ACon: A Learning-Based Approach to Deal with Uncertainty in Contextual Requirements at Runtime

Alessia Knauss, Daniela Damian, Xavier Franch[a], Angela Rook, Hausi A. Müller, Alex Thomo

Department of Computer Science, University of Victoria, Victoria BC, Canada
{alessiak, arook}@uvic.ca, {danielad, hausi, thomo}@cs.uvic.ca, franch@essi.upc.edu

[a]*Software Engineering for Information, Systems research group (GESSI)*
Universitat Politècnica de Catalunya (UPC), Barcelona, Catalunya, Spain

Abstract

Context: Runtime uncertainty such as unpredictable operational environment and failure of sensors that gather environmental data is a well-known challenge for adaptive systems.
Objective: To execute requirements that depend on context correctly, the system needs up-to-date knowledge about the context relevant to such requirements. Techniques to cope with uncertainty in contextual requirements are currently underrepresented. In this paper we present ACon (Adaptation of Contextual requirements), a data-mining approach to deal with runtime uncertainty affecting contextual requirements.
Method: ACon uses feedback loops to maintain up-to-date knowledge about contextual requirements based on current context information in which contextual requirements are valid at runtime. Upon detecting that contextual requirements are affected by runtime uncertainty, ACon analyzes and mines contextual data, to (re-)operationalize context and therefore update the information about contextual requirements.
Results: We evaluate ACon in an empirical study of an activity scheduling system used by a crew of 4 rowers in a wild and unpredictable environment using a complex monitoring infrastructure. Our study focused on evaluating the data mining part of ACon and analyzed the sensor data collected onboard from 46 sensors and 90,748 measurements per sensor.
Conclusion: ACon is an important step in dealing with uncertainty affecting contextual requirements at runtime while considering end-user interaction. ACon supports systems in analyzing the environment to adapt contextual requirements and complements existing requirements monitoring approaches by keeping the requirements monitoring specification up-to-date. Consequently, it avoids manual analysis that is usually costly in today's complex system environments.

Keywords: requirements engineering, self-adaptive systems, contextual requirements, operationalization, machine learning

1. Introduction

Self-adaptive systems (also referred to as dynamically adaptive systems [1]) are able to adjust their behaviour in response to changes in their environment and the system itself [2]. Their operating environment includes end-user input, external hardware devices, sensors, and program instrumentation [3].

While self-adaptivity promises to decrease the cost of handling the complexity of software systems at runtime [3], it challenges current software engineering practices, particularly the activities of requirements definition and satisfaction for systems that operate in uncertain environments [4]. Examples of such systems include software systems for smart cities that are interacting with thousands of individuals in a highly dynamic environment [5, 6], and intelligent vehicle systems that have to deal with unforeseen traffic and weather conditions, as well as obstacles that have to be detected and avoided. In this paper, we use the intelligent vehicle system as the running example to illustrate our concepts[1].

In such uncertain environments, runtime uncertainty [4] and unpredictability arise due to: (1) the environ-

[1]Note that we have implemented a proof-of-concept of ACon for this example as a tool demo [7]

ment and the resulting limitations in accounting for all possible environmental conditions at design time, and 2) the monitoring infrastructure (e.g., sensor imprecision, sensor noise, and sensor failures).

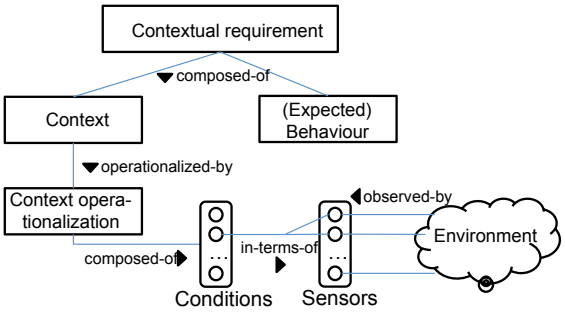

Figure 1: Runtime uncertainty affecting the context operationalization of contextual requirements.

In this paper we address the impact that unpredictable environments have on a particular type of requirements, namely contextual requirements. Defined as requirements that are valid only in a specific context [8, 9, 10], contextual requirements rely on measurable properties of the context state for a particular expected behaviour. In our example, consider the contextual requirement "support lane keeping when driver is sleeping". In order for the system to be able to satisfy such a contextual requirement, the context "driver is sleeping" has to be detected by the system at runtime. Therefore, the system needs to have an operationalization consisting of different conditions representing the context and able to be measured by available sensors (e.g., a camera installed in the car or some wearable devices). We depict the relationship between a contextual requirement, the environment and the context operationalization in terms of the observed environment in Figure 1.

Unpredictability in the environment due to events or conditions that were unforeseeable at design time result in conditions not considered for a certain context for a contextual requirement [4]. In our example, the condition of the street being wet and the sun shining results in the user squinting his eyes. Such conditions would usually be recognized by the system as end-user sleeping, but should not. Changes to the monitoring infrastructure may affect the system's ability to recognize the context in which contextual requirements are valid. In our example, the camera fails to record, resulting in system's inability to monitor the status of user's eyes. Both conditions result in the system not being able to satisfy contextual requirements and thus provide the expected system behaviour. Overcoming this limitation calls for runtime support for the adaptation of contextual requirements [11] (i.e., adapting the requirements' context to changes in the environment or in the monitoring infrastructure). To the best of our knowledge there are no approaches that address runtime uncertainty by ensuring an up-to-date knowledge of the relevant context for contextual requirements.

To address this problem we describe our approach, ACon (**A**daptation of **Con**textual requirements), in which we take the position that self-adaptive systems, in order to respond to unpredictable changes in their operating environment, need to learn from the environmental data that they have available at runtime and execute the adaptation of their contextual requirements accordingly. ACon uses a feedback loop to maintain up-to-date knowledge about contextual requirements based on current information about the context in which contextual requirements are valid at runtime. Upon detecting that contextual requirements are affected by runtime uncertainty, ACon integrates data mining algorithms that analyze contextual data to determine the context in which contextual requirements are valid, thus adapting the context in which contextual requirements are valid. ACon includes the interaction with end-users as part of a semi-automatic approach in which the human is in the loop.

We conducted an evaluation of ACon through an application of its feedback loop and data mining algorithms on a large dataset collected from the use of an activity scheduling system that operated in extremely demanding situations in wild and unpredictable environments with a complex monitoring infrastructure.

Our paper is structured as follows. In the following section we introduce and discuss background and related work that is relevant to the development of our ACon approach. Section 3 defines the formal framework for the ACon approach. With this background, ACon and its feedback loop for adaptation is described fully in Section 4. Section 5 details our evaluation of ACon. Finally, we discuss our results and its applicability in Section 6 and conclude the paper in Section 7.

2. Background and Related Work

Before we describe ACon in detail, we review here related work and the background terms necessary to understand our approach. Table 1 characterizes and compares research related to ACon.

According to Qureshi et al. [19], it is important that the engineering of self-adaptive systems distinguishes between requirements engineering activities at *design*

Table 1: Comparing characteristic properties of selected approaches related to ACon.

Work by	Type of requirement	Purpose of usage in system	Consider uncertainty	Triggers changes	Lifecycle	Automatic execution	Techniques applied
Souza et al. [12]	Evolution requirements	Requirements evolution	No	Yes	Feedback loop	Yes, predefined at design time	Event-Cond.-Action
Inverardi and Mori [10]	Context requirements	Evolution of context variations	Yes	Yes	N/A	Yes, predefined at design time or user specified	Model checking
Ali et al. [8]	Contextual requirements	Adaptation to context	No	No	N/A	No	Reasoning
Ramirez et al. [13]	Uncertainty-aware req.	Goal relaxation	Yes	No	N/A	Yes	Genetic algorithms
Oriol et al. [14]	Monitoring requirements	Monitor adaptation	No	No	Feedback Loop	No	Event-Cond.-Action
Qureshi et al. [15]	Adaptive requirements	Capturing adaptation	Yes	No	N/A	No	Variability modeling
Canavera et al. [16]	N/A	Determine right time for adaptation	Yes	N/A	N/A	Yes	Data mining
Gullapalli et al. [17]	N/A	Self-management of adaptive controllers	Yes	N/A	N/A	N/A	Data mining
Esfahani et al. [18]	Features	Feature-oriented adaptation	Yes	Yes	Feedback loop	Yes	Machine learning
ACon	**Contextual requirements**	**Adaptation of contextual req.**	**Yes**	**Yes**	**Feedback loop**	**Yes**	**Machine learning, data mining**

time and *runtime*. In traditional requirements engineering, the analyst is typically responsible for specifying requirements at design time. Self-adaptive systems are aware of their requirements and thus able to execute basic requirements engineering activities themselves to cope with changing conditions that appear at runtime. This demands requirements engineering to predict what can change at runtime during the design of self-adaptive systems. Therefore, implementation of such abilities for reflection of self-adaptive systems depends on a solid understanding of the runtime environment already at design time [20].

To deal with changing requirements and uncertainty in the operational environment, self-adaptive software systems have to evolve at runtime [11]. Indeed, self-adaptivity is linked to the process of software evolution [3]. Therefore, we consider related work on evolution of requirements. Souza et al. introduce *evolution requirements* for the purposes of requirements evolution (cf. first row in Table 1 for the characteristics of this concept) [12]. Evolution requirements define how requirements can change at runtime and under which conditions. When identifying such conditions at runtime, the system can enact the (predefined) evolution on its own.

The application of evolution requirements is restricted in uncertain environments that cannot be completely predicted at design time. In fact, uncertainty is the main reason that we cannot fully specify our knowledge of requirements and environment of a system during design time [4]. Due to uncertainty, self-adaptive systems need to evolve and even consider unforeseeable changes at runtime [11].

Inverardi and Mori present a framework on how to deal with unforeseen evolution [10]. Their framework is centred around features, which consist of three elements – a requirement, a context entity, and the service implementing this feature. Their framework considers a "subset of constraint requirements which are expressed in terms of context entities". A similar concept of *contextual requirements* is presented by Ali et al. through the introduction of contextual requirement models to specify requirements and their context at runtime [8]. In their work they describe contextual requirements as the interplay of two elements – requirements and context – and focus on the modelling of the variability of both context and requirements, and the detection of errors in contextual requirements models [21].

In building on this related work where the execution of requirements depends on the context, we use the fol-

lowing definition for contextual requirements:

> A *contextual requirement* consists of a 2-tuple of the expected system behaviour and the specific context within which this expected behaviour is valid.

Existing approaches to deal with *uncertainty in requirements* include new languages with dedicated features. Among them, the RELAX language stands out [22]. RELAX is a fuzzy logic-based specification language to be used in goal-oriented approaches that supports the explicit expression of uncertainty in requirements by means of temporal and ordinal operators (e.g., "as early as possible", "as many as possible"). Several methods have been proposed to exploit the capabilities of RELAX. We mention AutoRELAX [13], an automated approach that relaxes goal models that contain RELAX expressions with the objective of satisfying their functional requirements while reducing the number of necessary adaptations.

An adaptive system adapts to certain context conditions by monitoring the current operating environment (through sensors) to decide which actions it has to perform to fulfil system goals [2, 23]. Qureshi et al. [15] introduce adaptive requirements that contain variation points together with a monitoring specification, so that the system can make its own decision on how to best react to a particular goal with such variation points. Oriol et al. [14] argue that a *monitoring specification* (operationalization of context) is a prerequisite to designing requirements monitoring feedback loops to adapt to a certain goal, and derive this monitoring specification manually. Franch et al. [24] extend this work to integrate (again, manually) monitoring into an approach of goal-driven adaptation.

Even when using RELAXation of requirements and adaptive requirements, it is still challenging to detect certain context conditions in an uncertain operational environment. It can be impossible to define the monitoring specification, as context operationalization is not trivial and requires a full understanding of relevant context, which might not be possible at design time. Even if context conditions can be operationalized, assumptions might become invalid at runtime [25]. To address this challenge, we investigate how to update the monitoring specification of one (contextual) variation point automatically *at runtime* with the help of machine learning. In our previous work we obtained promising results when using data mining algorithms to make runtime context conditions measurable [26].

Related work shows that machine learning techniques are valuable for the development of self-adaptive mechanisms, where often great volumes of data is produced over time and can be used to derive decisions on self-adaptation. In this way, machine learning has been used successfully in different research areas on self-adaptive systems: Canavera et al. use data mining to determine the right time for system adaptation with the goal to avoid inconsistencies and system disruptions during and after adaptation [16]. They focus on situations (to which they refer to as "uncertainty factor") where the dependencies of system components are not captured in a model. They mine the execution history of a software system to infer a stochastic component dependency model, which represents interactions among the system's components and use this model to infer the right time for adaptation of a system component. Gullapalli et al. apply data mining to adaptive control in order to adapt feedback control based solutions to changes in the environment [17]. Their goal is to provide accurate decisions in tuning the control parameters in order to self-regulate distributed computing systems based on a time-series-analysis algorithm. Esfahani et al. use machine learning on a feature selection space to support system adaptation of features [18]. They focus on feature adaptation and suggest to treat the system as a blackbox. Thus, they do not base adaptation decisions on the managed system's internal structure. Instead, by defining a feature solution space, they are able to use machine learning to asses and reason about adaptation decisions. They map features to metrics that consider contextual factors to allow automatic learning of feature-oriented adaptation. While Esfahani et al. concentrate on making decisions on the adaptation of all possible features, we concentrate on only one particular system behaviour to adapt the context operationalization.

Feedback loops are a key aspect of engineering self-adaptation [27]. Kephart and Chess introduced the notion of an autonomic element with its famous MAPE-K loop (cf. Figure 2), which culminated in IBM's architectural blueprint for autonomic computing and the Autonomic Computing Reference Architecture (ACRA) [28, 29]. These are key architectural elements of modern self-adaptive systems. Several autonomic elements can be composed to fulfill a common system goal.

Each autonomic element consists of an *autonomic manager*, and one or more *managed elements*. The autonomic manager implements two *manageability interfaces – sensors and effectors*. Through sensors the autonomic manager gathers information from the environment or other autonomic elements. Through effectors

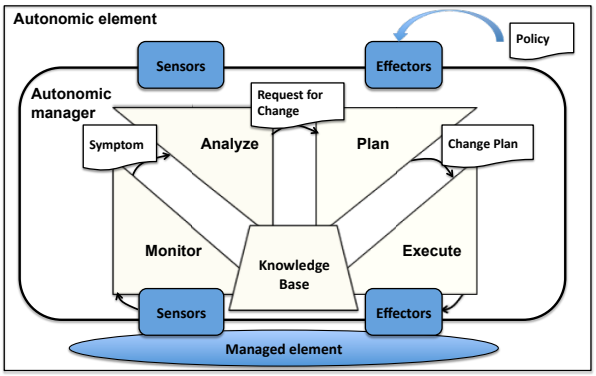

Figure 2: IBM's autonomic element consisting of an autonomic manager and a managed element with a MAPE-K feedback loop at its core [28, 29].

the autonomic manager adjusts the managed element as needed. An autonomic element itself can be a managed element, therefore consisting of sensors and effectors at the top of the autonomic manager. Through the effectors at the top the autonomic manager can receive policies that drive the adaptation and evolution of the system.

The autonomic manager implements a feedback loop that is known as the *MAPE-K loop* because of the four components monitor, analyze, plan, and execute and the knowledge base. The *knowledge base* represents the major communication mechanism between the four components of the autonomic manager. The *monitor* senses the managed element and the context, filters the collected sensor data, and decides on relevant events that can indicate the need for an action of the autonomic manager. These *symptoms* are communicated to the *analyzer* for further analysis [30]. The analyzer correlates the received symptoms and in case it decides about the need to adapt the managed element it sends a *request for change*. The *planner* defines the activity to execute by considering the policy and creates a *change plan*. The *executor* adapts or evolves the system accordingly following the change plan [11].

In comparison to these proposals (cf. Table 1), our approach ACon uses machine learning to mine contextual data at runtime so that the system can continuously adapt its contextual requirements. Similarly to the approaches from Inverardi and Mori [10] and Ali et al. [8], ACon focuses on requirements that depend on context. The purpose of the usage of ACon in the system is the adaptation of the context operationalization in which contextual requirements are valid. Inverardi and Mori focus on the evolution of context variations to deal with uncertainty by using model checking to execute either predefined evolution or by letting users specify the evolution needs. They do not support automatic evolution. The approach by Ali et al. concentrates on reasoning about system adaptation to varying context. They do not deal with requirements uncertainty at runtime. Related work on uncertainty proposes to use goal relaxation in the sense of uncertainty-aware requirements [13] as well as adaptive requirements that capture points of adaptation [15]. ACon uses a MAPE-K feedback loop [28] to trigger adaptation of context operationalization concerning contextual requirements to deal with runtime uncertainty. From the related approaches feedback loops are used by Souza et al. to trigger the execution of evolution requirements [12]. In their work evolution requirements are predefined and therefore they do not deal with requirements uncertainty at runtime. Oriol et al. use feedback loops to monitor the context and provide system adaptation to current context conditions by using a monitoring specification [14]. They do not consider the adaptation of the monitoring specification. Three approaches by Canavera et al. [16], Gullapalli et al. [17], and Esfahani et al. [18] use machine learning (data mining) to support system adaptation. Only one of these approaches concentrates on the integration of requirements through the focus on features [18]. They concentrate on making decisions on the adaptation on all possible features, while we only concentrate on the adaptation of the context in which one system behaviour is valid.

3. The ACon Framework: Fundamentals

This section formalizes the ACon framework by defining different functions over the domains introduced in Figure 1 and specified in Figure 3.

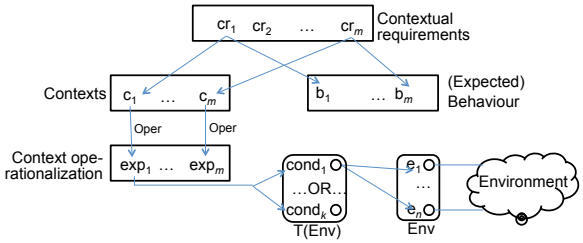

Figure 3: Variables and functions defined in ACon framework.

3.1. Definition of a System State

At any time t, the self-adaptive system will be in a given state concerning the requirements, the operationalization of their context and their satisfactibility. The state includes:

1. A set CR of m contextual requirements,

$$CR = \{cr_i = (c_i, b_i) \mid 0 < i \leq m\},$$

where c_i is the context and b_i the expected system behaviour for a particular contextual requirement i.
In our example, we have two contextual requirements cr_1 = (driver is sleeping, support lane keeping) and cr_2 = (it is raining, activate windshield wiper).

2. $Contexts_{CR}$ and $Behav_{CR}$ are the sets of all contexts and expected behaviours of CR,

$$Contexts_{CR} = \{c_i \mid 0 < i \leq m\},$$

$$Behav_{CR} = \{b_i \mid 0 < i \leq m\}.$$

In our example, $Contexts_{CR}$ = {driver is sleeping, it is raining}, $Behav_{CR}$ = {support lane keeping, activate windshield wiper}.

3. A set Env of typed variables that represents the environment measured through sensors,

$$Env = \{e_i \mid 0 < i \leq n\}.$$

The value of these variables is characterized by the measurement function

$$Meas : Env \rightarrow Object,$$

where *Object* is a supertype of all possible measure types.
In our example, e_1 = BlinkOfEye, e_2 = PositionOfHead, $Meas$(BlinkOfEye) = 0.19, $Meas$(PositionOfHead) = 0.56, etc.

4. The function $Oper$ assigns to requirements' contexts their operationalization in terms of the environment variables,

$$Oper : Contexts_{CR} \rightarrow T(Env),$$

where *T(Env)* is the term algebra formed from variables from Env combined by expression operators (i.e., relational, arithmetic, and logical). Each operationalization in $T(Env)$ is of the form $cond_1$ *OR ... OR* $cond_k$, where every $cond$ is a condition which can be measured by sensors. $vars(Oper(c))$ denotes the set of variables involved in the operationalization of c, $c \in Contexts_{CR}$.
In our example, $cond_1$ = ((BlinkOfEye <= 0.9) and (PositionOfHead <= 0.85) and (PositionOfHead >= 0.75))

We assume the following:

a. The contextual requirements in CR are elicited at design time and cannot change, which means that no unexpected system behaviour may emerge and also that contexts cannot change, only their operationalization (represented by the $Oper$ function) is allowed to change.

b. Variables in Env are fixed too, meaning that sensors are deployed at design time.

c. $Meas$ is a partial function, to reflect the fact that a particular sensor may become unavailable at a certain interval of time.

d. $Oper$ is a partial function, meaning that at some points, the operationalization of a context is not yet known (typically because it is not known at design time and not yet determined at runtime).

e. Even if a context is operationalized, it may (in parts) be not evaluable because some of the variables involved represent a sensor that is currently unavailable. To facilitate the detection of this situation, we introduce a predicate *evaluable* on contexts defined as: $evaluable(c) = \forall e : e \in vars(Oper(c)) : e \in dom(Meas)$, where *dom* is the actual domain of the specified function at a given point of time.

f. Even if the sensor is available, it may be sending wrong data. To detect such situations, we introduce a predicate

$$outlier : Env \rightarrow Bool.$$

over environment variables which returns true if the current value of the variable is considered to be wrong.

g. At design time, an initial operationalization $Oper_I$ of the defined contexts will be determined with all the operationalizations that are known beforehand.

3.2. Satisfaction of Contextual Requirements

The ultimate goal is to maximize the satisfaction of contextual requirements under uncertainty. With this objective, we introduce selected evaluation functions to the ACon framework:

1. $Eval_{Context}$, a function to evaluate the current operationalization of a context,

$$Eval_{Context} : Contexts_{CR} \rightarrow Bool,$$

defined as

$$Eval_{Context}(c_i) = evaluation(Oper(c_i)),$$

where *evaluation* is a function that evaluates the operationalization of the context in some given logics that interprets the expressions formed with the term algebra *T(Env)* previously chosen.

2. $Eval_{Behav}$, a function to evaluate the satisfaction of a given system behaviour,

$$Eval_{Behav} : Behav_{CR} \rightarrow Bool.$$

3. $Eval_{CR}$, a function to evaluate the satisfaction of a contextual requirement,

$$Eval_{CR} : CR \rightarrow Bool.$$

We define $Eval_{CR}((c_i, b_i))$ in the following manner:

- if $c_i \notin dom(Oper)$, then the context is not operationalized, therefore it cannot be evaluated. In this case, we consider that the contextual requirements is not satisfied
- if $\neg evaluable(c_i)$, the contextual requirement is considered not satisfied since the current context operationalization cannot be evaluated due to some sensor malfunctioning (unavailability or outliers). Note that even if some of the rules could be evaluated, our data-mining based approach is still requiring reoperationalization for an automatization.
- if $\neg c_i$, the contextual requirement is considered satisfied since the context acts as guard when evaluating the satisfaction
- if c_i and b_i, this means that both the context and the behaviour are satisfied, yielding thus to the satisfaction of the contextual requirement
- if c_i and $\neg b_i$, this yields clearly to insatisfaction of the contextual requirement

Given this, we define:

$$Eval_{CR}((c_i, b_i)) = (c_i \in dom(Oper)) \\ \wedge evaluable(c_i) \wedge (c_i \Rightarrow b_i)$$

Note that, in order to make ACon as generic as possible, we leave the choice of formal framework both for expressing and evaluating contexts and requirements open. These formalisms do not impact the design of ACon as long as the three required satisfaction functions are provided.

3.3. ACon objective function

ACon aims at reacting to changes that make requirements unsatisfied. In the case of contextual requirements, the particular situation to avoid is having requirements whose context is not operationalized or is not satisfied while the behaviour is, which are the two cases that violate the notion of satisfaction of contextual requirements. This objective can be formalized in the following way:

$$\text{minimise } cr_i : 0 < i \leq m : \neg Eval_{CR}(cr_i)$$

In order to accomplish this goal, ACon will continuously try to operationalize contexts not yet operationalized, and will react as soon as possible to unpredictable environment changes and monitoring problems.

4. Adaptation of Contextual Requirements to Deal with Uncertainty

Since ACon intends to support systems in self-adaptation, we adopt a feedback loop-based approach as an essential part to realize the adaptation. The feedback loop, which we call *Adaptation of Contextual Requirements Feedback Loop* (short: *ACRFL*), identifies conditions in which contextual requirements are affected by uncertainty and adapts the context operationalization for such contextual requirements.

In Figure 4 we illustrate the relationship of ACon to requirements monitoring approaches (e.g., [14]). In the requirements monitoring feedback loop, the system monitors current incoming sensor data and determines based on the operationalization which contextual requirements have to be satisfied. Contextual requirements, their operationalization, as well as the satisfaction of contextual requirements are stored in the knowledge base. As soon as context conditions are detected that fulfill the stored operationalization, the corresponding contextual requirement is satisfied – the system executes the system behaviour of the contextual requirement. ACon adapts contextual requirements (Adaptation of Contextual Requirements Feedback Loop in Figure 4) to provide an up-to-date context operationalization and which represents the monitoring specification described by Oriol et al. [14]. For this purpose, ACon monitors the sensor data as well as the satisfaction of contextual requirements to determine contextual requirements affected by runtime uncertainty. When it

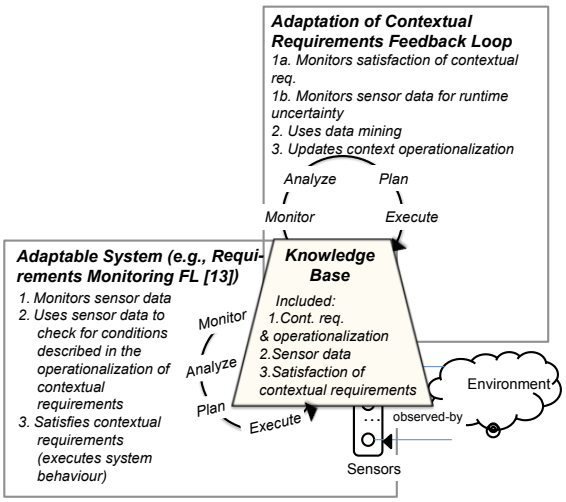

Figure 4: In a feedback loop, ACon updates the context of validity in contextual requirements with an up-to-date operationalization. A requirements monitoring feedback loop as described by Oriol et al. [14] uses the operationalization to detect context conditions, in which a certain system behaviour (represented in contextual requirements) is provided by the system.

detects such contextual requirements, it uses historical sensor data as well as the information about the satisfaction of contextual requirements from the knowledge base to determine an up-to-date context with the help of data mining.

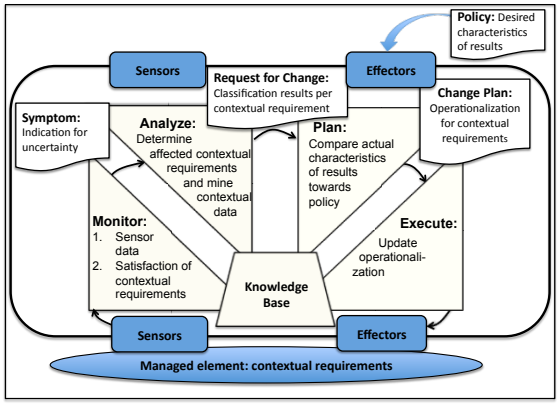

Figure 5: ACon responsible for the adaptation of contextual requirements.

Figure 5 gives an overview of the elements and activities taking place in ACRFL. The *autonomic manager* keeps track of the *managed elements* – all contextual requirements. The *monitor* senses the managed elements. At every time interval t_{int}, the monitor executes two main tasks: 1) evaluating the satisfaction of contextual requirements, and 2) collecting available sensor data related. After filtering the sensed data, the monitor stores *symptoms* in the knowledge base: every requirements violation and sensor relevant change (i.e., indications of potential uncertainty). A symptom contains relevant information for the indication on uncertainty so that the *analyzer* can analyze the situation and make a decision on whether to act on the indications or not. If the analyzer decides to act on it, the analyzer uses data mining on the collected sensor data to determine an appropriate operationalization. The *planner* decides whether the results of the data mining algorithm are good enough and generates a *change plan*. The change plan contains the rules produced by the data mining algorithms representing a new operationalization. The execution engine reads info from the knowledge base (e.g., a script computed and generated by the planner) and actuates, or effects (i.e., effects changes in the managed element) the managed element.

We describe the tasks of the knowledge base as well as of the four components of ACRFL step by step in the remainder of this section, with an emphasis on the monitor and analyzer as the two engines in which the automation takes place.

4.1. Knowledge Base

The knowledge base stores everything that belongs or is required for decision making in an autonomic manager. The knowledge base is an important part of ACon and stores relevant information about 1) contextual requirements, and 2) sensor data as preparation to apply data mining on.

4.1.1. Information Related to Contextual Requirements

For simplicity we store all information about the set of contextual requirements in the knowledge base. Every requirement is stored as a 2-tuple (context, expected behaviour). Table 2 shows the knowledge base containing two contextual requirements from the intelligent vehicle example. For the first contextual requirement the expected behaviour (b_1) is *support lane keeping* in the context (c_1) of *driver is sleeping*. To be able to satisfy this contextual requirement the system has to identify when a driver is sleeping using the sensors or cameras installed in a car. We illustrate a possible operationalization (represented through sensors and their values) of cr_1 by detailing two conditions:

Condition 1 ($cond_1$) : (BlinkOfEye <= 0.9) and (PositionOfHead <= 0.85) and (PositionOfHead >= 0.75)

Table 2: Each contextual requirement is stored as a 2-tuple of context and expected system behaviour together with the operationalization (rules) produced by data mining in the knowledge base.

Contextual requirements knowledge base

$cr_i \in CR$	Context (c_i)	Exp. Behaviour (b_i)	Operationalization of context ($Oper(c_i)$)	Sensors involved ($vars(Oper(c_i))$)
cr_1	Driver is sleeping	Support lane keeping	((BlinkOfEye <= 0.9) and (PositionOfHead <= 0.85) and (PositionOfHead >= 0.75)) or ((RightArmOffSteeringwheel = 1) and (LeftArmOff-Steeringwheel = 1)and (PositionOfHead > 0.95))	BlinkOfEye, PositionOfHead, RightArmOffSteeringwheel, LeftArm-OffSteeringwheel, PositionOfHead
cr_2	It is raining	Activate windshield wiper	...	...

Condition 2 ($cond_2$) : (RightArmOffSteeringwheel = 1) and (LeftArmOffSteeringwheel = 1) and (PositionOfHead > 0.95)

The operationalization contains both conditions (i.e., $Oper(cr_1) = cond_1\ OR\ cond_2$)), but only one condition is sufficient at runtime to trigger the satisfaction for cr_1. The first condition describes a situation in which the eyes of the driver are almost closed (BlinkOfEye <= 0.9), and the head directed to the left side ((PositionOfHead <= 0.85) and (PositionOfHead >= 0.75)). The second condition depicts a situation in which both hands of the user are off steering wheel ((RightArmOffSteeringwheel = 1) and (LeftArmOffSteeringwheel = 1)), and the head turned to the left back side (PositionOfHead > 0.95). The two conditions are measurable by the sensors BlinkOfEye, PositionOfHead, RightArmOffSteeringwheel, and LeftArmOffSteeringwheel.

The different operationalizations created along time for the requirement are stored in the operationalization column in the knowledge base. We introduce the function

$$former{-}operationalizations : Contexts_{CR} \rightarrow Set(T(Env))$$

which returns the set of such existing operationalizations.

4.1.2. Information Related to Sensor Data for Data Mining

Typically, the data used for data mining will come from sensors of different types. Before data mining can be applied on the sensor data, the incoming raw sensor data from the available sensors has to be preprocessed. Preprocessing of the contextual data depends on the characteristics of the sensors and data collected. The frequency of sensor readings that are taken into account for the operationalization has to be determined and can change at runtime (the time t, $t + 1$, $t + 2$ that is used in ACon). For some sensors, especially if the sensors are existing sensors (like sensors integrated in other devices) and reused for the purpose of system adaptation, the readings might be less often than the defined time interval t_{int}. In such cases missing data for these less frequent readings has to be determined for example by keeping the value of the last reading till the next reading. Finally, the sensor data has to be normalized to lay within a range between 0 and 1 so that data mining algorithms that rely on Euclidian distance can be applied for operationalization.

Table 3: Sensor data and related information are stored in the knowledge base.

Sensor data in knowledge base

Time	*IA* for cr_1	*IA* for cr_2	*IA* for cr_3	...	e_1	e_2	...
...	...	...	...	...	...	...	...
$t-1$	true	true	false	...	0.34	0.8	...
t	false	true	false	...	0.34	0.7	...
$t+1$	false	true	false	...	0.34	0.7	...
...	...	...	...	...	...	...	...

After preprocessing of sensor data, another step is necessary to prepare the contextual data for data mining. An additional attribute, the *indicator attribute* (short: *IA*), is stored together with the sensor data in the knowledge base (see Table 3) and is needed for the application of data mining to determine the operationalization of the context in which a contextual requirement is valid. Every time the expected system behaviour was satisfied it stores the current context together with a value true for the indicator attribute, false otherwise.

In our example, for the initial operationalization of the "driver is sleeping" context the driver might help by manually activating the lane keeping feature when he has the feeling to fall asleep soon. Tracking these situations allows to exactly define conditions which indicate

that the driver is falling asleep and in which the expected behaviour of supporting lane keeping is needed. When the system starts executing the lane keeping feature on its own, the system monitors the actions of the driver. Slow actions might indicate that the driver is sleeping. In addition this kind of end-user feedback can be used to determine whether the action was right or wrong.

ACon uses the evaluation function of the system behaviour ($Eval_{Behav}(b)$ for $cr = (c, b)$) as the indicator attribute. $Eval_{Behav}(b) = true$ marks contextual data in which the expected system behaviour is linked to the contextual requirement. $Eval_{Behav}(b) = false$ marks contextual data in which the expected behaviour is not needed. Note that we expect the user correct the system if the executed behaviour is not correct for example by terminating the system behaviour. The indicator attribute together with the contextual data define the input for data mining algorithms to identify patterns that show correlation between contextual conditions with $Eval_{Behav}(b) = true$.

4.2. Monitor State of the System

In the monitor, ACon keeps track of the satisfaction of contextual requirements in certain intervals of time. We define the time interval t_{int} between two measurements t and $t + 1$ of the monitor. ACon considers four different cases that can point to uncertainty affecting the satisfaction of contextual requirements (Table 4.1.2, case 1 - case 4).

4.2.1. Case 1: No operationalized context

This case will arise typically when the context has not been operationalized at design time and still there is not enough data for the data mining algorithm to propose a context at runtime. Eventually, it also may happen that a context becomes unoperationalized at runtime since no feasible operationalization can be found at a certain moment.

Affected requirements: $\{cr = (c, b) \in CR \mid c \notin dom(Oper)\}$

In our example, for each new car which does not include the sensor PositionOfHead, the analyst does not know how to operationalize the context *driver is sleeping*. As a consequence, the operationalization is left to runtime, expecting than the ACon approach will be able to discover other ways to operationalize the context.

4.2.2. Case 2: An unpredictable monitoring framework state

When the context is operationalized, some of the sensors might become unavailable temporarily or permanently. Even if the signal is not lost, the data may be incorrect because the sensor requires recallibration. In any case, the system will not be able to satisfy contextual requirements which contain the sensor that is malfunctioning in their operationalization. To identify this situation, all sensors that are part of the current operationalized context for the contextual requirements are monitored.

Three situations are possible:

a) The sensor stops sending data.
Affected requirements: $\{cr = (c, b) \in CR \mid c \in dom(Oper) \wedge (\exists e : e \in vars(Oper(c)) : e \notin dom(Meas))\}$

b) The sensor is sending wrong data.
Affected requirements: $\{cr = (c, b) \in CR \mid c \in dom(Oper) \wedge (\exists e : e \in vars(Oper(c)) : e \in dom(Meas) \wedge outlier(e))\}$

c) The sensor regains a correct state. Either because it gets recallibrated or comes back into operation again. Requirements that have this sensor in their operationalization are affected, as well as all other requirements that had it at some moment in their history. This second type includes both requirements whose context is not currently operationalized as well as others that are operationalized because it may happen that the current operationalization does not behave as well as the one involving the recovered sensor. To detect conditions that trigger this case we define $correctSensor(e) = e \in dom(Meas) \wedge \neg outlier(e)$.

Affected requirements: $\{cr = (c, b) \in CR \mid (e \in vars(Oper(c)) \vee (\exists\ op \in$ former-operationalizations(c): $e \in vars(op)))\wedge$ at time $t, correctSensor(e) \wedge$ at time $t - 1, \neg correctSensor(e)\}$

Example for case a: Consider a driver that is suddenly wearing sunglasses. So far the eye tracking sensor was the most accurate sensor to measure when a driver is sleeping. After wearing sunglasses, the sensor shows an error in capturing the sensor values. Hence, the monitor will communicate this loss of sensor. Later in the cycle, all other available sensors will be used for re-operationalization for "driver is sleeping" context so that the system can continue monitoring the satisfaction of this contextual requirement.

4.2.3. Case 3: Contextual requirement not satisfied

This situation may occur in two completely different circumstances. First, it may happen that the context

Table 4: Monitoring of requirements affected by runtime uncertainty.

Cases	Detection of uncertainty	Condition on context (c)	Condition on behaviour (b)
Case 1	No operationalized context.	$c \notin dom(Oper)$	
Case 2 a)	Sensor lost	$\exists e \in vars(Oper(c)) : e \notin dom(Meas)$	
Case 2 b)	Sensor decallibrated	$\exists e \in vars(Oper(c)) : outlier(e)$	
Case 2 c)	Sensor up	$\exists e \in vars(Oper(c)) :$ Time $t-1 : \neg correctSensor(e)$ Time $t : correctSensor(e)$	
Case 3	Violation	$Eval_{Context}(c)$ =true	$Eval_{Behav}(b)$ =false
Case 4	Potentially wrong context	$Eval_{Context}(c)$ =false	$Eval_{Behav}(b)$ =true

has been operationalized in an incorrect or inaccurate way (e.g., because the data mining algorithms have not learned enough yet or because the context has been re-operationalized recently due to some malfunctioning in a sensor of the previous operationalization) or because the action that leads to the satisfaction of the behaviour has not been executed yet.

Affected requirements: $\{cr = (c, b) \in CR \mid Eval_{Context}(c) = true \wedge Eval_{Behav}(b) = false\}$

In our example, imagine that the car was mostly used for daily trips where the driver is quite awake, training the classifier to only recognize situations where the driver does not blink very often, recognizing the "driver is sleeping" times with fast blinking of eyes. Now during the night it might happen that the driver needs to blink often due to the different light. Therefore the operationalization will be satisfied while the behaviour is not; the actions to make (reoperationalize context more accurately or to activate the "support lane keeping" option in the car) will be decide later in the cycle.

4.2.4. Case 4: Effects of environmental uncertainty

Cases occur in which the operationalization is not satisfied but the expected system behaviour is. Especially at the beginning, when the data mining classifier is not properly trained because the car is hardly used and therefore there is not enough historical data, this case will appear as the system has to learn over time. In this case the user executes an action that makes the specified system behaviour to be fulfilled, triggering the system to re-operationalize the context considering the current context state in the new operationalization.

Affected requirements: $\{cr = (c, b) \in CR \mid Eval_{Context}(c) = false \wedge Eval_{Behav}(b) = true\}$

In our example, a driver who is quite tired realizes that he might be falling asleep in the next couple of minutes, therefore she switches on the lane keeping feature which makes the behaviour "driver is sleeping" to become satisfied. The current context should be considered in the (re-)operationalization.

4.3. Analyze - Apply Data Mining to Operationalize Context for Contextual Requirements affected by Uncertainty

Given the output of the monitor, the analyzer determines which contextual requirements are affected by uncertainty. For these contextual requirements, it applies data mining to (re-)operationalize the context (determine $Oper(c_i)$).

In the analyzer, ACon relies on lightweight data mining algorithms (i.e., rule-based classifiers). We consider building rule-based classifiers to be lightweight because the algorithm only needs to project and sort the dataset with respect to each attribute (i.e., m projections and sorts for m attributes), and then sequentially iterate over sorted lists. While sorting is not a linear operation, it is nevertheless a fast operation using well-known algorithms such as quick sort. Other classifiers, such as Support Vector Machines (SVM), and Neural Networks (NN), solve optimization problems employing variants of gradient descent algorithms, which sometimes take long to converge to an optimal solution.

Training a lightweight classifier, such as a rule-based one (as opposed to more high-end ones, Neural Nets, Support Vector Machines, etc.) trades off accuracy for speed and computational resources. Therefore, it can be easily rerun many times as newer data instances become available. Since the classifier needs to be built on-the-fly, expensive optimization packages for high-end classifiers would pose a burden on devices used for classification.

Another reason we use rule-based classifiers is because they expose ”rules“, i.e. their classification decision is transparent, visible to the user, if he/she wants to have an idea how the decision was made by the machine on a classification. This cannot be said for SVM

and NN, which have a black-box nature in their classification. The rules can be periodically checked by a human operator to see if they make sense, i.e. correspond to his/her understanding of the environment.
In our example, a rule could be (BlinkOfEye <= 0.9) and (PositionOfHead <= 0.85) and (PositionOfHead >= 0.75).

The outcome of applying such data mining algorithm is the identification of patterns, so called *rules* which represent the operationalization of context $Oper(c_i)$. Rules are produced on contextual data collected and stored in the knowledge base at runtime. The operationalization is represented through rules of the desired context in which a specific contextual requirement is valid. Many different rules can exist for one contextual requirement, depending on how many patterns the data mining classifier finds in the contextual data. Each of the rules represents one desired context condition as depicted in Figure 3. Hence, the combination of different rules produced by the data mining algorithm represents the operationalization for the context of one contextual requirement (e.g., $Oper(c_1) = rule_1\ OR\ rule_2$).

4.4. Plan - Decide whether Operationalization is Good Enough

In the planer the actual error in classifying correct context conditions (e.g. number of context conditions misclassified by the rules generated by the data mining algorithm) is compared against the policy that is given. The policy can represent either an error threshold given by the system provider or given by the users. The users can define this threshold for the error at design time (and change it at runtime), to be used for these context conditions that are 'learned' at runtime. If the error is smaller than the threshold, the system updates the contextual requirements with the new rules produced by the data mining algorithm.

4.5. Execute - Update Contextual Requirements with Operationalized Context

Last, the system updates the information about contextual requirements in the knowledge base with the operationalized context. The rules generated by the data mining algorithm become the candidate operationalization of context for the particular contextual requirement. Using these rules, the system can automatically recognize the context in which contextual requirements are valid at runtime and satisfy the expected behaviour.

5. Evaluation of ACon

We evaluated ACon's performance in operationalizing the context for contextual requirements in a dataset collected from the execution of an activity scheduling system, ToTEM, being used by a crew of four athletes rowing in extreme conditions crossing the Atlantic Ocean, from Dakar (Senegal) to Miami (USA). During the rowing trip, we collected i) end-user needs to discover contextual requirements and ii) sensor data to apply ACon for operationalization of context.

The rowing crew was scheduled to reach Miami in less than 100 days. To achieve this goal, the rowers had to adhere, consistently, to a strict schedule of required activities and daily routines such as sleeping/rowing in four-hour shifts, as well as adequate rest time to maintain their biometric rhythms [31], [32]. The ToTEM system played an essential role in their successful trip performance because the rowers were to survive harsh and unpredictable conditions at sea during periods of extreme fatigue.

In return for the system supporting their trip, the rowers volunteered to keep, *during the trip*, a record of situations in which they needed new or apriori unanticipated system behaviour. This presented us with a unique opportunity to examine the situations when new requirements were needed, as well as the context in which they should be valid. Our interviews with the rowers *after the trip* as well as the analysis of their records from the trip allowed us to identify a number of contextual requirements.

In the remainder of this section, we describe the application of ACon to the data we collected. We depict the evaluation process in Figure 6. We note that ToTEM was not implemented with any adaptation support during the rowers trip. Instead, our evaluation consisted in a *post-trip* analysis of data collected *during* the rowing trip to validate the performance of the data mining algorithms in operationalizing the context for the contextual requirements we identified as relevant to ToTEM supporting the rowers' trip.

In summary, our evaluation consisted in a number of activities as follows. Each of these steps are described in detail in the next subsections:

A. *Preparation for the application of data mining algorithms:*

   1) Elicitation of ToTEM requirements *before the trip*: Based on the knowledge we acquired from a past rowing trip of the same crew, as well as through interviews with the rowers, we elicited a number of requirements and developed ToTEM.

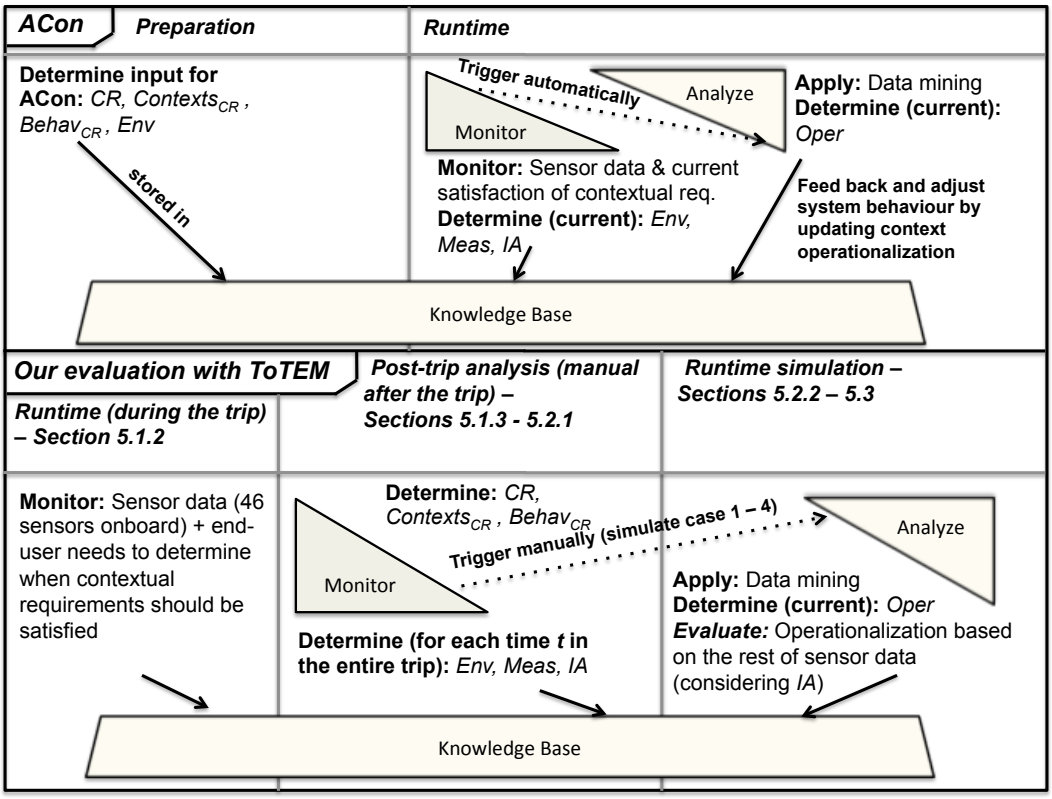

Figure 6: Our evaluation process used to evaluate ACon's performance in operationalising the context for contextual requirements based on sensor data and user needs collected during a rowing trip while using ToTEM.

2) Collection of data on rowers' adaption needs and sensor data *during the trip*: ToTEM was used by the rowers during the trip. We collected contextual data (sensors on the boat) and the rowers' records of runtime adaptation needs in context.

3) Elicitation of contextual requirements *after the trip*: We used the collected data as well as interviews with the rowers and elicited five contextual requirements.

B. *Operationalization of context:* We applied data mining for the operationalization of the context of these five contextual requirements.

C. *Evaluation of data mining (rules):* We validated the results of the operationalization of context 1) through the statistical analysis of the data mining algorithms and 2) in interviews with the rowers.

5.1. Preparation for the Application of Data Mining Algorithms

5.1.1. Elicitation of ToTEM requirements before the trip

Drawing on the domain knowledge gained in the analysis of the contextual data when shaping the ACon approach, we elicited the rowers' goals and requirements for the ToTEM scheduler to be used in their trip.

The major ToTEM functionality was to *alert the rowers about scheduled activities according to the current local (boat) time*. Other requirements included the system *allowing the rowers to manually assign activities*, *alerting rowers by using configured alarms*, and *allowing the rowers to configure the representations of the alerts*. ToTEM integrated one contextual requirement to *adjust the local boat time when the time difference between the current location and the last location is between 2 and 5 minutes*. Time difference was calculated using time zone changes. The goal was to divide the one hour time zones into smaller zones to avoid time shifts of one hour. However, we were not able to understand the impact the context would have on ToTEM require-

ments. The rowers indicated their preference for automated rescheduling in certain context conditions but were unable to indicate which activities to be rescheduled or under which conditions.

5.1.2. *Collection of data on rowers' adaptation needs and sensor data* during the trip

The boat was configured with 46 onboard sensors that recorded biometric and environmental data. Biometric data was captured through ReadiBands[2] on the arm of each rower, and measured actigraphy (movement), effectiveness (fatigue level), and whether the rower was in bed. Environmental data included GPS position, ship roll, wind direction/speed, and altitude.

Unfortunately the trip did not reach its destination due to capsizing at about 1600 km from Miami. However, the rowers recovered measurements from all 46 sensors, containing about 90,748 measurements per sensor, from the first 64 days of the trip. The rowers also recorded (in daily audio and written logs), as much as their busy schedule and harsh conditions allowed, their desired functionality for system self-adaptation in particular context conditions.

5.1.3. *Elicitation of contextual requirements*

After the trip we analyzed these adaptation scenarios (from the rowers' audio files as well as interviews with the rowers). We identified that the rowers' goals for ToTEM adaptation to certain context conditions at sea became clearer during the trip. We were able to elicit five contextual requirements (shown in Table 5).

5.2. *Operationalization of Context*

5.2.1. *Determine Times of Validity for Contextual Requirements*

With knowledge of these five contextual requirements, we turned our endeavours to the application of ACon. To operationalize context for each contextual requirement, we analyzed the contextual data from the entire trip, as well as the rowers' input, to identify correlations between contexts in which contextual requirements were valid and the actual sensor data collected.

For cr_2, cr_3 and cr_5 (shown in Table 5), where certain sensors clearly indicated relevant context conditions, the operationalization was trivial. c_2 (`two rowers are sleeping`) and c_5 (`one rower is sleeping`) directly correlated with the rowers band sensors capturing the times the rowers were sleeping; similarly for

[2]http://fatiguescience.com/solutions/readiband

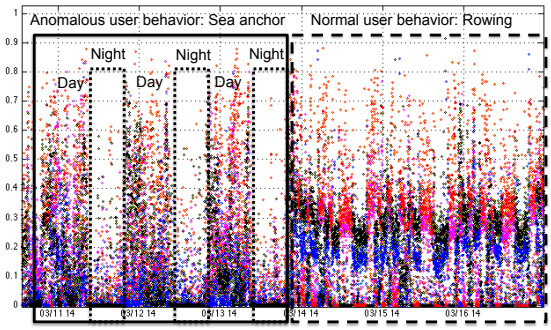

Figure 7: Anomalous user behaviour indicates sea anchor conditions

c_3 (`low performance`) we used the fatigue sensor-measurements to determine the rowers performance. The outstanding research challenge for measuring context for these contextual requirements were situations with loss of these sensors. Other sensors had to be identified, that could be used to determine the context of validity. Cases of sensor loss appeared in our sensor data set. ACon would have recognized such situations and would have (re-)operationalized the context, using sensors that are currently available. Because ACon uses historically sensor data, using data mining it can exactly give correlations to other sensors which can be used at runtime to measure the context situations c_2, c_3 and c_5.

In contrast, for cr_1 and cr_4 the situation was completely different because the boat was not equipped with a sensor to directly detect sea anchor conditions. As rowers reported, the contexts for sea anchor (at day or night) were totally unpredictable and variable. Possible (though not fully understood) conditions for sea anchor could be rower fatigue or sickness or bad weather conditions. In any of these cases, identifying which sensors can measure sea anchor conditions was not possible at design time.

For the purpose of evaluation, based on the steps in ACon we first had to identify contextual data in which the contextual requirement had to be satisfied. We manually identified sea anchor conditions through an iterative process of inspecting log data, listening to audio files, and analyzing the sensor data visually. The Speed Over Ground sensor was not relevant as its values close to 0 could have indicated sea anchor but also rowing against heavy winds. The visual inspection of the graph showing the combination of actigraphy measurements for all four rowers was more useful.

Figure 7 shows the rowers movements in different colours for each rower on the y-axis over a period of eight days (x-axis). Alternating clusters of similar movements show normal user behaviour in the right dot-

Table 5: Contextual requirements with most important sensors and number of rules generated for day 53.

$cr_i \in CR$	Valid context (c_i)	Expect. behaviour (b_i)	Most important sensors (up to first 13 important) ($vars(Oper(c_i))$)	# Rules
cr_1	c_1: On sea anchor at night	b_1: Turn alarms off	Rower1InBed, Rower2InBed, Rower3InBed, Hour, WindDirectionRelative, Rower2-Effectiveness, Rower1Effectiveness, ShipPitch, Rower4InBed, Rower3Effectiveness, gpsSpeedOverGround, WindSpeedRelative, gpsCourseOverGround	29
cr_2	c_2: Two rowers are sleeping	b_2: No non-sleeping alerts	Rower2SleepWake, Rower3SleepWake, Rower1SleepWake, Rower4SleepWake	7
cr_3	c_3: Low performance	b_3: Assign easy activities	Rower1Effectiveness, WindDirectionRelative, gpsSpeedOverGround, DistanceOver-Ground, AtmosphericPressure, Rower3InBed, Hour, Rower1InBed, WindDirection-Bow, Rower4Effectiveness	8
cr_4	c_4: On sea anchor during the day	b_4: Assign sea anchor daytime activities	WindDirectionRelative, Rower4Effectiveness, Hour, ShipRoll, Rower4InBed, Rower1-Effectiveness, Rower3Effectiveness, gpsCourseOverGround, SpeedOverGround, AtmosphericTemperature, WindDirectionBow, Rower1InBed, WindSpeedRelative	29
cr_5	c_5: One rower is sleeping	b_5: Set alerts to visible (no tone)	Rower1SleepWake, Rower2InBed, Rower4SleepWake, Rower3SleepWake, Rower2SleepWake, Rower4InBed, Rower1InBed	6

ted box — two pairs of rowers rowing in alternating shifts of 4 hours, 24 hours a day. The left continuous lined box shows anomalous user behaviour for three sequential days. This anomalous behaviour coincided with the times when the daily logs indicated that the rowers were actually on sea anchor. These are the conditions that represent the context for sea anchor at night and during the day. For the shown time at night it was discovered that at least three rowers are resting/sleeping during the times shown in "night" boxes. Given that there were only two sleeping spots in the cabin, these conditions could not be predicted by the analyst at design time.

5.2.2. Application of Data Mining

Having identified the context "on sea anchor at night" we next had to identify the sensors and their values that correlated with this context. The sensor data was formatted to be tabular, with one column per sensor, and a last column for the indicator attribute. We used *data mining algorithms on this sensor data to identify frequent patterns* correlating with the context of validity. First, we preprocessed the sensor data for the data mining by merging sensor data from all sensors into a single data set. As some sensors had measurements every minute, some every couple of minutes, and some every 15 minutes, we filled the entries in between by taking the last sensed measurement. Finally, we normalized all sensor readings.

Next, we considered rule-based data mining algorithms appropriate to the system's available processing resources (e.g., battery power and CPU). Because ToTEM is implemented on a mobile smartphone we chose JRip [33, 34], a rule-based classifier that uses relatively few system resources. JRip produces a series of rules that represent the sensors and associated threshold values characterizing context conditions for a particular contextual requirement. For the application of JRip, we used WEKA, the Waikato Machine Learning suite of algorithm implementations in Java. More details on the performance of JRIP algorithm, the sensor data set, and preprocessing are included in our AIRE workshop paper [26], as well as in Angela Rook's Master thesis [35].

We experimented extensively with J48 in our preliminary analysis described in the aforementioned thesis. However, we decided not to include results obtained using J48 for the following reasons:

(a) The accuracy, precision, recall, and F-measure were almost the same as when using JRip.

(b) The time complexity of J48 is greater than that of JRip. The complexity of C4.5 (which J48 implements) can reach $O(m^2 * n^3)$ with continuous attributes (cf. [36]), where m is the number of attributes and n is the number of instances. On the other hand, the complexity of RIPPER, which JRip implements, is in the order of $O(n * log^2 n)$ (cf. [37] for a complexity analysis of the precursor algorithm which RIPPER ([38]) optimizes).

(c) J48 treats all the classes the same, whereas JRip favors the less prevalent class (the target class) over the more frequent one. As such, decision trees are in our case less comprehensible due to a somewhat increased model complexity (see also Kotsiantis [34] for more discussion on this point).

We also experimented with SMO (a Support Vector Machines implementation), Logistic Regression, and Neural Nets. All of them performed worse than JRip in terms of accuracy, precision, and recall, not to mention their time complexity which is much higher than that of JRip, making them quite unsuitable for low-power devices.

The JRip algorithm is a separate-and-conquer algorithm that grows rules by greedily adding antecedents (or conditions) to the rule thus increasing the accuracy of the rule at the expense of coverage (the fraction of instances satisfying all the antecedents of the rule). Multiple rules are constructed in this way in order to cover the instances not covered by the previously built rules. The grow phase is followed by an optimization phase that prunes rules in order to reduce overfitting (see the description of RIPPER in [38] for more details).

Two samples of the rules produced by JRip for sea anchor conditions from our investigation include:

- (InBed $\geq$ 1) and (Effectiveness $\leq$ 0.6411) and (COG $\leq$ 0.247911)
- (Day $\leq$ 0.210526) and (COG $\leq$ 0.682451) and (Temperature $\leq$ 0.27027) and (Altitude $\leq$ 0.072993)

These rules represent the *operationalization of context* for the contextual requirements in our case study. Because both rules represent conditions for "on sea anchor", we only give the antecedents of the rules, excluding the right-hand side of the rule after => as it is the same for all of them (i.e., the target-class value "on sea anchor").

The JRip algorithm produces a different amount of rules for each of the five contextual requirements. To give an overview of the most important sensors, Table 5 shows the first 13 sensors with the highest frequency in the rules generated by JRip for each of the five contextual requirements for day 53 in the trip.

5.3. Evaluation of Data Mining (Rules)

We validated our context operationalization results on all five contextual requirements through 1) statistical analysis of the data mining algorithms and 2) interviews with the rowers.

1) Statistical analysis. We performed two statistical analyses based on accumulated data to validate the rules (operationalized context) generated by the JRip algorithm.

(A) For any day of the trip, we used the accumulated data to up to that day in a *10-fold cross validation* to check the data mining algorithms performance on the data collected up to that point. During the 10-fold cross validation the collected data is randomly partitioned into 10 data sets of equal size. The data mining classifier is trained on 9 data sets (the training data set) and tested on the data set that is left (validation data set). This procedure is repeated 10 times, making sure that each of the 10 data sets is used only once as the validation data set. The mean is determined out of the 10 results.

For the 10-fold cross validation we used Weka[3], which accounts for skewed data sets (relevant for most of our contextual requirements) by stratifying the folds accordingly (see [39], page 50). The 10-fold cross validation trains and classifies the context conditions on past collected data. The results were very high with average values for all five contextual requirements between 98-99% for each of the precision, recall, and f-measure respectively. We conclude from this analysis that the classifier performs very well in classifying events in the past.

(B) To assess the predictive power of the data mining classifier we also applied the classifier on "future sensor data". We were interested in how rules produced at one point in time would classify context conditions at future points during the rowing trip. An accurate runtime classification of ACon is the major step for the satisfaction of contextual requirements. Therefore, we conducted a time series analysis of the classifier which trains on past contextual data and tests on future contextual data for the evaluation of the data mining approach in ACon. This is a stricter but more realistic evaluation of the generated rules as it shows results as if ToTEM would have been used at runtime during the rowers trip. For our evaluation of the classifier we use the measurements *precision, recall and F-measure*. Precision shows how many of the cases that were identified as the desired context condition by the classifier are actually correct (compared to real life). Recall shows how many of all desired context instances existing in the sensor data set are also found by the classifier. F-measure is the harmonic mean of precision and recall.

We demonstrate our results from the analysis for precision, recall and F-measure for each of the five contextual requirements (cr_1 - cr_5) in Figure 8, represented each in a separate sub-graph. The x-axis of each sub-graph shows the time from the trip for which we had sensor data (January 22nd to March 26th), the y-axis the normalization of the represented measures precision/recall/F-measure. Each grey diamond that occurs along the x-axis indicates our evaluation of the data

[3] http://www.cs.waikato.ac.nz/ml/weka

mining classifier produced at that date (e.g., the first grey diamond represents our analysis for the day January 25th). The measurement is determined based on a classifier based on the sensor data up to that day (e.g., January 25th) and tested on the rest of the sensor data from that point on to March 26th.

The times when we did the validation of the operationalization (grey diamonds) represent times when ACon could have triggered the (re-)operationalization based on the four cases of the ACRFL when used at runtime. In our post-trip analysis we chose to operationalize the context on day three for the first time as some contextual data has to be collected first, followed by a (re-)operationalization every third day. Hence, day three covers *case 1* as there did not exist an operationalization before. Because the classification results of the 10-fold cross validation are quite good for these instances already, ACon would store this operationalization in the knowledge base. Every following grey diamond (besides the ones that are following the circles) represents cases where the user might have indicated cases where contextual requirements are not satisfied (*case 3*) or indicating effects of uncertainty (*case 4*). In the absence of real-time data on the user-system interaction we applied a heuristic in which such cases would appear every third day and therefore operationalize the context for these days.

In Figure 8, the times when sensor loss occurred are indicated by black filled circles. The first sensor loss (representing *case 2 a)*) was due to the need for energy conservation and affected the environmental measurements (wind speed, wind direction, ship roll, ship pitch, atmospheric temperature, and atmospheric pressure). The second black filled circle shows sensors gained for the previously lost environmental sensors (representing *case 2 c)*). The third circle shows the second sensor loss (again *case 2 a)*), the biometric sensor for one of the rowers.

For cr_2, cr_3 and cr_5 we observe that the classifier results improve in the first 18 days, going up to over 95% for each measurement. This indicates that ACon adapts to the runtime conditions and shows great results for the three contextual requirements after a learning phase of about 18 days. After the first sensor loss (first black filled circle) cr_2 drops in recall, but recovers very fast. Similar with cr_5, which also drops precision a bit, again it recovers fast. For the second sensor loss we do not have enough sensor data to evaluate the effect of this sensor loss for all contextual requirements, besides cr_2. For cr_2 recall drops about 10%, whereas precision stays the same. For contextual requirements that are prone to an unpredictable environment (cr_1 and cr_4), precision, recall, and f-measure fluctuate more than for the other three contextual requirements over time. Recall of cr_1 is better than for cr_4. This might be due to the fact that being on sea anchor allows the rowers to have an additional rest, resulting in patterns in the sensor data that measures the sleeping times, whereas the daily activities of the rowers when on sea anchor seem to be quite unpredictable (e.g., cleaning the boat, catching up with activities that they do not have time on regular days).

In summary, the time series analysis shows that for three out of five contextual requirements ACon would have adapted at runtime after 20 days if used in our ocean rowing example, producing very high results for precision, recall and F-measure. For all three contextual requirements it is possible to achieve similar results even after (the first) sensor loss occurs that seems to affect the classification of the context conditions in first place. For cr_1 the data mining classifier achieves at certain time periods results of about 90% and even more after 27 days of the trip for recall, but only about 60% in precision. Depending on the circumstances this result is better than having no support for the adaptation. Only for cr_4 the data mining classifier delivers poor results over the entire trip.

2) *Two interviews with the rowers to validate the discovery and operationalization of the five contextual requirements* confirmed that the context we identified was correct and that the rules produced by data mining were appropriate. Furthermore, the rowers indicated that they would like to use the functionality of the contextual requirements (even if the classification of the context delivers low performance results) as early as possible during their next trip. They were willing to help train the classifier (by giving feedback about the correctness of the classification) rather than not using the systems adaptation.

5.4. Threats to Validity

5.4.1. Internal Validity

While we took great care to not influence the outcome during (manual) preparation of data, it cannot be guaranteed that no problem was introduced in this step. In our evaluation, we had to manually separate the sensor data set to determine the indicator attribute. For example, for cr_1 – sea anchor at night – the times when they exactly went on sea anchor were sometimes not absolutely clear. We had to analyse the daily logs from the rowers to determine these times, which not always included the exact starting times of the context conditions. Also, we relied on interviews with the rowers for establishing the usefulness of our results. Although we

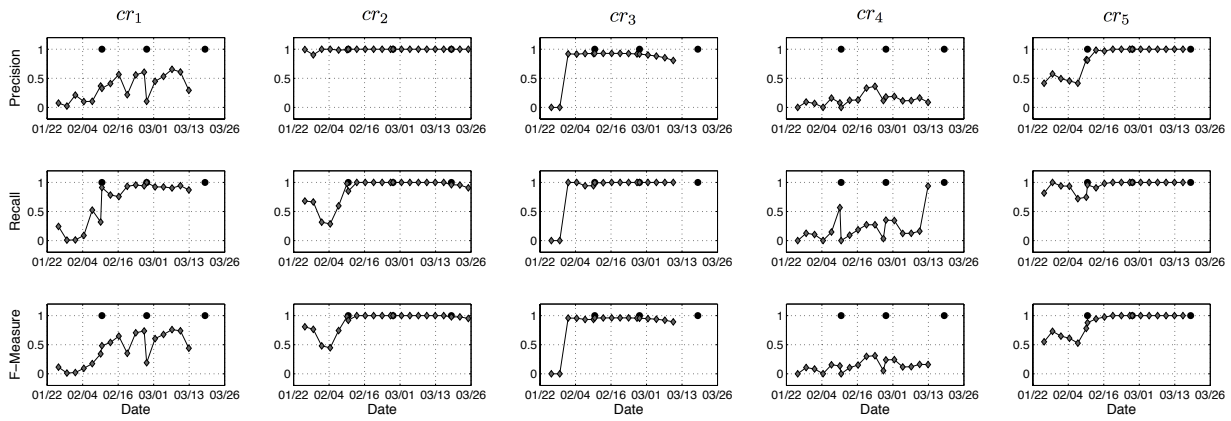

Figure 8: Time series analysis of contextual rules generated by the JRip algorithm for the five contextual requirements from Table 5. Each column shows precision, recall, and f-measure in the classification of the context in which each of the contextual requirements are valid. Sensor losses and gains are shown as three black filled circles (i.e. sensor losses shown as first and third circles; sensor gain the second circle respectively)

carefully designed the interview guide, it is possible that the way we asked could have influenced the outcome. In addition, the rowers might have been biased towards helping the researchers and towards confirming the researchers' goal. We believe that plans to actually use the system on the rowers' next trip mitigated this problem to a large extend.

5.4.2. Construct Validity

A threat to construct validity is that our evaluation of ACon was conducted through a post-trip analysis of the runtime contextual data in relation to the identified contextual requirements. We applied data mining on the contextual data to operationalize the context (i.e. identify measurable context conditions) in which the rowers' contextual requirements were valid. ACon was not used during TOTEM's runtime execution and our evaluation could be affected by our interpretation of the context where the contextual requirements appeared to be valid. At the same time however, the post-trip analysis provided us with an unique evaluation setup not possible if ACon would have been implemented and evaluated at runtime. It enabled us to apply data mining algorithms on part of the contextual data and evaluate the results on the rest of the data (which would have been future contextual data if ACon had been evaluated at runtime).

5.4.3. Conclusion Validity

In our evaluation we used five contextual requirements that we identified after the trip. We achieved results of over 90% for three out of five contextual requirements for the measurements precision, recall and F-measure. The classifier for the other two requirements was at about 90% and even higher after 27 days of the trip for recall, and about 60% for precision. However, for one contextual requirement the results were very poor. This might be due to two reasons: 1) the data set for the contextual data that had to be classified as the valid context was the smallest data set of all five contextual requirements and might not have been sufficient to find a correlation, and 2) the data points that we had for the valid context consisted of times with little repeatability (i.e., the rowers being on sea anchor during the day and executing random activities). Future work should investigate these two reasons closer and find suitable mechanisms to allow human interaction in cases where ACon is not able to automatically adapt.

5.4.4. External Validity

Our evaluation is highly specific to the case study and provides one example of how it is possible to automatically update context operationalization to improve the mapping of system behaviour to valid context. In the specific case, even moderate accuracy promised value to the rowers and failure to execute a requirement would not imply immediate danger. For safety critical systems, an extensive assessment would be needed, which might prove difficult because of the high level of uncertainty in system, environment, and technical approach (i.e. machine learning). In any case we would suggest to provide an option for users to override a decision of the system in the case that it fails to adapt correctly to the users' needs. Observing such user interference could prove to be a valuable information source for the adaptive system by itself and we encourage future research in that direction.

6. Discussion

In discussing ACon, we reflect on the applicability of ACon to other domains, ACon's relationship to other research approaches in the literature, as well as possible extensions of ACon.

6.1. Applicability of ACon to other domains

Although the operational setting and direct users of ToTEM are relatively unique (i.e., four elite athletes on an open-ocean rowing trip), the overall challenge of developing a system for an uncertain operating environment is not. In our evaluation only four users were involved. While this sample is not representative of all users that can be interacting with a system, our focus was on the uncertain operating environment and having involved as many sensors as possible.

The analysis of uncertain operating environments for context becomes increasingly significant as mobile and cloud system developers create products that are used *in unexpected settings*. Considering the example of a typical *smart city* scenario, with literally *thousands of sensors* continuously sending information that may be used, e.g., by e-mobility services, ACon may provide significant value. For instance, one of the most important challenges that smart cities face is to *counteract sensor damage* or calibration loss. Since the cost to repair an individual unit is high due to the human involvement required, it is often the case that the sensor is disconnected until there are several damaged sensors in a relatively small area. Developers in these cases cannot anticipate the full scope of scenarios, and context changes that may occur at runtime. ACon can support developers in designing and implementing systems for such uncertain environments to support self-adaptive systems in changing contexts evolving under unknown conditions.

The smart city example naturally raises the concern of *scalability*. On the one hand, it becomes necessary to manage hundreds of contextual requirements that are represented in the system. Managing a large amount of contextual requirements is inherently complex, but ACon partially mitigates this problem thanks to the *feedback loop* due to having one central place where adaptation is triggered and updates are executed.

On the other hand, we need to consider the *behaviour of data mining techniques* for these large data sets. Jacobs discusses performance for big data, including examples of sensor data bases [40]. Jacobs argues that sequential access is very fast and is suitable for big data. The algorithm we used in our evaluation (JRip) performs only "sequential passes" over contextual data. Therefore, we may reasonably expect that ACon would scale even in this extreme smart cities setting, although of course validation by experimentation is required.

6.2. Connection of ACon to other Research Approaches

ACon has been designed with the purpose of automatically operationalizing uncertain environments into rules based on context conditions that are integrated into contextual requirements. The knowledge about contextual requirements are thus kept continuously up-to-date in response to context changes or acquisition of new knowledge. ACon is covering a current gap in the state of the art and connects naturally with other lines of research. For instance, it complements existing requirements monitoring approaches, e.g., work by Oriol et al. [14]: The rules derived in ACon can be used as monitoring specifications for contextual requirements thus allowing *timely adaptation* of the monitor in response to context changes. Further, our work can be combined with techniques that link self-adaptation to end-user feedback for more customized adaptation. End-users can actively influence the adaptation to satisfy their needs better [41, 42, 43]. Last, integration of ACon with RELAX-based approaches to uncertainty mitigation is worth exploring [13]. In particular, we could consider *applying the principle of relaxing goals* for those situation where the data mining techniques proposed in ACon are not able to discover a clear correlation between context conditions and contextual requirement satisfaction.

6.3. Extensions of ACon

The research presented in this paper is a first step towards using machine learning to support self-adaptive systems in the adaptation of contextual requirements and opens up new research directions and questions.

In the case of the ocean rowing domain – which represents a dynamic, uncertain environment – we found that ACon can be applied for the operationalization of context and also trigger continuous adaptation of contextual requirements when runtime uncertainty is identified. Nevertheless, ACon only works semi-automatically as it needs the help of end-users in using the functionality manually before ACon starts to adapt to the context in which the user executes the functionality. Additionally, the end-user has to correct the system when requirements are not satisfied by the system or are affected by environmental uncertainty.

In our future research we will explore the following extensions:

1. *Full evaluation of ACon at runtime.* We plan to implement ACon into a self-adaptive ToTEM system (as well as other self-adaptive system) and evaluate it during a future system execution. Our preliminary evaluation showed great potential for four out of five contextual requirements. The systematic use of ACon in different uncertain environments will better exhibit the strengths and limitations of ACon and allow us to explore different options and improve the details of ACon. We will be able to investigate and improve the *technical debt* of applying machine learning techniques in further detail, as recommended by Sculley et al. [44].

   Furthermore, we will iterate to improve several parts of ACon, for example investigate the application of adaptive monitoring to adjust for situations where many requirements are violated. Furthermore, we can investigate different policies in how to determine when data mining produces good results. Currently, we are suggesting to use a threshold for the characteristics of data mining as policy. Determining the acceptance of data mining might be dependent on the users as well as the settings, the system is used in.

   After having shown the applicability of ACon for several users, we plan an investigation how ACon performs for different personalities and a huge amount of users. The smart city scenario includes *extremely broad audiences* in different settings. Considering the user characteristics might have an influence on how ACon performs and whether a personalization of ACon would be valuable.

2. *Investigate techniques to identify requirements never executed.* So far ACon triggers operationalization for cases when no operationalization is given, problems with the monitoring infrastructure occur, a contextual requirement is violated, or a contextual requirement executed in a wrong context. In addition to these cases, it is possible that a contextual requirement is never executed because the context never appears in reality. It is important that a self-adaptive system identifies such cases and acts on them, potentially with humans in the loop to investigate this situation. *A possible example:* Using ToTEM in a future rowing trip without ocean-specific conditions, for example along the Mississippi river, can cause the system to never recognize sea anchor conditions based on strong wind context. The current context might now be "strong currents". A potential solution to this challenge could be awareness requirements introduced by Souza et al. [45]. Awareness requirements are requirements that track the execution of other requirements and could be useful in identifying contextual requirements that are never or seldom executed.

3. *Develop techniques to automatically identify whether a given requirement should be defined as contextual requirement.* Currently ACon can only be applied on known contextual requirements. Supporting a self-adaptive system in the automatic identification of requirements that are in fact contextual requirements would increase the self-adaptive capabilities. Machine learning might show potential in this scenario as well: Cluster analysis might help in identifying requirements that are always executed in one specific context. *For example,* an alarm is set on the phone and while ringing nobody picks up the phone to confirm that the alarm was perceived. If such cases happen often, applying cluster analysis might show that this user behaviour typically happens during bad weather conditions (e.g., strong wind or rain). Therefore, the requirement "provide visual notifications" should be considered in the context of "bad weather".

7. Conclusion

In this paper we presented ACon, a novel approach to tackle runtime uncertainty affecting the execution of contextual requirements. ACon uses a feedback loop to detect contextual requirements affected by runtime uncertainty. Further, ACon integrates data mining algorithms that analyze contextual data to determine the context in which contextual requirements are valid, thus adapting the context in which contextual requirements are valid. ACon includes the interaction with end-users as part of a semi-automatic approach in which the human is in the loop. In a preliminary evaluation, we evaluated the performance of the data mining algorithms, which lie at the core of ACon. The application of data mining techniques in the evaluation scenario of the ocean rowing domain has demonstrated their great potential.

ACon is well suited for contextual requirements engineering, where requirements are enriched with context of validity. ACon enables the system to learn the desired context conditions in which a specific system behavior has to be satisfied. The best possible way to store such contextual requirements is using contextual goal models [46].

The use of established concepts in the field such as contextual requirements and feedback loops allow us to argue for the smooth integration of ACon with other works that tackle related issues such as evolution of requirements and requirements monitoring. The results obtained so far are promising and open up to further research avenues in continuously emerging scenarios in the brave new world, where cloud technologies, smart cities and apps in mobile environments create complex ecosystems.

Acknowledgements

We are deeply indebted to the OAR Northwest members, J. Cummer, L. Pasquale, and the SEGAL research group members. Furthermore, we would like to express our gratitude to E. Zavala for her implementation of ACon based on the example of the automotive domain. This research was funded in part by the Canadian National Sciences and Engineering Research Council (NSERC). This work was partially supported by the Spanish funded project EOSSAC, TIN2013-44641-P.

References

[1] K. Welsh, P. Sawyer, Understanding the Scope of Uncertainty in Dynamically Systems, in: Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Springer, 2010, pp. 2–16.

[2] B. H. Cheng, R. L. et al., in: Software Engineering for Self-Adaptive Systems, Springer-Verlag, 2009, Ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

[3] M. Salehie, L. Tahvildari, Self-Adaptive Software: Landscape and Research Challenges, Transactions on Autonomous and Adaptive Systems (TAAS) V (N) (2009) 1–40.

[4] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng, A Taxonomy of Uncertainty for Dynamically Adaptive Systems, in: Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2012, pp. 99–108.

[5] O. Brill, E. Knauss, Structured and Unobtrusive Observation of Anonymous Users and their Context for Requirements Elicitation, in: Proceedings of International Requirements Engineering Conference (RE), IEEE, 2011, pp. 175–184.

[6] R. Dameri, C. Rosenthal-Sabroux (Eds.), Smart City, Progress in IS, Springer International Publishing, 2014.

[7] E. Zavala, X. Franch, J. Marco, A. Knauss, D. Damian, SACRE: a tool for dealing with uncertainty in contextual requirements at runtime, in: Proceedings of International Requirements Engineering Conference (RE), IEEE, 2015, pp. 278–279.
URL `http://www.upc.edu/gessi/SACRE/SACRE.pdf`

[8] R. Ali, Modeling and Reasoning about Contextual Requirements: Goal-based Framework, Ph.D. thesis, University of Trento (2010).

[9] N. Bhaskar, P. Govindarajulu, Context Exploration For Requirements Elicitation In Mobile Learning Application Development, International Journal of Computer Science and Network Security (IJCSNS) 8 (8) (2008) 292–299.

[10] P. Inverardi, M. Mori, Requirements Models at Run-time to Support Consistent System Evolutions, in: International Workshop on Requirements@Run.Time, IEEE, 2011, pp. 1–8.

[11] H. Müller, N. Villegas, Runtime Evolution of Highly Dynamic Software, in: Evolving Software Systems, Springer, 2014, pp. 229–264.

[12] V. E. Souza, A. Lapouchnian, J. Mylopoulos, (Requirement) Evolution Requirements for Adaptive Systems, in: Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2012, pp. 155–164.

[13] A. J. Ramirez, E. M. Fredericks, A. C. Jensen, B. H. C. Cheng, Automatically RELAXing a Goal Model to Cope with Uncertainty, in: Symposium on Search-Based Software Engineering (SSBSE), Springer, 2012, pp. 198–212.

[14] M. Oriol, N. A. Qureshi, X. Franch, A. Perini, J. Marco, Requirements Monitoring for Adaptive Service-Based Applications, in: Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), 2012, pp. 280–287.

[15] N. Qureshi, A. Perini, Engineering Adaptive Requirements, in: International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2009, pp. 126–131.

[16] K. Canavera, N. Esfahani, S. Malek, Mining the Execution History of a Software System to Infer the Best Time for Its Adaptation, in: Proceedings of International Symposium on the Foundations of Software Engineering (FSE), ACM, 2012, pp. 1–11.

[17] R. Gullapalli, C. Muthusamy, A. Babu, Data Mining in Adaptive Control of Distributed Computing System Performance, International Journal of Computer Trends and Technology (IJCTT) 2 (2) (2011) 128–133.

[18] N. Esfahani, A. Elkhodary, S. Malek, A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems, Transactions on Software Engineering (TSE) 39 (11) (2013) 1467–1493.

[19] N. Qureshi, A. Perini, Requirements Engineering for Adaptive Service Based Applications, in: Proceedings of International Requirements Engineering Conference (RE), IEEE, 2010, pp. 108–111.

[20] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, A. Finkelstein, Requirements-Aware Systems: A research agenda for RE for Self-adaptive Systems, in: Proceedings of International Requirements Engineering Conference (RE), IEEE, 2010, pp. 95–103.

[21] R. Ali, F. Dalpiaz, P. Giorgini, Reasoning with contextual requirements: Detecting inconsistency and conflicts, Information and Software Technology (IST) 55 (1) (2013) 35–57.

[22] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, J.-M. Bruel, RELAX: A Language to Address Uncertainty in Self-Adaptive Systems Requirements, Requirements Engineering Journal (REJ) 15 (2) (2010) 177–196.

[23] A. Finkelstein, A. Savigni, A Framework for Requirements Engineering for Context-Aware Services, in: International Workshop From Software Requirements to Architectures, 2001, pp. 2–7.

[24] X. Franch, P. Grunbacher, M. Oriol, B. Burgstaller, D. Dhungana, L. Lopez, J. Marco, J. Pimentel, Goal-Driven Adaptation of Service-Based Systems from Runtime Monitoring Data, Proceedings of Annual International Computer, Software, and Applications Conference (COMPSAC) (2011) 458–463.

[25] R. Ali, F. Dalpiaz, P. Giorgini, V. Souza, Requirements evolution: From assumptions to reality, in: Enterprise, Business-Process and Information Systems Modeling, Vol. 81 of LNBIP, Springer, 2011, pp. 372–382.

[26] A. Rook, A. Knauss, D. Damian, A. Thomo, A Case Study of

Applying Data Mining to Sensor Data for Contextual Requirements Analysis, in: International Workshop on Artificial Intelligence for Requirements Engineering, IEEE, 2014, pp. 43–50.
[27] Y. Brun, G. S. C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. PezzÂŔ, M. Shaw, Engineering Self-Adaptive Systems through Feedback Loops, in: Self-Adaptive Systems, Springer-Verlag, 2009, pp. 48–70.
[28] I. Corporation, An architectural blueprint for autonomic computing., White paper Fourth Edition.
[29] J. Kephart, D. Chess, The Vision of Autonomic Computing, IEEE Computer 36 (1) (January 2003) 41–50.
[30] I. Corporation, Symptoms Reference Specification (2006).
URL {http://download.boulder.ibm.com/ibmdl/pub/software/dw/opensource/btm/SymptomSpec_v2.0.pdf}
[31] C. Samuels, Sleep, Recovery, and Performance: The New Frontier in High-Performance Athletics., Physical Medicine and Rehabilitation Clinics of North America 20 (1) (2009) 149–159. doi:10.1016/j.pmr.2008.10.009.
[32] K. E. Klein, H. M. Wegmann, Significance of Circadian Rhythms in Aerospace Operations, Tech. rep., AGARDograph No.247, Neuilly-Sur-Seine: NATO-AGARD (1980).
[33] P.-N. Tan, V. Kumar, M. Steinbach, Introduction to Data Mining, Pearson Publishing, 2005.
[34] S. B. Kotsiantis, Supervised Machine Learning: A Review of Classification Techniques, Informatica 31 (2007) 249–268.
[35] A. Rook, On the Feasibility of Integrating Data Mining Algorithms into Self Adaptive Systems for Context Awareness and Requirements Evolution, Master thesis, University of Victoria, 2014.
URL http://dspace.library.uvic.ca/bitstream/handle/1828/5580/Rook_Angela_MSc_2014.pdf
[36] J. K. Martin, D. Hirschberg, On the complexity of learning decision trees, in: Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH), 1996, pp. 112–115.
[37] J. Furnkranz, G. Widmer, Incremental reduced error pruning, in: In Proceedings of the Eleventh International Conference on Machine Learning, 1994, pp. 70–77.
[38] W. Cohen, Fast effective rule induction, in: In Proceedings of the Twelfth International Conference on Machine Learning, 1995, pp. 115–123.
[39] G. Forman, M. Scholz, Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement, Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) 12 (1) (2010) 49–57.
[40] A. Jacobs, The Pathologies of Big Data, Communications of the ACM 52 (8) (2009) 36–44.
[41] R. Ali, C. Solis, M. Salehie, I. Omoronyia, B. Nuseibeh, W. Maalej, Social Sensing: When Users Become Monitors, in: Proceedings of European Software Engineering Conference (ESEC), 2011, pp. 476–479.
[42] W. Maalej, D. Pagano, On the Socialness of Software, in: Proceedings of the International Conference on Dependable, Autonomic and Secure Computing (DASC), IEEE, 2011, pp. 864–871.
[43] R. Ali, C. Solis, I. Omoronyia, M. Salehie, B. Nuseibeh, Social Adaptation: When Software gives Users a Voice, in: Proceedings of International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2011, pp. 28–30.
[44] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, Machine Learning: The High Interest Credit Card of Technical Debt, in: Workshop on Software Engineering for Machine Learning (SE4ML), 2014.
[45] V. E. Souza, J. Mylopoulos, From Awareness Requirements to Adaptive Systems: A Control-Theoretic Approach, International Workshop on Requirements@Run.Time (2011) 9–15.
[46] R. Ali, F. Dalpiaz, P. Giorgini, A goal-based framework for contextual requirements modeling and analysis, Requirements Engineering Journal 15 (4) (2010) 439–458.