Formal model for user-centred adaptive mobile devices

Tommaso Di Noia¹, Eugenio Di Sciascio¹, Francesco Maria Donini², Marina Mongiello¹, Francesco Nocera¹

¹Dipartimento di Ingegneria Elettrica e dell’Informazione, Politecnico di Bari, Bari, Italy
²DISUCOM, Università della Tuscia, Via S. Carlo 32, 01100 Viterbo, Italy

Abstract: The authors present an approach to complex adaptive mobile applications modelling and implementation, able to dynamically change according to changed behavioural properties, state and/or context variables and user's preference. To this aim, they design a metamodel made up of an action repository (AR) to store triples composed by logical propositions to define criteria for selecting actions to be executed. An algorithm has been devised to retrieve a set of possible actions – apps, services or components – to be executed from the AR. The selection of a single action to be executed depends on a user's model. The metamodel validation is carried out through an instantiation in two real scenarios: a proximity environment and a smartphone.

1 Introduction and motivation

Interest in mobile applications has widely increased in the past years. Execution of these apps often depends on gestures, sensors and location data and allows adaptive behaviours. A variety of techniques and issues related to modelling, implementation and execution of such applications and to the well-known self-adaptive systems is summarised in [1]. Adaptation gained increasing attention to classify issues of self-management [2], and of architectural decision making at design and runtime [3]. Several reference models have been also proposed [4]. Anyway runtime adaptation of software components is still a challenging problem [4, 5].

Usually, adaptation actions are employed at an architectural level to add components, apps and services in a composite model.

<table>
<thead>
<tr>
<th>RQ and RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>RQ1</td>
</tr>
<tr>
<td>RQ2</td>
</tr>
<tr>
<td>RQ3</td>
</tr>
<tr>
<td>RQ4</td>
</tr>
</tbody>
</table>

At behavioural level, they enable dynamic changes in an app's behaviour, deployment and execution.

This paper was carried out starting from key research concepts within the self-adaptive systems and focused on some research questions (RQs) about the same class of applications as far as modelling and implementation are concerned. First of all, we identified as key concepts the following factors emerging as the more responsible of changes at runtime: (i) changes in state–context/environment, (ii) changes in requirements and (iii) user's habits and needs. The key concepts were analysed by highlighting their role in runtime design and implementation. Table 1 presents the RQs that will be addressed and the research process (RP) that has been taken.

We propose an approach to comprise relevant aspects of adaptation: knowledge about environment, context and user's habits and information extracted from external sources and sensors.

A formal metamodel that uses an action repository (AR) with stored triples of actions and related pre- and post-conditions about state and context is proposed to model properties of a device.

The metamodel is instantiated in real scenarios, by contextualising the main elements, thus obtaining an adaptive mobile software.

The rest of this paper is organised as follows. Section 2 introduces the proposed approach. Section 3 instantiates the approach in two real scenarios: i.e. a proximity environment application and an application to dynamically modify the home screen of a smartphone. Section 4 validates the approach on empirical setting experiments. Section 5 discusses advantages of the proposed framework according to RQs. Section 6 compares our approach with existing state-of-the-art works. The last section concludes this paper and outlines future work.

2 Approach

2.1 Action repository

We propose a formal approach to build runtime mobile applications based on a metamodel to support runtime modelling of adaptive applications. Modelling depends on behavioural/contextual changes and observable properties of the user's habits and profiles.

The metamodel is made up of a control level where data extracted from sensors are transmitted to effectors. Sensors and effectors are interpreted in the sense of the definition of the monitor-analyse-plan-execute-knowledge (MAPE-K) model [2]. Actions and tasks...
to be performed are derived from high-level properties, pre-
conditions about the state and context.

Knowledge about the adaptive software and its environment is
modelled using requirements such as: if the user is near a
restaurant, it is lunchtime, usually she eats Japanese food, then she
will be informed about a Japanese restaurant nearby.

The metamodel allows one to express runtime adaptation at
behavioural level: apps, services or components maybe runtime
loaded, deployed and executed.

Definition 1: (States): Given a set of state variables
\( SVars = \{v_1, \ldots, v_n\} \), and a set of corresponding domain values
\( \{V_1, \ldots, V_n\} \), a state is an assignment \( s: v_i \mapsto d \in V_i \) that for each
\( i = 1, \ldots, n \) maps each state variable \( v_i \) to its current value
\( s(v_i) \in V_i \).

An Action can change a state \( s \) to a state \( s' \), by changing the value of
any number of state variables.

For example, a state variable \( gps \) of a mobile device may record in
a Boolean value whether the global positioning system (GPS) is
ON or OFF, another may record whether or not the icon of an app
appears in the home screen, another may contain the present
brightness of the screen etc. Actions model the adaptations of the
device, turn OFF the GPS and reduce the brightness of the screen
(see below). State variables not affected by an action keep their
values.

Contexts are defined in the same way as states, but with the crucial
difference that context variables cannot be changed by actions; the
device environment changes them exogenously.

Definition 2: (Contexts): Given a set of Context Variables
\( CVars = \{w_1, \ldots, w_m\} \), and a set of corresponding domain values
\( \{W_1, \ldots, W_m\} \), a Context \( c \) is an assignment \( c: w_i \mapsto d \in W_i \) that for each
\( i = 1, \ldots, m \) maps each context variable \( w_i \) to its current value
\( c(w_i) \in W_i \).

No action can change a context variable.

Intuitively, contexts model every condition of the device adaptation
can have no effect on, instance, signal strength, battery level,
geographical location, nearest street address, date/time etc.
Actions are defined by their pre-conditions (conditions of
application that are verified before the action can start) and post-
conditions (which change the values of one or more state variables).

Definition 3: (Pre- and post-conditions): We call \( \mathcal{P} \) the language
containing all possible pre-conditions. For every action \( a \):

1. the pre-condition of \( a \) is a formula \( P_a \in \mathcal{P} \) which is a Boolean
   combination of:
   - comparisons of state variables \( (v_i, d) \), where \( v_i \in SVars, \)
     \( d \in D, \) and \( \in \{=, \neq, <, >, \geq, \leq\} \) is a comparison
     operator;
   - comparisons of context variables \( (c_i, d) \), where \( c_i \in CVars, \)
     \( d \in W, \) and \( op \) is as above.

2. the post-condition of \( a \) is a formula \( Q_a \) which is a conjunction
   of assignments \( (v_i = d) \), where \( v_i \in SVars, d \in D, \)

We require that \( Q_a = \neg P_a \) so that once an action has been
performed, its effects prevent the immediate re-application of the
action.

An action \( a \) can be performed when the state \( s \) of the device in the
context \( c \) makes its pre-condition true. After the action is
performed, we denote the resulting state as \( s' = s Q_a \), meaning that
variables not in \( Q_a \) keep their value, while every variable in \( Q_a \)
changes accordingly.

For example, a pre-condition could be \( [(gps = true) \lor (brightness \geq 0.7)] \land (battery level \leq 1v) \) –
denoting some high-consumption state in the context of a low
battery level – while a post-condition could be
\( (gps = false) \land (brightness = 0.2) \) – which changes the state to a
low-consumption one. Intuitively, pre-conditions model a Boolean
check on sensors, while post-conditions model the activation of
effectors. Clearly, while sensors can be checked for a wide range of
possible values at once with operators such as \( <, \neq, \geq \), effectors
are modelled as deterministic actions with the only operator \( = \),
setting a definite value for a state variable.

We stress that pre-conditions are not triggers, that is, the
satisfaction of \( P_a \) does not force \( a \) to be necessarily performed. In
fact, when several such pre-conditions are true at the same
moment, the adaptation will choose the most preferable action
according to the user model, which will be discussed later on.

Definition 4: (AR): An AR is a set of triples of the form \( (a, P_a, Q_a) \)
where \( a \) is a (unique) name of an action, \( P_a \) is its pre-condition
and \( Q_a \) is its post-condition.

The AR is the core knowledge of our knowledge-based runtime
adaptive system, which comprises other parts as listed below.

Definition 5: [Adaptive architectural metamodel (AAMM)]: An
AAMM is a tuple \( \text{AAMM} = (S, \text{AR}, \text{find}, U, E) \) where \( S \) and \( E \)
are, respectively, the sensors and effectors, \( \text{AR} \) is an
algorithm that finds a set of actions that might be applied in a given
state and context, and \( U \) is the user's model.

Fig. 1 shows the metamodel of the proposed approach.

2.2 Personalised action selection

In most cases, a perfect match between the actual state and context
and the ones required in the pre-condition is not to be expected.
Given a state–context \( P \) we need to evaluate if it is ‘similar
enough’ to the one specified in the pre-condition \( P_a \) of an action \( a \).

Given an AR, we want to execute the action whose pre-conditions
are more similar to \( P \). Moreover, it would be advisable that the
selection procedure behaves in a personalised way. That is, given
\( P \), the selection of the action to be executed may change depending
on the user. Hence, when evaluating the pre-condition \( P_a \) of an
action \( a \), most comparisons in \( P_a \) are evaluated as fuzzy conditions
in fuzzy logic [6]. Indeed, due to their inner nature, it may result
hard to evaluate a comparison of state variables in a Boolean
setting. In such a setting, the comparison \( (time = 12:30) \) would
be infrequent considered to have truth value 1 only in the case in
which the device time is exactly 12:30, even if \( around 12:30, \)
say 12:25, the comparison still yields a degree of truth which is nearly
\( 1 \) – say, 0.8. Fuzzy logics may surely help in modelling such
graded values of truth. They are based on the notion of fuzzy sets
which are defined, simply put, as functions \( f: D \rightarrow [0, 1] \)
assigning a grade (value) of truth \( f(d) \in [0, 1] \) to a certain value
\( d \in D \). With reference to the previous example, \( D \) is the domain
of time and \( d \) is a possible hour. In the literature, the most common
and used membership functions are depicted in Fig. 2.

The choice of the right function to be associated to a fuzzy set
depends on \( D \). Going back to our example, in some scenarios, it
may result quite natural to select a triangular function with
\( y = 12:30 \).

Given a pre-condition \( P_a \in \mathcal{P} \) represented as a Boolean
combination of state–context variables comparisons we now define
how to evaluate its truth value.

Definition 6: (Interpretation): An interpretation \( I \) for \( \mathcal{P} \) is a
function \( \cdot \cdot \cdot \) that maps each comparison of state variables
\( (v_i, op, d) \) occurring in \( P_a \) to a truth value \( (v_i, op, d)^I = f(d) \) and, analogously,
each comparison of context variables \( (c_i, op, d) \) to a truth value
\( (c_i, op, d)^I = f(d) \) with \( f \) being a fuzzy membership function. Given
\( P_a, P_a \in \mathcal{P} \) we recursively define the interpretation of a formula
as:
\[ (\neg P_a)^I = 1 - P_a^I. \]
\( (P_a \land P_a') = \min(P_a', P_a) \).

\( (P_a \lor P_a') = \max(P_a', P_a) \).

With reference to the above definition, given a set of pre-conditions \( \mathcal{P} \) we can compute a total order among its elements by means of the interpretation functions. Indeed, given \( P_a, P_b \in \mathcal{P} \) we can always evaluate whether \( P_a \geq P_b \) or \( P_b \geq P_a \). Actually, an order among pre-conditions can be easily reverted to a ranking among the corresponding actions. In other words, if \( P_a \geq P_b \) we assume \( a \) is more likely to be executed than \( b \).

**Definition 7:** (Executable action): Let \( \mathcal{AR} = \{ (a, P_a, Q_a), (b, P_b, Q_b), \ldots \} \) be an AR and \( t \in (0, 1) \) be a threshold value. We say \( a \) is an executable action iff both there is no action \( b \) such that \( P_b > P_a \) and \( P_a \geq t \).

Since we deal with a total order, we may have more than one executable action \( a, a', a'' \), etc. Indeed, it may occur that \( P_a = P_a' = P_a'' = \ldots \). We see that as we do not have any order among \( a, a', a'', \ldots \) we may execute any of them randomly. The reason why we introduce the threshold \( t \) is to avoid situations where the executed action has a low truth value (which corresponds to a high untruth value). Given a state–context \( \hat{P} \), in case there is no executable action, the system does nothing until the next change in \( \hat{P} \).

At the end of Section 2.1, we argued that, given a state–context \( \hat{P} \), the computation of the executable action should adapt to the user. With respect to the model presented in this section, we can encode user preferences within the fuzzy membership functions. In fact, looking at Fig. 2, we see they are defined in terms of a set of parameters \( x, y, z, t \). By changing these values, we modify the shape of the functions. Let us go back to our example (\( \text{time} = 12:30 \)) and suppose we define the fuzzy set associated to time by means of a triangular function with \( y = 12:30 \). We may distinguish between an "always on time" user and a "more relaxed" one by setting, for instance, in the former case \( x = 12:25 \) and \( z = 12:35 \) while in the latter case \( x = 12:00 \) and \( z = 13:00 \). Hence, depending on the user, the truth value associated to \( P_a \) may change and then the possible selection of \( a \) as executable action. It is noteworthy that \( x, y, z, t \) can be either set manually or be automatically learned by collecting information about the user's behaviour.

The history of the user's behaviour is stored in a repository through the values of the context and state variables describing the actions generally performed by the user and her preferences. A triangular function elicits the variables values to describe the user's behaviour: for example, the history of the places she usually visits (the state variable is position) or the times he usually visits that place (the state variable is time). The slopes of the functions (values \( x, y, z, t \)) are tuned by the user's model.
places. The supervisor will choose among the pool of actions identified in the AR the action that verifies the constraint with the threshold \( r \) with respect to the triangular function of variable in the pre-condition. In case of multiple properties in the pre-condition expressed with fuzzy variables the minimum or maximum operator as specified in the corresponding fuzzy interpretation.

To make the formulas fuzzy, we can express the pre-conditions using intervals, that is, the pre-condition is not true for only one value of the formula, but for the values in these ranges. For example, the choice of points to be displayed on a map will not be shown only for an exact value of the radius of the area, but depending on user habits, this value can be included in an interval. For example, \( y \) is the current position, while the interval \([x, z]\) defines the length of the circumference of the diameter to be displayed (centered at \( y \)). In this case, the triple is: Pre-condition: \((\text{gps} = \text{true})\) and \((\text{radius} > x)\) and \((\text{radius} < z)\) and \((\text{location} = y)\) and \((\text{now} > 24:20)\) and \((\text{Time} < 12:40)\) and \((\text{number of high interest points})\); Action: displays restaurants in the area of interest; and Post-condition: at least one point is displayed.

### 3 Instantiation of the model

#### 3.1 Proximity environment

The metamodel proposed in Section 2 was instantiated in the domain of proximity environments. Nowadays, proximity is being considered as an added value of most applications, especially in social environments.

This phenomenon is widely observed spreading in the social sphere, thanks to the enormous spread of smartphones with GPS. Using the GPS connection, users of social networks can inform in every moment in which attraction or local shop they are. When the user is in a certain place, she launches the application from her smartphone, after the connection with the GPS has been made, she looks at the list of places in the neighbourhood and the so-called check-in to tell users on her list of contacts where she exactly is.

In the field of proximity marketing, we developed adaptable proximity marketing tool (AProM) a mobile app to enable spreading advertisements to end-users according to their needs. The application starts and shows the map. Depending on the context (location, time), points can be displayed in different categories of interest. For example, restaurants at lunch time, hotels in the evening etc. are displayed. The time ranges are decided depending on user's habits (the time usually she has lunch, goes to sleep etc.). Besides, the area taken into account on the map and the points shown inside have a radius that varies according to the places already visited by the user.

### 3.2 AAMM instantiation

The instantiated metamodel exploits the goals, the objective that the user expects. Table 2 summarises how the elements in the tuple AAMM are instantiated.

Effectors are instantiated as actions extracted from the AR based on the ‘find’ selection algorithm; they are mainly implementation of the operations on the map (simple display or recommendation).

The repository is conceptually separated and can virtually be always on the device. Table 3 presents an example of the instantiated tuples of actions, pre-condition and post-condition of the AR.

State and context variables are sensors in the external environments, context variables available on the user mobile device and events extracted from the sensors. The architectural model derived from the instantiation of the metamodel defined in Section 3.2 is shown in Fig. 3.

AProM was developed on the Android platform and requires, as minimum supported version, Android Ice Cream Sandwich 4.0.3 [application programming interface (API) Level 15]. GPS sensor, Internet connection. The device on which the application is deployed is an LG Nexus 5 Android updated to version 5.1, which meets all the required specifications. The development environment (DE) is Android Studio, an open source tool for developing Android applications. We chose the Android platform to be able to develop the app without cost and to take advantage of some features of the operating system. AProM presents the typical structure of an application for Android; it is written in Java and extensible markup language, respectively, for dynamic and static parts.

Fig. 4 shows a screenshot of AProM: the area where the user is localised is the circle in which points of interest are shown belonging to the category of food; the right-hand side of this figure shows the list of categories that can be accessed by user for manual search or for inserting advertisements.
3.3 ‘SmartSmartphone’

The second instantiation refers to the implementation of a mobile application that manages dynamic characteristics of the home screen of a smartphone, depending on the user's habits, by the position detected by GPS in which the user is currently located using the device, by the external context and by user's current behaviour. The app, SmartSmartphone, monitors the user's context and shows a list of recommended applications depending on it.

3.4 AAMM instantiation

The instantiated metamodel exploits the goals, the objective that the user expects. Table 4 summarises how the elements in the tuple AAMM are instantiated.

The repository is conceptually separated, and can virtually be always on the device. Table 5 presents example of the instantiated tuples of actions, pre-condition and post-condition of the AR.

State and context variables are the apps installed on the device and time/position, respectively.

The architectural model derived from the instantiation of the metamodel defined in Section 3.2 is shown in Fig. 5.

To develop this application, we chose Android as guest operating system. We chose a native approach to model the architecture, which allows to exploit the system, but also the entire ecosystem associated with it. In fact, thanks to the choice of the Android operating system, it was possible to make use of Google Play Services, a Google proprietary system that runs in the background on devices and that provides developers a set of APIs.
that use on the one hand Google services and other hardware on the smartphone in a way transparent, to the developer.

The programming language used is Java. The integrated DE used is Android Studio (version 2.0). This tool is released directly from Google and allows excellent integration with the Android ecosystem.

Fig. 6 shows three examples of the smartphone home screens related to three different contexts: home, travel and work.

In the home context, apps providing functionalities related to the home living are directly available to the user such as Skype, Facebook; in the travel contexts apps such as Tripadvisor, Google Maps are more frequently used by the user and hence are available; in the work context the user needs apps such as Dropbox, email, Skype and Chrome.

4 Experiment and validation
In this section, we describe a controlled experiment designed to perform a validation of the proposed approach. We aim to prove how the contribution of the proposed framework is relevant.

Two teams of students were assembled, each one composed of three second year graduate students. All students were trained on software design, architectural modelling, mobile and adaptive applications implementation.

The teams had to model and develop two adaptive mobile applications according to scenarios described in Section 3 using only their own experience. Afterward the same teams had to solve the same problem supported by the metamodel, by mapping the main elements of the metamodel to the main elements of an architecture.

The solution to each problem was provided as an adaptive mobile application. A set of non-functional requirements was formerly identified based on general purpose goals of an adaptive software. From the end-user perspective, we fixed usability,
functionality and correctness; from the developer/designer perspective, we considered maintainability, correctness, data access and adaptability. Anyway, the two solutions are not expected to be the same, because of details that derive from creativity, experience and from reasoning approaches. Hence, the measure of the goodness of the method was very complex to quantify.

We performed two different experiments: in the first one we compared the solutions provided by the two teams supported by the metamodel with these obtained using only the human experience. In the second experiment, real users evaluated their degree of satisfaction about adaptive functionalities of the applications obtained with/without support of the metamodel. Besides, we aimed to evaluate the impact of user's behaviour modelling on the system operation.

We measured the advantage of using the metamodel according to designer/developers opinion in terms of: (a) level of difficulty in development for the developers; (b) assessment of requirements satisfaction – in percentage; and (c) efficiency: time elapsed between the starting point and the solution.

Hence, a questionnaire was administered to each member of each team to compute the degree of satisfaction of developers for the metamodel supported development procedure that stands at a level of 96%. Table 6 summarises results for the first experiment.

The test was conducted as a usability test by considering developers/designers as final users.

In the second experiment, a set of end-users was provided with the two applications developed without support of the metamodel, for a fixed period of time. Afterwards the same experimenters were provided with the same application obtained by applying the metamodel during its development. At the end of the time period, a qualitative evaluation of the applications was required to the experimenters. Six students were chosen for testing the application. To conduct this experiment, we used a slightly modified kind of think-aloud usability test [7], and administered two questionnaires. The first questionnaire was about the degree of satisfaction expressed through a rating scale ranging from 1, strongly unsatisfied to 10, strongly satisfied.

The degree of satisfaction for metamodel-based applications stands at a level of 97%.

The second questionnaire was made on a system usability scale-like schema. The questions were not statements about features related to expected non-functional requirements, and the answers had to be expressed through a rating scale from 1, strongly disagree to 5, strongly agree with the proposed statements.

Table 7 summarises results.

5 Discussion

We describe hereby how the four RQs defined in Section 1 have been addressed throughout the work.

To take into account the first two key concepts – changes in state–context/environment and changes in requirements – we posed the first two RQs in Table 1. To achieve a complete answer to these questions, we studied state of the art concerning main challenges in modelling and implementing self-adaptive systems and selected two main categories of related work: ‘formal methods in modelling adaptive architecture’ and ‘composition and interaction between applications in adaptive software’.

By considering the key concept 3 – user's habits and needs – we posed RQ3 in Table 1. To answer this question, we reviewed the state of the art for user-centred approaches in modelling mobile and adaptive applications. The answer to RQ3 was derived from studying the previously cited related work and devising a formal method to relate the first two key concepts to the third one – state–context/environment changes, requirement changes to user's habits and needs. This issue was very challenging since none of the approach in the state of the art related the key concepts in a single model. This paper allowed us to build the theoretical framework of our approach that is the answer to RQ4. We answered this RQ by defining a metamodel, a tuple that comprised all the elements and the relative relations. Each element of the metamodel can be instantiated in real application scenarios thus realising an adaptive architecture on the fly.
The approach intrinsically satisfies some relevant general purpose requirements of an adaptive application. One of the granted requirement is correctness that is wired in the pre/post-condition pairs whose definition is submitted to the constraint that once an action is performed, its effects prevent the immediate re-application of the action. Also, an action can be performed when the state of the device in the context satisfies its pre-condition. A second requirement is functionality that is elicited by the third element of the tuple, that is, the action to be executed when the state and context requirements formalised in the pre-condition/post-condition are satisfied. The centralised control and management of the framework ensures maintainability, in fact the resulting application when instantiating the metamodel will have an application manager to coordinate the remaining elements of the tuple for the event-based flow of action. As external qualities we identified a high level of usability due to the strong weight the metamodel gives to the user profiling.

6 Related work

Modelling and analysis of self-adaptive systems has gained increasing attention in the past years. Several approaches intend to face the main challenges of modelling such systems; hence, there is a wealth of the literature about studying, modelling and implementing this category of systems.

This paper was carried out starting from key research concepts within the self-adaptive systems and focused on some RQs about the same class of applications as far as modelling and be concerned. We identified three main categories of state-of-the-art approaches. With respect to these categories we studied shortages or possible ideas for improvement.

6.1 Formal methods in modelling adaptive architecture

Formal approaches remain the more powerful ones in order to ensure correctness and guarantee of quality properties. A complete study is proposed in [8], while a survey of architectural modelling in self-management is proposed in [4]. Formal methods have been used in [9] to model MAPE-K loops through a conceptual and methodological framework based on abstract state machines. Riccobene and Scandurra [10] define a lightweight formal framework to express adaptive behaviour of service components able to monitor and react to environmental changes (context-awareness) and to internal changes (self-awareness). Anyway these approaches are defined at abstract and conceptual level.

In the other approaches, that provide an architectural dimension are in [11]. The authors propose a software architecture-based approach to perform code synthesis as an assembly of actual components which respect the architectural component behaviour. The architectural framework Rainbow [12] uses external mechanisms and a model to carry out adaptation actions at explicit customisation points. The Genie approach [13] manages structural variability of adaptive systems at architectural level and do not provide a way to guarantee desired properties of the systems after each adaptation execution. With respect to the previously described methods, we proposed a framework that is applicable at different levels of abstractions, both conceptual and architectural. A similar advantage is found in [14]. The approach uses attributed graph grammars and define consistency and operational properties that are maintained despite adaptation. With respect to this graph-based adaptation logic, we introduce a semantic elicitation of knowledge about environment, user's habit and behaviours.

Differently from all the previously described methods, in our framework we comprise both architectural and conceptual ideas together with user's behaviour and a semantic approach.

6.2 Modelling of user's behaviour and requirements

User-centric approach for adaptive applications was proposed in [15] to model adaptation for future Internet application. Mobile user-aware root planner is proposed in [16]. The method collects everyday locations elicited by the user's usual travel patterns. A data-oriented, context-aware architecture is proposed in [17]. The context is validated at runtime. With respect to (w.r.t) this model we introduce state variables to model changes in the system or in the device.

Besides modelling of user's properties in these approaches is related to a specific context or environment or behaviour, while we model user's behaviour that is general purpose. The meta information is split into various categories of context-aware information modelled in the state and context variables.

Context-awareness modelling techniques are summarised in [18]. We point out that none of the existing techniques mixes up context, user's behaviour and requirements.

A more recent stream of research is focusing on leveraging the new sources of information becoming available through ubiquity of systems [19]. Our approach is compatible with this idea; anyway, we overcome a user's perspective, by considering a formal approach based on pre- and post-conditions.

To specify requirements, great interest is focused on goal modelling [20].

Among the goal modelling languages, knowledge acquisition in automated specification (KAOS) supports a linear temporal logic (LTL)-based formalism and goals and requirements can be specified by pre- and post-conditions [21]. Instead non-functional requirements maybe represented as soft goals or probabilistic patterns [22]. Fuzzy expression of requirements is adopted in [23]. Modelling and requirement-based adaptation has been proposed in [24]. Compared with these approaches our method takes into account variation in requirements and variables that may influence the adaptation logic of application. Zhang et al. [25] proposed an approach to monitoring non-functional requirements.

Adaptation is based only on requirements. Instead we use fuzzy properties to model user's behaviour in conjunction with requirements specifications.

6.3 Compositional approaches in adaptive applications

Models of service choreography and composition are proposed in [26]. The deployed application is tailored w.r.t. the context at binding time. Composition of self-adaptive systems for dependability is proposed by Cubo et al. [27]. Compositional adaptation based on technological dimension is in [3] without a semantic modelling and reasoning. Semantic approaches are proposed in [28] to model self-adaptive architectural model in internet of things (IoT) and [29] for runtime verification. We focused our model on the tuples of an AR where pre-conditions and post-conditions are linked to the actions to be performed, so we guarantee the adaptability also including the tuples to perform the actions. If more actions are selected, the user's behaviour discriminates on the actions to be performed.

Summarising, we introduced a formal framework that is innovative with respect to existing approaches since it merges all the relevant changes in state or context, changes in requirements, user's behaviour and user's habit and preferences, formalised in a fuzzy logical perspective.

Moreover, it wires the correctness requirement in the structure of the metamodel by defining the terms pre-condition, post-condition and action.

7 Conclusion and future work

In the future Internet era, the way software will be produced and used will depend on the new challenges deriving from the huge number of software apps and services that can be composed to build new applications. To face the problem of dynamic architectural modelling and of runtime composition of distributed complex applications, we proposed a semantic approach to define a metamodel for taking into account relevant aspects adaptation: context, user's habits and profiles, information detached from external sources and sensors. The use of a knowledge-based approach allows modelling and reasoning on complex adaptive software architecture according to changed behavioural properties or context variables. RQs based analysis of the state of the art demonstrates the improvements of the metamodel w.r.t. existing approaches. A controlled experiment was conducted with designers.
and users. Experimental results proved a high level of satisfaction for designers and developers that used the metamodel, and a high level of satisfaction for end-users that used the application in real environments.

We are currently working to strengthen the theoretical framework by adapting new reasoning services with temporal interaction among software components. We are also performing extensive experiments on a structured benchmark.

8 Acknowledgments

The authors acknowledge partial support of PON03PE 00136 1 Digital Services Ecosystem: DSE. The authors acknowledge management of teams for implementations and students of Angelo Parchitelli and Massimo Sciancalepore for coordination.

9 References


