Context-Oriented Domain Analysis

Brecht Desmet, Jorge Vallejos, Pascal Costanza,
Wolfgang De Meuter, and Theo D'Hondt

Programming Technology Lab, Vrije Universiteit Brussel,
Pleinlaan 2, B-1050 Brussel, Belgium

Abstract. Context-aware systems are software systems which adapt their behaviour according to the context of use. The requirements engineering phase is recognized as a primordial step to develop robust implementations of context-aware systems since it establishes a comprehensive understanding of the problem space. This paper proposes the Context-Oriented Domain Analysis (CODA) model which is a specialized approach for analyzing, structuring, and formalizing the software requirements of context-aware systems.

1 Introduction

The Ambient Intelligence vision (IST Advisory Group, 2003) describes scenarios in which people are pervasively surrounded by interconnected embedded and mobile devices. As the context of such devices continuously changes over time, context-aware systems adapt their behaviour accordingly in order to suit the user's expectations more closely. The robust implementation of context-aware systems is founded on a comprehensive understanding of the problem domain at the early stages of software development. Requirements engineering is specifically concerned with producing specifications for software systems that satisfy the stakeholders needs and can be implemented, deployed, and maintained.

Traditional methods in requirements engineering, like use cases [1], aim at capturing functional requirements by looking at the interactions between actors and systems. Our practical experiments however point out that these methods do not closely match the niche domain of context-aware systems. This is mainly caused by the fact that context-aware systems add a new dimension to the actor-system interaction by incorporating additional information from the (physical or software) environment: so-called *context information*. We define the latter as any piece of information which is computationally accessible. [2]

Such a refined parameterization of the actor-system interaction has a strong impact on the specification of functional requirements. Whereas in monolithic applications, an actor's action typically corresponds to a single behaviour, context-aware systems have multiple behavioural variations associated to a single action. The choice of the appropriate variation is determined by the context in which the system is used.

We claim that as soon as context-aware requirements become the rule rather than the exception, more adequate modelling techniques are required to capture

In: B. Kokinov, D.C. Richardson, T.R. Roth-Berghofer, L. Vieu (Eds.):
CONTEXT 2007, LNAI 4635, pp. 178-191, 2007.

http://dx.doi.org/10.1007/978-3-540-74255-5_14

the contextual influence on software systems. The contribution of this paper consists of a new modelling approach, called Context-Oriented Domain Analysis (CODA), which is a systematic approach for gathering requirements of context-aware systems. CODA is intended to be relatively simple and concise to lower the accessibility barrier for various kinds of stakeholders while being expressive enough to evolve towards the solution space. In contrast to general-purpose methods for requirements analysis, like use cases, goal models [3], or problem frames [4], CODA is solely specialized for context-aware (functional and non-functional) requirements.

This paper is organised as follows. Section 2 presents a context-aware scenario which is used throughout this paper. Next, in Section 3, we explain the CODA approach thoroughly by means of this context-aware scenario. We validate our approach in Section 4 by showing how the various concepts of CODA can be mapped to decision tables. Finally, Section 6 identifies some future work and gives the conclusion.

2 Motivating example: context-aware cell phone

We introduce an intelligent cell phone as an illustration of a context-aware system. In the following subsections, we briefly discuss the *requirements* of this context-aware cell phone in an *informal manner*. We take up again these requirements in Section 3 to illustrate our proposed CODA approach.

2.1 Basic behaviour

We first present the default context-unaware behaviour of the cell phone which we call the *basic behaviour*. This behaviour consists of the following functionalities:

- Incoming communication (R1)
  - play ring sound whenever somebody calls or sends a message (R1.1);
  - provide the means to answer phone calls and read messages (R1.2);
- Outgoing communication (R2)
  - provide means to make phone calls and send messages (R2.1);
  - use default mobile connection for outgoing communication (R2.2);
- Shared by incoming and outgoing communication (R3)
  - maintain a contact list and journal (R3.1).

2.2 Behavioural variations

We now increase the user experience of this cell phone by making it context-aware. In the following requirements description, we introduce some behaviour which deviates from the basic behaviour, depending on the context in which the cell phone is used. First, we present a group of behavioural variations which affect the *Incoming Communication* (R1).

- If the battery level is low, ignore all phone calls except for contacts classified as VIP (R1.3).
- If the time is between 11pm and 8am, activate the answering machine for incoming phone calls and the auto-reply service for messages. Add voice and text messages to the journal. (R1.4) The outcome of this behaviour is one of the following cases:
  - Everything turned out ok (R1.4.1).
  - A predefined list of callers can circumvent the answering machine by pressing the # button e.g. for emergency reasons (R1.4.2).
  - If the answering machine is unavailable because there is no memory left for voice messages, the cell phone gives an auditive signal (R1.4.3).
- If the user is in a meeting, redirect all calls and messages to the secretary (R1.5).

Next, there is a series of behavioural variations which affect the *Outgoing Communication*:

- The user can switch on a service which counts the amount of outgoing communication. This information is interesting e.g. for estimating costs. The concrete behaviour depends on the type of outgoing communication. (R2.3)
  - In case of phone call, measure the duration of the calls (R2.3.1).
  - In case of messages, count the number of sent data packages (R2.3.2).
- If there is a WiFi connection available, it is tried to make phone calls or send messages via VoIP since this is cheaper for the user (R2.4).
- If there is a GPRS connection available, it is tried to send messages using TCP/IP also since this is cheaper (R2.5).

In general, switches between behaviour are only possible between incoming or outgoing phone calls or messages (R4).

3 Principles of CODA

Context-Oriented Domain Analysis (CODA) is an approach for modelling context-aware software requirements in a structured, well-defined, and unambiguous way. The CODA model enforces software engineers to think of context-aware systems as pieces of basic context-unaware behaviour which can be refined. The driving force of the refinement is the context in which the system is used. We therefore prefer the term *context-dependent adaptation* which is defined as follows: *A unit of behaviour which adapts a subpart of a software system only if an associated context condition is satisfied.* The principle of distinguishing basic behaviour and context-dependent adaptations lays at the heart of our CODA approach.

In this paper, we apply our CODA approach to the requirements description of a context-aware cell phone (cfr. Section 2), yielding the CODA diagram of Figure 1. In the following, we discuss the vocabularium of CODA by means of this concrete example.

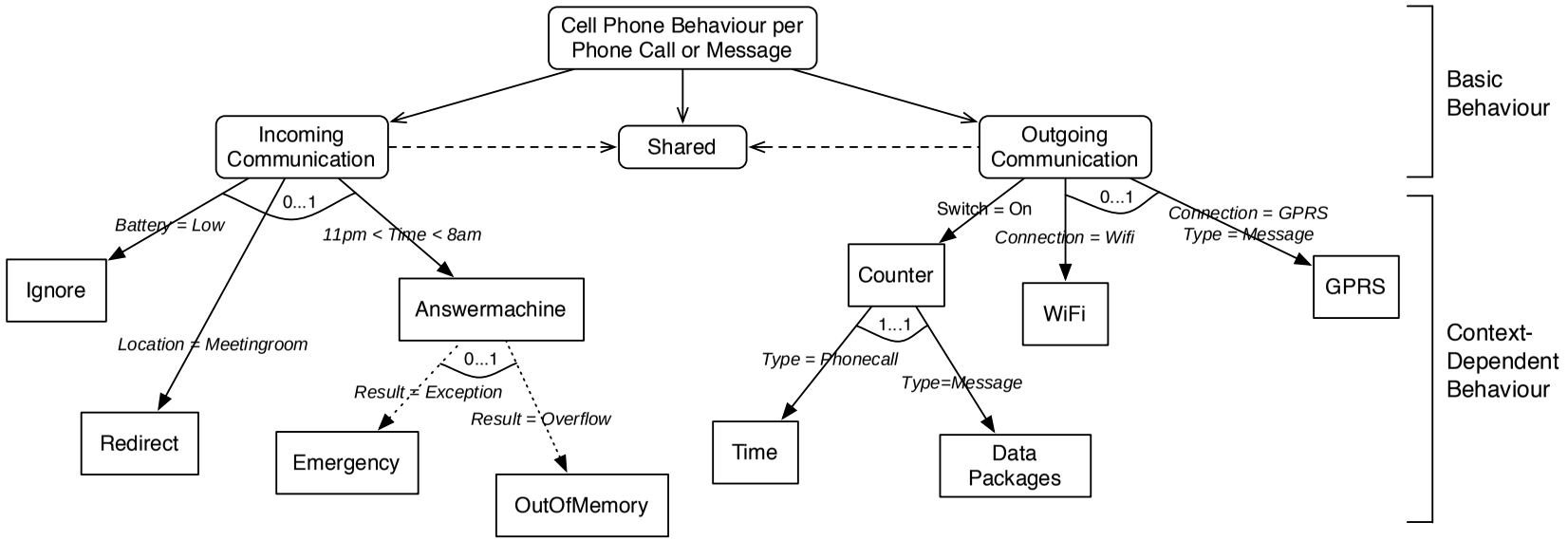

Fig. 1. CODA diagram of the context-aware cell phone.

3.1 Vocabularium

The root node of the CODA diagram refers to all possible combinations of context-aware behaviour on a per phone call or message basis. The topmost levels always contain the basic context-unaware behaviour, represented by means of rounded boxes. In the cell phone example, we divided the basic behaviour into three parts: *Incoming Communication* (R1), *Outgoing Communication* (R2), and *Shared* (R3). All these subparts are connected to the root via the "consists of" relationship ($\rightarrow$). *Incoming Communication* and *Outgoing Communication* are connected to *Shared* via the "uses" relationship ($\dashrightarrow$). Since these are the leaf nodes of the hierarchical decomposition of the basic behaviour, we also call them *variation points* which are subject to further refinement. The level of granularity to which the basic behaviour should be hierarchically decomposed is an important design choice for the modeller. The rule of thumb is to decompose until the leaf nodes are small and meaningful enough to serve as variation points.

Context-dependent adaptations are represented by means of rectangular boxes which are attached to relevant variation points (see Figure 2). For example, the variation point *Incoming Communication* of Figure 1 has three refinements: *Ignore* (R1.3), *Redirect* (R1.5), and *Answermachine* (R1.4). Each such context-dependent adaptation consists of two parts: a *context condition* which specifies the applicability of the adaptation (displayed on parent link) and a *label* which summarizes the adaptive behaviour (displayed within rectangular box). It is not allowed to add basic behaviour nodes below context-dependent adaptations, since this would break the principle of putting the basic behaviour at the topmost levels of the CODA diagram.

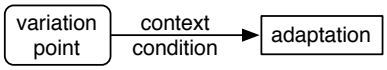

Fig. 2. Variation point refined by context-dependent adaptation.

Relationships The tree structure can be recursively expanded with additional context-dependent adaptations. To this end, CODA defines three kinds of relationships: inclusions, conditional dependencies, and choice points. The former two operate only among context-dependent adapations. The latter can be used among variation points and context-dependent adaptations.

- **Inclusion** The inclusion relationship (see Figure 3) means that only if adaptation A is applicable, the applicability of adaptation B should be verified. Possibly, adaptations A and B are simultaneously active. For example, if the counter switch is on (i.e. $Switch = On$), either *Time* (see R2.3.1) or *Data Packages* adaptation (see R2.3.2) should be included.

– **Conditional dependency** The conditional dependency relationship (see Figure 4) has a temporal character: If the return value of adaptation A equals r, then B should be executed subsequently. For example, if the special button # is pressed (i.e. $Result = Exception$) while using the answering machine, callers can circumvent the answering machine (see R1.4.2). Or, if the memory of the answering machine is full (i.e. $Result = Overflow$), some signal starts ringing (R1.4.3).

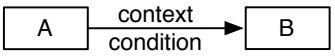

Fig. 3. Inclusion.

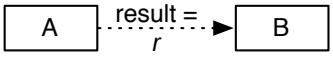

Fig. 4. Conditional dependency.

– **Choice point** Variation points and context-dependent adaptations can have multiple context-dependent adaptations associated to them. For example, *Incoming Communication* is refined by *Ignore*, *Redirect*, and *Answermachine*. Although the three adaptations can be simultaneously applicable (i.e. $Battery = Low$ while $Location = Meetingroom$ while $11pm < Time < 8am$), the adaptations are semantically conflicting (i.e. one cannot ignore and redirect phone calls simultaneously). Since it is the responsibility of a context-aware system to *choose* a non-conflicting set of adaptations out of a set of available candidates, we use the term *choice point* to mark such places in our CODA diagram. They are graphically denoted with ⏃.
Choice points have a *multiplicity* associated to them. This is a pair consisting of the minimal and maximal number of adaptations to be activated. For example, the variation point *Incoming Communication* has multiplicity "0...1" which means that at most one context-dependent adaptation can be activated, i.e. either *Ignore*, *Redirect*, or *Answermachine*.

3.2 Resolution strategies

In case of semantic interactions at choice points, a context-aware system should be able to make autonomous decisions based on some user-defined policy. For example, in Figure 1, if $Battery = Low$ and $Location = Meetingroom$, both *Ignore* and *Redirect* adaptations are applicable. However, the multiplicity of the choice point indicates that at most one adaptation can be activated. For these situations, we incorporate the ability to associate *resolution strategies* [5] with choice points. These strategies unambiguously describe which context-dependent adaptations should be activated or deactivated in case of semantic interactions.

CODA incorporates by default four resolution strategies: priority, frequency, timestamps, and case-by-case which are discussed in the remainder of this section. From our experience, these strategies seem to appear frequently for a wide range of scenarios. However, they are not universal. We therefore allow modellers

to combine or refine existing strategies and define new strategies whenever necessary. Graphically, resolution strategies are represented by means of UML-style stereotypes [6] which are attached to choice points.

Case-by-case The most straightforward option is to enumerate all possible interactions and their resolutions using relationships like exclusion, inclusion, etc. Case-by-case is considered as the default strategy and does not require the mentioning of a stereotype. The details of this resolution strategy are discussed thoroughly in Section 3.3.

Priority A commonly used strategy is to associate priorities with the alternatives. Priorities are good because they are the easiest way to understand by most stakeholders. For example, Figure 5 associates a priority with each context-dependent adaptation. These priorities are graphically represented by means of circles. If multiple adaptations are applicable, the one with the highest priority will be elected. Unfortunately, priorities are not an all-round solution because of limited expressiveness.

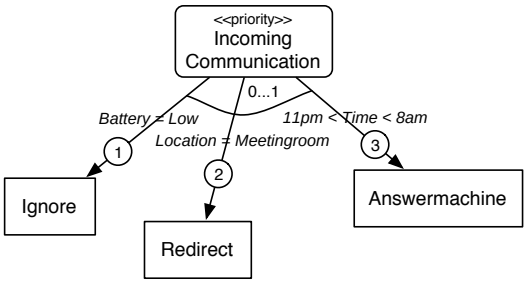

Fig. 5. Priority resolution strategy.

Frequency The frequency strategy is an adaptive method which selects context-dependent adaptations based on their frequency of use in the past.

Timestamps One can associate time stamps to context values to keep track of the order in which the context evolves. A possible timestamp strategy is to give preference to the most recent context information.

3.3 Vocabularium of case-by-case resolution strategy

If the user preference does not match an overall resolution strategy (like frequency, priorities, or timestamps), one can use the case-by-case strategy which

is more like a *general-purpose approach*. The idea of this strategy is to add *cross-reference relationships* among interacting context-dependent adaptations to the CODA diagram. For example, Figure 6 is an extension of the CODA diagram of Figure 1 which exhibits an example of the case-by-case resolution strategy. The new relationships are put in bold for clarification. In the remainder of this section, we discuss the semantics of these relationships.

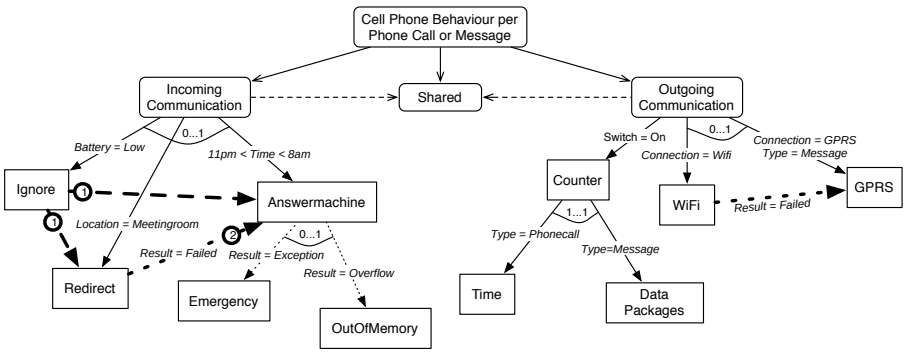

Fig. 6. CODA diagram of context-aware cell phone extended with an example of case-by-case resolution strategy.

Independence All adaptations that do not have a specific relationship specified are defined as being *independent*. This means that they do not semantically interfere with other adaptations. For example, as derived from Figure 6, it is possible that $Switch = On$ while $Connection = WiFi$, so both *Counter* and *WiFi* adaptations are active, operating independently.

Exclusion and Inclusion An adaptation can *exclude* or *include* another one. This is respectively presented in Figure 7 and 8 with a dashed (in case of exclusion) and full (in case of inclusion) arrow between interacting context-dependent adaptations. Their semantics are as follows:

- **Exclusion** If both adaptations W and Z are applicable, W is activated and Z is deactivated.
- **Inclusion** If adaptation W is applicable (either because the context condition a is true or because W is included by another adaptation), adaptation Z should also be active at the same time regardless of the thruth value of context condition b. The distinguishing feature between the inclusion defined in Section 3.1 and the inclusion defined here is that the latter operates cross-referencing.

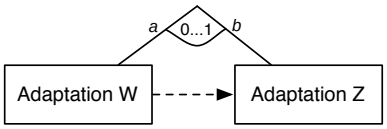

Fig. 7. Exclusion.

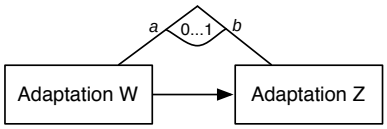

Fig. 8. Inclusion.

The CODA diagram of Figure 6 contains an illustration of an exclusion: Ignoring a phone call or message excludes both the redirection and answering machine adaptation. This is represented by means of a dashed arrow from *Ignore* to *Redirect* and *Answermachine*.

Conditional dependency Section 3.1 introduces the *conditional dependency* as part of the basic vocabularium of CODA. We now show that this relationship is also useful as part of the case-by-case resolution strategy. Consider for example the situation in which the user wants to send a message (i.e. $Type = Message$) while both WiFi and GPRS connection are available (i.e. $Connection = WiFi$ and $Connection = GPRS$). A possible policy in this case is to first try to send the message via WiFi because this is the cheapest way. If this fails, the GPRS connection can be tried instead. If this fails as well, the basic behaviour *Outgoing Communication* is used to send the message via the default mobile connection. This user policy is concretized in Figure 6 by means of a cross-referencing dotted arrow from *WiFi* to *GPRS*.

The same policy is applied to the *Redirect* and *Answermachine* adaptations: If the redirection of the phone call fails (e.g. secretary is not available), the caller gets in touch with the answering machine. Priorities are added to the various relationships that are used among the *Ignore*, *Redirect*, and *Answermachine* adaptations to avoid ambiguities. For example, if $Battery = Low$, $Location = Meetingroom$, and $11pm < Time < 8am$ simultaneously, only the exclusions between *Ignore-Redirect* and *Ignore-Answermachine* are applicable.

4 Validation

4.1 Design rationale

The vocabularium of CODA is intentionally kept concise since we want it to be *accessible for various kinds of stakeholders* including end users, domain experts, application developers, etc. Special attention has been paid to the expressiveness of the CODA model: It tries to be *as human-friendly as possible* by avoiding "enumeration like" descriptions and working with high-level abstractions instead. In this way, CODA offers a high-level view on the runtime contextual variability of a software system without burdening the stakeholders with low-level technical details.

CODA can be used by requirement analysts as an *instrument for communication* with clients to grasp the problem domain. It is furthermore an important *document of reference* for designers and implementors to better understand the technical challenges. Our CODA approach has already proven its usefulness for modelling a wide variety of context-aware scenarios going from intelligent vending machines, domotic systems, and shopping guidance systems to advanced user interfaces.

4.2 Mapping to decision tables

Although CODA might seem as being far removed from the solution space, since it has a well-defined syntax and semantics, it can be easily mapped to decision tables [7] which brings it very close to the computational level. For example, Tables 1 and 2 reflect the CODA diagram of Figure 6. Each decision table corresponds to a particular variation point of the CODA diagram. In our particular example, we distinguish *Incoming Communication* and *Outgoing Communication*.

The columns of the decision tables contain an enumeration of all possible context situations (at the top) and the associated actions (at the bottom). The actions represent the adaptations of CODA. Strictly speaking, we should always include an action called "basic behaviour," but we omit this for clarity reasons. Furthermore, Table 2 is incomplete because the case in which *Switch = Off* is not included - also for clarity reasons.

The mapping of CODA to decision tables is a *lossy transformation*. This is because the high-level concepts (e.g. multiplicities, resolution strategies, etc.) of CODA are translated to plain enumerations of yes/no-questions. Although such a mapping is important to evolve towards the solution space, the decision tables themselves are not useful for humans to understand the insights of the problem domain.

In the following, we validate our CODA approach by describing in high-level terms how it can be mapped to decision tables. The transformation algorithm has been implemented in Java using an XML representation of CODA. To this end, we developed an XML Schema Definition which provides *concrete syntax for CODA*. Furthermore, the mapping of CODA to decision tables creates a *formal basis for the semantics of CODA*.

1. Create a decision table per variation point in CODA which is the root of a disjunct subtree.
2. Place all context conditions of the CODA diagram in the conditions column of the decision table.
3. Generate all possible yes/no combinations of the context conditions. Redundancy should be avoided.[1]
4. The following rules apply for the generation of actions in the decision table:
   - All context-dependent adaptations which have a *full arrow* as parent link (i.e. either a refinement or inclusion) correspond to a *single action*.

[1] For example, if $\neg(11pm < Time < 8am)$, it is not necessary to include all yes/no combinations of the return values OK, Exception, and Overflow.

- Context-dependent adaptations with a *dotted arrow* as parent link (i.e. conditional dependency) are translated to an action called "cond-dep from-adaptation → to-adaptation".
- It is possible that a context-dependent adaptation has *both a full and dotted arrow* as parent link which implies the definition of two different actions.

5. For each possible context description in column s_i of the decision table, perform the following steps.
   (a) Let n be the variation point of a particular decision table. Furthermore, let solution set $S = \emptyset$.
   (b) At choice point n, determine the set A of applicable (b, p, a) triples based on the context description s_i. The variable b is the root node (i.e. variation point or context-dependent adaptation), p is the parent link type (i.e. full or dotted arrow) and a is the applicable context-dependent adaptation.
   (c) Verify if A fulfills the multiplicity constraint. If not, the resolution strategy associated with choice point n should be applied to A.
   (d) Add set A to the solution set S. Recursively call step (5b) for all $n = a : (b, p, a) \in A$.
   (e) Per (b, p, a) triple of the solution set S, mark the corresponding actions. If $p = dotted$, the appropriate action "cond-dep $b \rightarrow a$" should be marked.

Table 1. Decision table for incoming communication.

Conditions	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}
Battery is low	N	Y	N	N	N	N	Y	Y	N	N	N	N	Y
Location is meetingroom	N	N	Y	N	N	N	Y	N	Y	Y	Y	Y	Y
Time is at night	N	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
Answermachine returns OK				Y	N	N				Y	N	N	
Answermachine returns Exception				N	Y	N				N	Y	N	
Answermachine returns Overflow				N	N	Y				N	N	Y	
Redirection succeeded (Y) or failed (N)									Y	N	N	N	
Actions													
Ignore		×					×	×					×
Redirect			×						×	×	×	×	
Answermachine				×	×	×				×	×	×	
cond-dep Answermachine → Emergency					×						×		
cond-dep Answermachine → OutOfMemory						×						×	
cond-dep Redirect → Answermachine										×	×	×	

5 Related work

5.1 CODA versus FODA

The CODA approach is heavily inspired by the already existing Feature-Oriented Domain Analysis used in product-line development. The FODA modelling ap-

Table 2. Decision table for outgoing communication.

Conditions	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
Switch is on	Y	Y	Y	Y	Y	Y	Y	Y	Y
Connection is WiFi	N	N	Y	Y	N	N	Y	Y	Y
Connection is GPRS	N	N	N	N	Y	Y	Y	Y	Y
Type phone call (Y) or message (N)	Y	N	Y	N	Y	N	Y	N	N
WiFi call/message fails								N	Y
Actions									
Counter	×	×	×	×	×	×	×	×	×
Time	×		×		×		×		
Data Packages		×		×		×		×	×
WiFi			×	×			×	×	×
GPRS						×			×
cond-dep WiFi → GPRS									×

proach is originally proposed by Kang et al. [8] to model a family of related software products in a very concise manner. Although the CODA diagram looks syntactically very close to FODA, the overall aim of CODA and the semantics of its building blocks differ significantly. Table 3 contains a detailed comparison between CODA and FODA based on the following characteristics:

Goal What is the intended purpose of the model?
Concept What does the model describe?
Commonalities and Variabilities Software variability in general can be characterized by means of common and variable elements. What is the concrete meaning of these elements?
Actor Who uses the model?
Mode Is the model a static or dynamic analysis?

5.2 State charts

Proposals like [9] and [10] already identified the importance of a sound formal basis to develop robust context-aware systems. Central to their approach is the use of state charts (or activity diagrams) to model the application domain. The conceptual difference between CODA and state charts is the way of thinking that is induced. Whereas in CODA one thinks in terms of (hierarchical decomposition of) basic behaviour and refinements of this behaviour at certain variation points, statecharts are about states (e.g. denote a particular way of behaving) and transitions between them.

Strictly speaking, all concepts of CODA can be imitated with state charts simply because state charts are turing complete. However, the concern of CODA is not *what* can be modelled, but *how*. In that regard, we observe that the vocabularium of state charts do not include concepts like multiplicities, resolution strategies at choice points, inclusions, and exclusions. In CODA, these concepts are crucial for establishing a high-level view on the problem space.

Table 3. Comparison of CODA and FODA.

Characteristic	**FODA**	**CODA**
Full name	feature-oriented domain analysis	context-oriented domain analysis
Goal	product-line development	context-aware systems
Concept	all possible product variations of a family of related products	context-aware behavioural variations within (a subpart of) a single system
Commonalities	behaviour shared by all family members of product	basic context-unaware behaviour which is always applicable, but might be refined
Variabilities	*feature*: any prominent and distinctive aspect or characteristic that is visible to various stakeholders	*context-dependent adaptation*: behaviour refinement of some basic behaviour that is only applicable if a certain context condition is satisfied
Actor	software designer decides on a feature composition	the software system itself makes autonomous decisions about the composition of basic behaviour and context-dependent adaptations based on context conditions, multiplicities and resolution strategies
Mode	*static analysis*: describes static properties of features which enable the generation of all product variations at compile time	*dynamic analysis*: contains context conditions, conditional dependencies and resolution strategies which enable automatic run-time computation of behavioural variations based on context information

6 Conclusion and Future Work

Context-oriented domain analysis is an approach for identifying and modelling *context-aware software requirements*, which is a niche domain within the field of requirements analysis. It enforces modellers to think in terms of *basic context-unaware behaviour* which can be further refined by means of *context-dependent adaptations* at certain *variation points*. A context-dependent adaptation is a unit of behaviour that adapts a subpart of a software system only if a certain context condition is satisfied. By context, we mean every piece of information which is computationally accessible.

This work identifies a number of relationships that may exist among context-dependent adaptations. A context-dependent adaptation can *include* another adaptation which means that the applicability of the second adaptation is verified only if the first one is activated. Next, a context-dependent adaptation can *conditionally depend on* another adaptation. In this case, the applicability of the second adaptation depends on the result of the first adaptation, yielding a sequential execution. We finally introduce the notion of a *choice point* which is a variation point or context-dependent adaptation which has multiple adaptations

associated to it. Optionally, one can associate a *resolution strategy* to deal with semantically interacting adaptations.

The CODA approach can be represented in three ways: *Graphically*, where the system's basic behaviour and its context-dependent adaptations are presented in a tree structure; *Textually*, using XML technology to declaratively write down CODA diagrams; and *Structurally*, where the semantics of CODA elements are mapped to decision tables.

The aim of the CODA approach is to have a concise modelling language for context-aware systems which is accessible to various kinds of stakeholders. Since CODA has a well-defined syntax and semantics, it possesses a sound basis for evolving towards the solution space. However, a deeper understanding of the mapping from CODA to the computational level is still under investigation. In this regard, we believe that the mapping of CODA to decision tables is an important step in the right direction.

References

1. Kurt Bittner. *Use Case Modeling*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
2. Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented Programming. *Submitted to Journal of Object Technology*, 2007.
3. Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. In *6IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 3–50, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.
4. Michael Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
5. Luc Steels. *Kennissystemen*. Addison-Wesley, Reading, MA, USA, 1992.
6. Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
7. R.M. Feagus. Decision tables - an application analyst/programmer's view. *Data Processing 12*, pages 85–109, 1967.
8. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
9. Annika Hinze, Petra Malik, and Robi Malik. Interaction design for a mobile context-aware system using discrete event modelling. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pages 257–266, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.
10. Mark Mahoney and Tzilla Elrad. Distributing statecharts to handle pervasive crosscutting concerns. In *Building Software for Pervasive Computing Workshop at OOPSLA '05*, San Diego, CA, October 2005.