A Model-Driven Approach for Developing Adaptive Software Systems

Thomas Hamann, Gerald Hübsch, and Thomas Springer

Technische Universität Dresden, Department of Computer Science,
Institute for Systems Architecture, Computer Networks Group
{Thomas.Hamann, Gerald.Huebsch, Thomas.Springer}@tu-dresden.de

Abstract. Context-awareness and adaptation are highly interrelated key concepts to build applications for heterogeneous and dynamic execution environments. While gathering, distribution, abstraction, and management of context is examined in research for several years, development of context-aware, adaptive applications, and the relations between context and adaptation are rarely considered. We present a model-driven approach for developing adaptive software. It comprises a design methodology, a set of software engineering artefacts, and a runtime platform for adaptive, multimodal software. Our approach focusses on modelling context information, context providers, and their relations to system functionality and user interface adaptation. We developed an adaptive plant maintenance application to show the feasibility of our methodology.

1 Introduction

Today, mobile and wireless technologies are an integral part of distributed computing environments building up a convergent platform for traditional as well as innovative services and applications. As a consequence new service and application areas are enabled, but also new challenges for application development arise from a mobility induced frequently changing infrastructure and the heterogeneity of: device capabilities; reliability and performance of network connections; user requirements and computing context.

Adaptation and context-awareness are closely interrelated key concepts for software executed in pervasive computing environments. The term adaptation describes the adjustment of a system to specific conditions or changes in its environment. The question to ask is: What is adapted to what? referring to *object* and *target* of the adaptation. Objects from an application's viewpoint are its processed data, communication for data exchange, and internal structure (functional components, their interconnections, and placement). The target of adaptation is the environment (i.e. available resources, user information and preferences, and context of system usage), characterized by *context information* (or short: context), which represents information about the state and changes of the environment.

A typical example illustrating the need for adaptation is an adaptive plant maintenance application. Maintenance workers have to visit the plant locations

to carry out their tasks. In this scenario, workers as well as management could benefit from adapting mobile and wireless technologies. Mobile workers could access location and task information or construction documents for particular plant equipment while being connected to a wireless network. Accessed data as well as its presentation could be adapted to the capabilities of the devices a worker is using and to the environmental conditions (e.g. selecting relevant document parts only for reducing the amount of transferred data or choosing the interaction modalities according to available modalities and ambient noise and light conditions). Moreover, application functionality could be reduced according to the processing power and storage capacity of the used device. The other way around, the management could track mobile worker's positions, capture their current activity and situation, to dispatch incoming service requests to a worker (a) with an empty task queue, (b) nearby the requesting customer or (c) with appropriate expertise.

This paper is organized as follows: Related work is discussed in section 2. Section 3 introduces our adaptive software design methodology. While section 4 discusses its major artefacts representing tasks, application data, and context. These models are refined over various abstraction levels and are transformed into code. The runtime environment for execution of the latter is described in section 5. After a prototype-based validation of our approach in section 6 we conclude the paper with a summary of our ideas and an outlook to future work.

2 Related Work

The presented approach has relations to several research domains. Beside the concept of model-driven development, especially the definition of domain-specific languages and approaches for developing adaptive applications in conjunction with concepts for context-awareness are relevant.

The modelling of context considered in adaptation processes and the adoption of a context middleware service are addressed in many research projects during the last years. Recent projects [11,5] covered the creation of comprehensive and generic context models with the goal of identifying and integrating characteristics of context. Especially, ontology-based modelling is addressed [3,7,4] with focus on knowledge sharing and reasoning. In [3,7], approaches for defining a common context vocabulary based on a hierarchy of ontologies are described. An upper ontology defines general terms while domain-specific ontologies define the details for certain application domains. Both approaches use a centralized architecture and work on local scenarios from the smart home or intelligent spaces domain. Furthermore, implications of modelling for context service design, integration of heterogeneous context sources and distribution support are not addressed. In our approach we adopt ontologies for modelling context information, but the major focus of our models is to create a common basis for context providers and context consumers. Especially, a dynamic discovery and binding of context providers should be supported. Moreover, the models are used for generating code for context access.

Besides modelling and integration issues, architectural aspects are considered. Current context-aware systems are mostly centralized. Thus mobile clients either rely on a server providing them with context or gather required context on their own. The Java Context Aware Framework (JCAF) [1] is a Java-based approach for a generic context middleware service. Service components communicate in a peer-to-peer manner with support of an event-based notification mechanism. Nexus [6] deals with the efficient management of heterogeneous context information in large-scale infrastructures. Especially scalability is addressed by a federated context middleware and special-purpose servers for optimized management of large amounts of context. Moreover, approaches like [9] and [10] concentrate on the abstraction process of context and the integration of sensing devices. Our context service as part of a runtime environment follows a peer-to-peer approach. It adopts concepts from architecture and abstraction but is based on the abstraction of context providers which can be hierarchically organized to derive more abstract context.

Our modelling approach is based on the concepts of Model-Driven Architecture (MDA) and UML. As abstract view to our applications we use a task model. Its basic classes are adopted from an approach described in [15]. User interaction modelling builds on concepts developed for device-independent [8,17] or multimodal user interfaces [2,12].

3 Design Methodology for Adaptive, Multimodal Applications

Development of adaptive software involves interrelated steps, which require specific expertise provided by different developers. According to the model-driven software development approach, these steps are carried on models of different abstraction layers. Hence, the development process are performed by developers acting in different roles that must be coordinated. Our approach is embedded into a design methodology (cf. 1), which defines a set of artefacts (mainly models), developer roles and a process model, to coordinate the development process.

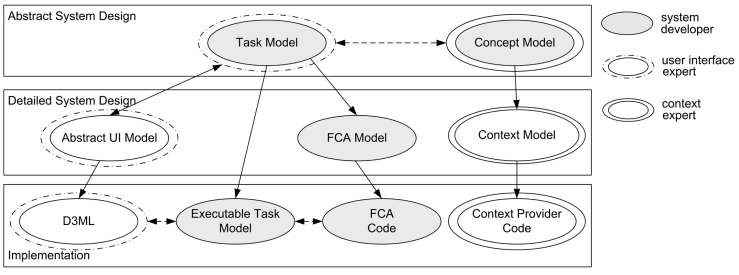

Fig. 1. Design methodology for adaptive, multimodal systems

The development process starts with a *requirements analysis*. Focusing on requirements concerning system functionality and user interaction as well as their adaptation this phase lays the foundation for later system specification.

In the *abstract system design* phase two models are created. The *task model* specifies control and data flow as well as temporal relationships between tasks. Whereas, the *concept model* comprises the application data and knowledge, including context relevant for adaptation of the system. The task model refers to the concept model for the definition of application data, input and output parameters as well as for involving context for adaptation.

Both models are transformed into the models of the next development phase – the *detailed system design*. Depending on their nature tasks are either transformed into the *Abstract UI model (AUI)* or the *Functional Core Adapter model (FCA)*. Context-related concepts from the concept model are transformed into the *context model* for specification of context provision based on context providers.

The detailed system design models are used for code generation in the *implementation phase*. All code is generated toward a well defined runtime environment, which is part of the presented solution. It contains a task execution engine, a user interface engine, and a context infrastructure. The *executable task model* is derived from the task model. It governs the control and data flow of the application and it refers to *D3ML* representations of the interaction tasks and *FCA code*, for executing tasks respectively. It also calls to the *context provider code* that is generated from the context model to access context for adapting user interactions and control flow. The corresponding context providers are managed by the context service at runtime.

We assume that the roles of a *system developer*, a *user interface expert*, and an *context expert* are necessary to perform all steps of our development methodology. While the task and concept models can be designed by a system developer, special expertise is needed to consider the requirements for adaptation and the relations to context in that design phase. Since refinement of the AUI model and D3ML code comprises multimodal interaction as well as user interface adaptation, this is the responsibility of a user interface expert. The creation and refinement of the context model and the context provider code is the task of the context expert, for the latter requires detailed knowledge of the underlying context service.

4 Models

Models are the formal basis to capture software design at a conceptual level. Just like UML is tailored to the specific requirements of object-oriented software design, dedicated models are needed to account for the peculiarities of adaptive software. We have devised a set of five models to enable modelling of all aspects that are necessary to create task-driven, adaptive multimodal applications. They can be used to model the interrelated aspects of domain modelling (concept model), context provision (context model) and context consumption in the form

of application logic (task model, functional core adapter model) and adaptive multimodal user interfaces (abstract user interface model) – sections 4.1 to 4.5.

Following the philosophy of model-driven software-engineering, the models support the range from a highly abstract level down to generated code as the most concrete level. Transitions between the different levels are implemented as model-to-model and model-to-code transformations – based on MOF QVT [14] or JET [16]. Technology independence is achieved across all models and levels of abstraction.

4.1 Concept Model

The common base of all models is represented by the concept model. It allows for modelling of application data and knowledge based on ontologies. Therefore the Ontology Definition Metamodel (ODM) [13] was chosen for integrating the Web Ontology Language (OWL) into object-oriented design. Thus application developers can rely on the expressiveness of XML Schema, RDFS, RDF, and OWL-DL when modelling data that is being processed in and presented by software.

Furthermore the model may contain contextual concepts and their data types, which might be relevant for adaptivity. This allows for two major advantages: (1) uniform modelling of application data and context knowledge, and (2) paving the way for logic-based reasoning over context.

4.2 Task Model

In our approach, we use a task model to express the dynamic aspects of adaptive applications from both the user and the system perspective. Following the approach presented in [15], our task model features three task types. *User Tasks* are performed solely by the user (e.g. perceiving and evaluating system output). *Interaction Tasks* represent interactions between the user and the application via a user interface. Their complexity can vary from entering a simple piece of information, e.g. a single numeric value, to complex tasks such as editing a plant maintenance report. *System Tasks* are performed entirely by application logic (e.g. validating the maintenance report and storing it in a database).

The temporal relationship between tasks is modelled by directed edges that represent the control flow. They are triggered when a system task completes or when the user completes an interaction task. Control flow parallelism is modelled by Fork, Merge and Join Nodes. Data transport to and from tasks is modelled by object flows. Like control flows, object flows are directed edges between tasks. They transport concepts defined in the concept model (see section 4.1) and are connected to tasks by input and output pins (cf. 2).

Context-awareness is supported by three novel concepts that we have introduced in task modelling. These concepts are *event consumers*, *context queries* and *guard conditions*.

Event consumers represent event subscriptions in the task model. By event consumers, subscriptions to the following two types of events can be modelled.

Context events are issued by context providers. They allow push access to context. For example, a context provider may trigger an event when a sensed temperature exceeds a critical value. *Observer events* allow monitoring state changes of concepts. Whenever a change occurs, an observer event is triggered. It conveys the new state to subscribed tasks. Due to the abstraction introduced by the concept model, the origin of the state change is not visible to the tasks, allowing homogeneous processing of observer events. For example, an observer event may be thrown when a system task updates a concept by writing the result of a calculation or when the value sensed by a context sensor changes.

Tasks can consume events in two ways. Event consumers can be bound to a system task by a control flow. In this case, the system task has the role of an event handler. When the subscribed event occurs, a new instance of the system task is created and its execution is started. Event consumers that are subscribed to observer events can be connected to input pins via an object flow. Upon an observer event, the value of the input pin is updated with the new state of the observed concept.

Whenever context must be fetched by an application, means to actively pull information from the concept model are needed. In our task model, these means are provided by concept queries. A concept query is bound to a concept in the concept model and connects to a task via an object flow and an input pin. The task can access the value of the input pin regardless of the data sources bound to the input pin.

Guard conditions can be bound to control flows and object flows that originate from event consumers. Guard conditions are formulated over concepts. In case of control flows, the control is passed to a task only if the guard condition of the control flow is fulfilled. Using identical modelling techniques, control flows can therefore be controlled by context and application data. For example, assume that the number of records stored in a database is modelled as a concept. A guard condition that requires the number of records in the database to be zero can prevent the invocation of a system task that deletes the database if it contains any records.

Guard conditions bound to object flows that originate from event consumers serve as filters for observer events. They can be formulated over the concept transported by the observer event. If the guard condition is not fulfilled, the observer event does not update the value of the input pin.

Fig. 2 shows the novel modelling concepts for context-aware task modelling in a task model consisting of two interaction tasks (InteractionTask1, InteractionTask2) and two system tasks (SystemTask1, SystemTask2). User tasks are omitted for reasons of brevity. InteractionTask1 and InteractionTask2 can be interacted order independent. SystemTask2 is the event handler for the context event consumer bound to it. An observer event consumer for the numeric concept SensedValue is connected to InteractionTask2. The guard condition filters out all events with negative or zero values of SensedValue, i.e. only positive values of SensedValue are written to the input pin of InteractionTask2. The Interaction-

Task2 can only be completed by the user if the value of the concept SensedValue is larger than ten.

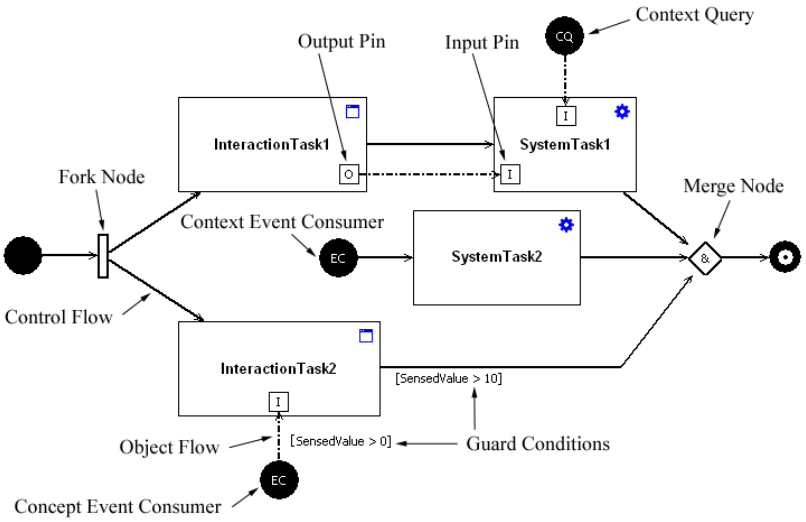

Fig. 2. Task model

4.3 Abstract User Interface Model

The abstract user interface model (AUI) allows modelling abstract multimodal user interfaces based on abstract interactors as proposed in [2] and [12]. These are representation- and technology-independent descriptions of user interface widgets. They are composed to complex abstract user interfaces, which can be transformed into multiple, technology-specific representations, including versions for different modalities. In most cases, technology independence alone is not sufficient to create highly usable user interfaces from abstract user interfaces. For this reason, AUI refinements that take the context in which a user interface presented into account are needed [8].

In our approach, AUI refinement is based on the concept of manual adaptation to so called *context profiles*. A context profile is a list of name-value pairs that characterize a complex condition (a *situation*) in terms of device properties, built-in sensors, and available input and output mechanisms. Fig. 3 shows two context profiles. The *PDA* profile describes a device whose keyboard, display, microphone, and speaker can be utilized for user interaction. The keyboard must be thumb keyboard with English layout. The display must have QVGA resolution, which is typical for most PDAs, and landscape format. Audio output can be via earphone or speaker. The *Hands-Free/PDA* profile describes a typical hands-free

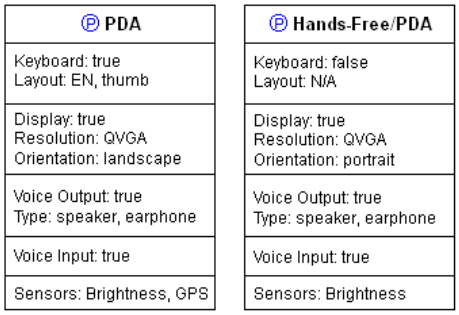

Fig. 3. Context profiles

situation in which the keyboard can not be used for input. The display must have QVGA resolution in portrait format.

Context profiles can be bound to refined versions of the *initial AUI model*. The initial AUI model is generated by a task-to-AUI transformation of interaction tasks. For each interaction task, a generic AUI interactor is generated. Fig. 4 shows the transformation of a login dialog task model and the initial AUI model to which it is transformed.

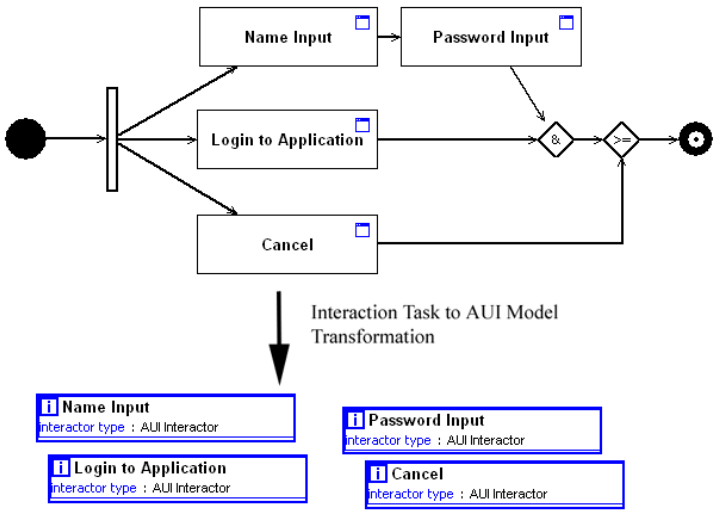

Fig. 4. Interaction task to initial AUI model transformation

AUI refinement is performed on copies of the initial AUI model. The developer binds each copy to at least one context profile and starts its refinement process. In this process, concrete interactor types (text input field, select con-

trol, button, ...) must be assigned to the generic interactors generated by the task-to-AUI transformation. Furthermore, layouts for an appropriate visual appearance of the user interface in the situation described by the context profile can be added (cf. 5). Interactors can also be removed from a refined model if they are inappropriate for the situation, e.g. if the device does not support its rendering. Modality-specific properties like voice input grammars, voice prompts, etc. can be set as properties of a concrete AUI interactor where applicable.

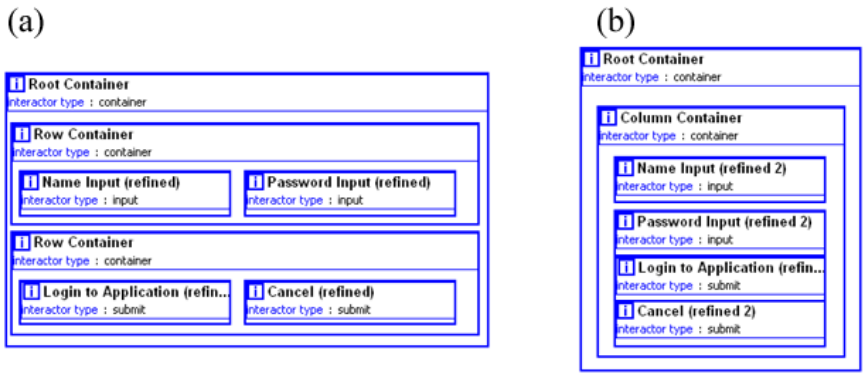

Fig. 5. Refined AUI models of the login dialog for (a) landscape and (b) portrait displays

At runtime, evaluation of context profiles is performed by the context service to select the appropriate version of the user interface. A context profile is transformed into a context query, which is evaluated against a concept that describes the properties of the execution environment. This concept is added automatically to every concept model.

4.4 Functional Core Adapter Model

Besides enhancing the two particular aspects task and user interface a more generic way exists for achieving context-awareness of the designed software. This means is provided by the model representing the functional core adapter (FCA) – the interface to the business logic of the software.

The FCA model provides two element types – *FCA methods* and *FCA calls*. Methods contain the mapping to methods of the business logic, including their arguments and results. Calls are instantiations of those methods and therefore refer to the methods.

4.5 Context Model

Building upon the concept model as common ground the context model refers to its conceptual facet only. Subject of this model is the provision of context.

A variety of context sources exists all differing in retrieval, storage, and presentation of their information. Hiding that heterogeneity the sources are described in a uniform way – as so-called context providers – to allow for matching with potential context consumers.

The context model describes the various types of context providers, which may be of two different kind – low- or high-level. Low-level providers that retrieve sensed, profiled or stored context directly from a hardware or software context source (e.g. sensor, monitor, database) can be described by characterizing their provided context. Thus the data types and concept from the concept model are referred to by the context model. High-level providers apply various derivation schemes for retrieving their provided context and thus must consume other context prior to the derivation of more abstract (or high-level) context. Therefore high-level providers need additional description of their consumed context.

5 Runtime Environment

The afore mentioned modelling concepts are mapped onto our runtime environment, which is targeted toward resource-constraint mobile and embedded devices – implemented in Java ME (CDC 1.1) based on the OSGi platform. Its main components are: task process engine, context service, and multimodal services component (cf. 6). For brevity reasons we focus on the context service only.

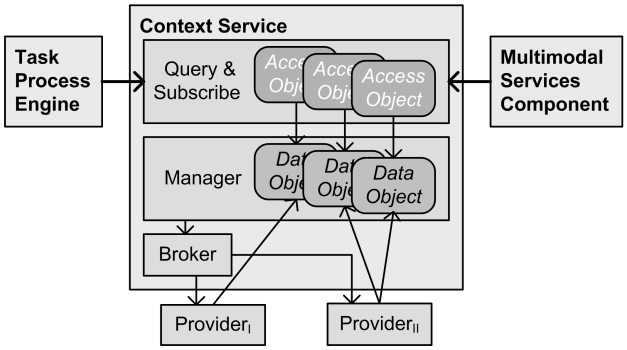

Fig. 6. Runtime architecture – main components

The context service provides on-demand access to context. Service instances running on different devices connect to each other and form a distributed context service. Thus, allowing for context access between sources and consumers – the adaptive software – even on different devices.

Context propagated from the sources to the consumers is represented based on a metamodel [13], which is adopted from the topic map and entity-relationship metamodels. It defines three model element types: *Entities* represent either real-world objects or abstract concepts (e.g. a process). *Attributes* contain the specific

features of the entities and can be of simple or structured type – arranged in so-called *attribute groups*. *Associations* are relationships between two entities.

The different facets of designtime context models are reflected in the runtime too. There are context, data, and usage models. Provided and consumed context is specified by so-called context patterns each of them being a triple (m, t, r) with: a meta type m (entity, attribute, or association), a domain-specific type t, and an optional restrictive expression r. The pattern $(ENTITY,' person','')$ would refer to all entities of type 'person' without any restriction.

Runtime Context Model: For uniform management each context source is wrapped in a so-called context provider – a lightweight component with uniform interface. Due to the nature of its underlying context source (e.g. wireless sensor, database, application) a provider may be subject to software de-/installation, plug-and-play, or wireless connectivity problems and thus dynamically available.

A provider is self-descriptive since a single service instance cannot be expected to a priori "know" all potential providers that may be encountered during its life cycle. Upon detection a provider is examined by the broker component using the provider API to retrieve its description. The main part of this description consists of the provided and consumed context, each represented by a set of context patterns.

Runtime Usage Model: Context consumers specify their needs by using context patterns too. They are passed as call parameters to the Query & Subscribe API resulting in either synchronous responses or asynchronous notifications – both containing references to the current context.

The context usages of the task, AUI, and FCA models are implicitly contained in the context patterns and the specific API calls performed by the respective code at runtime. In figure 6 this is shown by the arrow from the task process engine and the multimodal services component accessing the Query & Subscribe component.

Runtime Data Model: Since the context service is executed on resource-constraint devices it does without expensive model validation mechanisms. Instead the service performs a matching algorithm based on consumed and provided context patterns. This matching is applied for binding suitable providers when a consumer specifies a pattern in an API call.

Java classes exist corresponding to all data types defined in the concept model – either by mapping to native Java classes or by generating application-specific ones. The dynamic availability of providers applies to their provided context too. Instead of caching context we opted to decouple the information access in order to prevent consumption of stale context. Therefore, context is kept in data objects that are hidden from the consumers by access objects. The latter are maintained by the Query & Subscribe component whereas the former are updated by the providers, while being created by the manager component.

When the provider of a certain data object becomes unavailable subsequent consumer calls to its access object will cause the binding to an alternative provider. When this is unsuccessful the consumer receives an according notification.

6 Validation

We have successfully validated our approach by developing a context-aware plant maintenance application. This application consists of three tools that (a) support maintenance workers in organizing their work and (b) allow the remote monitoring of plant parameters and (c) their history. It exploits all features of context-aware software design provided by our models.

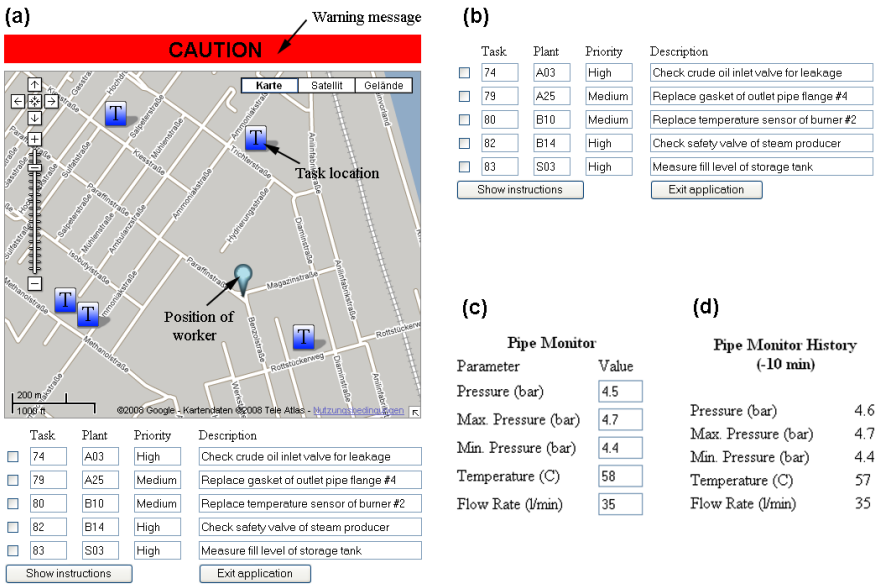

Fig. 7. Screen shots of the plant maintenance application: (a) task organization tool, (b) task organization tool without location context, (c) sensor monitoring tool, (d) sensor history tool

The task organization tool (cf. 7a) demonstrates the use of location information and UI adaptation. In our scenario, a list of tasks is assigned to a maintenance worker by his manager. A maintenance task is composed of information about the location (name and geo-coordinates of the plant) where the task is to be performed, its priority, a short description, and list of detailed instructions. This information is conveyed by a warning message area, an electronic map, and a task list. Symbols in the map indicate the task locations. Observer events generated by a location context provider, which wraps the GPS sensor of the PDA running the application, are utilized to update the marker that indicates the worker's position on the map (cf. 7a). The warning message area is updated from "OK" to "CAUTION" whenever the worker is in the proximity of plants that process inflammable or explosive substances. This update is initiated by observer events that are produced by a remote context provider that consumes the

output of the location context provider and determines the hazard classification of the area around that geo-coordinate.

UI adaptation is demonstrated in fig. 7b. When location information is not available, e.g. because there is no GPS signal or because the PDA is not equipped with a GPS sensor, a reduced UI of the task organization tool is loaded. It omits the map and the warning message area since they require location information.

Several components of a plant are monitored by sensors. A sample remote monitoring tool (Pipe Monitor, cf. 7c) aggregating pressure, temperature and flow rate information from a hot water pipe has been realized for validation purposes. The values shown by the Pipe Monitor are delivered by observer events from corresponding sensors and update the system output accordingly.

To demonstrate pull access to context via context queries, we have implemented sensor history providers for the pipe monitor's sensors. Sensor history providers can be queried for the sensor readings of the last 15 minutes. The query results returned by the history providers can be accessed by the Pipe Monitor History tool (cf. 7d).

7 Conclusion and Outlook

We have presented a software development methodology for adaptive, multimodal applications. It comprises a process model, a set of models and transformations and a runtime environment. Especially, we presented a concept for modelling system and user interface adaptation based on context. Thus, with our approach, the relation between context and adaptation processes is explicitly definable. Using a model-driven approach, these definitions can be transformed semi-automatically into code, which significantly eases the development of adaptive software. Our validation example has demonstrated the feasibility of the approach presenting a context-aware application able to adapt its functionality and user interaction to the execution environment. Therefore, we have involved several context sources, which are managed by a peer-to-peer context service.

In the future we will extend the existing methodology and development environment. Furthermore, we plan to implement further applications to validate how our model-driven approach can reduce development effort.

References

1. J. E. Bardram. The Java Context Awareness Framework (JCAF) – a service infrastructure and programming framework for context-aware applications. In H. Gellersen, R. Want, and A. Schmidt, editors, *Proceedings of the 3rd International Conference on Pervasive Computing*, Lecture Notes in Computer Science, Munich, Germany, May 2005. Springer.
2. R. Burmeister, C. Pohl, S. Bublitz, and P. Hugues. Snow - a multimodal approach for mobile maintenance applications. *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 131–136, 2006.

3. H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments, 2003.
4. E. Christopoulou, C. Goumopoulos, and A. Kameas. An ontology-based context management and reasoning process for ubicomp applications. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 265–270, New York, NY, USA, 2005. ACM Press.
5. F. Fuchs, I. Hochstatter, M. Krause, and M. Berger. A meta-model approach to context information. In *Proceedings of Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 8–14. Cambridge University Press, 2005.
6. M. Grossmann, M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, and T. Schwarz. Efficiently managing context information for large-scale scenarios. In *Proceedings of the 3rd IEEE Conference on Pervasive Computing and Communications*, Kauai Island, Hawaii, March 2005.
7. T. Gu, H. K. Pung, and D. Q. Zhang. Toward an osgi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, 3(4):66–74, 2004.
8. G. Hübsch, T. Springer, A. Spriestersbach, and T. Ziegert. *An Integrated Platform for Mobile, Context-Aware, and Adaptive Enterprise Applications*, pages 1105–1124. Physica-Verlag, 2005.
9. K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. *Lecture Notes in Computer Science*, 3760:846–863, 2006.
10. P. Korpipää, J. Mäntyjärvi, J. Kela, H. Kernen, and E.-J. Malm. Managing context information in mobile devices. *IEEE Pervasive Computing*, 2(3):42–51, 2003.
11. K. H. S. Livingstone and J. Indulska. Towards a hybrid approach to context modelling, reasoning and interoperation. In *Ubi-Comp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 54–61, 2004.
12. W. Mueller, R. Schaefer, and S. Bleul. Interactive multimodal user interfaces for mobile devices. page 90286.1, 2004.
13. Object Management Group, Inc. Ontology definition metamodel. OMG Adopted Specification ptc/2007-09-09, OMG, November 2007.
14. Object Management Group, Inc. MOF QVT. Final Adopted Specification ptc/05-11-01, OMG, November 2005.
15. F. Paterno, C. Mancini, and S. Meniconi. ConcurTaskTrees: A diagrammatic notation for specifying task models. In *INTERACT 97: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, pages 362–369, London, UK, 1997. Chapman and Hall, Ltd.
16. R. Popma. JET tutorial part 1 (introduction to jet). Technical report, Azzurri Ltd., 2003.
17. T. Ziegert, M. Lauff, and L. Heuser. Device independent web applications – the author once - display everywhere approach. *Lecture Notes in Computer Science*, 3140:244–255, 2004.