Architecture Adaptation Based on Belief Inaccuracy Estimation

Rima Al Ali[1], Tomas Bures[1,2], Ilias Gerostathopoulos[1], Jaroslav Keznikl[1,2], Frantisek Plasil[1]

[1] Charles University in Prague
Faculty of Mathematics and Physics
Prague, Czech Republic

[2] Institute of Computer Science
Academy of Sciences of the Czech Republic
Prague, Czech Republic

{alali, bures, gerostathopoulos, keznikl, plasil}@d3s.mff.cuni.cz

***Abstract*—Cyber-physical systems (CPS) are systems of cooperating autonomous components which closely interact with and control the physical environment. Being distributed and typically based on periodic activities, CPS have to cope with the problem that data capturing a distributed state of the system and its environment are inherently inaccurate (they represent *belief* on the state). In particular, this poses a problem when dependability is being pursued. In this paper we address this issue by modeling belief at the architecture level. In particular, we enhance the architecture by models describing belief inaccuracy over time. We exploit these models to quantify at runtime the impact of belief staleness on its inaccuracy. We then use this quantification to drive architectural adaptation with the aim to increase dependability of the running CPS system.**

Keywords—cyber-physical systems; component architectures; self-adaptivity; state-space models; belief

I. INTRODUCTION

Cyber-physical systems (CPS) [1] are systems of collaborating entities, typically materialized as software components that closely interact and control the physical environment. In addition to being distributed, CPS often have a dynamic architecture and are to a degree autonomic and self-adaptive to handle changes in their environments.

A specific property of CPS is that components usually maintain *belief* about the state of other components and/or of the *real state* of their physical environment. Because of distribution and periodic nature of real state sensing, a belief is necessarily outdated (*stale*). Naturally, this implies *belief inaccuracy* (i.e., a deviation of the belief from the real state) and impacts the correctness and safety attributes of CPS. An important property of belief is that it often reflects real state that changes gradually (e.g., physical distance of an obstacle). Consequently, the staler the belief, the more inaccurate it is. This is also often the implicit assumption in a CPS architecture, which enables components to operate correctly in the "normal" case when their belief is not "too stale".

Obviously, key problems are how to quantify the maximum allowed staleness and how to handle situations when it exceeds such maximum. The first problem pertains to how to establish a relation between belief staleness and its inaccuracy since it heavily depends on the nature of belief data domain (e.g., position may change much faster than ambient light level). The second problem is related to architectural adaptations; for instance, detecting that belief inaccuracy exceeded a given threshold may trigger architecture mode switching [2] resulting in a configuration guaranteeing a safe state.

Addressing these problems essentially requires (i) to provide a model that predicts evolution (*prediction model*) of the real state captured by a belief, (ii) to associate the prediction model with the component architecture to allow describing architecture adaptations at the design level, and (iii) to employ the prediction model at runtime to estimate the belief inaccuracy dynamically. Additionally, it is also necessary to design and perform the actual architectural adaptations; since this topic is adequately covered in existing literature [3]–[5] , we do not deal with this topic here and focus only on challenges (i) to (iii).

To our knowledge, there are no approaches that would specifically address the problem of capturing belief inaccuracy at the architecture level and that would connect the inaccuracy estimation with architectural adaptation in support of retaining CPS dependable under changing conditions.

The contribution of this paper lies in proposing a method which specifically addresses the challenges (i)-(iii) above. We do so in the context of DEECo [6] – a component model that specifically targets the design and development of CPS by explicitly supporting dynamic architecture evolution in physical environment and by handling belief of inherently distributed components which interact with and control this environment.

II. MOTIVATING EXAMPLE

To illustrate belief inaccuracy and its prediction, we use as a motivating example a simplified version of the cooperative adaptive cruise control system (CACC) [7]. CACC is an extension of an adaptive cruise control (ACC). CACC maintains a specified distance of a rear vehicle (*follower*) from the vehicle ahead (*leader*). If there is no leader, follower maintains a desired velocity. Contrary to ACC, which typically relies solely on headway sensors (radar-, laser-, camera-based) to continuously measure the distance to the leader, CACC assumes wireless communication of, e.g., the current position and velocity between vehicles. This enables CACC to make

decisions regarding congestion avoidance in a broader time frame than a conventional ACC. In this paper, we assume for brevity that position and velocity are one-dimensional magnitudes.

The key fragments of the CACC architecture are captured in Figure 1 by means of the DEECo architecture model. A vehicle is represented as an instance of the Vehicle component. A DEECo component constitutes state, referred to as *knowledge*, and behavior, expressed via periodic *processes* operating upon the knowledge. For each role of the Vehicle component (i.e., leader and follower) the figure shows the relevant knowledge and processes.

Facilitating dynamic architectures, component interaction in DEECo is encapsulated into *ensembles*, which take the role of dynamic connectors realizing attribute-based communication. Such communication is centered around dynamic component binding based on component attributes (knowledge exposed for this purpose), rather than explicit component identification. Specifically, the set of components to take part in an ensemble is defined declaratively via the *membership condition*. Among components that meet the membership condition, the interaction takes the form of implicit *knowledge exchange* (handled by the execution environment). The architecture of our example comprises a single ensemble – UpdateLeaderPositionAndVelocity, which groups together the leader and its follower with the goal of sharing the leader's velocity and position with the follower. Specifically, this ensemble simply ensures copying the leader's velocity and position (real state) to the knowledge of the follower – this copy becomes the belief of the follower on these magnitudes. Details on the semantics of ensembles can be found in [6], [8].

The CACC architecture described in Figure 1 brings about potential staleness and consequent inaccuracy of the follower's belief (i.e., leader's position/velocity), due to the unreliability of wireless communication and delays introduced by periodic execution of processes and knowledge exchange (not necessarily executed immediately after completion of sensing in the leader). However, safe operation of such CACC architecture can be assumed only if the belief inaccuracy fits within certain "expected" margins. Thus, it is imperative to capture these inaccuracy margins at the level of the architecture, enable for run-time inaccuracy estimation, and, in case of an excess, perform architecture adaptation towards, e.g., the traditional ACC. Note, that ACC employs less-sophisticated control as a trade-off for avoiding leader-to-follower communication (therefore ACC is not impacted by the inaccuracy of the follower's belief).

III. MODELING BELIEF ACCURACY

When investigating the above-mentioned challenge in the frame of our example, we observe that the leader cannot brake the vehicle to full stop immediately, since there is an inherent physical process (determined by the kinetics of the vehicle) that governs how fast the vehicle can slow down[1]. This process

[1] Here, we assume normal conditions; exceptional conditions, such as emergency stops, have to be handled separately (e.g., via obstacle detection).

```
component Vehicle
  // leader's role
  knowledge:
    position, velocity, ...
  process measurePosition(out position):
    function:
      position ← GPS.getCoordinates()
    scheduling: periodic( 100ms )
  // follower's role
  knowledge:
    position, velocity, leaderPosition, leaderVelocity, targetAcc, ...
  process measurePosition(out position)
  process computeAccelerationCACC(in position, in velocity, in leaderPosition,
          in leaderVelocity, out targetAcc):
    function:
      targetAcc ←PIDController.
        computeTargetAcceleration(position, velocity, leaderPosition,
                leaderVelocity, DESIRED_DISTANCE)
    scheduling: periodic( 200ms )
  // switching between CACC and ACC to be decided
  process computeAccelerationACC(in distance, in velocityDifference, out targetAcc):
    ...
ensemble UpdateLeaderPositionAndVelocity:
  coordinator: Vehicle as follower
  member: Vehicle as leader
  membership:
    distance(coordinator.position, member.position)
      ≤ 2 * DESIRED_DISTANCE
  knowledge exchange:
    coordinator.leaderPosition ← closest(coordinator.position, members).position
    coordinator.leaderVelocity ← closest(coordinator.position, members).velocity
  scheduling: periodic( 200ms )
...
```

Figure 1: Excerpt of the motivating example in DEECo DSL.

can be described as a time-invariant state-space model [9], which is a standard representation of a number of physical processes. In a simplified form it can be mathematically captured as $dX/dt = F(X, C)$, where X is a vector that represents the state of the system (e.g., vehicle's velocity and position), C is a vector that represents the control (e.g., brake/gas pedal level), and F is a function that represents the physical process. In our running example F computes the acceleration/deceleration and position change of the vehicle based on its state (i.e., current velocity and position) and the control (i.e., brake/gas). In practice, state-space models are widely used – typically empirically measured under nominal conditions and tabulated by manufactures for their products (e.g., the braking distance of a vehicle or its maximum acceleration at different velocities).

The knowledge of the physical process makes it possible to predict the real state of the leader (i.e., its minimal potential velocity and the minimal potential distance to it) based on the value of the belief about the leader's state and the staleness of it. This prediction is based on the worst-case assumption that since the time when the belief's value was observed, the leader had applied maximal braking.

The idea of providing a prediction model that defines potential evolutions of a real state given the belief value and its staleness is of course not specific to CACC. It is relevant to all systems employing belief regardless of the nature of the real state (be it based on a physical process or be it a state of software entity). In the following we show how this idea can be generalized for states that can be modeled by a time-invariant state-space model $dX/dt = F(X, C)$.

We capture the prediction model by a pair of functions $F^{min}, F^{max}: \mathbb{R}^n \to \mathbb{R}^n$, which for a given state $X \in \mathbb{R}^n$ (e.g., a tuple of position and velocity) return the minimal and maximal derivative of the state. Formally, $F^{min}(X) = F(X, C^{min})$ and $F^{max}(X) = F(X, C^{max})$ where C^{min} and C^{max} correspond to extremal control (e.g., maximal braking and maximal acceleration). In the running example (Figure 2), F^{min} computes the deceleration (i.e., rate of velocity decrease) and velocity (i.e., rate of position change) for maximum braking, while F^{max} computes the acceleration and position change when the gas pedal is maximally pressed. Note that specifically for the running example, since the derivative of position (velocity) is part of the state X, F^{min} and F^{max} do not depend on the position explicitly.

Next, we explicitly associate the functions F^{min}, F^{max} on the level of the component architecture with corresponding component knowledge fields and provide means for calculating the accuracy of the real value based on the model captured by F^{min}, F^{max}.

Formally, we attach a pair of $F^{min}, F^{max}: \mathbb{R}^n \to \mathbb{R}^n$ to a tuple of component knowledge fields $K = (k_1, \dots, k_n)$ whose valuation $B = (b_1, \dots, b_n) \in \mathbb{R}^n$ forms the belief about a real state modeled by F^{min}, F^{max}. For example, in the state-space-models section of Figure 2, we associate leaderPosition and leaderVelocity with F^{min}, F^{max} modeling the actual position and velocity of the leader vehicle. The accuracy of a belief B can then be computed as an interval $A = [X^{min}(t); X^{max}(t)]$, where X^{min} is the solution to the differential equation $dX/dt = F^{min}(X)$ with initial condition $X(0) = B$, and where t is the staleness of B. Similarly, X^{max} is the solution to $dX/dt = F^{max}(X)$ with the same initial condition. In other words, if B was observed (i.e., B was equal to the real state) time t ago, the real state has to be within the bounds of A.

Note that the time-invariance of the model (i.e., that functions F^{min}, F^{max} do not depend on time even though X does) and the fact that we quantify the accuracy only based on extremal control, make it possible to conveniently construct F^{min}, F^{max} as an interpolation in tables obtained by empirical measurements. In Figure 2, F^{min} is based on interpolation of a maximum-deceleration table; e.g., for velocity of 35 m/s, the maximum deceleration is -5 m/s^2. This table can be established simply by bringing the vehicle to its top velocity and then intensively braking and measuring the rate of velocity change over time until the full stop. If an analytical model is available, it can alternatively be employed for constructing F^{min} and F^{max} directly.

The last step is to enrich the component architecture with rules for architecture adaptation based on the safe margins of belief accuracy supported by the architecture. Figure 2 gives an example of specifying a condition that triggers architecture adaptation via mode switching [2]. Specifically, the process corresponding to CACC, resp. the process corresponding to traditional ACC, is annotated with a condition specifying the mode in which the process is activated; i.e., when the estimated inaccuracy of the distance between leader and follower (computed from their positions) is below, resp. above, a given

```
component Vehicle
  knowledge:
    leaderPosition, leaderVelocity, ...
  state-space-models:
    [leaderPosition, leaderVelocity]:
    // Tables for  lookup of max. deceleration/acceleration based on current velocity.
    // The tables consist of tuples (velocity (in m/s) -> acceleration (in m/s²))
    maxDecTable = {0 -> -6, 35 -> -5, 51 -> -3}
    maxAccTable = {0 ->  4, 35 ->3, 51 -> 0}
    Fmin(leaderPosition , leaderVelocity )
              = (leaderVelocity , interpolate(maxDecTable , leaderVelocity ))
    Fmax(leaderPosition , leaderVelocity )
              = (leaderVelocity , interpolate(maxAccTable, leaderVelocity ))
    ...
  process computeAccelerationCACC(in position, in velocity, in leaderPosition,
          in leaderVelocity, out targetAcc):
    mode-trigger: inaccuracy(distance(position, leaderPosition)) <= THRESHOLD
    ...
  process computeAccelerationACC(in distance, in velocityDifference, out targetAcc):
    mode-trigger: inaccuracy(distance(position, leaderPosition)) > THRESHOLD
    ...
```

Figure 2: Representation of state-space model in our example.

threshold. Here, inaccuracy means the size of the accuracy interval A for the current staleness of the distance belief.

IV. Composition of Belief Inacuraccy

An important observation when modeling belief accuracy is that a component process/ensemble knowledge exchange P may aggregate multiple input beliefs $B_1, \dots, B_m$ (i.e. beliefs of different accuracy and model). As a consequence, the result of executing P with $B_1, \dots, B_m$ as inputs, i.e., $P(B_1, \dots, B_m)$, is again a belief B_{out}, whose value, accuracy, and model are given by the composition (determined by P) of $B_1, \dots, B_m$. Thus P acts as a composition function upon the accuracy of input beliefs.

In a general case, the definition of P thus consists of both (i) a specification of how to compute its output based on the inputs, and (ii) a specification of how to compute the accuracy of output based on the accuracy of inputs. Advantageously, when P can be represented as a composition of continuous functions (e.g., +, -, *, /, max, min, sin, cos, etc.), which is indeed a typical case, we can deduce (ii) from (i) and thus (ii) does not have to be specified explicitly (which is also the case of the accuracy of distance in our example).

V. Realization in Java

To evaluate the feasibility of our approach we have implemented[2] a prototype of the motivating example in the jDEECo framework, which is the Java realization of the DEECo component model. The implementation contains the Java code of the components and ensembles, accompanied by the respective Simulink models, which graphically represent the control in the components and responses of the environment. For numerical solving of the differential equations of the prediction models, the Apache Commons Mathematical Library is employed. Solving takes place prior to the execution of every process, for the relevant input knowledge fields.

We are currently working on integrating the proposed extensions to jDEECo. Specifically, we are working on (i)

[2] http://d3s.mff.cuni.cz/projects/components_and_services/deeco/

associating the state-space models with knowledge fields and component modes via Java annotations, (ii) keeping track of the last time the value of a field was updated, so that the staleness can be determined, and (iii) integrating the differential equation solver into the jDEECo runtime framework. The final goal is to make the minimum and maximum predicted values of a knowledge field accessible via an API, which would be available in both processes and mode switching conditions (mode-triggers).

VI. Related Work

The work presented in this paper is positioned in the broad context of self-adaptive autonomic systems [10][11]. Such systems typically feature execution based on the Monitor-Analyze-Plan-Execute (MAPE-K) control loop [12]. Our approach is directly connected to MAPE-K adaptation mechanisms by providing a form of "augmented" sensing. Sensing approaches proposed in literature span from direct (hardware) sensing [13] to monitoring of extra-functional properties (e.g., performance monitoring [14][15]) and from environment sensing to monitoring of internal state via introspection (often termed *self-awareness*). On top of reasoning over present and past data, prediction-based approaches (e.g., [14]) allow for foreseeing a faulty or harmful situation (e.g., high load on server farm, traffic congestion along a lane) and for adapting in order to prevent such situations. The main distinction of our approach is that adaptation decisions are based not on predicted future values, but on anticipating what the current values of the input data could be (with respect to their staleness).

Time-based analysis of hybrid systems [16] (e.g., via timed automata) is also applicable to CPS. Whereas the goal there is model checking, in our case the associated time analysis aims at guiding runtime adaptation. Dealing with inaccuracy in the sensed data is also tackled in the data-filtering domain, typically within the context of control systems. For example, approaches based on Kalman filtering [17] employ linear quadratic estimation to remove the measurement noise and produce a statistically optimal estimate of the underlying system state. Even though these methods can be considered as alternatives to pure state-space-based estimations, they would have to be adapted to explicitly deal with staleness of belief and to be seamlessly integrated with the architecture.

Finally, although incorporating the above-discussed prediction models into other component models (e.g., Simulink) is technically feasible, it is difficult as these models usually do not provide appropriate abstractions for explicit handling of belief and its accuracy at the level of architecture.

VII. Conclusion

In this paper, we have presented an approach for modeling and estimating inaccuracy of belief in distributed cyber-physical systems (CPS) for the purpose of architecture adaptation. Specifically, we have (i) proposed a prediction model to capture the evolution of the real state captured by a belief, (ii) enriched architecture definition with the prediction model to allow describing architecture adaptations based on belief inaccuracy at the design level, and (iii) shown how to employ the prediction model at runtime to estimate the belief inaccuracy dynamically, allowing for runtime adaptation.

Our approach of associating the model at the level of architecture can be easily adapted for other types of models (e.g., those based on machine learning). Nevertheless, a significant advantage of the proposed prediction models (specifically the time-invariant state-space models) is that these models are invariably used in modeling physical properties of CPS and thus well known. They can also be easily obtained by empirical measurements.

Acknowledgment

This work has been partially supported by the European Union Seventh Framework Programme FP7-PEOPLE-2010-ITN under grant agreement n°264840, and partially by the EU project ASCENS 257414.

References

[1] B. K. Kim and P. R. Kumar, "Cyber–Physical Systems: A Perspective at the Centennial," *Proc. IEEE*, vol. 100, no. Special Centennial, pp. 1287–1308, 2012.

[2] D. Hirsch, J. Kramer, J. Magee, and S. Uchitel, "Modes for software architectures," in *Proc. of the 3rd European conference on Software Architecture, EWSA '06*, 2006, pp. 113–126.

[3] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure," *Computer (Long. Beach. Calif).*, vol. 37, no. 10, pp. 46–54, 2004.

[4] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering*, 2007, pp. 259–268.

[5] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *Intell. Syst. their Appl. IEEE*, vol. 14, no. 3, pp. 54 – 62, 1999.

[6] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "DEECo – an Ensemble-Based Component System," in *Proc. of CBSE'13*, 2013, pp. 81–90.

[7] C. Desjardins and B. Chaib-draa, "Cooperative Adaptive Cruise Control: A Reinforcement Learning Approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1248–1260, Dec. 2011.

[8] J. Barnat, N. Benes, T. Bures, I. Cerna, J. Keznikl, and F. Plasil, "Towards Verification of Ensemble-Based Component Systems," in *Proc. of FACS'13*, 2013.

[9] B. Friedland, *Control System Design: An Introduction to State-Space Methods*. Dover Publications, Inc. Mineola, New York, 1986.

[10] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, May, pp. 1–40, 2009.

[11] C. Ghezzi, L. Pinto, P. Spoletini, and G. Tamburrelli, "Managing Non-functional Uncertainty via Model-Driven Adaptivity," in *Proc. of ICSE '13*, 2013, pp. 33–42.

[12] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer (Long. Beach. Calif).*, vol. 36, no. 1, pp. 41–50, 2003.

[13] J. Eidson, E. Lee, S. Matic, S. A. Seshia, and J. Zou, "Distributed Real-Time Software for Cyber–Physical Systems," *Proc. IEEE*, vol. 100, no. 1, pp. 45 – 59, 2012.

[14] S. Kounev, "Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets," *IEEE Trans. Softw. Eng.*, vol. 32, no. 7, pp. 486–502, 2006.

[15] S. Kounev, F. Brosig, N. Huber, and R. Reussner, "Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems," *2010 IEEE Int. Conf. Serv. Comput.*, pp. 621–624, Jul. 2010.

[16] A. van der Schaft and H. Schumacher, *An Introduction to Hybrid Dynamical Systems*. LNCS vol. 251, Springer, 2000, pp. 111–132.

[17] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. I. Jordan, and S. S. Sastry, "Kalman Filtering with Intermittent Observations," *IEEE Trans. Automat. Contr.*, vol. 49, no. 9, pp. 1453 – 1464, 2004.