Towards an Infrastructure for Context-Sensitive Intelligence

Holger Mügge, Tobias Rho, Daniel Speicher,
Julia Kuck, Armin B. Cremers

Institute of Computer Science III
University of Bonn

A Vision of Generic Adaptive Mobile Devices

Our vision is that mobile computational devices should behave always appropriate for the situation the user is currently in. The device will become a *generic tool* accompanying the user in all its situations of his everyday life and it will embrace the complete scale of electronic functionality. We will use mobile devices equally for communication as for payment, as keys, as entertainment gadgets, navigation tools, and capable of providing arbitrary specific features.

Compared to personal computers we think there is something very special about mobile devices with regard to their generic usability. On the one hand, *mobile devices demand automatic adaptivity*, because its user is always on the move and does not have time for manual reconfigurations when he suddenly finds himself in a situation where he requires additional functionality. On the other hand, mobile devices are increasingly capable of *perceiving their current use context*. I.e. they can perceive context data as light, noise, temperature, time, user, location, and co-located devices etc.

Context-Sensitive Intelligence

Our current research project Context-Sensitive Intelligence (CSI) aims to exploit both, the demand for automatic adaptivity and the perceivableness of context data. Goal of the project is a framework for the development of generically adaptive applications. The main research topics of the project are:

- How shall we model context?
- How should we design context-driven adaptation? In particular: how can we mediate between the conflicting forces of anticipation and adaptivity?
- What do developers of applications and adaptation need as infrastructure?

Modeling Context

What is Context? First of all, the concept of context is always relative, i.e. it relies on a certain subject which is the current center of reference. For a given subject under observation its context comprises all perceivable data that is external to itself. We identified some important characteristics of contexts which should be taken into account for modeling:

- Clear Subject Definition
- Direct and Indirect Context
- Time
- Relevance
- Practical Perceivableness
- Explicit and Implicit Context
- History

Besides these phenomenological topics we collected a first typology of concrete contexts:

- *Computing Context*: e.g. network connectivity, bandwidth, CPU characteristics, running or installed services
- *Physical Context*: e.g. temperature, lighting, noise level
- *Spatial Context*: e.g. location, movement characteristics (direction, speed, acceleration), co-location (distance to other objects)
- *Temporal Context*: e.g. current time, duration of activities
- *User Context*: e.g. user profile, history of user activity
- *Social Context*: e.g. user role in current group, people nearby and their roles.

Our approach to modeling context is logic-based. We use facts to represent context values and rules to reason about contexts, for example to infer implicit context data. We also apply logic for mapping new contexts to adequate software adaptations in the form of event-condition-action rules.

Context-Driven Adaptation – Requirement Cases

Based on a survey on context modeling by Strang et al. (c.f. [Strang2004])[1] we set up a list of general requirements for context-driven adaptation:

- Distributed composition
- Partial validation
- Information quality and richness
- Incompleteness and ambiguity
- Level of formality
- Applicability to existing environments
- Flexibility of mapping context to adaptations
- Level of needed anticipation
- Richness of adaptivity
- Retrieval and selection of appropriate adaptations

[1] Strang et al. set up the first six requirements of our list. They focused on context modeling, and did not take adaptation – let alone unanticipated adaptation – into account.

- Expressiveness and usability of specification language

The emerging CSI development framework integrates five adaptation techniques to tackle the mentioned requirements.

- *Object-Oriented* adaptation (OO): specialization, delegation, design patterns. We assume a static OO language (Java, C#, C++, etc.) as primary implementation language.
- *Aspect-Oriented* adaptation (AO): statically quantified advices (before/after/instead), dynamically quantified advices (based on call stack)
- *Logic-Based* adaptation (Logic): asserting or retracting facts and rules of a deductive data base
- *Architecture-Based* adaptation (Arch.): dynamic plug-ins, dynamic service retrieval, interception of service dependency
- *Ontology-Based* adaptation (Onto.): compatible service descriptions, semantic driven adaptation

The following table shows how we expect combinations of these techniques to work together and fulfill the requirements.

	Distributed Composit.	Partial Valid.	Inform. Quality	Incomplete-ness	Formality	Applicability	Flexible Mapping	Level of Anticipation	Adaptation Richness	Retrieval, Selection	Lang. Express.
OO	**+**[2]	**+**[2]	**+**[2]			**+**[2]		**+**			
AO	**+**	**+**			**+**		**+**	**+**			**+**
Logic	**+**[2]		**+**[3]	**+**[2]	**+**[2]	**–**[2]	**+**		**+**		**+**
Arch.	**+**	**+**		**+**		**+**	**+**	**+**			**+**
Onto.		**+**[2]	**+**[2]	**+**[2]	**+**[2]	**+**[2]		**+**	**+**	**+**	**+**

We would like to discuss some selected requirements and how we tackle them by a certain combination of techniques during the workshop. In the following section we give brief explanations for each requirement case (column in the table).

Adaptation Techniques for the Requirement Cases

Distributed composition: context-sensitive adaptation always employs distributed composition by its nature. E.g. when you enter a large building and your mobile device becomes adapted to integrate a specific navigation system the software

[2] Argumentation relies on [Strang2004]
[3] Argumentation disagrees with [Strang2004]

on your device is as well a part of the new appliance as the additional navigation part. Typically location will rely on external software components interacting with internal software. OO comes into play as basic implementation language of the participating components, aspect oriented programming (AOP) contributes point cuts enabling to adapt multiple components when a cross cutting adaptation is required. The resulting join points can be distributed and act on inter- and intra-component level. Logic-based expression of context and reasoning can be distributed over the participating devices, e.g. sensors can deliver their models and reasoning implementations (as rules) to the adapted device. A component-based and service-oriented architecture serves as backbone of adaptivity allowing for adding and removing service components and changing their interactions.

Partial validation is important to the developer of adaptations since the interaction of context reasoning and diverse adaptations involves a high degree of complexity. Statically typed OOP provides with its type-checking facility some help to validate the executabilty of adapted programs a priori. The first-class concept of point cuts expands this validation option to cross cutting adaptations. On a coarser level a component-based architecture can validate the adaptation of the software composition e.g. by checking required and provided ports. On the semantic level, ontological annotations of the software components and adaptations can help to guarantee that a certain adaptation will be feasible and have the desired effect for the user.

Context information quality and richness: Context data is perceived by sensors and often inherently inaccurate. E.g. location information often depends strongly on the technique used and on the current physical environment. Bandwidth varies with distance and as a result the richness of the transferred data might change accordingly. Ontologies serve as backbone for modeling context data and meta data, e.g. accuracy, perception time. They can be implemented for example using OWL as specified in [Smith2004]. OOP allows for implementing dynamic strategies to cope with changes in data richness or quality. Provided context sensors deliver meta data for their accuracy etc. Logic can be applied to take these dynamics into account by differentiated reasoning.

Incompleteness and ambiguity: Te context awareness of mobile devices is not restricted to internal sensors. To the contrary, many applications will take virtual sensors into account which might be provided by the immediate environment or even through web-based services, e.g. weather forecasts, co-location sensors. Therefore, applications for generically adaptive devices should be able to cope with fluctuating context data. A logic-based context model allows dynamically asserting and retracting fact and rules for context data and reasoning. Integrating new sensors must of course be supported by the architecture, which is given for component-based and service-oriented architectures. Ontological descriptions of the sensors enable to select the required sensors and evaluate their interrelation to others.

Level of formality: The cooperation of original software with unanticipated adaptations requires a common understanding between all participating parts. Together with the high degree of interaction complexity formalization of concepts definitions is of much interest[4]. A logic-based description of context-awareness is relatively close to a formal representation and exposes concept definitions much clearer than an OOP-based implementation usually does. Ontological annotations provide for a clear definition of concepts on the semantic level. AOP provides with point cuts a first-class concept to express cross-cutting adaptations and therefore contributes to formality on the implementation level. This is increased as we have developed a logic-based extension of AOP which allows to use logic reasoning within the declaration of point cuts as described in [Rho2004].

Applicability to existing environments: With J2ME and several other platforms OOP can be applied on mobile devices easily. Even component-based service-oriented architectures can be realized, e.g. with several OSGi-based implementations like Kopflerfish or Equinox. Ontological interpreters are readily available for mobile platforms. Nevertheless we face problems with employing logic-based deductive databases on mobile devices and will probably be restricted to a client-server architecture for integrating them.

Flexibility of mapping context to adaptations is an important issue, in particular when the adaptivity has not been anticipated by the developer of the original software. As an enabler the architecture must allow for changes. This is granted in principle by component-based service-oriented architectures. Furthermore we require a loose coupling of services, which we support by extending OSGi towards distributed tracking and interception of services (c.f. [Gu2004] for similar approaches). The logic database allows for exchanging context reasoning at runtime, e.g. for interpreting newly defined situations as the user enters a certain setting. Aspect-oriented techniques allow describing context-driven adaptations in one place that adapt the software at several places and hence enhance flexibility for the developers.

Level of needed anticipation: for our vision of generically adaptive software anticipation of particular adaptations should be avoided or at least minimized. Nevertheless, in most cases adaptations require some preparation by the original software in order to be safely applicable and provide useful functionality. E.g. a co-location-based sorting of emails is certainly a useful feature that eases the users access to the most relevant mails for the user's current situation. To be applicable the sorting adaptation needs access to the mails to compute their relevance and it needs a way to modify the previous sorting algorithm. The CSI framework aims to provide the developer of the original software to open it up to later adaptation without too much specificities by generic variation points. We use

[4] Obviously the level of formality is strongly related to partial validation, since a higher level of formality will make partial validation easier.

OOP techniques like design patterns to provide such variation points with semantics as generic as possible, e.g. strategies, decorators etc. AOP techniques combined with logic description provide further generality for variation points and reduce the need to explicitly define them. Ontological annotations should also be used in a most general manner. The later needed specificity should arise from combinations of annotations. The service-oriented architecture we employ allows for integrating previously unknown components.

Richness of adaptivity: the richness of an adaptation denotes to what extend it exploits the potential integration with the original software. E.g. new functions may cross-cut several original features and should exploit that. The user interaction paradigms used before an adaptation should be respected and continued as far as possible. These samples show how the richness of adaptations depends on available information about the original software. This adaptation interface of the software should not only rely on the component level, but employ a gray box view instead. AOP features in particular when they are capable to describe fine-grained point cuts help to exploit the adaptability information (c.f. [Rho2006]). Hence, we should support the developers in providing suitable information easily. Ontological annotations provide the backbone for this kind of information on three levels: syntax, semantic and pragmatic[5]. Logic-based reasoning can also contribute to exploit available information about the software to be adapted.

Retrieval and selection of appropriate adaptations: In most adaptation scenarios a large number of services and potential adaptations are available. Based on the user's profile and his context appropriate adaptations should be selected. Ontological descriptions of software components on a syntactic level (i.e. compatibility in the OOP sense) and a semantic level (i.e. suitability of functionalities) serve as the backbone of adaptation retrieval. We plan to use additionally community feedback on a pragmatic level.

Expressiveness and usability of the specification language is important since we aim to support the developer of generically adaptive software. AOP point cuts combined with logic facts and rules provide a very expressive set of language concepts. Semantic annotations based on ontologies provide an easy to use and effective way of preparing the software towards adaptivity without anticipating concrete adaptations.

Status Quo and Urging Open Research Questions

In the previous sections we depicted the development framework for generically adaptive mobile software which we work towards in the CSI project. The scope under consideration is quite wide and we can not overlook by now to what extend

[5] The three levels of ontological information are shortly defined in the paragraph on retrieval and selection of appropriate adaptations in the same section.

we will be able to give suitable answers in terms of a guideline for developers or even in terms of a usable implemented framework.

We have elaborated the conceptual landscape and identified a list of requirements. Furthermore, driven by practical experiments we came up with a combination of several different software engineering techniques which we believe can be fruitfully combined to tackle the requirements. We currently see the following three research questions as most urging:

- *How can we balance between the conflicting requirements of a low anticipation level and the demand for rich adaptivity? How can such a balancing process be systematized or even formalized?*
- *What do typical developers of adaptive and adapting software expect as means for their work? How can we provide a usable integration of the mentioned techniques?*
- *How should the adaptation techniques be employed to let the end users gain advantages of adaptivity most effectively?*

References

[Gu2004] Gu, T., Pung, H.K., Zhang, D.Q.: Toward an OSGi-based infrastructure for context-aware applications. In: IEEE Pervasive Computing, vol. 03, no. 4. (2004)

[Laddad2005] Laddad, R.: Aop@work: Aop and metadata: A perfect match, part 1—concepts and constructs of metadata-fortified aop. Technical report, IBM Developer Works (2005)

[Rho2004] Rho, T., Kniesel, G.: Uniform genericity for aspect languages. Technical Report IAI-TR-2004-4, Computer Science Department III, University of Bonn (2004)

[Rho2006] Rho, T., Kniesel, G., Appeltauer, M.: Fine-grained generic aspects, foundations of aspect-oriented languages workshop, aosd 2006. (2006)

[Smith2004] M.K. Smith, C.W., McGuinness, D.: Owl web ontology language reference. http://www.w3.org/TR/owl-ref (2004) World Wide Web Consortium (W3C) recommendation.

[Strang2004] Strang, T., Linnhoff-Popien, C.: A context modeling survey. In: UbiComp Workshop on Advanced Context Modelling, Reasoning And Management. (2004)