A transformation-based approach to context-aware modelling

SW & System Modeling with Graph Transformations manuscript No.
(will be inserted by the editor)

A Transformation-Based Approach to Context-Aware Modeling

Degrandsart, Sylvain[1,2], Demeyer, Serge[1], Van den Bergh, Jan[3], Mens, Tom[2]

[1] Department of Mathematics and Computer Science, Universiteit Antwerpen,
Universiteitsplein 1, B-2610 Antwerpen, Belgium
e-mail: {sylvain.degrandsart, serge demeyer}@ua.ac.be
[2] Institut d'Informatique, Université de Mons
Place du Parc 20, B-7000 Mons, Belgium
e-mail: tom.mens@umons.ac.be
[3] Hasselt University – tUL – IBBT,
Expertise Centre for Digital Media, Wetenschapspark 2, B-3590 Diepenbeek, Belgium
e-mail: jan.vandenbergh@uhasselt.be

DRAFT — The final accepted article is available at http://www.springerlink.com/content/l1718p7035852610/

Abstract Context-aware computing is a paradigm for governing the numerous mobile devices surrounding us. In this computing paradigm, software applications continuously and dynamically adapt to different "contexts" implying different software configurations of such devices. Unfortunately, modeling a context-aware application for all possible contexts is only feasible in the simplest of cases. Hence, tool support verifying certain properties is required. In this article, we introduce the Context-Aware Application model (CAA), in which context adaptations are specified explicitly as model transformations. By mapping this model to graphs and graph transformations, we can exploit graph transformation techniques such as critical pair analysis to find contexts for which the resulting application model is ambiguous. We validate our approach by means of an example of a mobile city guide, demonstrating that we can identify subtle context interactions that might go unnoticed otherwise.

Key words context-aware model – model transformation – critical pair analysis – context adaptation – context coverage

1 Introduction

Context-aware computing refers to the idea that mobile devices (such as smart phones and tablet PCs) can sense what is happening around them and respond accordingly. One example application is a mobile city guide: city visitors are carrying a portable device that alerts them when they approach a point of interest (e.g. an architectural curiosity, a special restaurant, ...). While alerting the visitors, the city guide adapts itself to the visitor's preferences (e.g., language of use — English or Arabic), profiles (e.g., child or adult) and abilities (e.g., pedestrian, bike, wheel-chair, ...).

Modeling such context-aware applications is a real challenge, certainly given the rapidly increasing number of sensors in mobile devices (e.g., light, compass, GPS, accelerometer, gyroscope, ...). Separation of concerns remains a guiding design principle for such applications, hence good designers will model the application such that the variables composing the contexts are as independent as possible. For example, in the mobile city guide, the choice of landscape or portrait mode for the display screen is independent from the type of network connection used to retrieve data to be displayed. Nevertheless, the very nature of context-aware applications implies that some variables will affect one another. For example, the mobile city guide should adapt its data download behaviour depending on the available network connection: using a wifi-connection with unlimited download capacity should result in videos about the nearby points of interest, while with a paying mobile data connection the device should restrict itself to showing low-resolution pictures only.

In that sense, a context-aware application can be represented as a multi-dimensional design space where the axes represent the variations that may occur for a given context variable, and where each point in the design space represents the expected behaviour of an application for a given configuration of context variables. A model of a context-aware application then specifies what happens when a given context variable changes, i.e. what happens when we move from one point to another. Making the realistic assumption that the number of dimensions is large, it should be clear that even when most axes of the design space are effectively indepen-

dent, exploring and analyzing such a multi-dimensional design space is a real challenge.

In this article, we present the Context-Aware Application model (CAA), where context adaptations are specified explicitly as graph transformation rules and critical pair analysis [1] is used as a verification tool. Starting from an initial application model (named the *origin* context), the critical pair analysis allows to enumerate all reachable contexts, this way exploring the context design space. Moreover, for those context variables that depend on one another, the critical pair analysis also identifies the transformation sequences resulting in conflicts, this way revealing contexts for which the resulting application model is ambiguous.

The paper itself is set up as a feasibility study, where we investigate how one could use critical pair analysis to find contexts for which the resulting application model is ambiguous. To that extent, we validate the Context-Aware Application model (CAA) using a "proof-by-construction" of a mobile city guide application which has been deployed on an Android phone. Starting from a non-trivial application model specifying the behaviour of the mobile city guide in a single context called the origin context (1 UML class diagram specifying 18 classes; 3 UML activity diagrams specifying 12 tasks; 1 user-interface model specifying 4 screen layouts) we define 11 context adaptations, each one of them specifying what should happen when a single context variable changes. With these transformations, we perform a critical pair analysis, which reveals that 131 out of 256 contexts of the 5 dimensional context space can be reached. The remaining 125 contexts are unreachable because their context-specific models are ambiguous: the sequence of context transformations needed to obtain them contains conflicts. As such, we demonstrate that using the CAA model, a designer can identify subtle context interactions that might go unnoticed otherwise.

The remainder of this article is structured as follows: Section 2 presents the exemplar that is used throughout the article to illustrate the core elements composing the CAA model. Next, Section 3 provides a definition of a context and relates it to other definitions drawn from the literature. This serves as an introduction to the CAA model in Section 4, which is then mapped onto graphs and graph transformations in Section 5. To validate the exploration and analysis mechanism, we demonstrate the critical pair analysis on the exemplar of the mobile city guide in Section 6. We discuss about the lessons learned, the limitations of our approach and some design choices in Section 7 leading to Section 8 that relates the CAA model to existing model-driven engineering solutions for specifying and verifying context-aware applications. Section 9 concludes by summarising the contributions of this article.

2 A Specification Exemplar: The Mobile City Guide

The archetypical example for context-aware computing is a mobile museum guide, where "museum visitors are endowed with a portable device which reacts to changes of contexts, [...] alerting visitors with hints and stimuli on what is going on in each particular ambient" [2]. In that sense, the mobile museum almost obtained the status of a "specification exemplar": a self-contained, informal description of a problem in some application domain proposed as unique input for the specification process [3]. Unfortunately, the mobile museum guide in its current incarnation lacks the necessary reality check to make it a true specification exemplar.

Indeed, on the one hand a good specification exemplar should be sufficiently small to allow for condensed representation in academic papers and textbooks and allow for manual inspection and comparison of results. In this respect, the mobile museum guide clearly qualifies, as it has been used to demonstrate modelling practices for context-aware applications [2,4]. On the other hand, in spite of miniaturization, a good exemplar should also represent a real-world specification task in such a way that it has properties which are not immediately obvious by inspection. This reality check also avoids the problem that specifiers interpret reality in order to better suit the specification. Reviewing the original mobile museum guide exemplars (i.e. [2,4]) this reality check is somehow missing as the example so far was mainly used to illustrate the advantages of a single modelling approach. Consequently, three Belgian research groups aiming to facilitate the development of context-aware applications have contacted the City of Mons tourism office, which —in preparation of the Cultural Capital of Europe 2015 event— is preparing a mobile *city* guide. These three research groups have mixed the touch of realism provided by the city of Mons with the archetypical mobile museum guide example to deduce the mobile city guide as a more realistic specification exemplar "promoting research and understanding among multiple researchers or research groups" [3].

In this section we list those elements of the exemplar relevant for the remainder of the article. Since we do not have the space to provide a complete and detailed description of the mobile city guide we refer the interested reader to `http://lore.ua.ac.be/Research/Artefacts/mobilecityguide`.

2.1 Mobile City Guide — Scope Description

The mobile city guide is an interactive application running on mobile devices (such as a smart phone or a tablet PC) that presents information about points of interest in the vicinity of the person carrying the device. To increase interactivity, this information is displayed when

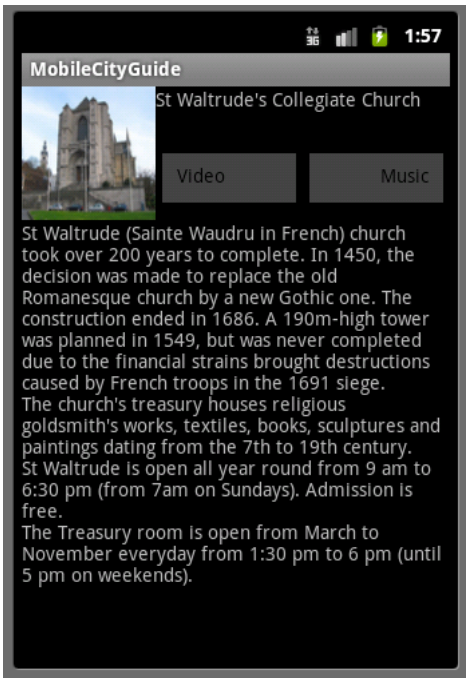

Fig. 1 Mobile city guide presenting information about a point of interest.

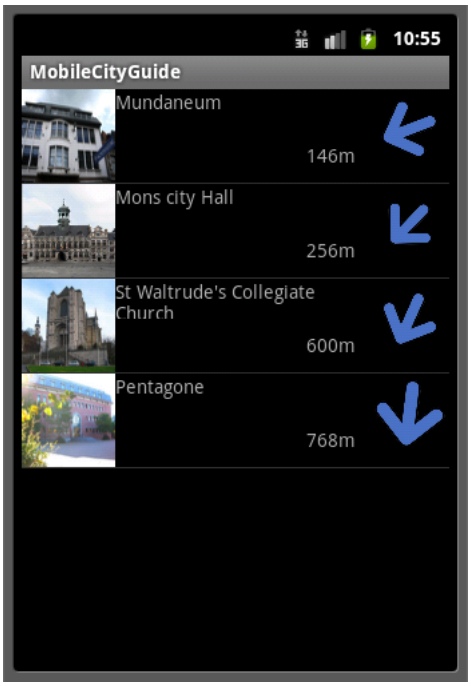

Fig. 2 Mobile city guide guiding the user to other points of interest.

the user is approaching one of these points of interest. The presented information can be of various kinds: pictures, text, video, sound,... When approaching a point of interest, the application should quasi-instantaneously present a picture, a name and a small description and allow a user to request more information if necessary. Video and sound may be played on demand, depending on the availability of a connection to an external database and memory available on the device.

Figure 1 shows a screenshot of how such a mobile application might display a particular point of interest (in this case, a church). The display shows active buttons linking to video and music content concerning this point of interest because there is an open connection to the database providing extra media files. To guide the user towards other points of interest situated in the neighborhood, the mobile city guide displays the name, direction and distance to all points of interest in the vicinity. In this particular case, Figure 2 illustrates what happens when the connection to the maps database is not available; the application then uses information from the global positioning system (GPS) to have arrows pointing in the general direction.

2.2 Mobile City Guide — Context Variables

The mobile city guide is context-aware and therefore tries to optimize its interactivity by taking into account its current context of use. In this exemplar the mobile city guide must adapt itself to the following variables:

User language preference: All texts and menus of the mobile application should be presented depending on the language preferences selected by the user. Depending on the language, texts might flow from left to right (e.g., French, English) or right to left (e.g., Hebrew and Arabic). The mobile city guide must support at least one language in each category.

Screen orientation: the orientation of the device on which the mobile city guide is running should affect the way in which the information is displayed. The mobile city guide must support at least portrait and landscape mode.

Mobile data connection: Many (but not all) of the mobile devices use a mobile data connection to connect to the Internet over the mobile phone network. Depending on the tariff plan of the mobile data connection the mobile city guide may adapt how much data is downloaded over this connection and whether it should open/close the connection as quickly as possible, or leave it open for long periods of time. Therefore, the mobile city guide must at least support the following modes: (a) no data connection available; (b) a pay per connection fee; (c) a pay per Mb connection with restricted download capacity; (d) a connection with unlimited download capacity.

Wifi availability: The availability of a wifi connection may be exploited to proactively download and cache information about points of interest in the vicinity, in order to enhance and speed up user interaction, and to reduce the need of using a slower and more expensive mobile data connection. A wifi connection can either be available or unavailable.

Precision of geo-positioning: To guide the user to nearby points of interest (An example is shown in Figure 2) the precision of the built-in geo-positioning hardware (GPS or GPRS) is a crucial factor. To adapt the behaviour for optimal guidance, the mobile application guide must at least support the following precision modes: 2 meters, 5 meters, 10 meters or 20 meters.

2.3 Mobile City Guide — Platform Independent Model

With the above requirements, the design space for the context-aware model consists of five dimensions; each dimension representing between 2 and 4 possible values for a context variable. As a result, the design space for the complete mobile city guide spans $4 \times 2 \times 4 \times 2 \times 4 = 256$ configurations of context variables. A complete requirements specification should state for each combination what the desired application behaviour should be.

One possible way to describe the desired behaviour for each of these configurations is modelling each configuration in a so-called *platform-independent model* (PIM) [5]. Such a platform-independent model is suited for a requirements specification for a mobile platform, as it allows to abstract away from the device specific details (operating system, display, ...) yet describes precisely how an application should behave in each context.

Three aspects of a mobile application are especially relevant in the context of such a platform-independent model.

1. Structure model: Describing the structure of the application in terms of classes and their relations; UML class diagrams are well suited for this purpose. Figure 3 shows an example.
2. User Interface model: Describing the graphical user interface independently of the particular widgets to be used. Since the UML does not provide a notation for screen layouts, we relied on a special purpose modeling language named CAP3 [6]. Figure 4 shows an example; we elaborate on the use of CAP3 in Section 7.
3. Activity model: Describing how the application may be used to perform activities in order to reach users goals; UML activity diagrams are well suited for describing these. Figure 5 shows an example.

Given that the design space for the complete mobile city guide spans 256 configurations of context variables and that each of these configurations in principle contains an entire platform-independent model of the mobile city guide, it is clear that automated support is desirable. We adopt a transformation-based approach, which is explained in the following sections.

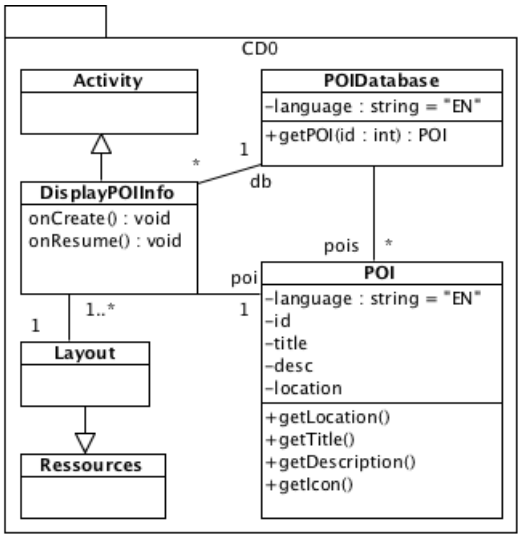

Fig. 3 The part of the class diagram CD_0 specifying the classes used to display information about a point of interest.

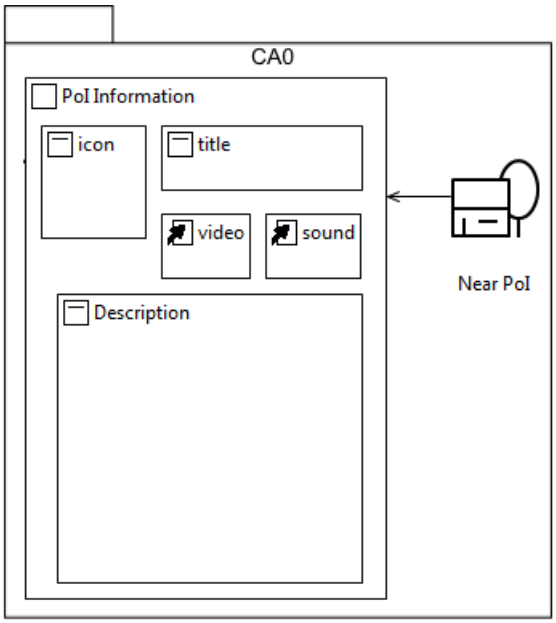

Fig. 4 The part of the CAP3 model CA_0 specifying that when the application is in the vicinity of a point of interest, the display screen in portrait mode should include an icon, a title, a description and a link to a video or sound clip.

3 Definition of Context

When people interact, they are using more than the explicit information that is exchanged. People are able to capture and understand surrounding information, the context of the discussion, and use it to interact more effectively. But when humans-computers takes place, this crucial surrounding information has to be defined precisely in order for the application to understand it. A lot of research has been conducted in the field of human-computer interaction to define and model contexts as well as its implication on software artifacts [2,7,8,9,10,

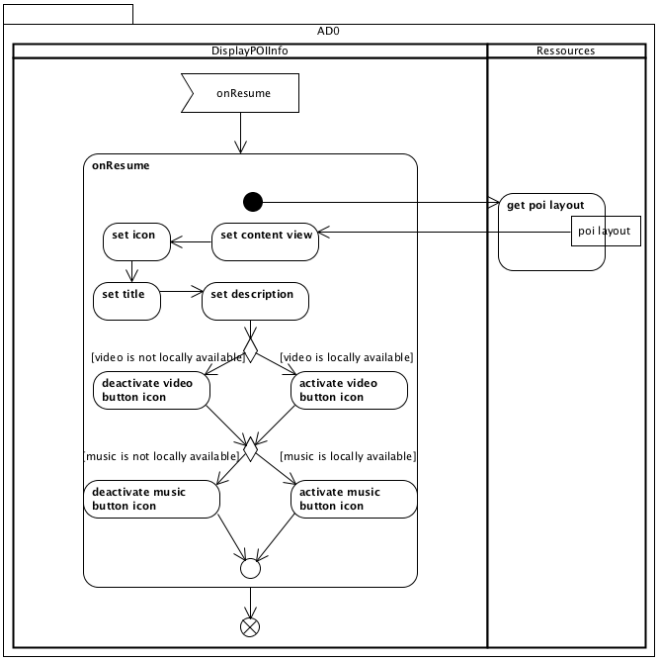

Fig. 5 The part of the activity diagram AD_0 used to activate or deactivate the buttons when the video or sound information is available.

11]. It is not the purpose of this article to discuss or condense all the classifications of context, but it is important to elicit some of them to position the context representation we propose to use. Therefore, it is important to stress that we focus on context-awareness that results in context adaptation impacting the interaction between an application and its users.

The notion of context was introduced by Schilit and Theimer [7] as location, identities of nearby people and objects, and changes to those objects. Brown et al. [12] define context as location, identities of the people around the user, the time of day, season, temperature,... Preuveneers et al. [8] define context as the user, environment, platform and services. Elicitation definitions are discussed in more detail by Dey and Abowd [9] who come up with the following generalized definition:"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Bolchini et al. [2] expose a general analysis framework for comparing these context models. In contrast to these definitions, we are not interested in the meaning or important entities nor the way context information is collected, but more on its concrete representation.

Bolchini et al. [2] propose a context classification framework and use it to classify what they consider the most interesting approaches to describe context available in literature. We refer to their work for a more elaborate overview of context modeling techniques. The context-representation mechanism used in this paper falls in their category E, "Context as a matter of selecting relevant data, functionalities and services (data or functionality tailoring)" for which they state among others the following "Context definition is typically centralized, context history and reasoning are often not provided" and "the application as subject, the possibility to express both variable context granularity, valid context constraints, and multi-context models" [2]. Similar to the context representation presented in this paper, they take a formal approach. Most of these statements correspond to our description and use of context.

We define context in a variable perspective, that does not define entities that are considered part of the context, but instead focuses on their status representation in terms of variables. For a given application and at a given time, the context will be unique. If two entities are relevant for the application, the values of the variables representing the state of these two entities are composing the context. Schmidt et al. [10] already use a variable perspective, but at a lower level of abstraction, as they are mapping each device sensor to a variable. They aggregate different variables in order to obtain meaningful contexts. They are considering that an application can be in a multitude of contexts at the same time and that these contexts do not interfere with one another. Schmidt et al. group their contexts into sets of exclusive contexts depending on their impact on the system, which is closely related to our definition of variable.

Coutaz and Rey [11] also define context in terms of variables, but they consider context to be a composition of situations observed between a reference time t_0 and the actual time t. Each situation at a time t is related to a user performing a certain task. This definition formalizes several important aspects of the definition by Dey and Abowd and introduces the time dimension. All these aspects are important when trying to build a runtime infrastructure, but are less relevant for the goal of this article, which focuses on design time and merely considers the size of the composition of all these situations. We thus use a more generic version of the definition of Coutaz and Rey. Our definition of context is as follows:

Definition 1 ***[Context and context domain]***

Let $n \in \mathbb{N}$ be the finite number of variables that represent the state of the entities relevant for a context-aware application. For all $i \in [1, n]$, each variable *is defined by a finite set V_i that represents all possible values. The* context domain *is defined by $\mathcal{C} = V_1 \times V_2 \times \ldots \times V_n$. A* context *$c = (v_1, v_2, \ldots v_n) \in \mathcal{C}$ is a tuple of n values, one for each variable domain V_i. Using the projection operator $\pi_i : V_1 \times V_2 \times \ldots \times V_n \rightarrow V_i$ one can obtain the i-th value of a context $\pi_i(c) = v_i$.*

For the mobile city guide, the context domain is $\mathcal{C} = Language \times Orientation \times Data\_connection \times Wifi \times Positioning\_precision$ where:
$Language = \{English, Dutch, French, Arabic\}$,
$Orientation = \{portrait, landscape\}$,

$Data\_connection = \{noCon, pay/con, pay/Mb, unlimited\}$, $Wifi = \{wifiAvailable, noWifi\}$ and $Positioning\_precision = \{2m, 5m, 10m, 20m\}$.
Hence the mobile city guide context domain is a 5-dimensional space, covering 256 contexts in total. Each variable corresponds to one dimension, and each point of the space defines a particular context composed of one value for each variable. For example, a possible context could be $c_0 = (English, portrait, noCon, noWifi, 2m)$. We refer to it as the *origin context* as we use it as a starting point for reaching all other contexts.

Note that the theoretical number of contexts an application has to adapt to increases exponentially in function of the number of variables, and polynomially in function of the values these variables can take. Consequently, scalability is certainly an issue and must be addressed.

4 The Context-Aware Application Model

To explore and analyze the context space we use a transformation-based specification called the Context-Aware Application (CAA) model. Before exposing our techniques to explore the context space, we formally define the concepts used in a CAA model and the CAA model itself.

Definition 2 ***[Context-specific model]*** *The specification of an application for a given context $c \in \mathcal{C}$ is expressed using a model, that we denote as the* context-specific model M_c.

Typically, a context-specific model is specified as a set of different platform-independent models. For example, if we use UML as modeling language, a context-specific model could be defined as a set of class diagrams, activity diagrams, state machines, use case diagrams and so on. Consequently, the domain of possible models that can be expressed is infinite.

Definition 3 ***[Model Domain]*** *The* model domain $\mathcal{M}$ *is the set of all possible models that can be described using the chosen modeling language.*

In this article, we will restrict the models of $\mathcal{M}$ to sets containing at least one UML class diagram, one UML activity diagram and one CAP3 model. Taking the origin context c_0 of the previous section as an example, its context-specific model $M_{c_0} = \{CD_0, CA_0, AD_0\}$ is composed of the diagrams partially shown in Figure 3, Figure 4 and Figure 5.[1] The data to be displayed to the user when he is approaching a point of interest is loaded in the user language (i.e., English) as reflected in the class diagram CD_0. Moreover, as no wifi connection is available and the user does not have access to a mobile data connection, when some artifacts (either music or video) are not locally available their corresponding buttons are deactivated as shown in the activity diagram AD_0. The CAP3 model CA_0 specifies the user interface when the mobile device is used in portrait mode. The positioning precision is not perceptible in the part of the diagrams shown in Figures 3, 4 and 5 as the precision is only used by the algorithm and user interface during the activity of guiding the user to the next point of interest.

When a context changes, the context-aware application has to adapt accordingly, resulting in a new context-specific model that only differs slightly from the previous one. Therefore, it is logical to represent a context adaptation of a context-specific model by using a model transformation. More precisely, given a context-specific model M_c for the context c, the modification induced by change of the application's context to context d can be specified by a *model transformation rule* denoted $r_{c,d}$. Applying this transformation rule to M_c results in a context-specific model M_d belonging to context d. We assume that the transformation rule $r_{c,d}$ can always be applied on the context-specific model M_c. This condition, that is not restrictive in practice as discussed in Section 7, simplifies the CAA model.

Definition 4 ***[Context change]*** *Let $\mathcal{C}$ be the context domain of a context-aware application. A* context change *over $\mathcal{C}$ (or simply a 'context change' when $\mathcal{C}$ is implicit) is a pair $(c, d) \in \mathcal{C} \times \mathcal{C}$ with $c \neq d$, representing the change of the application context from context c to d.*

Definition 5 ***[Model transformation (rule)]***
A model transformation rule *$r_{c,d}$ specifies the impact of a context change (c, d), i.e., it yields M_d if it would be applied on the context-specific model M_c. The set of all possible transformation rules is denoted $\mathcal{R}$.*
A model transformation *is defined as a function $t : \mathcal{M} \to \mathcal{M}$.*

For example, if we apply the rule $r_{c,d}$ to context-specific model M_c, we obtain the model transformation $t_{c,d} : M_c \to M_d$. If the rule $r_{c,d}$ is also applicable to another context-specific model M_e, we obtain another model transformation $t_{e,f} : M_e \to M_f$ corresponding to a context change (e, f).

In the example of the mobile city guide, what should happen if at a certain point in time the wifi network becomes available? Starting from the aforementioned context-specific model M_{c_0} for the origin context c_0, one can obtain a context-specific model M_d for the context $d = (English, portrait, noCon, wifiAvailable, 2m)$ by applying the transformation rule $r_{c_0,d}$ of Figure 6. In this figure, the model transformation rule is presented as a graph transformation rule with a left-hand side L and a right-hand side R specifying the abstract syntax of the model transformation rule. The effect of applying the rule $r_{c_0,d}$ to model M_{c_0}, resulting in model M_d, is visualised in Figure 6. As the impact of discovering a wifi

[1] The entire context-specific model M_{c_0} is available at `http://lore.ua.ac.be/Research/Artefacts/mobilecityguide`.

network does not affect the user interface, only some elements of the activity diagram (node `1:Activity` of Figure 6) and the class diagram (node `2:Class` of Figure 6) are transformed.

For any given context, a special relation, called context mutation, links this context with all the contexts that only differ from it by one variable value.

Definition 6 ***[Context mutation]*** *We define the function* $\mathsf{mut}_i : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{C})$ *such that* $\mathsf{mut}_i(c)$ *is the set of i-mutations of the context c, i.e.,* $\mathsf{mut}_i(c) =$

$$\{d \in \mathcal{C} \mid \pi_i(c) \neq \pi_i(d) \wedge \forall k \in [1,n] \setminus \{i\}, \pi_k(c) = \pi_k(d)\}$$

Based on the context mutation function we can define the more specific notion of context projection as follows:

Definition 7 ***[Context projection]*** *We define the i-th context projection function* $\mathsf{proj}_i : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ *such that* $\mathsf{proj}_i(c,d)$ *is the context with all variables' values equal to the corresponding value in c except for the i-th variable value that is equal to its respective value of d. More precisely,* $\mathsf{proj}_i(c,d) = e \in \mathsf{mut}_i(c)$ *such that* $\pi_i(e) = \pi_i(d)$.

Definition 8 ***[CAA model]*** *A CAA model is a triple* $(c_0, M_{c_0}, \mathcal{X}) \in \mathcal{C} \times \mathcal{M} \times \mathcal{P}(\mathcal{R})$ *where* $c_0 \in \mathcal{C}$ *represents the origin context for which this application is entirely specified in the context-specific model* $M_{c_0} \in \mathcal{M}$.

The set $\mathcal{X} \in \mathcal{P}(\mathcal{R})$ *contains transformation rules specifying the impact of a context change where only one variable value is modified while the values of all the other variables are equal to their respective value in* c_0. *More precisely,* $r_{c_0,d} \in \mathcal{X}$ *if* $\exists i \in [1,n]$ *such that* $d \in \mathsf{mut}_i(c_0)$, *and* $r_{c,d} \in \mathcal{X}$ *if* $\exists j \in [1,n]$ *such that* $c \in \mathsf{mut}_j(c_0)$ *and* $d \in \mathsf{mut}_j(c)$.

Conceptually, if we consider the context space with c_0 as origin, the set $\mathcal{X}$ contains the transformation rules specifying a context change impact where the starting context and the destination context are on the same variable axis. This last constraint is induced by the separation of concerns principle a modeler would follow specifying a context-aware application. The rules $r_{c,d}$ with $c \in \mathsf{mut}_i(c_0)$ and $d \in \mathsf{mut}_i(c)$ are considered part of $\mathcal{X}$ because a modeler would want to specify what the impact is of a context mutation from a context-specific model different from M_{c_0}. For example, it is natural to think that after modeling a user's language change from 'English' to 'Arabic', a modeler would prefer to specify a rule representing a change from 'Arabic' to 'Hebrew', two right-to-left read languages. To abstract away the complexity of this sort of transformation rules from the context coverage, a closure function is needed.

Definition 9 ***[Closure]*** *We define the function* $\mathsf{closure}_c : \mathcal{P}(\mathcal{R}) \rightarrow \mathcal{P}(\mathcal{R})$ *such that* $\mathsf{closure}_c(\mathcal{Y})$ *is the subset of transformation rules* $r_{c,d}$ *of* $\mathcal{Y}$ *closed under sequential composition* $\mathsf{seqComp}$. *More precisely,* $\mathsf{closure}_c(\mathcal{Y})$ *is the set* $\mathcal{Z} = \mathcal{Y} \cup \{\mathsf{seqComp}(r_{c,f}, r_{f,e}) \mid r_{c,f} \in \mathcal{Z} \wedge r_{f,e} \in \mathcal{Z}\}$ *In the remainder of this article we use* $\mathcal{Y}_c^*$ *to denote* $\mathsf{closure}_c(\mathcal{Y})$.

Given the set of transformation rules $\mathcal{X}$ and the context-specific model for the origin context c_0, obtaining the specification of the application for a context d seems simple. For each variable i, it is possible to find in $\mathcal{X}_{c_0}^*$ a transformation rule that specifies the context i-mutation from the i-th value of c_o (i.e., $\pi_i(c_0)$) to the i-th value of d (i.e., $\pi_i(d)$). To obtain the context-specific model of the application for context d, it is then necessary to apply consecutively the rules for each i. In practice, however, it is likely that for some context d the variables that change value between c_0 and d are interdependent. This dependency can be detected at the model level by a conflict occurring between a pair of transformation rules corresponding to the mutations that have to be sequentially composed in order to obtain the model transformation $t_{c_0,d} : M_{c_0} \rightarrow M_d$.

Definition 10 ***[Conflicting]*** *For a context-specific model* M, *we define the partial boolean function* $\mathsf{conflicting}_M : \mathcal{R} \times \mathcal{R} \rightarrow \mathbb{B}$ *such that* $\mathsf{conflicting}_M(r,s)$ *is true if and only if the transformation rule s cannot be applied after r on M i.e. r is not sequentially composable with s in M.*

Let us consider again the situation where a context adapts to a wifi network becoming available, thus moving from $c = (English, portrait, noCon, noWifi, 2m)$ to $d = (English, portrait, noCon, wifiAvailable, 2m)$ by applying the transformation rule $r_{c_0,d}$ of Figure 6. On the other hand, consider the context adaptation of an unlimited mobile data connection becoming available, from $c = (English, portrait, noCon, noWifi, 2m)$ to $e = (English, portrait, unlimited, noWifi, 2m)$ by applying the transformation rule $r_{c,e}$ of Figure 7. Both transformation rules $r_{c,e}$ and $r_{c,d}$ are in conflict because both are modifying the `onResume` activity. Conceptually, we decided to model the change of behaviour of an activity by a change of the activity's name. The real behaviour is specified at implementation level.

This example highlights the need for a verification of the contexts covered by a given specification. The conflict between $r_{c,d}$ and $r_{c,e}$ reveals that it is impossible to obtain the corresponding context-specific model from the CAA specification, where a wifi is available and an unlimited mobile data connection is established. This reveals a partial dependency between the variables related to the wifi and the mobile data connection. The goal of the context coverage construction is therefore twofold. Firstly, it allows to validate the modeled design against the requirements, not in terms of behaviour but in terms of supported contexts. Secondly, it allows an analysis of the independence of the variables constituting the context.

Given a set of transformation rules $\mathcal{X}$ and the context-specific model for the origin context c_0, it is possible to

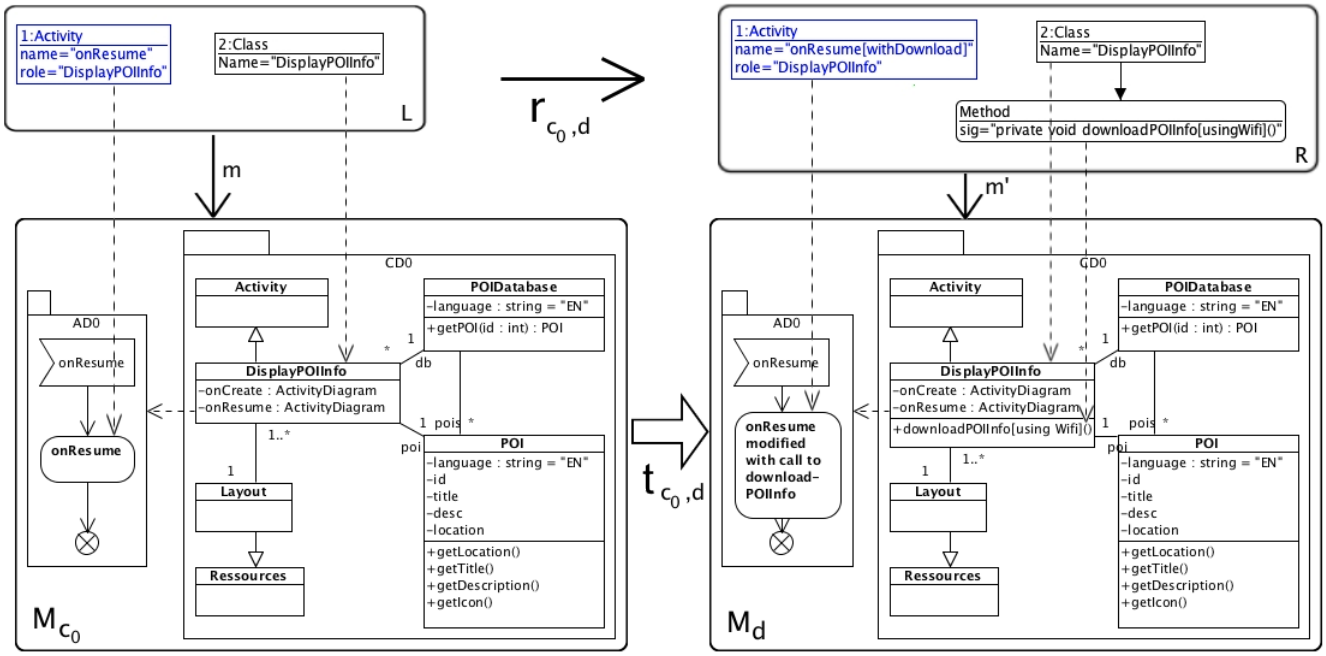

Fig. 6 Application of the transformation rule $r_{c_0,d}$ on the context-specific model M_{c_0}. $r_{c_0,d}$ specifies the context change when a public wifi is detected.

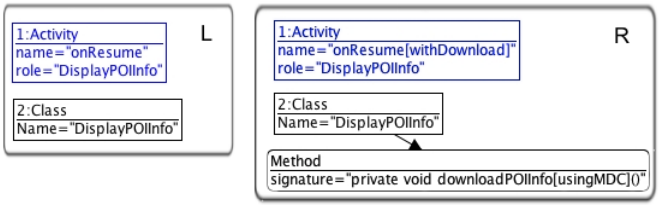

Fig. 7 Transformation rule specifying the adaptation of the mobile city guide when an unlimited mobile data connection is established

know if a given context d is part of the reachable context. Intuitively context d is reachable if for each variable i it is possible to find a transformation rule in $\mathcal{X}^*_{c_0}$ that specified the context i-mutation from the i-th value of c_0 to the i-th value of d, and that these transformations are not in conflict.

Definition 11 ***[Context coverage]*** *We define the function* $\mathsf{coverage} : \mathcal{C} \times \mathcal{M} \times \mathcal{P}(\mathcal{R}) \rightarrow \mathcal{P}(\mathcal{C})$ *such that* $\mathsf{coverage}(c, M_c, \mathcal{Y})$ *contains all possible contexts for which it is possible to obtain the associated context-specific model from context-specific model* M_c *by applying non-conflicting transformation rules in* $\mathcal{Y}$*. Formally,*

$$\mathsf{coverage}(c, M_c, \mathcal{Y}) =$$

$$\{c\} \cup \{d | \exists r_{c,d} \in \mathcal{Y}^*_c\}$$

$$\cup \{e \mid \forall i, j \in \{1 \dots n\}, i \neq j,$$

$$r_{c,\mathsf{Proj}_i(c,e)}, r_{c,\mathsf{Proj}_j(c,e)} \in \mathcal{Y}^*_c \wedge$$

$$\neg \mathsf{conflicting}(r_{c,\mathsf{Proj}_i(c,e)}, r_{c,\mathsf{Proj}_j(c,e)})\}$$

5 Graph transformation

To enable the computation of the context coverage computation, the abstract CAA model of Definition 8 needs to be mapped to a formal executable language. Many of the definitions of Section 4 do not need such a mapping. Only those notions need to be mapped that are explicitly used and needed in the definition of context coverage (Definition 11). This is the case for the definitions of context-specific model, model domain, model transformation rule, closure and conflicting (Definitions 2, 3, 5, 9 and 10).

A graph transformation language is suitable as formal executable language. On the one hand, it allows to specify the abstract syntax of a context-specific model as a typed attributed directed graph. On the other hand, the model transformation rules specifying context adaptations can be represented as graph transformation rules. An additional benefit of graph transformation is that several tools exist that offer mechanisms to facilitate the computation of context coverage. The language and tool that has been used in our validation is AGG [13].

Given a CAA model $(c_0, M_{c_0}, \mathcal{X})$, the abstract syntax of the context-specific model M_{c_0} is specified by a graph denoted G_{c_0}. We decide to represent all diagrams that make up the context-specific model M_{c_0} using a single graph G_{c_0} in order to facilitate keeping track of the existing relations between the different views (i.e., the different diagrams) of the same element. For example, for context $c_0 = (English, portrait, noCon, noWifi, 2m)$, the graph of Figure 8 specifies the abstract syntax of the part of the context-specific model M_{c_0} pre-

sented in Figure 6. Consequently the graph domain $\mathcal{G}$ corresponding to the model domain $\mathcal{M}$ defined in Definition 3, includes the abstract syntax of all the models included in $\mathcal{M}$. Similarly, the set of model transformation rules $\mathcal{X}$ is mapped to a set of graph transformation rules $\mathcal{GX}$.

Definition 12 (graph transformation) *We denote a* graph transformation *by* $t : G \xrightarrow[r,m]{} H$ *where* G *is the source graph,* $r : L \to R$ *a graph transformation rule,* $m : L \to G$ *a match and* H *the result graph.*

The match m is a function that links the elements of the left-hand side L of the graph transformation rule r to elements of the graph G the rule is applied to. A constraint of our approach is that all rules $r_{c_0,d}$ of $\mathcal{X}$ have a unique match in the graph G_{c_0}. While this could seem restrictive from a graph transformation perspective, this is not the case for the specific use we make of it. We discuss this further in Section 7.

Graph transformation rules can be enhanced with application conditions to further constrain their application [14]. Because this enhanced expressiveness is not needed for specifying the impact of a context change, we will not use it in the remainder of this article.

An example of a graph transformation rule $r_{c_0,d}$, specifying the adaptation of the mobile city guide when a public wifi connection is established is given in the top part of Figure 6. To be precise, the bottom part $T_{c_0,d} : M_{c_0} \to M_d$ of the figure should also be replaced by its graph equivalent $t : G_{c_0} \xrightarrow[r,m]{} G_d$.

Following Definition 9, the computation of the closure requires to sequentially compose graph transformation rules. Since version 2 of AGG [15], the sequential composition seqcomp of two graph transformation rules $r_{c,d}$ and $r_{d,e}$ can be achieved by creating the concurrent rule for a rule sequence that corresponds to a new rule $r_{c,e}$. Figure 9 illustrates this, with $r_{c,d}$ the rule specifying a change of the user language from 'English' to 'Dutch' and $r_{d,e}$ a change of the user language from 'Dutch' to 'French'. The sequential composition results in a rule $r_{c,e}$ specifying a language change from 'English' to 'French'.

We used this sequential composition $\mathsf{seqcomp}(r, s)$ in the closure computation of Algorithm 1. Note that seqcomp is a partial function since it is only computable for two graph transformation rules r and s if they can be applied in sequence.

Algorithm 1 *[*$\mathsf{closure}_c(\mathcal{GX})$*]*

$\mathcal{GX}^* \leftarrow \{\}$
for all *context variables* V_i *of* $\mathcal{C}$ ***do***
  $\mathcal{I} \leftarrow$ *all* $r_{c,d} \in \mathcal{GX}$ *with* $d \in mut_i(c)$
  create a queue Q
  enqueue all $r_{c,d} \in \mathcal{I}$ *in* Q
  while Q *is not empty* ***do***
    dequeue a graph transformation rule $r_{c,d}$ *from* Q
    for all $r_{d,e} \in \mathcal{GX}$ *such that* $r_{c,e} \notin \mathcal{I}$ ***do***
      if $\mathsf{seqcomp}(r_{c,d}, r_{d,e})$ *is defined* ***then***
        $r_{c,e} = \mathsf{seqcomp}(r_{c,d}, r_{d,e})$
        enqueue $r_{c,e}$ *in* Q
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{r_{c,e}\}$
      end if
    end for
  end while
  $\mathcal{GX}^* \leftarrow \mathcal{GX}^* \cup \mathcal{I}$
end for

Algorithm 1 is a transitive closure algorithm executing in a polynomial time that use the seqcomp as composition operator for graph transformation rules. This algorithm is used to compute the closure for the mobile city guide example in Section 6.

Finally, to compute the context coverage, the computation of the conflicting function of Definition 10 needs to be made applicable on a pair of graph transformation rules. The critical pair analysis of AGG is used for this purpose. Critical pairs formalize the idea of a conflicting situation in a minimal context [16,17]. Moreover, from the set of all critical pairs we can retrieve the result of the context coverage by extracting the objects and links that cause the conflicts leading to uncovered context.

Definition 13 (conflicting transformations)
Two graph transformations $t_1 : G \xrightarrow[r_1,m_1]{} H_1$ *and* $t_2 : G \xrightarrow[r_2,m_2]{} H_2$ *are in conflict if* t_1 *cannot be performed after* t_2 *(i.e., rule* r_1 *cannot be applied to* H_2*) or vice versa (i.e., rule* r_2 *cannot be applied to* H_1*).*

Definition 14 (critical pairs) *A critical pair is a pair of conflicting graph transformations* $t_1 : G \xrightarrow[r_1,m_1]{} H_1$ *and* $t_2 : G \xrightarrow[r_2,m_2]{} H_2$ *such that* G *is a minimal graph, i.e., there is no proper subgraph* G' *of* G *such that there are conflicting transformations* $t'_1 : G' \xrightarrow[r_1,m'_1]{} H'_1$ *and* $t'_2 : G' \xrightarrow[r_2,m'_2]{} H'_2$ *with* $m'_i(x) = m_i(x)$ *for all* $x \in L_{p_i}$ *and* $i = 1, 2$.

To each pair of graph transformation rules r_1 *and* r_2 *we can associate a (possibly empty) set of critical pairs* $t_1 : G \xrightarrow[r_1,m_1]{} H_1$, $t_2 : G \xrightarrow[r_2,m_2]{} H_2$.

There are two reasons why graph transformation rules can be in conflict. First a rule application deletes a graph element (i.e., a node or edge) that is in the match of another rule application. Secondly a rule application changes attributes being in the match of another rule application.

The existence of critical pairs between two graph transformation rules can be used to derive if the corresponding model transformation rules are conflicting with respect to our definition 10. If one or more critical pairs are detected between two graph transformation rules, these transformations can be conflicting. It is however possible that some detected critical pairs will never occur in the graph on which the transformation rules are

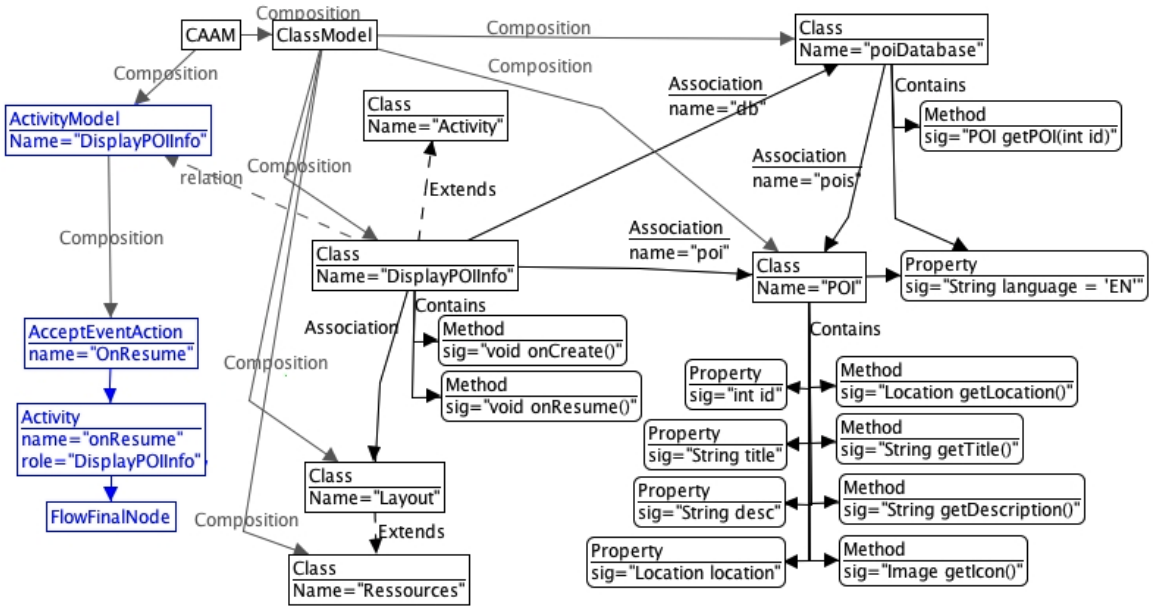

Fig. 8 Graph G_{c_0} specifying abstract syntax of the part of the context-specific model M_{c_0} presented in Figure 6

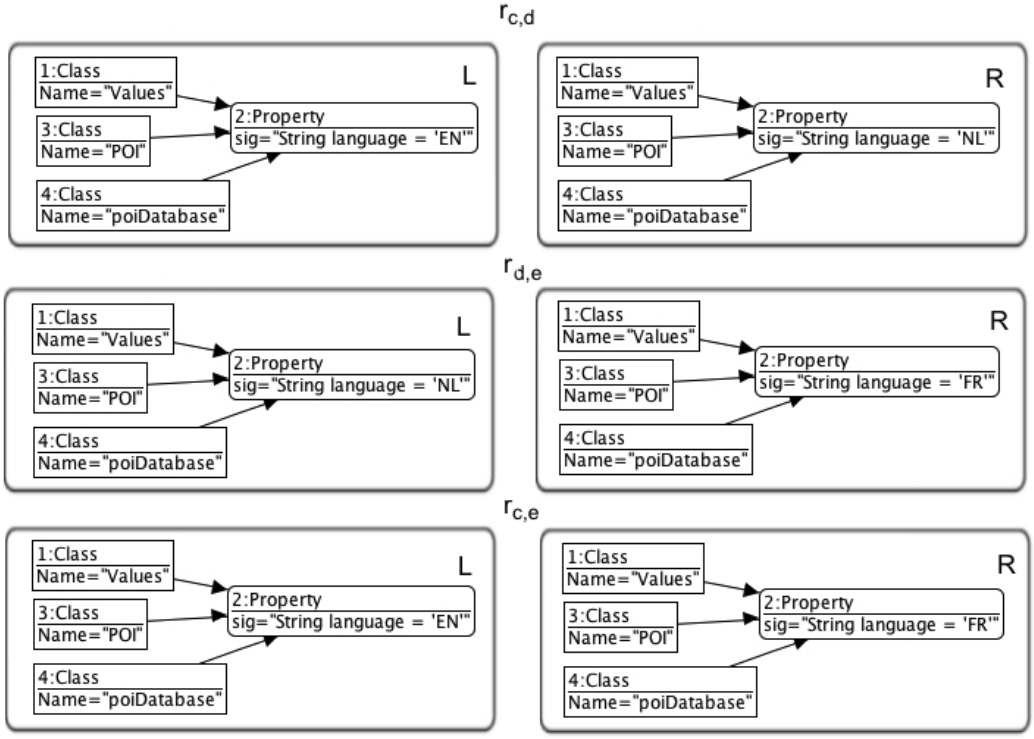

Fig. 9 Sequential composition of two rules $r_{c,d}$ and $r_{d,e}$ resulting in a new rule $r_{c,e}$.

applied. AGG can reduce the number or reported critical pairs, by checking for the existence of a match for each critical pair with respect to a given graph. In our case, the graph G_{c_0} on which the transformation rules are applied is well known, so we can use it to remove automatically the irrelevant critical pairs.

Given the ability to compute the conflicting function and given the closure of the transformation rules specifying the context mutations (i.e., modifications of one context variable value), it is possible to determine the set of reachable contexts. By combining context mutations, it is possible to obtain a specific context d from the origin context c_0. For each variable V_i whose value differs between context c_0 and context d, $closure_{c_0}(\mathcal{GX})$ must contain a graph transformation rule $r_{c_0,e}$ with $e = \mathsf{proj}_i(c_0, d)$. If $r_{c_0,e}$ is not part of $closure_{c_0}(\mathcal{GX})$, the

context d is not reachable. But when all the transformations corresponding to the mutations necessary to transform the origin context to d are part of the closure and have no critical pair (i.e., are not conflicting), the context d is reachable. In this case, the resulting context-specific graph G_d, and by extension its corresponding context-specific model M_d can be obtained in a non-ambiguous way. As the critical pair analysis checks a pair of graph transformation rules, all transformation rules corresponding to a mutation required to obtain d from c_0 are checked pairwise.

Algorithm 2 computes the set of contexts that are reachable given a CAA model in a polynomial time.

Algorithm 2 ***[Coverage(*** c_0, G_{c_0}, $\mathcal{GX}^*$***)]***

$\mathcal{CC} = \{c\} \cup \{d | \exists r_{c,d} \in \mathcal{Y}^*_{c_0}\}$
for all $e \in \mathcal{C}$ ***do***
  $inCC \leftarrow true$
  for all $i \in [1, n]$ ***do***
    if $\exists r_{c_0,f} \in \mathcal{GX}^*$ *with* $f = \mathsf{proj_i}(c_0, e)$ ***then***
      for all $j \in [1, n]$ *with* $j \neq i$ ***do***
        if $\exists r_{c_0,g} \in \mathcal{GX}^*$ *with* $g = \mathsf{proj_j}(c_0, e)$ ***then***
          if $|\mathsf{criticalpairs}_{G_{c_0}}(r_{c_0,f}, r_{c_0,g})| \geq 0$ ***then***
            $inCC \leftarrow false$
          end if
        else
          $inCC \leftarrow false$
        end if
      end for
    else
      $inCC \leftarrow false$
    end if
  end for
  if $inCC$ ***then***
    $\mathcal{CC} = \mathcal{CC} \cup \{e\}$
  end if
end for

This algorithm is used in Section 6 on the mobile city guide validation.

6 Validation

To validate our approach, we explore the design space of the mobile city guide, searching for the unreachable contexts caused by the dependencies of context variables for some of their values. Some of these dependencies might be intuitively predictable without formal support, yet since the approach enumerates all possibilities systematically, it is guaranteed to reveal also those dependencies which might go unnoticed otherwise.

We start with an overview of the mobile city guide and the contexts it should adapt to. The CAA model representing the mobile city guide is then constructed using the graph transformation language and presented partially. Finally, the context coverage is computed for the CAA model, and the results are used to characterize the design space and the relations between context variables.

As introduced in Section 2, the mobile city guide is composed of an activity presenting information about the nearest point of interest and an activity to guide the user to reach points of interest. The mobile city guide encompasses two more activities, one taking care of the mobile city guide main menu, and one to manage the user preferences. All these activities are context-aware and adapt to the mobile city guide context. The variables impacting the mobile city guide have been described in Section 2, and their formalizations were presented in Section 3.

The context adaptations of the CAA model for the mobile city guide are specified using graph transformation rules. The origin context has been arbitrarily defined as the context $c_0 = (English, portrait, noCon, noWifi, 2m)$ in which the user's language is $English$, the mobile device is used in *portrait* mode, the mobile device has no mobile connection access, $noWifi$ is available and the precision of the built-in geopositioning sensor is 2 meters. The mobile city guide is then entirely modeled for this origin context in order to create the context-specific model M_{c_0}. This model is composed of three different kinds of diagrams: a class diagram specifying the structure of the mobile city guide, activity diagrams specifying how activities are organized in order to fulfill application tasks, and finally some user interface diagrams using the CAP3 notation [6]. The complete model is too large to show in the article. Interested readers can find it on our mobile city guide website: `lore.ua.ac.be/Research/Artefacts/mobilecityguide`

As an illustration of the used diagrams, part of the class diagram is presented in Figure 3, part of an activity diagram is presented in Figure 5, and Figure 4 presents the user interface of the presentation activity using the CAP3 notation. Table 1 shows the size of the context-specific model for the origin context in terms of class diagrams, activity diagrams and user interface diagrams but also in terms of the elements they define: classes, tasks, screen layouts and their respective abstract syntax node elements.

For this context-specific model, the following set $\mathcal{GX}$ of 11 graph transformation rules has been created to specify the impact of context variable changes:

- 3 graph transformation rules to represent language related adaptations:
  $r_{l1} = r_{c_0,(Dutch,portrait,noCon,noWifi,2m)}$
  $r_{l2} = r_{(Dutch,portrait,noCon,noWifi,2m),}$
  $(French,portrait,noCon,noWifi,2m)$
  $r_{l3} = r_{c_0,(Arabic,portrait,noCon,noWifi,2m)}$
- 1 graph transformation rule to change the mobile device orientation:
  $r_{o1} = r_{c_0,(EN,landscape,noCon,noWifi,2m)}$
- 3 graph transformation rules to change the mobile data connection mode:

Type of diagram	#diagrams	# diagram elements	#graph nodes	#graph edges
Class diagram	1	18 classes	102 nodes	126 edges
Activity diagram	3	12 tasks	36 nodes	36 edges
UI diagrams	1	3 layouts	21 nodes	23 edges

Table 1 Size of context-specific model for the origin context.

$r_{g1} = r_{c_0,(EN,portrait,pay/con,noWifi,2m)}$
$r_{g2} = r_{c_0,(EN,portrait,pay/Mb,noWifi,2m)}$
$r_{g3} = r_{c_0,(EN,portrait,unlimited,noWifi,2m)}$

– 1 graph transformation rule to change the wifi availability:
$r_{w1} = r_{c_0,(EN,portrait,noCon,wifiAvailable,2m)}$

– 3 graph transformation rules to adapt the positioning precision:
$r_{p1} = r_{c_0,(AR,portrait,noCon,noWifi,5m)}$
$r_{p2} = r_{c_0,(AR,portrait,noCon,noWifi,10m)}$
$r_{p3} = r_{c_0,(AR,portrait,noCon,noWifi,20m)}$

As an example, the transformation rule r_{g3} of Figure 7 specifies the adaptation of the mobile city guide to a change from no data connection to an unlimited data connection. Table 2 shows the complexity of the transformation rules in terms of the size of the left-hand side, the number of created nodes and edges and the number of modified attribute values.

In order to compute the context coverage of the mobile city guide CAA model, the closure $\mathcal{GX}^*$ of the $\mathcal{GX}$ set has to be computed first. Executing the closure algorithm (Algorithm 1) results in one possible combination of transformation rules r_{l1} and r_{l2} (respectively represented in Figure 9 by $r_{c,d}$ and $r_{d,e}$) that can be sequentially composed to obtain the transformation specifying the adaptation of the mobile city guide when the language preference changes from English to French. The closure $\mathcal{GX}^*$ is then composed of all transformation rules of $\mathcal{GX}$, excluding T_{l2} and including the sequential composition of T_{l1} and T_{l2}.

The $coverage(c_o, \mathcal{GX}^*)$ of the mobile city guide's CAA model can be computed using critical pair analysis. The critical pair analysis results in a set of critical pairs detected between each pair of graph transformation rules in $\mathcal{GX}^*$ and is shown in Figure 10. As the number of rules involved does not permit to display the entire table of critical pairs, we disabled all rules related to the positioning precision from the figure since they were not in conflict with any other transformation rules.

Based on the detected critical pairs, Algorithm 2 computes the coverage of the mobile city guide CAA model, resulting in a coverage set containing 131 contexts (out of 256) that are reachable from the origin context given the mobile city guide specification. The remaining 125 contexts of the mobile city guide context domain are not present in the coverage set and therefore not reachable from the origin context. An in-depth exploration of the design space of reachable versus unreachable contexts reveals that all the contexts in which the data connection preference value is different from *noCon* and having $wifiAvailable$ are not reachable. The contexts combining a mobile device in *landscape* mode with the *Arabic* language are also unreachable.

Guided by the results of the coverage computation, further investigations of the transformation rules to adapt the mobile city guide to establish a data connection or to discover a wifi network reveal a complete dependency between these two variables (i.e., all their possible values are in conflict). Indeed, in case of the mobile city guide, these two variables have exactly the same effect on the behaviour of the mobile city guide. The 'onResume' activity (shown in Figure 5) is modified to reflect the possibility the user has to download music and video, by activating the video and music buttons. In addition, a call to a download functionality is added to the same 'onResume' activity to pro-actively download music and video when a wifi is available or when an unlimited data connection is established.

This reveals a first defect, representative for an entire class of faults — overlaps between two variables representing context switches. In this particular case the variables $Data\_connection$ and $Wifi$ overlap because, when a wifi is available, the mobile city guide has to prefer the Wifi connection over the mobile data connection for downloading the data related to a point of interest independent of the status of the data connection. To resolve this overlap it is sufficient to merge the variables $Data\_connection$ and $Wifi$ into a single variable (for example named *Connection*). After this modification the number of contexts for which the mobile city guide needs to adapt itself decreases to 160 and the number of reachable context increases to 140.

Concerning the last 20 unreachable contexts, these reveal a second defect, representative for another class of faults — context switches where the order in which they are applied matters. In this particular case, when the user changes to *Arabic* language and the device switches to *landscape* mode — the change to *Arabic* rearranges all text from right to left, but the device rotation will also affects the display of the text. This represent unintended side-effects that a designer should be aware of.

To summarise, while the intuition of a designer modeling a context-aware application can be used upfront during the specification of a CAA model, a complete systematic exploration of the design space remains necessary to detect situations that have been modeled incorrectly. In particular, we have demonstrated that our approach can detect two classes of faults which may result in ambiguous specifications (a) overlaps between two variables representing context switches; (b) context

Rule name	# nodes in LHS	#edges in LHS	#attributes modified	#created nodes	#created edges
r_{l1}	4	3	1	0	0
r_{l2}	4	3	1	0	0
r_{l3}	8	3	5	0	0
r_{o1}	6	5	10	0	0
r_{g1}	3	0	3	0	0
r_{g2}	3	0	3	0	0
r_{g3}	2	0	1	1	1
r_{w1}	2	0	1	1	1
r_{p1}	3	2	2	0	0
r_{p2}	3	2	2	0	0
r_{p3}	3	2	2	0	0

Table 2 Size of transformation rules representing context variable changes.

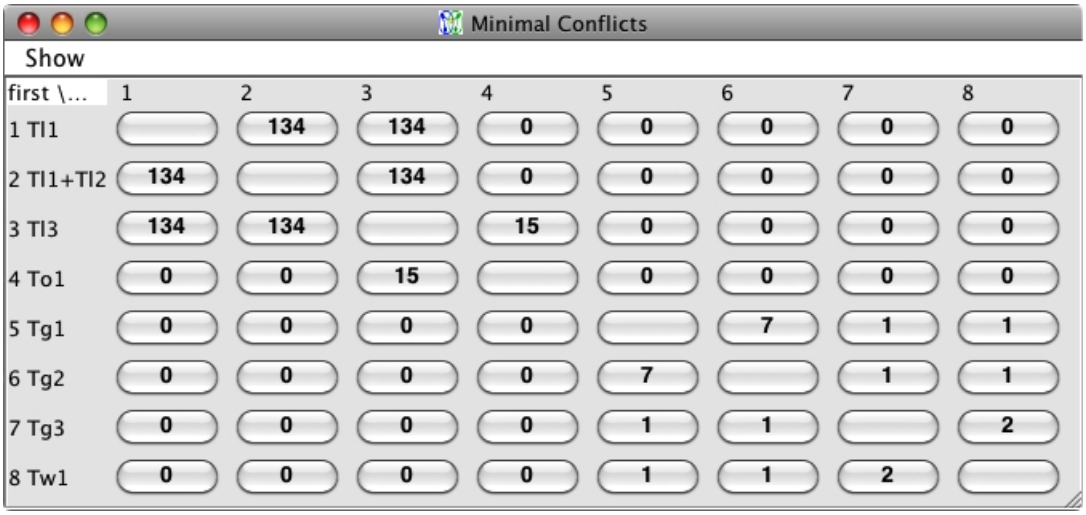

Fig. 10 Result of the critical pair analysis

switches where the order in which they are applied matters. Consequently, verifying context-models seems indispensable during the design phase, in order to iteratively improve design models for context-adaptive applications.

7 Discussion

Since this paper is set up as a *feasibility study*, investigating how one could use critical pair analysis to find contexts for which the resulting application model is ambiguous, obviously some restrictions on the applicability of the CAA model are in order. In this section, we list the design choices made for the feasibility study, the most important limitations we are aware of, and the lessons learned during the validation.

CAP3 notation. Of the three notations used to specify the platform independent model, one perhaps requires some extra explanation — the MOF-compliant CAP3 language [6]. We use this notation to specify the coarse-grained layout of the user interface independently of the particular widgets to be used, as should be the case for a platform-independent model. There are other modeling languages that allow to specify user-interface layouts. In this paper we opted to use CAP3 because it is explicitly designed to support user interfaces for context-aware applications and because it can be easily transformed into the format used to represent layout in the Android platform. The reader should be aware that the choice for CAP3 is a pragmatic one — it is possible to use other notations instead of CAP3 and still be able to exploit critical pair analysis for exploring the design space.

Scalability. At the time of writing, we have validated the CAA approach on a small context model — 256 configurations of context variables where each configuration corresponds to 1 UML class diagram, 3 UML activity diagrams and 1 CAP3 user-interface model. The AGG tool could load models of that size flawlessly and running the critical pair analysis was still a matter of seconds. Nevertheless, since the theoretical number of contexts an application has to adapt to increases exponentially with the number of variables, and polynomially with the discrete values these variables can take it is at the moment unclear how far the approach would scale for realistic context models spanning thousands of configurations, each of them representing platform independent models with hundreds of diagrams.

Applicability and matching of rules. In this paper, we consider a transformation rule $r_{c,d}$ always applicable on the context-specific model M_c. This assumption could lead to failure in the entire reasoning to compute the context coverage. In practice however, the modeler will typically define the set of transformation rules $r_{c,d}$ by recording the modification he has to do on the model M_c to obtain the context-specific model M_d. This method is feasible thanks to the context-specific model M_{c_0} from which it is possible to start creating transformations. We could have chosen to incorporate a checking mechanism that would simply check this applicability constraint using AGG and therefore would allow us to remove this assumption, but we preferred not to complicate the context coverage computation. An implication of the applicability is the existence of a match for the rule $r_{c,d}$ in the context-specific model M_c. Another assumption that we made is that this match is unique. Once again, this is not restrictive in practice as the model elements modified by $r_{c,d}$ (i.e., the match) are uniquely identified inside M_c. It is therefore not possible to substitute in the match, one of these elements with another element of M_c.

Expressiveness of rules. AGG allows a greater expressiveness to specify graph transformation rules than the one we used in this article. In our feasibility study, we have not been confronted with a situation requiring the use of negative and positive application conditions. Moreover, we believe that these application conditions are rarely needed when specifying the impact of a context change. Nevertheless, as the critical pair analysis as well as the concurrent rule generation mechanism of AGG can be used on graph transformation rules in presence of positive and negative application conditions, the context coverage could still be computed on such extended transformation rules. For even more general (nested) conditions, AGG currently does not provide critical pair analysis support[15].

Transformation rule The complexity and effort needed to define transformation rules is quite subjective: it depends on the experience, skills and knowledgeability of the designer with the transformation language that is needed to specify these rules. Instead of using a graph transformation language, other model transformation languages could have been chosen such as ATL [18,19] or QVT [20]. These languages have the advantage of being more widespread and would therefore reduce the effort needed to define transformation rules. On the other hand, AGG's graph transformation language is, to our knowledge, the only formalism offering some ready to use implementation of critical pair analysis for model transformations.

The CAA model proposed in this article represents a necessary first step to help the designer during the model-driven specification of context-aware applications. In particular, it helped us to explore the context space of the mobile city guide and consequently increased our knowledge about the relations between the variables composing its context. As a side effect we have gained a better understanding of the conflicting context adaptations involved in the computation of the ambiguous context-specific models. This helped us substantially while developing the mobile city guide on the Android platform, as it allowed us to identify which functional behaviour was context-dependent.

8 Related work

Context-awareness introduces some specific difficulties in the development process. At the level of source code, some mechanisms and frameworks have been developed to assist the programmer. For example, Henricksen et al. [21] propose a complete framework to support the development of context-aware applications. Raw data from sensors is collected and aggregated to obtain meaningful situation variables that can be used by the developer directly in the source code. Gonzalez et. al [22] propose context-oriented programming support for mobile devices called Subjective-C that enable the use of runtime adaptation through layer activation.

The actual development of such new mechanisms and frameworks at the source code level is indicating that traditional languages are not well suited to express context-aware applications. In a similar vein, modeling languages need to be improved in order to express context-specific models and their adaptations in a more effective way. In the human-computer interaction community, the importance of applying model-driven engineering to design and model context-aware applications has been broadly recognized. The Cameleon framework [23] proposes to create a model of the application for each context in which the application can be used. The context itself is modeled separately and can be linked to a specific application model.

Pribeanu et al. [24] propose several alternatives for the specification of interactions that are partly context-aware. They conclude that an integrated task model with context-dependent subtrees that are specific for a certain context would be the best way to model user interaction with application in multiple contexts. The context-dependent subtrees share a common parent task and have choice operators specified between them. UsiXML [25, Chapter 3] makes this approach concrete through the use of three distinct models: a context model, a mapping model and the specific user interface model that contains context-aware parts. Clerckx et al. [26] build further on this and create a separate notation for this common parent task, the decision node. This node specifies the contexts of use in which the subtrees should be executed. This allows them to create a user interface that adapts to the active context at runtime by linking the tasks to concrete interaction objects.

All the approaches mentioned above are treating context as a set of different situations and model the user interface for each situation. This results in a solution that does not scale as the number of contexts increases. The Contextual ConcurTaskTrees notation [27] takes a different approach and specifies what should happen when a certain context change happens through the definition of different task types. The idea of treating context as an active entity was inspired by programmatic support for context-aware applications as offered by, e.g., the Context Toolkit [28]. This approach was also used in CUP 2.0 [29] to model context-aware user interfaces at different levels of abstraction.

Lohmann et al. [30] define an approach to generate context-aware web applications at runtime. They do this by specifying a conceptual model, containing a context model, a domain model, a context-relations model, and user interface models. These user interface models contain rules on how the system behaviour should be adapted to a specific context.

All the aforementioned solutions are restricted to a specific model. In contrast, our CAA model approach is independent of the modeling language used to specify the context-aware application, hence enabling the use of domain-specific languages.

In complement to the above approaches, some mechanisms have been developed to verify context-aware applications. For example, Fleurey et al. [31] define a model in which contexts are specified in terms of variable values and application configurations in terms of variants. An objective function is then used to automatically determine which variants compose the best configuration of the application for a specific context. In contrast to this model that focuses on the relation between context variables and configuration variants, our CAA model allows to identify the relation between context variables. These relations can be identified because the adaptations to context change are modeled as executable specifications (i.e. the transformation rules). Nevertheless, the model defined by Fleurey et al. is similar to the CAA model, since the authors reduce the inherent complexity of the context-aware applications using a context variable perspective.

Sama et al. [32] define a model and some algorithms in order to identify fault patterns in the context adaptation of context-aware applications (CAA). CAAs are typically supported by a context-aware middleware composed of a context manager and an adaptation manager. The context manager collects and maintains context information that can be queried by a CAA. Using context value changes provided by the context manager, the adaptation manager maintains, evaluates and applies a set of rules defining adaptive actions to take. This approach, based on an analysis of the adaptation rules, reveals nondeterministic adaptations as well as unreachable configurations. In contrast to the CAA model that focuses on the application logic, Sama et al. focus their verification on the adaptation logic.

9 Conclusion

In this article, we presented the Context-Aware Application model (CAA), in which a context-aware application is represented as a multi-dimensional design space with axes representing the variations that may occur for a given context variable, and where each point in the design space represents the expected behaviour of an application as a platform-independent model. By mapping this model to graphs and graph transformations, we exploit critical pair analysis to enumerate all reachable contexts, this way exploring the context design space. Moreover, for those context variables that are dependent on one another, the critical pair analysis also identifies sequencing of transformations resulting in conflicts, this way revealing contexts for which the resulting application model is ambiguous.

The approach has been validated on a so-called specification exemplar: a self-contained, informal description of a problem in some application domain proposed as unique input for the specification process [3]. The exemplar is that of a mobile city guide, which is sufficiently small to allow for condensed representation and manual inspection, yet is sufficiently realistic to be representative for context-aware applications developed by a small team of software engineers. On this example, we illustrated that (a) exploration of the design space helps to reveal omissions in the model transformations; (b) conflicts during the critical pair analysis reveal ambiguities in the design space that might go unnoticed otherwise.

Of course, good designers will apply the separation of concerns principle, hence for realistic applications the variables composing the contexts will often be independent of one another. Nevertheless, the very nature of context-aware applications implies that some variables will have an effect on one another. With our Context-Aware Application model (CAA), we can reveal such dependencies at the end of the requirements analysis or design phase (i.e., once a platform-independent model has been created), thus early in the life-cycle of application development for mobile devices.

Acknowledgements

This work is supported by the FWO project Transforming human interface designs via model driven engineering (G. 0296.08), the MoVES project financed by the Interuniversity Attraction Poles Programme - Belgian State Belgian Science Policy, and the Action de Recherche Concertée Model-Driven Software Evolution financed by the Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique, Belgique. We express our gratitude to H.

Vangheluwe, T. Molderez, B. Meyers, M. Goeminne and O. Gauwin for reviewing earlier drafts of this article.

References

1. Reiko Heckel, Jochen Küster, and Gabriele Taentzer. Confluence of typed attributed graph transformation systems. In Andrea Corradini, Hartmut Ehrig, Hans Kreowski, and Grzegorz Rozenberg, editors, *Graph Transformation*, volume 2505 of *Lecture Notes in Computer Science*, pages 161–176. Springer Berlin Heidelberg, 2002.
2. Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A data-oriented survey of context models. *SIGMOD Rec.*, 36:19–26, December 2007.
3. Martin S. Feather, Stephen Fickas, Anthony Finkelstein, and Axel Van Lamsweerde. Requirements and specification exemplars. *Automated Software Engineering*, 4:419–438, October 1997.
4. Fabio Paternò and Carmen Santoro. One model, many interfaces. In Christophe Kolski and Jean Vanderdonckt, editors, *Proceedings of CADUI 2002, the 4th International Conference on Computer-Aided Design of User Interfaces*, pages 143–154. Kluwer Academic Press, 2002.
5. Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39:25–31, 2006.
6. Jan Van den Bergh, Kris Luyten, and Karin Coninx. Cap3: Context-sensitive abstract user interface specification. In *Proc. 3rd ACM SIGCHI symposium on Engineering interactive computing systems (EICS 2011)*, pages 31–40. ACM Press, June 2011.
7. Bill Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, 1994.
8. Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koen De Bosschere. Towards an extensible context ontology for ambient intelligence. In Panos Markopoulos, Berry Eggen, Emile Aarts, and James L. Crowley, editors, *Ambient Intelligence*, volume 3295 of *Lecture Notes in Computer Science*, pages 148–159. Springer, 2004.
9. A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, 2000.
10. Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In Hans-Werner Gellersen, editor, *HUC*, volume 1707 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 1999.
11. Joëlle Coutaz and Gaëtan Rey. Foundations for a theory of contextors. In Christophe Kolski and Jean Vanderdonckt, editors, *Proc. 4th International Conference on Computer-Aided Design of User Interfaces (CADUI 2002)*, pages 13–34. Kluwer, 2002.
12. P. J. Brown, J. D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 4(5):58–64, 1997.
13. Gabriele Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Proc. AGTIVE 2003*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer-Verlag, 2004.
14. Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundam. Inform.*, 26(3/4):287–313, 1996.
15. Olga Runge, Claudia Ermel, and Gabriele Taentzer. AGG 2.0 – New Features for Specifying and Analyzing Algebraic Graph Transformations. In A. Schürr and D. Várro, editors, *Fourth International Symposium of Application of Graph Transformation with Industrial Relevance (AGTIVE'11)*, LNCS. Springer, 2011. To Appear.
16. Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. Confluence of typed attributed graph transformation systems. In Corradini et al. [17], pages 161–176.
17. Andrea Corradini, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Proc. 1st Int'l Conf. Graph Transformation (ICGT 2002)*, volume 2505 of *Lecture Notes in Computer Science*. Springer, 2002.
18. Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. ATL: a QVT-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 719–720, New York, NY, USA, 2006. ACM.
19. Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. In Jean-Michel Bruel, editor, *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138. Springer Berlin / Heidelberg, 2006.
20. I. Kurtev. State of the art of QVT: a model transformation language standard. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 5088 of *Lecture Notes in Computer Science*, pages 377–393, Berlin, October 2008. Springer Verlag.
21. K Henricksen and J Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64, February 2006.
22. Sebastian Gonzalez, Nicolas Cardozo, Kim Mens, Alfredo Cadiz, Jean-Christophe Libbrecht, and Julien Goffaux. Subjective-C. In Brian Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering*, volume 6563 of *Lecture Notes in Computer Science*, pages 246–265. Springer Berlin / Heidelberg, 2011.
23. Gaelle Calvary, Joelle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
24. Costin Pribeanu, Quentin Limbourg, and Jean Vanderdonckt. Task modelling for context-sensitive user interfaces. In Chris Johnson, editor, *DSV-IS*, volume 2220 of *Lecture Notes in Computer Science*, pages 49–68. Springer, 2001.
25. Quentin Limbourg. *Multi-Path Development of User Interfaces*. PhD thesis, Université catholique de Louvain, 2004.
26. Tim Clerckx, Kris Luyten, and Karin Coninx. Generating context-sensitive multiple device interfaces from design. In Robert J. K. Jacob, Quentin Limbourg, and Jean

Vanderdonckt, editors, *CADUI*, pages 281–294. Kluwer, 2004.
27. Jan Van den Bergh and Karin Coninx. Contextual concurtasktrees: Integrating dynamic contexts in task based design. In *PerCom Workshops*, pages 13–17. IEEE Computer Society, 2004.
28. Anind K. Dey and Jennifer Mankoff. Designing mediation for context-aware applications. *ACM Trans. Comput.-Hum. Interact.*, 12(1):53–80, 2005.
29. Jan Van den Bergh and Karin Coninx. Cup 2.0: High-level modeling of context-sensitive interactive applications. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2006.
30. Steffen Lohmann, J. Wolfgang Kaltz, and Jürgen Ziegler. Model-driven dynamic generation of context-adaptive web user interfaces. In Thomas Kühne, editor, *MoDELS Workshops*, volume 4364 of *Lecture Notes in Computer Science*, pages 116–125. Springer, 2006.
31. Franck Fleurey and Arnor Solberg. A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In Andy Schrr and Bran Selic, editors, *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pages 606–621. Springer Berlin / Heidelberg, 2009.
32. Michele Sama, Sebastian Elbaum, Franco Raimondi, David S. Rosenblum, and Zhimin Wang. Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Trans. Softw. Eng.*, 36:644–661, September 2010.