. Context information provided by the mobile device is evaluated in the central unit. According to a matchmaking algorithm (as exemplified in Section 5), a list of SmartObjects is presented. They are potential addressees since the stakeholder is in their context. The final selection of the addressee SmartObject is made by the stakeholder. Depending on the definition of that SmartObject, a few questions may be displayed on the mobile device. Answers are sent back to the SmartObject as part of the feedback interaction.

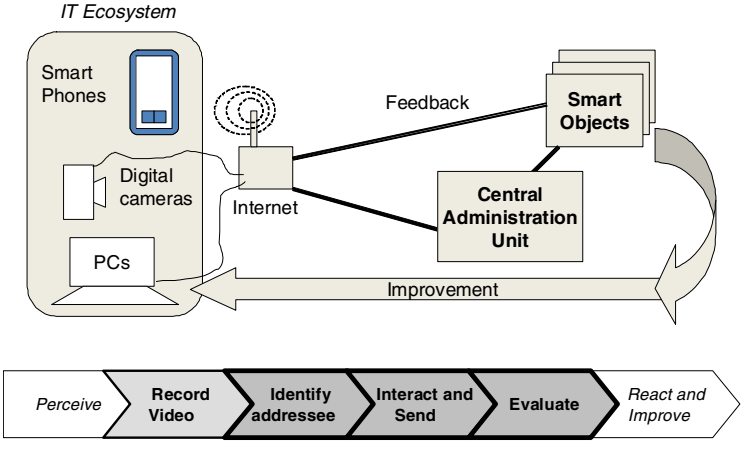

Fig. 2. Architecture of a framework and process for feedback in context

See Fig. 2 for an overview of the distributed architecture and the improvement process for feedback in context. Dark process steps are at the core of this paper, while video recording is mentioned only briefly. White process steps are required to close the improvement cycle, but they are beyond the scope of this paper: A stakeholder must perceive a trigger for giving feedback; and a service or software provider company must react to feedback. We do not discuss these steps here.

The architecture as sketched in Fig. 2 is characterized by several aspects:

- Feedback in context is achieved by a distributed framework of interacting parts: (1) The Central Administration Unit, (2) SmartObjects, and (3) software embedded in participating mobile device or digital cameras. This architecture reflects the distribution and flexibility of the IT ecosystem.
- Each of the three parts can be integrated with provider or mobile device systems. All parts together make up the *feedback in context* framework.
- There is a difference between a real-world object (subsystem, software, service) and the SmartObject representing it in the framework: Note that even software-free objects like restaurants or escalators can be represented by SmartObjects.

For example, a stakeholder can be specified to be in the context of a house whenever their GPS coordinates are within 100 m of each other.

- Feedback for all SmartObjects is first received by the central unit. Once the addressee SmartObject is identified, the feedback call is handled by the SmartObject software. It also collects data for evaluation.

Fig. 3 shows an overview sequence chart of the feedback interaction. While Fig. 2 is a static view of the architecture, Fig. 3 highlights dynamic aspects.

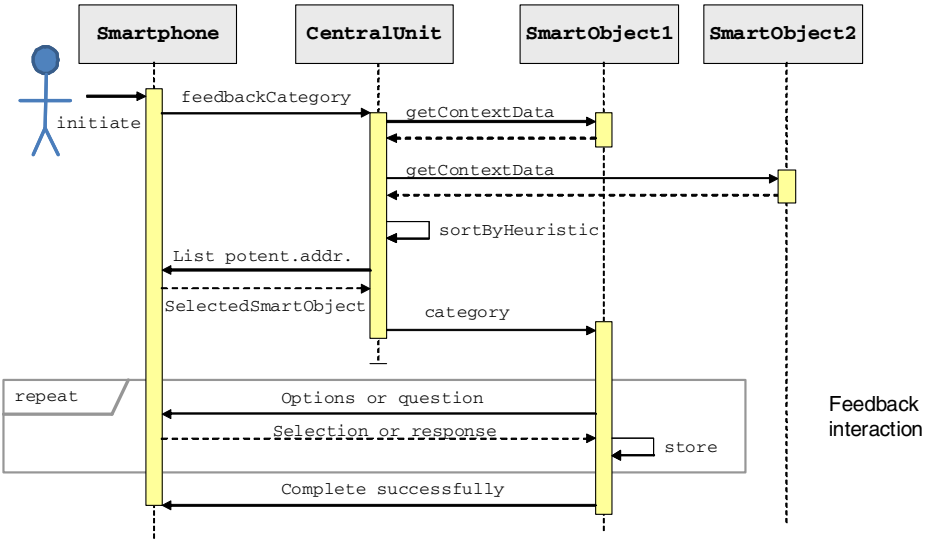

Fig. 3. Sequence chart of general interaction during feedback in context

Submitting feedback in context can vary in detail, but it always takes the steps sketched in Fig. 3:

1. Something triggers a stakeholder to send feedback, e.g. a problem concerning the behavior of a ticket machine or a software service.
2. The mobile device has software which connects it to the Central Unit.
3. According to the multi-dimensional context specifications, several SmartObjects may qualify as addressees ("within range of our service Wi-Fi" or "in 20m range of GPS…"). Their context definition is evaluated heuristically by the Central Unit.
4. The Central Unit determines a list of SmartObject "in context" and submits it to the user's Smartphone. This is the list of potential addressees.
5. The Smartphone displays the list, and the stakeholder selects the objects he or she wants to provide feedback to, e.g. SmartObject1 in Fig. 3.
6. The selected SmartObject may establish a short feedback interaction by presenting options for selection, or by asking simple questions.

5 Implementation of the ConTexter Framework

ConTexter is an implementation of the architecture and technical concepts described above. The ConTexter framework was implemented in four parts which can run on independent computer systems. They can be mapped to the architecture.

1. The GUI is used by the administrator to create, define, and edit SmartObjects. It is a Java Swing Application and works as a client that connects to the Central ConTexter Unit as well as to the SmartObjectAdministration (part of the Central Unit) via Java Remote Method Invocation (RMI).
2. The mobile part is used by the Smartphone user to submit feedback by mobile phone. We used a G1 Smartphone running the Android operating system.
3. The Central Unit identifies relevant SmartObjects by the context data provided by the G1. It returns information on how to connect to different SmartObjects.
4. The SmartObjectAdministration in the Central Unit provides an interface between the G1 Smartphone and selected SmartObjects.

Mobile ConTexter components use standard Android APIs. The accuracy of different modules of the G1 was a problem in general. In particular, the calculated GPS positions differ and depend on the current environment. To handle this problem, the administrator can define and adjust the range of tolerance with each SmartObject that has a GPS context. The actual context (via GPS, URL, WLAN, Bluetooth) is identified by a separate program thread to let the stakeholder use the GUI meanwhile.

URL context is retrieved by searching the browser's cache for the last visited website. URL context is only classified as relevant if the visit of this website was within the last three minutes.

The **SmartObjectAdministration** is necessary because the Android API does not provide any remote method invocation (RMI) functionality. We had to run the communication based on standard sockets communicating with our own protocol. In real applications, it is infeasible to assign each SmartObject a separate port. To avoid the need for one port per SmartObject the SmartObjectAdministration uses only one port and listens for incoming requests from the G1. It forwards its socket to the currently chosen SmartObject. Along the same lines, the SmartObjectAdministration is implemented as one process only, which dynamically instantiates SmartObjects on demand - instead of running one process per SmartObject permanently. This also opens the opportunity to run several SmartObjectAdministrations in parallel on different machines. This may be useful in a commercial setting where more than one software provider uses ConTexter feedback independently.

Heuristic for sorting potential addressee SmartObjects. When a call reaches the server, it calculates a preference value for each potential addressee (SmartObject). To determine the relevance of a SmartObject as potential addressee, a "context relevance index" is calculated. More relevant objects are displayed higher on the mobile phone SmartObject list. For each SmartObject: The administrator can adjust priorities of different context type via adjustment factors. A SmartObject's relevance is not only determined by its context alone, but also by the number of calls in which it was finally selected to be the intended addressee by the stakeholder. This heuristic is based on the

assumption that an existing problem will trigger several feedbacks. Counting calls and finally selected SmartObjects is an element of learning and adaptation.

Many other heuristics can be formalized and compared: Time and environment conditions, even weather and history could be included. We decided to start simple and investigate refinements later, during usage evaluation.

In our implementation, we decided to implement the management of actual context data in a central database located in the Central Unit: The request for context data is implemented by database queries in the Central Unit instead of involving SmartObjects. Fig. 3 is a good representation of the logical interaction, while our implementation uses one of many performance optimizations one could imagine.

6 Example Case Study: UniImprove

Our concept of multi-dimensional context of feedback and the framework architecture are independent of any particular implementation. In the previous section, core aspects of our implementation in the ConTexter framework were presented. In this section, we illustrate the concepts introduced above.

In this example, Leibniz Universität Hannover is considered an IT ecosystem. Our scenario pretends the university board decided to start the UniImprove initiative. It uses ConTexter to guide improvement of university services, software, and other objects under university control. Enrolled students are invited to register (as stakeholders) at the Central ConTexter Unit. Registration is carried out over the internet. Fig. 4 illustrates some contexts and SmartObjects on a Google Earth map: The main gate in front of our university building (a) is specified by its GPS coordinates and by a Bluetooth sender (b). When students follow path (c), they reach the information display in the entrance hall, which is a registered SmartObject. Its context is defined by a Wi-Fi network (d). There is also the University Restaurant (e: GPS context and URL for menu) and the pathway to the next building (f: GPS). The pathway is included and represented by a SmartObject since there were many complaints about dirt and poor lighting in the past. Finally, there is a Bluetooth antenna of our department (g).

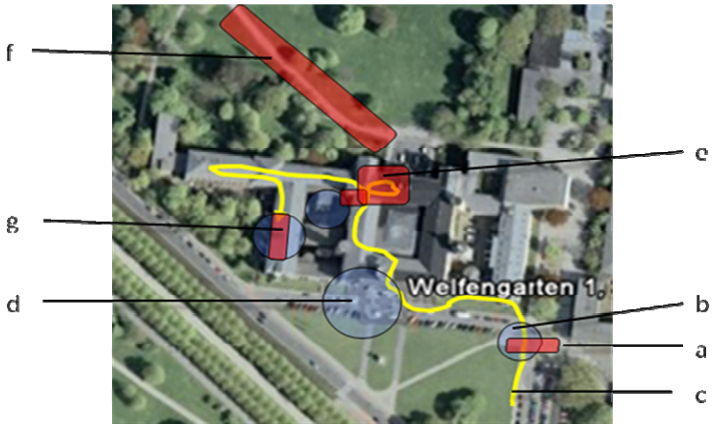

Fig. 4. Example of registered objects and context areas along a path at Universität Hannover

Fig. 4 is for illustration only. Real ranges and areas may differ, and there are far more registered objects in a real IT ecosystem. When a new *object* is registered at the Central ConTexter Unit, multi-dimensional context conditions are specified. A registered real object (e.g. tree, pathway, display, software, service) can be relevant when the stakeholder is "close" in any of the available dimensions. Fig. 5 shows a situation in which the online syllabus (list of offered classes) is being registered as "Syllabus" SmartObject. For demonstration purposes, URL, GPS, and Wi-Fi dimensions have already been specified. A specific Bluetooth connection is just being added. Whenever a stakeholder is in range of any of those context dimensions, the Syllabus will be considered a potential addressee of feedback.

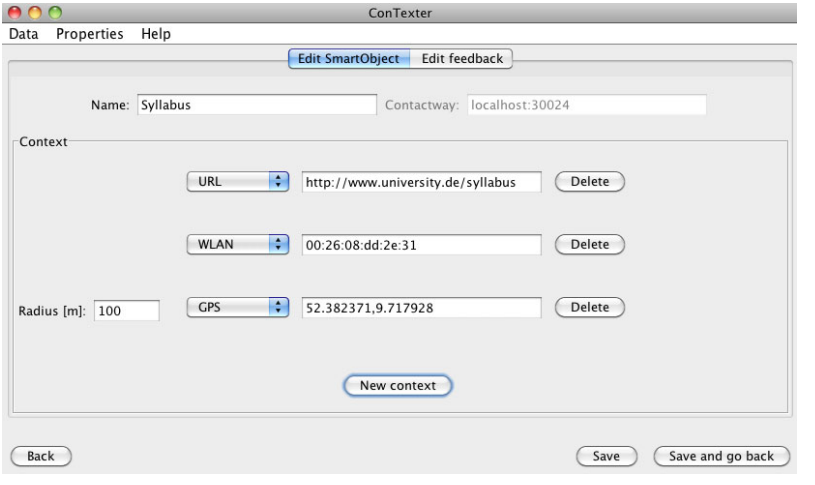

Fig. 5. Central ConTexter Unit during specification of multi-dimensional context

As soon as some SmartObjects have been registered, UniImprove can be used. When ConTexter is activated on a Smartphone, it asks for the category of feedback: complaint, compliment, or neutral remark. In Fig. 6 (left) a complaint is about to be made. In the next step (centre) all SmartObjects in the current context are displayed. In this example, the stakeholder selects Syllabus as the addressee. Since the SmartObject list was sorted by a heuristic, stakeholders make the final selection. Depending on the SmartObject definition, a few feedback options may be offered (right).

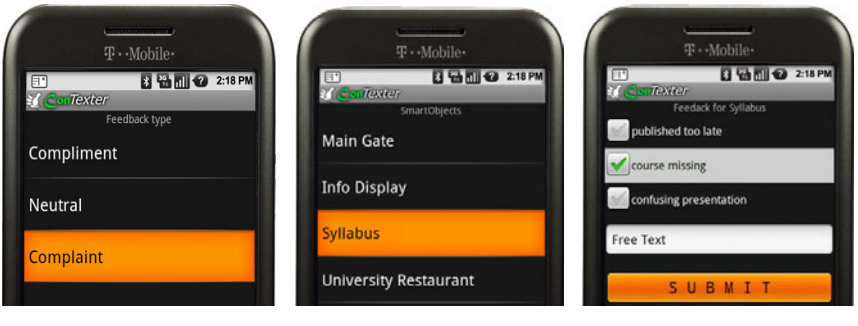

Fig. 6. Choosing category of feedback, final addressee, and submitting feedback

Stakeholders can check boxes and type free text if they wish. In an extended version of ConTexter, even previously recorded video clips can be attached to the feedback message. This short example covers most concepts introduced above. However, it illustrates only one possible implementation. We implemented other variants of the fine-grained interaction and heuristics. For example, the initial choice of a feedback category can be dropped. There is room for more optimizing heuristics. The goal is to present a reasonable list of possible addressees that most likely contains the object intended by the stakeholder.

7 Semi-automatic Feedback Evaluation

Soliciting feedback is an important yet difficult task. Submitted feedback must be used to the benefit of providers and stakeholders in order to justify the effort invested. In the introduction (Section 1), semi-automatic evaluation of submitted feedback was identified as one of the core opportunities. If many stakeholders participate, a large number of feedbacks will be received. By that time, a strategy for evaluation must be implemented and ready.

We present an example solution that supports simple pre-evaluation in many cases. As Fig. 7 shows in the realm of the UniImprove example, feedback options are defined in the Central ConTexter Unit when a SmartObject is registered and defined. So far, two compliments, a neutral type of feedback, and two types of complaints have been introduced. A third type of complaint ("confusing presentation") is just about being added. In all cases, stakeholders may include free text with their selections. During operation of the ConTexter framework, administrators and providers can use a similar interface to review the current status of feedbacks received. At this point, pre-defined options help classifying and visualizing the distribution of feedbacks.

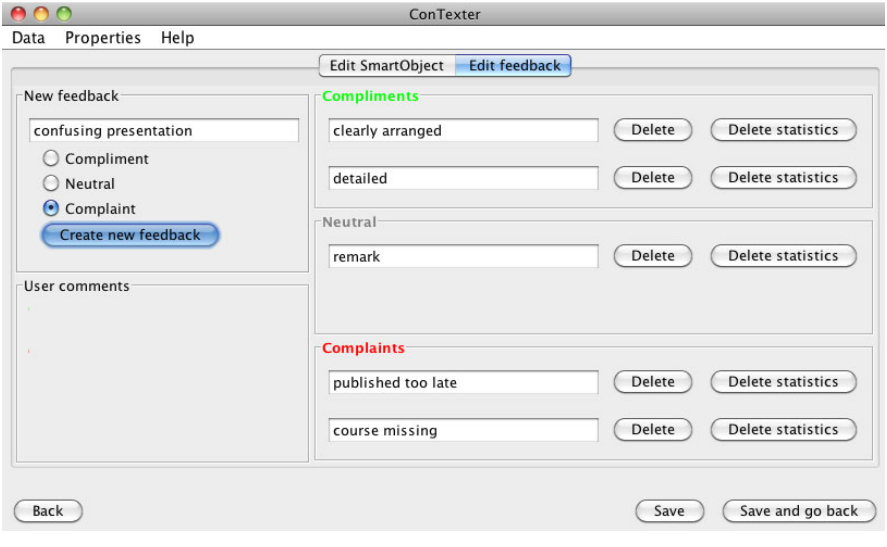

Fig. 7. Feedback options are defined during SmartObject registration

In Fig. 8, there is an overview of feedback types on the left, and a detailed view on all pre-defined options on the right. Free text annotations are shown in the lower left corner. In a more detailed view, original feedbacks (selected options with free text) can be seen for fine-grained analysis. In Fig. 8, two complaints refer to the title of a missing course.

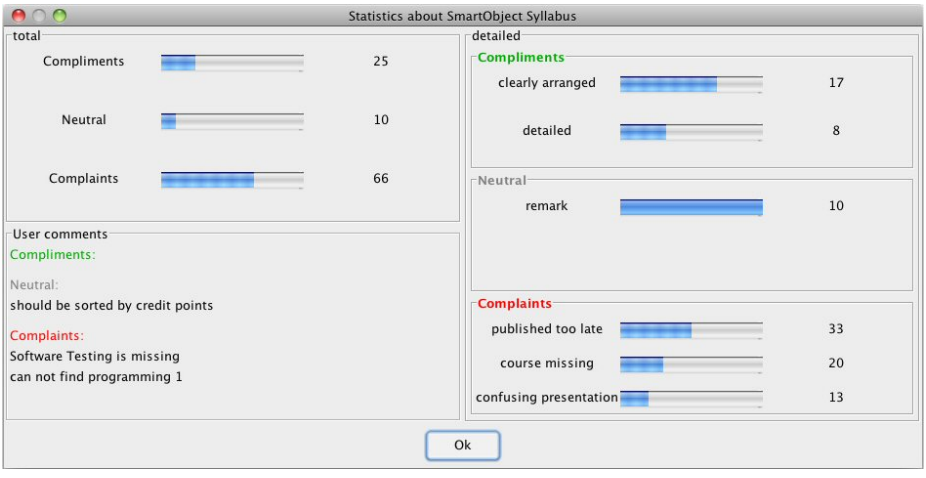

Fig. 8. Automatic classification of incoming feedback based on selected options

Obviously, the simple counters and visualizations can be refined and combined by marketing or requirements experts in a provider organization. Providers who registered more than one SmartObject can compare and use their respective feedback for advanced analysis. ConTexter is an implementation of a *framework* for contextualized feedback. Thus, providers can integrate the SmartObject statistics API into their internal evaluation systems by using the ConTexter framework. As presented in this paper, all aspects are integrated and facilitate installation and operation of that framework in a new application domain or software environment. After several feedbacks have been received, the above statistics screen Fig. 8 is created from specified feedback options and received feedbacks.

The components of the ConTexter framework offer the infrastructure for seamless interaction around feedback. When a new application is set up, administrators (i.e., providers) must register and specify SmartObjects with feedback types etc. Immediately afterwards, feedback can be sent and received. Writing or compiling code is not required.

8 Summary and Conclusions

Technical systems and software-controlled subsystems continue to interact more and more. Once this interaction exceeds central control, IT ecosystems start to emerge and evolve. The interaction of their subsystems is difficult to understand and may be impossible to anticipate.

Commercial software and service providers need to update and improve their subsystems if they want to stay competitive. Feedback from users and stakeholders is an essential input to continuous improvement. Stakeholders perceive an IT ecosystem as a smart environment and may not be able to distinguish all its parts. For the first time, the new generation of customers and stakeholders have the technical prerequisites and personal ability to recognize sub-optimal system behaviour – and to report it through contextualized feedback. This is a new opportunity, and we present an approach of seizing that opportunity.

Our approach consists of a technical framework, infrastructure, and concepts for applying them in an improvement processes. We first describe the architecture and framework in general. Then, we present the ConTexter implementation of that framework. ConTexter framework and UniImprove application example demonstrate that our concepts can be implemented with current technology. They establish a feedback and learning cycle for continuous improvement of IT ecosystem. We will continue exploring applications and extensions, such as the potential of ad-hoc video clips attached to contextualized feedback.

There are numerous open research questions in this field. Many interesting questions were actually stimulated by the work presented in this paper: How many stakeholders will provide feedback? Is it sufficient to lower effort, or do providers need to grant incentives for good feedback? What is the optimal proximity heuristic for SmartObjects? How should related SmartObjects be organized? Empowering semi-automatic interpretation is another research area that we have only touched upon in Section 7. We expect answers to vary significantly over different application areas and implementation alternatives. For example, acceptance might depend on perceived reaction and improvement time as much as on feedback mechanisms. The impact of our approach will be influenced by the personality of users such as their affinitiy to technology. Even the brand of supported Smartphones could have an influence. We are currently developing an iPhone interface to be used in addition to Android Smartphones to study that effect.

By applying our approach, subsystems of IT ecosystems can be explored and continuously evaluated by affected stakeholders. For example, an airport or an entire city could open that new feedback channel. Frameworks like ConTexter can support the evolution of IT ecosystems. Both service providers and their users can benefit from seamless feedback in context.

Acknowledgments. This work was inspired by our work in the NTH School for IT Ecosystems. NTH (Niedersächsische Technische Hochschule) is supported by Leibniz Universität Hannover, TU Braunschweig, and TU Clausthal.

References

1. Singer, L., Brill, O., Meyer, S., Schneider, K.: Leveraging Rule Deviations in IT Ecosystems for Implicit Requirements Elicitation. In: Second International Workshop on Managing Requirements Knowledge (MaRK 2009) at RE 2009 (September 2009)

2. Bosch, J.: Software Product Lines to Software Ecosystems. In: 13th International Software Product Line Conference (SPLC 2009), San Francisco, CA, August 24-28 (2009)
3. Lentz, J.L., Bleizeffer, T.M.: IT Ecosystems: Evolved Complexity and Unintelligent Design. In: CHIMIT 2007, Cambridge, MA, USA, March 30-31 (2007)
4. Karlsson, L., Dahlstedt, Å.G., Natt och Dag, J., Regnell, B., Persson, A.: Challenges in Market-Driven Requirements Engineerng - an Industrial Interview Study. In: Proceedings of Eighth International Workshop on Requirements Engineering: Foundation for Software Quality, Essen, Germany (2002)
5. Schneider, K.: Anforderungsklärung mit Videoclips. In: Proceedings of Software Engineering 2010, Paderborn, Germany (2010)
6. Davenport, T.G.P.: Knowledge Management Case Book - Best Practises. Publicis MCD, John Wiley & Sons (2000)
7. Brill, O., Schneider, K., Knauss, E.: Videos vs. Use Cases: Can Videos Capture More Requirements Under Time Pressure? In: Proceedings of REFSQ 2010, Essen, Germany (2010)
8. Hughes, J., O'Brien, J., Rodden, T., Rouncefield, M., Sommerville, I.: Presenting ethnography in the requirements process. In: Second IEEE International Symposium on Requirements Engineering, March 27-29, IEEE Computer Society, York (1995)
9. Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Proc. Second IEEE International Symposium on Requirements Engineering, March 27-29, pp. 140–147 (1995)
10. Creighton, O., Ott, M., Brügge, B.: Software Cinema: Video-based Requirements Engineering. In: 14th IEEE Internat. Requirements Engineering Conference (2006)
11. Zachos, K., Maiden, N.: ART-SCENE: Enhancing Scenario Walkthroughs With Multi-Media Scenarios. In: Proceedings of Requirements Engineering Conference (2004)
12. Zachos, K., Maiden, N., Tosar, A.: Rich-Media Scenarios for Discovering Requirements. IEEE Software 22, 89–97 (2005)
13. Sitou, W., Spanfelner, B.: Towards Requirements Engineering for Context Adaptive Systems. In: COMPSAC 2007: Proceedings of the 31st Annual International Computer Software and Applications Conference, Washington, DC, USA, pp. 593–600. IEEE Computer Society, Los Alamitos (2007)
14. Dey, A., Abowd, G., Salber, D.: A Context-based Infrastructure for Smart Environments. In: Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE 1999), pp. 114–128 (1999)
15. Sutcliffe, A., Fickas, S., Sohlberg, M.M.: PC-RE: a method for personal and contextual requirements engineering with some experience. Requirements Engineering 11(3), 157–173 (2006)
16. Fox, S., Karnawat, K., Mydland, M., Dumais, S., White, T.: Evaluating implicit measures to improve web search. ACM Trans. Inf. Syst. 23(2), 147–168 (2005)
17. Dupret, G., Liao, C.: A model to estimate intrinsic document relevance from the clickthrough logs of a web search engine. In: WSDM 2010: Proceedings of the third ACM international conference on Web search and data mining, pp. 181–190. ACM, New York (2010)