Department of Distributed and Dependable Systems
Technical Report no. D3S-TR-2015-04
December 22, 2015

Statistical Approach to Architecture Modes in Smart Cyber Physical Systems

Tomas Bures, Petr Hnetynka, Jan Kofron, Rima Al Ali, Dominik Skoda

Abstract: Smart Cyber-Physical Systems (sCPS) are complex distributed decentralized systems of cooperating components. They typically operate in uncertain environments and thus require means for managing variability at run-time. Architectural modes have traditionally been a proven means for the runtime variability. They are easy to understand, easy to realize in resource-constrained systems and (contrary to more sophisticated methods of learning) provide an explicit specification that can be inspected and validated at design time. However, in uncertain environments (which is the case of sCPS), they tend to lack expressivity to take into account the level of uncertainty and factor it in the mode-switching logic. In this paper we present a rich language to specify mode-switch guards. The semantics of the language is based on statistical tests, which, as we show, is a convenient way to reason about uncertainty in the state of the environment.

This work was partially supported by the project no. LD15051 from COST CZ (LD) programme by the Ministry of Education, Youth and Sports of the Czech Republic.

Statistical Approach to Architecture Modes in Smart Cyber Physical Systems

Tomas Bures[1,2], Petr Hnetynka[1], Jan Kofron[1], Rima Al Ali[1], Dominik Škoda[1]

[1] Charles University in Prague
Faculty of Mathematics and Physics
Prague, Czech Republic
{bures, hnetynka, kofron, alali, skoda}@d3s.mff.cuni.cz

[2] Institute of Computer Science
Czech Academy of Sciences
Prague, Czech Republic
bures@cs.cas.cz

***Abstract*— Smart Cyber-Physical Systems (sCPS) are complex distributed decentralized systems of cooperating components. They typically operate in uncertain environments and thus require means for managing variability at run-time. Architectural modes have traditionally been a proven means for the runtime variability. They are easy to understand, easy to realize in resource-constrained systems and (contrary to more sophisticated methods of learning) provide an explicit specification that can be inspected and validated at design time. However, in uncertain environments (which is the case of sCPS), they tend to lack expressivity to take into account the level of uncertainty and factor it in the mode-switching logic. In this paper we present a rich language to specify mode-switch guards. The semantics of the language is based on statistical tests, which, as we show, is a convenient way to reason about uncertainty in the state of the environment.**

Keywords—architecture modes, smart cyber physical systems, statistical testing

I. INTRODUCTION

Smart Cyber-Physical Systems (sCPS) are complex distributed decentralized systems of cooperating mobile and stationary devices, which closely interact with the physical environment. Examples of sCPS include smart home/office, smart cities, smart traffic, smart manufacturing, etc. The mobility aspect of sCPS, their openness and potential open-endness bring about a high level of dynamicity to the system. Therefore, traditional software development techniques have been shown not to be very suitable for developing such systems. Instead, novel approaches and techniques (e.g., [1], [2], [3]) have been proposed to address issues of sCPS development.

Due to operation in uncertain environment, sCPS typically require means of runtime variability (adaptation). Though there is a multitude of approaches to variability and adaptation, not many really address the specifics of sCPS, which on one hand require powerful means of dealing with environmental uncertainty, on the other hand, they are resource-constrained and need strong notion of dependability which essentially excludes learning-based algorithms in normal operation. (However, as we argue in our other works [4], learning can still be strongly benefited from in exceptional situations when no other viable pre-defined adaptation is available anyway.)

Traditional ways to manage variability [5] are not suitable for embedded systems. Typical solution for embedded and dependable systems relies on architectural modes [6]. They are featured by many components systems and technologies – e.g., MyCCM-HI [7], SOFA-HI [8], AADL [9], ProCom [10]. In all these systems, a mode is a property of a component and the component behavior depends on the active mode. At a single instant, a single mode can be active. Switching between modes is controlled by kind of a mode automaton that is associated with a component.

Though attractive in their simplicity and explicit and well-specified behavior, which can be validated before runtime, the existing approaches to modes turn out to be relatively weak in addressing the environmental uncertainty. This is mostly because they decide on the current state and current observations, while normally when addressing the uncertainty, one has to work with trends and historical observations generating likelihood of a future event happening.

While history, future and adversaries are normally tackled by systems that employ learning and Markov processes, we argue that modes are still very suitable (due to their simplicity and explicit specification) even in this case, only the logic for describing transitions has to be extended (e.g. by notion of hysteresis).

In this paper, we propose such an approach to modes and showcase it in the frame of one promising direction for designing sCPS – namely the component ensembles as introduces in the scope of the EU FP7 ASCENS project[1]. For technical details we rely on the DEECo component model [11], however, the results are directly applicable to any ensemble-based component model (e.g., Helena [12]) and with some integration also to general component models for embedded system (e.g., AUTOSAR [13]).

The paper is structured as follows. In Section II, we present a running example that is used as a motivation and for explanation and evaluation of new concepts. Section III explains the core concepts and semantics of our approach, while in Section IV, we discuss issues of the proposed approach. Section V evaluates the proposed approach using the running example and Section VI discusses the related work. Section **Chyba! Nenalezen zdroj odkazů.** concludes the paper.

[1] http://ascens-ist.eu/

II. MOTIVATION EXAMPLE – SMART HOME/OFFICE

As a motivation example, we present here a smart home/office scenario, in which a set of fully automated cleaner robots cooperate in order to keep a room clean while balancing the robots utilization.

The entities in the example are the robotic Cleaners, robots' Chargers, and Cameras, which monitor areas of the room and collect information about dirt in the room. A single Cleaner has an area assigned within the room, in which it should clean dirt. In the case no area is assigned to the Cleaner, it remains idle and waits for area assignment. Also, Cleaners are aware of their battery status. In the case the energy level of the Cleaner's battery is too low, the Cleaner looks up for a suitable Charger and moves to it to be charged. To sum up the Cleaner behavior, it is either (i) cleaning, or (ii) idle, or (iii) looking for a charger, or (iv) moving to a charger, or (v) charging.

A Charger itself is rather a passive entity; it either waits for a Cleaner to dock in or, with a Cleaner in it, it charges the Cleaner.

Cameras observe the room and provide information about areas with dirt that are obtained by Cleaners.

All of these entities can be modeled as components. An example of such an architecture is given in Figure 1. It is given in the DSL of the DEECo [11] component model, however, almost any component model could be used to model the example.

A component is defined by its state and activities. The state is given by a list of data fields (termed *knowledge* in DEECo). The activities of components are modeled as processes (in some component models called *tasks*). These are periodic or event-triggered activities that typically involve sensing, computation, mutation of the component data field, and actuating.

The communication among components is realized by exchanging data about dirty areas to be cleaned (between Cameras and Cleaners) and about availability of Chargers (between Chargers and Cleaners). In DEECo, this is modeled by communication groups (called *ensembles*). An ensemble dynamically determines which components are in the communication group via a membership condition. (Though other communication pattern – e.g., SOA – could be used as well.) The contract between components is captured by a component *role* which lists the data provided and required. The list of roles is then specified in component type definition and in the ensemble definition. A component may naturally provide or require several different roles.

There are three types of ensembles in the example: (i) the Charging ensemble, which groups a particular charger with a robot in the charger, (ii) the Monitoring ensemble between a robot and camera, which communication locations of dirty spots to the robot, and (iii) the Room ensemble, which groups all the entities in the room and primarily allows for balancing the charges and robots utilization.

To reflect the different situations in which a component may occur, components have naturally a number of modes. In the example above, they are implicit and (partially) hidden in the implementations. The Cleaner component has five processes but

```
component role Cleaner
  int id
  int energyLevel
  bool inCharger
  Charger charger
  Area cleaningArea
  Camera[] cameras

component role Charger
  int id
  Cleaner cleaner

component role Camera
  Area[] dirtLocations

component type Cleaner features Cleaner
  process findCharger()
  process move()
  process checkEnergyLevelInCharger()
  process waitForDirtyArea()
  process moveAndClean()

component type CleanerChager features Charger
  process setChargingVoltage()
  process unsetChargingVoltage()

component type Camera feature Camera
  process findDirtyAreas()

ensemble Charging
  roles
    Charger charger
    Cleaner cleaner
  condition
    charger.id == cleaner.charger.id

ensemble Monitoring
  roles
    Camera camera
    Cleaner cleaner

ensemble Room
  roles
    multiple Camera  cameras
    multiple Charger chargers
    multiple Cleaner cleaners
```

Figure 1 Robot cleaners example in DEECo

all of them are in fact mutually excluded, i.e., only executed in different modes. However the exclusion is not visible from the specification and is hidden in the process implementation (i.e., a condition whether the process should start or not). Another example of implicit modes is connected with the Charging ensemble – the Cleaner and Charger are grouped in this ensemble only if the Cleaner is in the Charger, i.e. the Cleaner is in the "Charging mode".

Clearly, an explicit mode specification, which describes modes at the level of the architecture, has a significant documentation value and also eases development and analysis. There are a number of approaches to make the modes explicit (as already mentioned, e.g., MyCCM-HI [7], SOFA-HI [8], AADL [9], ProCom [10], AUTOSAR [13] component systems). They generally list the possible modes of a component and tie process scheduling and component communication to each mode (an example of this is given in Figure 2). The relation between modes (if given) is typically described by a state machine (as found e.g. in Simulink [14] or Scade [15] -based

```
component type Cleaner features Cleaner
  @inMode(findcharger)
  process findCharger()
  @inMode(headingforcharger)
  process move()
  @inMode(charging)
  process checkEnergyLevelInCharger()
  @inMode(idle)
  process checkForMess()
  @inMode(cleaning)
  process moveAndClean()

component type CleanerChager features Charger
  @inMode(charging)
  process setChargingVoltage()
  @inMode(notcharging)
  process unsetChargingVoltage()

ensemble Charging
  roles
    Charger charger
    Cleaner.mode==charging cleaner
  condition
    charger.id == cleaner.charger.id
```

Figure 3 Association of processes and ensembles with modes

architectures of control systems) where transition guards are expressions over internal state or sensed data.

Though such approaches are relatively suitable for traditional embedded systems where the environment is known and can be modeled (e.g. to provide a model for a Kalman filter to cope with noise), it turns out that in the context of sCPS the mode switching would benefit from a bigger expression power to describe and take into account the uncertainty in the sensed data and in the state of the environment in general.

A trivial example is for instance given in Figure 3, which shows the measured energy level curve of a battery during continuous discharge. Suppose this happened in the case of the Cleaner robot. The robot monitors its energy level and switches to "looking for a charger" mode once the energy level drops below a specified point. Should the mode switch be guarded by something as "energy level < 20%", the robot would interrupt work and go for recharge prematurely thus compromising the optimality of the system. In fact, even applying a low-pass filter on the data may not help in this case, it would only introduce a small delay to the mode switch. The problem can be detected only if the data are analyzed more closely and the sudden drop is evaluated as erratic behavior that lowers the credibility of the reading.

Building further on this scenario, assume that there is another transition guarded by "energy level >= 20%" which switches from "looking for a charger" to "cleaning". This guard would alleviate the problem of premature looking for a charger when a sudden erratic drop in the energy level happens, but could itself cause oscillation when the energy level gets close to 20%. Such an oscillation could be again removed by a low-pass filter, however that would assume that the potential noise in the data is known in order to correctly tune the cut-off frequency of the filter.

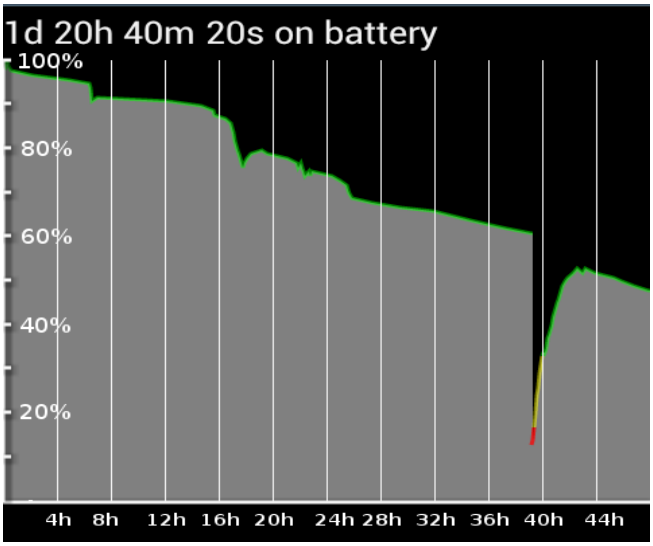

Figure 2 Sample battery energy level during continuous discharge

III. Statistical Approach to Mode Switching

In this section, we explain the core concepts and semantics of our approach. We overview the basic assumptions in Section III.A. Then in Section III.B we define the logic underlying the ability to switch modes based on trends and historical observations and in Section III.C we evaluate the defined operators.

A. Preliminaries

Similar as in existing approaches to modes in embedded systems (e.g. AUTOSAR), we assume that component comes with a set of modes, where a mode determines the tasks (or processes) executed by a component and the interfaces that the component requires or provides. We further assume that a component has a set of data fields (e.g. the component knowledge as shown in Figure 1) and that the mode switches are guarded by the expressions over the data fields.

We allow several mode-groups per component, where a component is exactly in one particular mode per group. For the sake of simplicity, we unite the concepts of a mode-group and a component role and thus we the mutually exclusive modes to be declared within a component role – see Figure 4. Note however, that this assumption is only for simplicity of explanation. It does not have an impact on the generality of our approach presented further.

The defined modes are employed at two places (Figure 2): (i) for processes definitions (the @inMode lines in the component types Cleaner and CleanerCharger) and (ii) for ensembles definitions (Cleaner.mode==charging in the ensemble definition). A process with a mode annotation is executed only if its component is in the particular mode while a component participates in an ensemble only if it is in the required mode.

To actually describe modes switching, the component role contains the mode-switch table (see Figure 5, lines 9-15). This is similar to AADL, where the mode automaton essentially takes a shape of mode switch table (i.e., a list of conditions and

resulting mode). The conditions in the mode-switch table are evaluated every time before an associated process or ensemble-based communication could be executed (i.e. typically periodically with the period of the process).

To cope with uncertainty in sCPS, we generally stick to the rule that there is always a default mode provided. This addresses the cases when the sCPS is caught in an unanticipated situation or environment. This is also the reason of our preference for a mode-switch table over a full-fledge mode automaton with transitions. The default modes typically turn the automaton to an almost complete graph which is less comprehensible than a simple mode-switch table.

With the mode-switch table it is generally possible that more modes are available to select from. (Note that only one such mode can be activated at a time as they belong to the same role and thus are mutually exclusive.) At least, the default mode will be typically available together with other modes. To resolve these situations, we assume that the conditions in the table are prioritized from the top to bottom. In Section XX, we show how to analyze cases in which the default mode applies; this may be a source of potential errors in the mode design, which, due to the presence of the default mode, might demonstrate themselves as late as at runtime.

B. History and Future

A cornerstone of our approach is the ability to switch modes based on historical development of observed values. This is important not only to filter out temporary disturbances, but also to predict trends in the observed data. To this end, we use time-series instead of single-valued data (as would be the case of traditional approach to modes). On top of the time-series, we provide several operators to perform statistical reasoning and to construct expressions that can be used as guards for the modes.

We define a many-sorted logic for expressing the mode guards as follows:

Variables $A = A_1, \dots, A_n$; B; C, ... are time-series. We denote $T(A) = T(A_1), \dots, T(A_n)$ the series of time when $A_1, \dots, A_n$ was sampled.

Operators that return a time-series are:

- Selection $A_i \dots A_j$ denoting a sequence consisting of elements A_i through A_j. This can be also one element time-series – e.g. A_1 denotes a time-series where only the first element has been preserved.
- Selection $[A]_x^y = A_l \dots A_r$ such that $l = \min\{i | T(A_i) \geq x\}$ and conversely $r = \max\{i | T(A_i) \leq y\}$. We call this selection an (x, y) window over A.
- Selection $[A|\Phi]$ denoting a sub-series containing only those A_i for which $\Phi(A_i)$ is true.
- Resampling $A \sim T(B)$ of a timeseries A to sample times $T(B)$ by linear interpolation.

```
component role Cleaner
  [cleaning, charging, findingcharger,
       headingforcharging, idle] modes
  int id
  int energyLevel
  bool inCharger
  bool vacuumStatus
  Charger charger
  Area cleaningArea
  Camera[] cameras

component role Charger
  [charging, notcharging] modes
  int id
  Cleaner cleaner
```

Figure 4 Modes definition

- Resampling $A \simeq T(B)$ of a timeseries A to sample times $T(B)$ by using the closest previous value.
- Basic arithmetics over time-series (assumes that $T(A) = T(B)$, and $c \in \mathbb{R}$):
  - cA – multiplication of each element by a constant
  - $A + c$ – addition of a constant
  - $A + B$, $A - B$ – element-wise addition/subtraction
  - $A \cdot B$ – element-wise product

  Relational operators over timeseries:
- $=$ – equality of two timeseries

  Operators that return a cumulative distribution function F:
- $mean(A)$ – distribution used for comparing the sample mean of A.

  Technically (in the light of Section III.C), it is a distribution of sample means of the time-series elements under the hypothesis that the mean is the sample mean of A. Note that the resulting distribution serves as a plug-in distribution used for evaluation of $\leq_\gamma$, ... relational operators (defined below) by means of hypothesis testing. The quantiles of this distribution are used to establish p-values for the test. For $mean(A)$, we assume that the samples are i.i.d. random variables with normal distribution. $mean(A)$ is thus a shifted and scaled Student's t-distribution (see Section III.C).
- $lra(A)$ – distribution used for comparing the value of intercept $\hat{\alpha}$ in linear regression $\hat{\alpha} + \hat{\beta}x$ fitted to the time-series A via ordinary least squares (OLS)[2].

  Technically, it is a distribution of intercepts $\hat{\alpha}$ under the hypothesis that the true intercept is the one estimated by OLS.

[2]The ordinary least squares (OLS) estimation for $y = \hat{\alpha} + \hat{\beta}x$ is given by $\hat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$, $\hat{\alpha} = \overline{y} - \hat{\beta}\overline{x}$.

- $lrb(A)$ – distribution used for comparing the slope $\hat{\beta}$. Technically, it is a distribution of slopes $\hat{\beta}$ under the hypothesis that the true slope is the one estimated by OLS.
- $lr(A, x)$ – distribution of $y = \hat{\alpha} + \hat{\beta}x$ as above. (Note that since $\hat{\alpha}$ and $\hat{\beta}$ are random variables, y is a random variable as well.)

Relational operators over the distribution functions realized by statistical testing:

- $F \leq_\gamma c, F_1 \leq_\gamma F_2$ – if the null hypothesis $X \leq c$ or $X_1 \leq X_2$ respectively cannot be rejected at confidence level γ, where X, X_1, X_2 are random variables with distributions F, F_1, F_2 respectively .
- $F <_\gamma c, F_1 <_\gamma F_2$ – if the null hypothesis $X \geq c$ or $X_1 \geq X_2$ respectively can be rejected at confidence level γ, where X, X_1, X_2 are as above.
- $\geq_\gamma$ and $>_\gamma$ are defined correspondingly
- $=_\gamma$ is defined as $\leq_\gamma$ & $\geq_\gamma$

Additionally, we include standard logical operators &,∨, ¬.

Examples: With this apparatus in hand, we can express mode-switching guards such as:

- $mean([tmp]_{now-10s}^{now}) <_{0.95} 20$ – with confidence 95% the expected value of temperature tmp in the past 10 seconds has been lower than 20 degrees.
- $mean([tmpB \sim T(A)]_{now-10s}^{now}) - 20 <_{0,95} mean([tmpA]_{now-10s}^{now})$ – with confidence 95% the expected value of temperature $tmpA$ has been in last 10 seconds at least by 20 degrees lower than the expected value of $tmpB$. Note that $tmpB$ is first resampled to sample times of $T(A)$ by linear interpolation.

This assumes that within the 10 seconds window, the temperature is constant and the measurement is subject to normally distributed error with constant mean and variance. If it is assumed that the samples within the window have a linear trend, the comparison can be executed based on statistics of confidence intervals obtained from the linear regression.

- $lr([bat]_{now-10s}^{now}, now) <_{0.95} 11$ – with confidence 95% the current expected value of battery voltage is less than 11 volts evaluated over the past 10 seconds.

The lr operator can be also exploited to predict values in the near future based on the current linear trend. Of course, care has to be taken in interpreting the confidence level of the extrapolation as the confidence speaks only about the current trend, not about its accuracy in predicting the future.

- $lr([bat]_{now-60s}^{now}, now + 120s) >_{0.95} 11$ – based on the trend observed in the past minute, the value of battery voltage after 120 seconds from now will be, with confidence of 95%, over 11 volts.

Generally, the x parameter of $lr(A, x)$ can be arbitrary, however, it is necessary to remember that the variance of the prediction grows with the distance of x from the mean of $T(A)$. The confidence bounds around $\hat{\alpha} + \hat{\beta}x$ form the usual "hourglass" shape (see Figure XX). This means that the statistical test used in the comparison will be the strongest roughly in the middle of the window and will get weaker towards the boundaries of the window (see Sect. XXX where these underlying mathematical formulas are given).

The $mean$ and lr operators can be also exploited to approximately describe that a value is increasing/decreasing:

- $mean([bat]_{now-10s}^{now}) <_{0.95} mean([bat]_{now-70s}^{now-60s})$
- $lr([bat]_{now-10s}^{now}, now - 10s) <_{0.95} lr([bat]_{now-10s}^{now}, now)$

However, the use of $mean$ is often incorrect as it assumes no trends in the observation window. The use of lr is correct, but does not permit to easily reason about the rate of decrease/increase. A better control is achieved by lra and lrb, which expose the distribution of the linear regression coefficients

- $lrb([bat]_{now-10s}^{now}) <_{0.95} -1$ – with confidence 95%, the battery voltage decreases faster than $f(x) = -x$.

Note that the statistical nature of the comparison operators brings a few unexpected features. In particular, it does not hold that:

a) $F_1 \leq_\gamma F_2$ & $F_2 \leq_\gamma F_3 \Longrightarrow F_1 \leq_\gamma F_3$

b) $F \leq_\gamma c_1$ & $c_2 \leq_\gamma F$ & $c_1 \leq c_2 \Longrightarrow c_1 = c_2$

The violation of (a) is caused by the fact the while the distance between F_1, F_2 and F_2, F_3 was small enough to prevent the hypothesis test from rejecting (i.e. $\leq_\gamma$ is true), the distance between F_1, F_3 may be big enough to allow the test to reject, thus evaluating $\leq_\gamma$ to false.

The violation of (b) has a similar cause – c_1, c_2 are close to mean of F thus no rejection takes place. However c_1 may lie below the mean of F while c_2 may be above it (though $F \leq_\gamma c_1$ & $c_2 \leq_\gamma F$).

The violation of (b) nevertheless leads to an elegant way of expressing uncertainty:

- $lr([bat]_{now-10s}^{now}) <_{0.95} 11$ – "provably low battery, drive to the charger"
- $lr([bat]_{now-10s}^{now}) >_{0.95} 11$ – "provably sufficient battery, move to the spot to be cleaned"
- $lr([bat]_{now-10s}^{now}) =_{0.95} 11$ – "nothing definite can be said about the battery, monitor area around, but don't get too far from the charger"

Note that this is a rather significant difference to the traditional semantics of comparison operators with real-valued quantities. There the likelihood that a real-valued observation is exactly equal to a certain quantity is extremely low (in fact, it is 0). In our interpretation, the equality $F =_{0.95} 11$ means "it cannot be shown by a statistical test with enough confidence that a mean is strictly higher or strictly lower". Thus, the mean is not compared exactly to number 11, but to a confidence interval around 11. The confidence interval widens with increasing

variance of samples. Consequently, the likelihood that $F =_{0.95} 11$ may be rather high, since both $F >_{0.95} 11$ and $F <_{0.95} 11$ are rejected.

To simplify the specification, we provide a short-hand notation for the most common cases:

- $below(x, y, w) \Leftrightarrow lr([x]_{now-w}^{now}, now) <_{0.95} y$ … with confidence 0.95% the current value of x is less than y. It assumes that there is a linear trend (potentially a zero trend) in x over the past time interval of length w.
- $above(x, y, w) \Leftrightarrow lr([x]_{now-w}^{now}, now) >_{0.95} y$ … similar as above with x greater than y.
- $fbelow(x, y, w, f) \Leftrightarrow lr([x]_{now-w}^{now}, now + f) <_{0.95} y$ … the future value of x in time f from now is expected to be less than y.
- $fabove(x, y, w, f) \Leftrightarrow lr([x]_{now-w}^{now}, now + f) >_{0.95} y$ … similar as above with the future x greater than y.

C. Evaluation of the Relational Operators

The interpretation we use for $F \leq_\gamma c$ and related operators relies on checking whether value c lies within the confidence bounds given by F, i.e. the $1 - \gamma$ and γ quantiles of F.

Recall that F denotes here the distribution of the statistical quantity under the hypothesis that the true mean is the sample mean. In case of the $mean(A)$ operator, it is a distribution of $\bar{A} + sT/\sqrt{n}$, where T is a random variable with Student's t-distribution with $|A| - 1$ degrees of freedom, $\bar{A}$ is the sample mean of A and s^2 is a sample variance of A.

In case of the $lrb(A)$ operator, assuming the model $\alpha + \beta x + \epsilon$ and normality of the error terms ϵ, the $lrb(A)$ is a distribution of $\hat{\beta} + s_{\hat{\beta}} T$, where $\hat{\beta}$ is the ordinary least square estimate of the slope, $s_{\hat{\beta}}$ is the standard error of the estimator $\hat{\beta}$, and T has Student's t-distribution of $|A| - 2$ degrees of freedom. Similar relation holds for the distribution of intercept $\hat{\alpha}$ returned by $lra(A)$ and the prediction $\hat{\alpha} + \hat{\beta} x$ returned by $lr(A, x)$.

The actual test $F \leq_\gamma c$ is interpreted as:

$$F \leq_\gamma c \quad \Leftrightarrow \quad F(2\mu - c) \leq \gamma$$

where μ is the mean of F and $F(x)$ denotes the cumulative distribution function of F.

The expression $2\mu - c$ is derived from the fact that we shift F such that its mean is c, which is the null hypothesis. This yields a distribution $F - \mu + c$. We then reject the null hypothesis if μ is greater than γ quantile of $F - \mu + c$. With a few trivial rearrangements, we arrive at the $F(2\mu - c) \leq \gamma$.

The test $F_1 \leq_\gamma F_2$ is interpreted as:

$$F_1 \leq_\gamma F_2 \quad \Leftrightarrow \quad (F_1 - F_2)(2\mu_1 - 2\mu_2) \leq \gamma$$

where μ_1, μ_2 denote the means of F_1, F_2 respectively; $(F_1 - F_2)$ denotes the distribution of a subtraction of random variables $X_1 - X_2$ where random $X_1 \sim F_1$ and $X_2 \sim F_2$.

The expression is derived in a similar way as above. We shift each of the two distribution to have mean 0 and subtract them: $F_1 - \mu_1 - F_2 + \mu_2$. This forms a distribution for the null hypothesis that the mean of $F_1 - F_2$ is less or equal 0. We reject if $\mu_1 - \mu_2$ is greater than γ quantile of $F_1 - \mu_1 - F_2 + \mu_2$.

IV. Discussion

In this section, we will discuss in more detail the issues already risen in the previous sections. Particularly, these are (i) verification of completeness of conditions in mode-switching table and (iii) quantile-based interpretation.

A. Verification of conditions

When designing the mode switching table of a component, an important aspect is to cover the option space sufficiently. A disadvantage of the default (true ->) switch option is that the designer can omit a specific situation that should be covered by a special mode switch, or simply makes a mistake in the mode switching specification. There is no way to algorithmically decide whether this was an intention or not. Therefore, we provide a support for enumerating cases in which the default mode switch applies. This is done by means of the Z3 SMT solver [16], when interpreting the high-level operators as independent Boolean variables; it is then up to the system designer to decide whether and which situations currently covered by the default mode switch should be covered explicitly by a special one. We believe that such a check can reveal unforeseen situations, as the number of combinations in more complex systems can be enormous.

B. Quantile-based interpretation

The operators $mean$, lra, lrb, and lr all give distribution of a mean value. This is a typical use-case as the mean value if well understood by practitioners. One however has to be aware of the fact that mean is typically rather sensitive to outliers. The computation of the mean can be thus preceded with some form of outlier detection and exclusion. Care however has to be taken because the definition of an outlier is purely domain-specific and requires good understanding of the cause for outliers. This is because removal of the outliers inherently changes the sample mean and the variance of the mean, which influences the results of the statistical tests used for the relational operators. Typically, this renders the test overconfident and increases the number of false positives.

An alternative to filtering outliers is to use a statistical value which is inherently robust to outliers. In particular, a median is a favorite choice. Also, generalizing the median to an arbitrary quantile has a nice advantage of giving the ability to reason about extremal values – e.g. with confidence γ, 90% of the measurements fall below a given threshold.

```
1.  component role Cleaner
2.    [cleaning, charging, findingcharger,
          headingforcharging, aitingfordirt,
          waitingforclosedirt] modes
3.    int id
4.    <int> energyLevel
5.    bool inCharger
6.    Charger charger
7.    Area cleaningArea
8.    Camera[] cameras

9.    mode-switch-table
10.     aabelow(energyLevel, ENERGYLOWLIMIT)
     && unset(charger) -> mode = findcharger
11.     aabelow(energyLevel, ENERGYLOWLIMIT)
     && isset(charger) ->
        mode = headingforcharger
12.     faaabove(energyLevel, ENERGYLOWLIMIT,
    2min) && isset(cleaningArea) ->
        mode = cleaning
13.     inCharger && aabelow(energyLevel,
    ENERGYCHARGED)  -> mode = charging
14.     unset(cleaningArea) &&
    aaabove(energyLevel, ENERGYLOWLIMIT) ->
    mode = waitingfordirt
15.     true -> mode = waitingforclosedirt

16.  component role Charger
17.    [charging, notcharging] modes
18.    int id
19.    Cleaner cleaner
20.    mode-switch-table
21.      isset(cleaner) -> charging
22.      unset(cleaner) -> notcharging

23. component role Camera
24.   Area[] dirtLocations

25. component type CleanerChager features
    Charger
26.   @inMode(charging)
27.   process setChargingVoltage()
28.   @inMode(notcharging)
29.   process unsetChargingVoltage()

30. component type Cleaner features Cleaner
31.   @inMode(findcharger)
32.   process findCharger()
33.   @inMode(headingforcharger)
34.   process move()
35.   @inMode(charging)
36.   process checkEnergyLevelInCharger()
37.   @inMode(waitingfordirt)
38.   process waitForDirtyArea()
39.   @inMode(waitingforclosedirt)
40.   process waitForDirtyAreaCloseToCharger()
41.   @inMode(cleaning)
42.   process moveAndClean()

43. component type Camera feature Camera
44.   process findDirtyAreas()

45. ensemble Charging
46.   roles
47.     Charger charger
48.     Cleaner.mode==charging cleaner
49.   condition
50.     charger.id == cleaner.charger.id

51. ensemble Monitoring
52.   roles
53.     Camera camera
54.     Cleaner cleaner
55.   condition
56.     ...

57. ensemble Room
58.   roles
59.     multiple Camera  cameras
60.     multiple Charger chargers
61.     multiple Cleaner cleaners
62.   condition
63.     ...
```

Figure 5 Example of the mode specification attached to an architecture in DEECo specification language

The use of median or quantiles in general however comes with a relatively high computational cost. Though there exist relatively simple non-parameteric tests for comparing a median of a set of i.i.d. observations (i.e. an operator in assumptions similar to $mean$), the quantile regression (i.e., yielding operators similar to lra, lrb, and lr) is much more complex. It turns out that its parameters $\hat{\alpha}$, $\hat{\beta}$ cannot be computed directly by a formula (as in the case of ordinary least-squares), but requires minimization, for instance by means of linear programming.

V. Evaluation

To evaluate our approach, we apply the logic for the mode transitions on the smart home/office use-case presented in Section II. Figure 5 shows the example of the mode specification attached to an architecture in DEECo specification language (note that due to space constraints, only parts relevant to modes are given, leaving out DEECo implementation specifics).

As already mentioned in Section II, all the entities are modeled as components – Cleaner, Charger and Camera. Recall that a component is defined by its role and type. A role specifies the component's interface, i.e., data (*knowledge* in the DEECo terminology) that is communicated with other components (via ensembles). A particular component type then features a number of the roles and defines components processes that are executed either periodically or as a reaction to the knowledge values changes. The processes operate with the component knowledge. A single component type can be instantiated many types.

The knowledge of the Charger role consists of its ID and the Cleaner in the Charger (lines 18 and 19 in Figure 5). Its modes are either charging or not-charging (line 17). The mode-switch-table (lines 20-22) is thus quite simple: if there is the Cleaner in the Charger, it is in the charging mode and vice-versa. The Camera role is even simpler; it just offers an array of areas with dirt (line 24) and there are no multiple modes.

The most complex role is the Cleaner. Its knowledge consists of its ID (line 3), energy level (which is a time-series field, line 4), an assigned Charger (line 6), information whether the Cleaner is in the Charger (line 5), an assigned area to be cleaned (line 7), and finally all the cameras in the room (line 8). Regarding the possible modes (line 2), the Cleaner has six of them – cleaning, charging, finding-charger, heading-for-charging, waiting-for-dirt and waiting-for-close-dirt. Switches between these modes, as described in the mode-switch-table (lines 9-15), are as follows. If the battery energy level is below prescribed limit and no charger is set, then the Cleaner tries to find an available charger (line 10). If the energy level is below the limit but the charger is set, the Cleaner proceeds to the charger (line 11). If the energy limit is sufficient and, importantly, will be sufficient within following 2 minutes (without sufficient energy the Cleaner would not be able to even start the cleaning) and the cleaning area is set, the Cleaner starts with cleaning (line 12). If the Cleaner is in the charger, then it chargers until the energy level is not above the prescribed limit (line 13). If there is no cleaning area set and the energy level is sufficient, then the waits for an area to be cleaned (line 14). If no from the above conditions hold, then the sufficiency of the energy level for cleaning cannot be evaluated and the Cleaner waits for an area to be cleaned but accepts only areas close to it so it spends minimum amount of energy for moving to the dirty area (line 15).

VI. Related work

As far as we are aware, there are no other approaches featuring a language for specifying mode transition that would be based on statistical testing. However, there are a number of approaches that are closely related and/or also employ statistical methods for managing variability.

Regarding our own works, recently, we have used a similar approach in Stochastic Performance Logic (SPL) [17], [18], which is a formalism for expressing performance requirements, together with interpretations that facilitate performance evaluation. To reason about performance, SPL considers historical data (i.e., time series of periodic performance measurements) and offers high-level operators to be used in a system code. Contrary to approach presented in this paper, SPL operates with history only and does not consider future.

A closely related approach is Stitch [19], which is a language for describing architecture-based self-adaptation. One of the Stitch's basic concepts is a tactics. It consists of a condition over the architecture state, an action that has to be performed if the condition holds, and finally an effect that is a condition, which should hold after applying the action. Tactics are used in strategies, which describe dynamic adaptation processes. Compared to modes, Stitch offers more fine-grained adaptation of an architecture, which however may not be always ideal for sCPS thanks to potentially limited hardware of the embedded elements of the system. Compare to our approach, Stitch does not allow reasoning about history/future and time-series in conditions. Probabilities are in Stitch used in strategies to describe likelihood that the condition will evaluate to true and subsequently employed in selecting a strategy to be executed.

A dynamic adaptation at runtime is discussed in [20]. To avoid issues with oscillations of context measures, the authors suggest to use an event processing engine, namely Esper [21]. Esper offers a SQL-based event processing language, which allows for defining queries on runtime events with time windows and aggregation functions (like min, max, average). However, the analysis via linear regression and/or predictions of future events is not supported.

References

[1] M. Hölzl, A. Rauschmayer, and M. Wirsing, "Software Engineering for Ensembles," in *Software-Intensive Systems and New Computing Paradigms*, M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, Eds. Springer, 2008, pp. 45–63.

[2] B. Morin, F. Fleurey, and O. Barais, "Taming Heterogeneity and Distribution in sCPS," in *Proceedings of SEsCPS 2015, Firenze, Italy*, 2015, pp. 40–43.

[3] I. Ruchkin, B. Schmerl, and D. Garlan, "Architectural Abstractions for Hybrid Programs," in *Proceedings of CBSE 2015, Montreal, Canada*, New York, NY, USA, 2015, pp. 65–74.

[4] I. Gerostathopoulos, T. Bures, P. Hnetynka, A. Hujecek, F. Plasil, and D. Skoda, "Meta-Adaptation Strategies for Adaptation in Cyber-Physical Systems," in *Proceedings of ECSA 2015, Dubrovnik/Cavtat, Croatia*, 2015, vol. 9278, pp. 45–52.

[5] F. Bachmann and L. Bass, "Managing Variability in Software Architectures," in *Proceedings of SSR '01, Toronto, Canada*, 2001, pp. 126–132.

[6] D. Hirsch, J. Kramer, J. Magee, and S. Uchitel, "Modes for Software Architectures," in *Proceedings of EWSA 2006, Nantes, France*, 2006, vol. 4344, pp. 113–126.

[7] E. Borde, G. Haik, and L. Pautet, "Mode-based reconfiguration of critical software component architectures," in *Proceedings of DATE '09, Nice, France*, 2009, pp. 1160 –1165.

[8] T. Pop, F. Plasil, M. Outly, M. Malohlava, and T. Bures, "Property Networks Allowing Oracle-based Mode-change Propagation in Hierarchical Components," in *Proceedings of CBSE 2012, Bertinoro, Italy*, 2012, pp. 93–102.

[9] P. Feiler, D. Gluch, and J. Hudak, "The Architecture Analysis & Design Language (AADL): An Introduction," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Technical Note CMU/SEI-2006-TN-011, 2006.

[10] H. Yin, H. Qin, J. Carlson, and H. Hansson, "Mode switch handling for the ProCom component model," in *Proceedings of CBSE 2013, Vancouver, Canada*, 2013, pp. 13–22.

[11] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "DEECo: An ensemble-based component system," in *Proceedings of CBSE 2013, Vancouver, Canada*, 2013, pp. 81–90.

[12] R. Hennicker and A. Klarl, "Foundations for Ensemble Modeling – The Helena Approach," in *Specification, Algebra, and Software*, S. Iida, J. Meseguer, and K. Ogata, Eds. Springer, 2014, pp. 359–381.

[13] “Autosar Specification, Release 4.2,” Jul-2015. [Online]. Available: http://www.autosar.org/specifications/release-42/. [Accessed: 18-Jan-2016].
[14] “Simulink.” [Online]. Available: http://www.mathworks.com/products/simulink/. [Accessed: 18-Jan-2016].
[15] G. Berry, “SCADE: Synchronous design and validation of embedded control software,” in *Proceedings of GM R&D Workshop, Bangalore, India*, 2007, pp. 19–33.
[16] L. De Moura and N. Bjørner, “Z3: An Efficient SMT Solver,” in *Proceedings of TACAS’08, Budapest, Hungary*, 2008, vol. 4963, pp. 337–340.
[17] L. Bulej, T. Bureš, J. Keznikl, A. Koubková, A. Podzimek, and P. Tůma, “Capturing performance assumptions using stochastic performance logic,” in *Proceedings of ICPE 2012, Boston, USA*, 2012, pp. 311–322.
[18] L. Bulej, T. Bureš, V. Horký, J. Kotrč, L. Marek, T. Trojánek, and P. Tůma, “Unit testing performance with Stochastic Performance Logic,” *Autom Softw Eng*, pp. 1–49, 2016.
[19] S.-W. Cheng and D. Garlan, “Stitch: A language for architecture-based self-adaptation,” *Journal of Systems and Software*, vol. 85, no. 12, pp. 2860–2875, Dec. 2012.
[20] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, “Models@ Run.time to Support Dynamic Adaptation,” *Computer*, vol. 42, no. 10, pp. 44–51, Oct. 2009.
[21] “Esper.” [Online]. Available: http://www.espertech.com/products/esper.php. [Accessed: 18-Jan-2016].