Department of Distributed and Dependable Systems
Technical report no. D3S-TR-2013-04
February, 2013

Department of Distributed and Dependable Systems D3S

Formalization of Invariant Patterns for the Invariant Refinement Method

Tomáš Bureš, Ilias Gerostathopoulos, Jaroslav Keznikl, František Plášil

Abstract: This report provides formalization of the Invariant Refinement Method. In particular, it formally describes the individual invariant patterns w.r.t. (soft) real-time constraints and provides formulation, as well as proofs of theorems related to refinement among these patterns.

This work was partially supported by the Grant Agency of the Czech Republic project P202/11/0312.
This work was also partially supported by the EU project ASCENS 257414.

Contents

1 Introduction

1.1 Motivation

Invariant-based design is advantageous for designing ensembles of components – see e.g., SOTA [1] – because it focuses on the valid states of the system, i.e., the invariant properties of a correct system. This is relevant since these systems operate autonomously in an open-ended environment, and invariants are well-suited for capturing the properties of a component with respect to its environment.

The problem of such invariant refinement is that the requirements have a much higher level abstraction than the properties (invariants) of the individuals of the system architecture (ensembles, components). The design transition from the high level to the low level includes a lot of design choices without firm borders and guidelines, and thus is prone to errors.

In our work we propose to skip this gap by gradual step-wise refinement (decomposition) of invariants, which ends up with architectural elements (ensembles, components). We call this approach *Invariant Refinement Method* (IRM).

IRM however requires the steps of the refinement to be well-defined (ideally formally), so that the refinement itself represents a proof of the correctness of the design.

> *In other words, it is necessary to have (formal) means allowing for deciding upon the correctness of the refinement.*

Having a formal framework that formalizes these relations would allow for:

- Design-time guarantees of design correctness = the system design truly addresses the high-level requirements
- Runtime monitoring = detection of discrepancies in system design during execution.

1.2 Case Study

To illustrate the challenges in IRM, we exploit the e-mobility case study [3]. Electric vehicles (e-vehicles) compete for e-mobility resources, such as charging stations (CSs) in order to achieve optimal journeys with respect to the drivers' daily activities (calendars). A calendar consists of a set of points of interest (POIs), together with timing constraints specifying the expected POI arrival and departure times. An e-vehicle uses a planner in order to create its individual journey plan, stemming from the driver's calendar and including charging periods when necessary. The system is fully decentralized – every e-vehicle plans and executes its route individually.

For brevity, we assume that each driver is bound to his/her private vehicle. Moreover, we assume that each vehicle has a single destination POI. The case study is thus reduced to a scenario where the goal is to reach the destination in time, while visiting charging stations during the trip if necessary. The charging stations may however become unavailable in time and thus it is necessary to introduce monitoring of charging stations and potential re-planning.

2 Invariant refinement

In principle, IRM employs *invariants* to describe a desired state of the system-to-be at every time instant; i.e., to describe the *operational normalcy* of the system-to-be, essential for its continuous operation.

The objective of IRM is to start the refinement with the overall system goal and end up by determining the invariants reflecting detailed design of the particular system constituents – components, component processes, and ensembles.

The core of IRM is a systematic, gradual refinement of a higher-level invariant by means of its decomposition (i.e., structural elaboration) into a conjunction of lower-level sub-invariants. Formally, decomposition of a parent invariant I_p into a conjunction of sub-invariants $I_{s1}, \ldots, I_{sn}$ is a refinement if the conjunction of the sub-invariants entails the parent invariant, i.e., if it holds that:

$$I_{s1} \wedge \ldots \wedge I_{sn} \Rightarrow I_p \quad (entailment)$$
$$I_{s1} \wedge \ldots \wedge I_{sn} \not\Rightarrow false \quad (consistency)$$

This definition complies with the traditional interpretation of refinement, where the composition of the children exhibits all the behavior expected from the parent and (potentially) some more. The

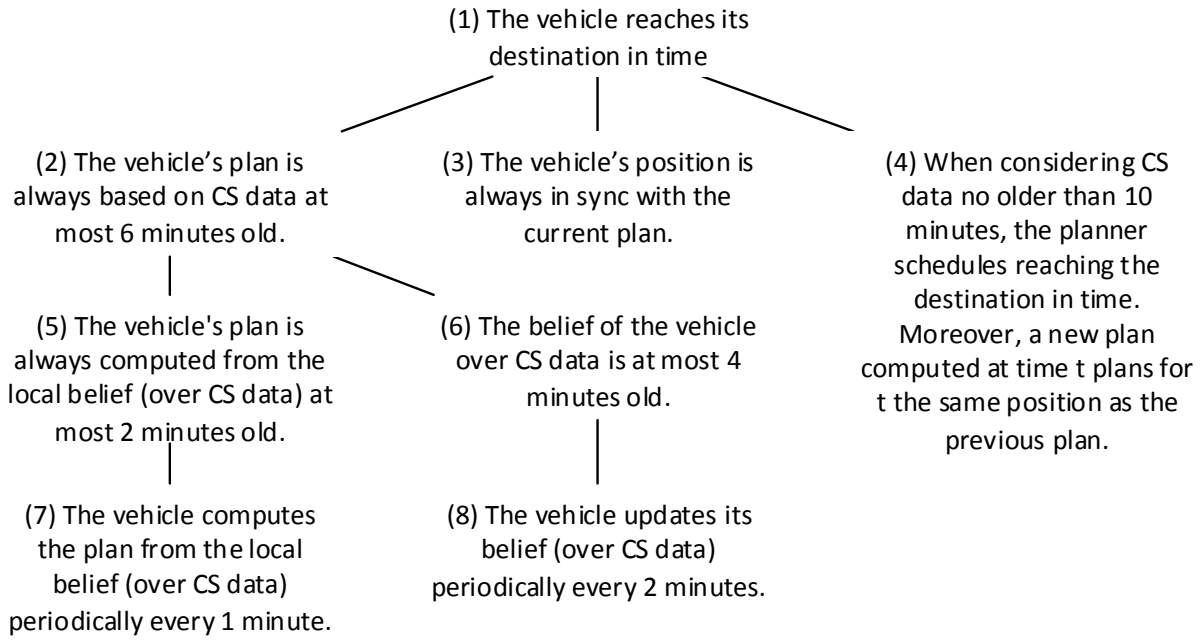

Figure 1: *Invariant refinement of the case study.*

refinement is applied recursively, starting with high-level invariants reflecting the overall system goals and involving a number of components and ending with low-level ones involving a single component or an ensemble of components. Note that since a decomposition step may involve a design decision, it is critical to ensure that this decision complies with the entailment and consistency conditions.

2.1 Invariant refinement of the case study

An invariant-based design of a system targeting the case study is presented in Figure 1. Below, the individual invariants are described in more detail:

(1) *The vehicle reaches its destination in time.* This is the main goal of the scenario.

(2) *The vehicle's plan is always based on CS data at most 6 minutes old.* This expresses a specific requirement on the designed system and the vehicle's planner input in particular. In this context, a plan is a black-box giving for each time instance the expected position of the vehicle at that time.

(3) *The vehicle's position is always in sync with the current plan.* This reflects the assumption, that the plan is always realistic (i.e., that it is actually possible to follow it given the traffic and car characteristics), and that the driver would follow it precisely.

(4) *When considering CS data no older than 10 minutes, the planner schedules reaching the destination in time. Moreover, a new plan computed at time t plans for t the same position as the previous plan (i.e., the two plans intersect in the moment of re-planning).* This expresses the assumption, that charging station availability does not change too quickly and that the initial set-up of the environment is "planning-friendly". Moreover, it expresses a particular assumption about the planner output.

(5) *The vehicle's plan is always computed from the local belief (over CS data) at most 2 minutes old.* A specific system requirement that constraints the input and timing of the planner. In particular, we assume read consistency with respect to the belief (i.e., new plan is always based on the same or newer belief than the previous plan). Moreover, 5 and 6 together represent the design decision of dividing the activity of computing the plan from remote data into two activities of creating a local belief of the remote data and computing the plan from the local belief.

(6) *The belief of the vehicle over CS data is at most 4 minutes old.* A specific system requirement that constraints the timing of charging station monitoring.

(7) *The vehicle computes the plan from the local belief (over CS data) periodically every 1 minute.* A specific system requirement precisely determining the input and timing of the planner. In particular, we assume real-time periodic computation.

(8) *The vehicle updates its belief (over CS data) periodically every 2 minutes.* A specific system requirement precisely determining the timing of CS monitoring. In particular, we assume (distributed) real-time periodic monitoring.

3 Invariant formalization

In general, the goal of invariant-based system design is to formally capture properties of a valid system. Thus, we will first discuss the necessary characteristics of such formalization (i.e., characteristics implied by the domain).

In the domain of (soft) real-time component ensembles, the way of expressing properties of a valid system is, as indicated by the case study, to capture a valid evolution of knowledge values in time. To do that, the underlying formalism has to provide means for referring to knowledge values at arbitrary time instants. When generalized, we can say the formalism needs to refer to timed sequences of knowledge values (i.e., timed streams of data), which provide a complete view on the knowledge value evolution in time.

This is explicitly formalized in the following definitions, where we consider time to be a non-negative real number, i.e., $\mathbb{T} \stackrel{\text{def}}{=} \mathbb{R}_0^+$.

Definition 1. *(Knowledge and its valuation)* Knowledge *is a set $K = \{k_1, \ldots k_n\}$ of knowledge elements, where the domain of k_i is denoted as V_i.* Knowledge valuation *of element k_i is a function $T \rightarrow V_i$ which for each time t yields a value of k_i (denoted $k_i[t]$).*

Definition 2. *(Invariant) An* invariant *is a predicate (in a higher-order predicate logic with arithmetic) over a knowledge valuation and time.*

In general, an invariant may refer to the knowledge valuation in an arbitrary time point/interval.

As further illustrated by the case study, when formalizing system design, it is critical to introduce formal assumptions about the system's environment. Although this is often omitted in informal design approaches, without explicit assumptions the formalized system design is not complete, nor correct. Thus there are two specific types of invariants:

- *System invariants* reflect properties of the individual architectural elements of the system and their validity is to be ensured by the system implementation.
- *Assumptions* reflect the properties of the system environment assumed by the system invariants. Validity of these invariants is usually out of control of the designer and is necessary for correct operation of the implementation.

For example, invariant 2 from the case study is a system invariant while invariant 4 is an assumption.

3.1 Invariant formalization of the case study

With respect to the previous definitions, the case-study design can be formalized as follows (note that this example is presented at this point just to illustrate the significant complexity of such a formalization, it will be fully explained in the following sections):

(1) *The vehicle reaches its destination in time:*

$$\exists t \in \mathbb{T}, t \leq DEADLINE : v.pos[t] = DEST$$

(2) *The vehicle's plan is always based on CS data at most 6 minutes old:*

$$\forall t \in \mathbb{T}, \exists t_t, t_{pos}, t_{charge}, t_1, \ldots, t_n :< t - t_i \leq 6min :$$
$$Plan(t, v.pos[t_{pos}], v.charge[t_{charge}], CS_1[t_1], \ldots, CS_n[t_n], v.plan[t])$$

where $Plan$ predicate denotes post-condition of the planning algorithm (i.e., that $v.plan[t]$ corresponds to the outcome of the planner applied to t, $v.pos[t_{pos}]$, $v.charge[t_{charge}]$, $CS_1[t_1]$, …, $CS_n[t_n]$).

(3) *The vehicle's position is always in sync with the current plan:*

$$\forall t \in \mathbb{T} : v.pos[t] = v.plan[t](t)$$

(4) *When considering CS data no older than 10 minutes, the planner schedules reaching the destination in time. Moreover, a new plan computed at time t plans for t the same position as the previous plan:*

$$\forall t \in \mathbb{T}, \forall t_t, t_{pos}, t_{charge}, t_1, \ldots, t_n : 0 < t - t_i \leq 10min :$$
$$Plan(t_t, v.pos[t_{pos}], v.charge[t_{charge}], CS_1[t_1], \ldots, CS_n[t_n], v.plan[t])$$
$$\Rightarrow$$
$$\exists t' \in \mathbb{T}, t' \leq DEADLINE : v.plan[t](t') = DEST$$

and

$$\forall t, t_{prev} \in \mathbb{T}, t_{prev} < t,$$
$$\forall t' \in (t_{prev}, t) : v.plan[t'] = v.plan[t_{prev}] \neq v.plan[t] :$$
$$v.plan[t_{prev}](t) = v.plan[t](t)$$

(5) *The vehicle's plan is always computed from the local belief (over CS data) at most 2 minutes old.*

$$\exists a_t, a_{pos}, a_{charge}, a_{belief}, a_1 : 0 < x - a_i(x) \leq 2min,$$
$$a(x) \leq a(y) \forall x \leq y, \forall t \in \mathbb{T} :$$
$$Plan(t, v.pos[a_{pos}(t)], v.charge[a_{charge}(t)], v.belief[a_{belief}(t)], v.plan[t])$$

where a_i reflects the read consistency with respect to knowledge element i.

(6) *The belief of the vehicle over CS data is at most 4 minutes old.*

$$\forall t \in \mathbb{T}, \exists t_1, \ldots, t_n, 0 < t - t_i \leq 4min :$$
$$Belief(CS_1[t_1], \ldots, CS_n[t_n], v.belief[t])$$

(7) *The vehicle computes the plan from the local belief (over CS data) periodically every 1 minute.*

$$\exists R, F : \mathbb{N} \to \mathbb{T}, P(x-1) \leq R(x) < F(x) \leq P(x),$$
$$\forall p \in \mathbb{N}, \forall t \in (P(p-1), P(p)) :$$
$$t < F(p) \Rightarrow Plan(R(p-1), v.pos[R(p-1)], v.charge[R(p-1)], v.belief[R(p-1)], v.plan[t])$$
$$t \geq F(p) \Rightarrow Plan(R(p), v.pos[R(p)], v.charge[R(p)], v.belief[R(p)], v.plan[t])$$

where $P(n) : \mathbb{N}_0 \to T = n * 1min$. $R(n)$ and $F(n)$ denote the release and finish time of the real-time process executing the planner in the n-th period.

(8) *The vehicle updates its belief (over CS data) periodically every 2 minutes.*

$$\exists R_1, \ldots, R_n, F : \mathbb{N} \to \mathbb{T}, P(x-1) \leq R_i(x) < F(x) \leq P(x),$$
$$\forall p \in \mathbb{N}, \forall t \in (P(p-1), P(p)) :$$
$$t < F(p) \Rightarrow Belief(CS_1[R_1(p-1)], \ldots, CS_n[R_n(p-1)], v.belief[t])$$
$$t \geq F(p) \Rightarrow Belief(CS_1[R_1(p)], \ldots, CS_n[R_n(p)], v.belief[t])$$

where $P(n) : \mathbb{N}_0 \to \mathbb{T} = n * 1min$. $R_i(n)$ denotes the read time on the i-th charging station, while $F(n)$ denotes the write time on the vehicle of the real-time process executing charging station monitoring in the n-th period.

4 Invariant patterns

In general, the form of invariants is not explicitly restricted. However, at particular levels of abstraction (when describing architectural elements) there are several patterns virtually omnipresent in any invariant-based design. It is thus beneficial to have means for concise and consistent representation of such invariant patterns.

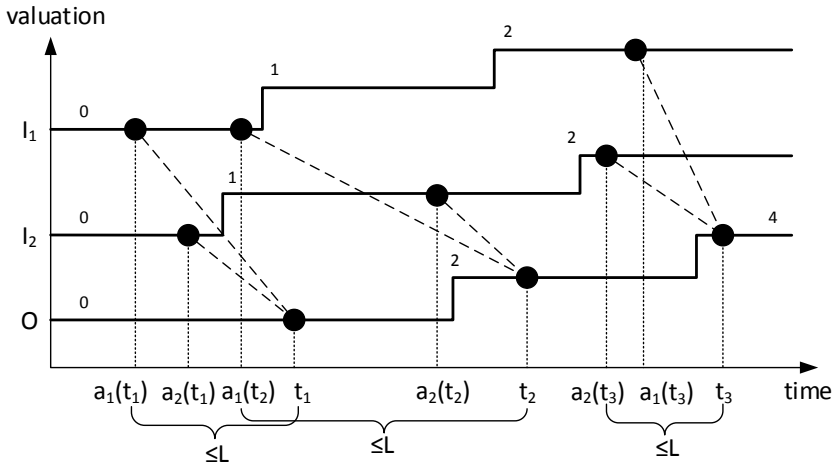

Figure 2: *Illustration of a valid knowledge valuation with respect to an activity where the output O represents sum of inputs I_1 and I_2, while meeting lag L.*

4.1 General invariant

General invariants at the top-level of abstraction capture the operational normalcy in terms of relating the past and current knowledge valuation to a future knowledge valuation. Therefore, a general invariant can have an arbitrary internal structure.

4.2 Present-past invariant

Frequently, the high-level system invariants express that some knowledge is based on knowledge no older than a particular time interval – *lag*. This reflects the fact (abstracted away at the level of general invariants) that software systems cannot cope with future data, but have to depend on current and/or past data instead.

In this case, such invariants typically capture that there is a particular relation (frequently capturing a post-condition P of a computation) between current knowledge and knowledge no older than the lag L. Similar to real-time software control systems, we assume that the smaller the lag, the bigger precision and robustness; lag equal to 0 denotes an idealized case where the beliefs of all components are always up-to-date and their actions are instant.

A concise representation of such an invariant is presented in the following definition.

Definition 3. *(Present-past invariant) For a predicate P capturing the relation between valuation of knowledge elements $I_1, \ldots, I_n$ and $O_1, \ldots, O_m$, and the lag L, the expression $P^L_{p-p}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ denotes the following* present-past invariant*:*

$$\forall t \in \mathbb{T}, \exists t_1, \ldots, t_n : 0 < t - t_i \leq L, i \in \{1, \ldots, n\} : \\ P(I_1[t_1], \ldots, I_n[t_n], O_1[t], \ldots, O_m[t])$$

In this context, we will call $I_1, \ldots, I_n$ the input variables and $O_1, \ldots, O_m$ the output variables of the invariant.

4.3 Activity invariant

Another frequent form of timed invariants, used at a lower level of abstraction, closely reflects properties of a (soft) real-time activity while assuming read consistency with respect to the input knowledge of this activity, i.e., that each output knowledge valuation is based on the same or newer input knowledge valuation than the previous one. This is illustrated in Figure 2. We call such invariants *activity invariants*.

In this case, an activity invariant captures that the output knowledge valuation changes only as a result of performing the activity. Moreover, although reading the input knowledge of the activity, as well as computing and writing the output knowledge, takes some time, it never (altogether) exceeds the corresponding time limit (i.e., lag).

More rigorously, at any time the output knowledge valuation corresponds to the outcome of the activity applied on input knowledge valuation not older than the lag. Moreover, each output is based on newer inputs than the previous output.

This follows the idea that an activity maintains the corresponding operational normalcy periodically by reading the input knowledge, performing the computation and writing the output knowledge.

A concise representation of such an invariant is given by the following definition.

Definition 4. *(Activity invariant) For a predicate P reflecting the post-condition of an activity with inputs $I_1, \ldots, I_n$ and outputs $O_1, \ldots, O_m$, and for lag L, the expression $P^L_{act}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ denotes the following* activity invariant*:*

$$\exists a_1, \ldots, a_n : \mathbb{T} \to \mathbb{T}, 0 < x - a_i(x) \leq L, a_i \textit{ non-decreasing } \forall t \in \mathbb{T}, i \in \{1, \ldots, n\} : \\ P(I_1[a_1(t)], \ldots, I_n[a_n(t)], O_1[t], O_m[t])$$

Here, the non-decreasing function a_i gives for each time t the corresponding time t' such that the valuation of I_i at t' was "used to compute" the valuation of $O_1, \ldots, O_m$ at t, as shown in Figure 2.

4.4 Process invariant

At the lowest level of abstraction (i.e., in the leaves of the invariant decomposition), an activity invariant that captures local computation (i.e., with no distributed knowledge involved) while assuming read consistency is refined into an invariant capturing a periodic real-time component process – a *process invariant*.

The difference to activity invariants lies in the fact that not only may the outputs change strictly as a result of performing the activity and must conform to current-enough inputs, but also that the activity is performed exactly once in each period. In this context, the period is an elaboration of the activity-predicate lag. Specifically, since we assume a component process to be periodic and (soft) real-time, the output knowledge valuation is determined by the release time and finish time of the process in each period [2].

Specifically, such an invariant captures that if the current time is before the finish time of the process in the current period, then the outputs are the same as in the previous period (i.e., they correspond to the inputs used in the previous period). Otherwise, the outputs correspond to the inputs at the release time of the process in this period.

A concise representation of such an invariant is presented in the following definition.

Definition 5. *(Process invariant) For a predicate P reflecting the post-condition of a periodic real-time process with inputs $I_1, \ldots, I_n$, outputs $O_1, \ldots, O_m$, and period L, the expression $P^L_{proc}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ denotes the following* process invariant*:*

$$\exists R, F : \mathbb{N} \to \mathbb{T} : E(x-1) \leq R(x) < F(x) < E(x)\ \forall x \in \mathbb{N}, \\ \forall p \in \mathbb{N}, \forall t \in \langle E(p-1), E(p)) : \\ t < F(p) \Rightarrow P(I_1[R(p-1)], \ldots, I_n[R(p-1)], O_1[t], \ldots, O_m[t]) \\ t \geq F(p) \Rightarrow P(I_1[R(p)], \ldots, I_n[R(p)], O_1[t], \ldots, O_m[t])$$

where $E(n) : \mathbb{N}_0 \to \mathbb{T} = n \cdot L$, i.e., the end of the n-th period. $R(n)$ and $F(n)$ denote the release and finish time of the real-time process in the n-th period.

Note that unlike activity invariants, there is the same R for each I, reflecting that at the release time the process reads all the inputs atomically.

4.5 Ensemble invariant

Similar to a process invariant, an activity invariant at the lowest level of abstraction that captures establishment of a belief (that can be addressed by ensemble knowledge exchange) while assuming distributed read consistency is refined into an invariant capturing periodic knowledge-exchange of an ensemble – an *ensemble invariant*.

The difference to the process invariants lies in the fact that invariants capturing knowledge exchange assume that the input values might have been read at different times, since the inputs are potentially distributed (however, the times have to fit into the same period). Therefore, an ensemble invariant further

accounts for the delay connected with potential transfer of the knowledge over the network (as required in distributed systems). The invariant thus describes a composite computation activity consisting of the knowledge transfer (with an upper time bound on its duration) followed by periodic evaluation of the membership condition and the knowledge exchange. Here, it is assumed that each component executes the incoming knowledge exchange (i.e., knowledge exchange that updates the component's knowledge) on its own, while the other components asynchronously send the required input knowledge. Further, it is assumed that such composite activities may be partially overlapping (mostly in situations when the knowledge transfer takes longer than the period of the knowledge exchange).

A concise representation of such an invariant is presented in the following definition.

Definition 6. *(Ensemble invariant) Let P be a predicate reflecting the post-condition of a periodic knowledge exchange with inputs $I_1, \ldots, I_n$, outputs $O_1, \ldots, O_m$, and period L. Provided that it takes at most T for the knowledge to become available at the component executing the knowledge exchange, the expression $P^{L,T}_{ens}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ denotes the following* ensemble invariant*:*

$$\begin{gathered}\exists a_1, \ldots, a_n : \mathbb{T} \to \mathbb{T}, 0 < x - a_i(x) \leq T, a_i \textit{ non-decreasing } \forall t \in \mathbb{T}, i \in \{1, \ldots, n\} : \\ \exists R, F : \mathbb{N} \to \mathbb{T} : E(x-1) \leq R(x) < F(x) < E(x)\ \forall x \in \mathbb{N}, \\ \forall p \in \mathbb{N}, \forall t \in \langle E(p-1), E(p)) : \\ t < F(p) \Rightarrow P(I_1[a_1(R(p-1))], \ldots, I_n[a_n(R(p-1))], O_1[t], \ldots, O_m[t]) \\ t \geq F(p) \Rightarrow P(I_1[a_1(R(p))], \ldots, I_n[a_n(R(p))], O_1[t], \ldots, O_m[t])\end{gathered}$$

where $E(n) : \mathbb{N}_0 \to \mathbb{T} = n \cdot L$, i.e., the end of the n-th period. $R(n)$ and $F(n)$ denote the release and finish time of the real-time knowledge exchange in the n-th period. Finally, a_i gives for each time t the corresponding time t' such that the valuation of I_i that was available to the component executing the knowledge exchange at t was sent to the component at t'.

Note, that there is a (potentially) different a_i for each I_i, reflecting that the inputs can be sent to the component executing the knowledge exchange at different times. Moreover, there is the same t for each O_i, which corresponds to the assumption, that knowledge exchange is unidirectional, i.e., it writes only into the knowledge of one component, and thus the writes can be atomic.

4.6 Case study revisited

Using the above-defined invariant patterns, the case-study invariants 2, 4 - 8 can be rewritten as follows (the patterns are not applicable for invariants 1 and 3).

(2) *The vehicle's plan is always based on CS data at most 6 minutes old:*

$$Plan^{6min}_{p-p}[t, v.pos, v.charge, CS_1, \ldots, CS_n][v.plan]$$

where in the expanded form of this expression $t[t']$ denotes t'.

(4) *When considering CS data no older than 10 minutes, the planner schedules reaching the destination in time. Moreover, a new plan computed at time t plans for t the same position as the previous plan:*

$$\begin{gathered}Plan^{10min}_{p-p}[t, v.pos, v.charge, CS_1, \ldots, CS_n][v.plan] \\ \Rightarrow \exists t' \in \mathbb{T}, t' \leq DEADLINE : v.plan[t](t') = DEST\end{gathered}$$

The second part of the invariant is the same as in the original case.

(5) *The vehicle's plan is always computed from the local belief (over CS data) at most 2 minutes old.*

$$Plan^{2min}_{act}[t, v.pos, v.charge, v.belief][v.plan]$$

(6) *The belief of the vehicle over CS data is at most 4 seconds old.*

$$Belief^{4min}_{p-p}[CS_1, \ldots, CS_n][v.belief]$$

(7) *The vehicle computes the plan from the local belief (over CS data) periodically every 1 minute.*

$$Plan^{1min}_{proc}[t, v.pos, v.charge, v.belief][v.plan]$$

(8) *The vehicle updates its belief (over CS data) periodically every 2 minutes.*

$$Belief_{ens}^{2min}[CS_1, \ldots, CS_n][v.belief]$$

It is obvious that usage of invariant patterns particularly simplifies the lower-level, more technical invariants that capture computation activities. This allows for more concise and consistent invariant-based design.

5 Correctness by construction

A simplification of invariant-based design is not the only benefit of using the invariant patterns during invariant-based design. The main advantage is the ability of formal reasoning on the level of patterns instead of reasoning on the level of predicate logic upon knowledge valuations (since state-of-the-art theorem provers for such complex logics still do not have the necessary performance).

Thus, we propose a formal framework allowing for formal reasoning on the level of invariant patterns.

5.1 Basic pattern relations

First, we elaborate on the basic relations of the invariant patterns which correspond to the natural relations among the related software concepts of activity/activity with read consistency/process/ensemble.

A straightforward observation for a present-past invariant is that, given a particular knowledge valuation, if the outputs are always based on inputs within the given time limit, increasing the limit maintains this property. A similar observation holds for activity invariants. This is formalized in the following theorem.

Theorem 1. *(Maximal lag refinement) For $K \leq L$:*

$$P_{p-p}^{K}[I_1, \ldots, I_n][O_1, \ldots, O_m] \Rightarrow P_{p-p}^{L}[I_1, \ldots, I_n][O_1, \ldots, O_m]$$
$$P_{act}^{K}[I_1, \ldots, I_n][O_1, \ldots, O_m] \Rightarrow P_{act}^{L}[I_1, \ldots, I_n][O_1, \ldots, O_m]$$

Proof. A direct corollary of the lag/activity invariant definition. In particular, the existence of t_i such that $0 < t - t_i \leq K$ in $P_{p-p}^{K}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ guarantees the existence of t_i such that $0 < t - t_i \leq L$ in $P_{p-p}^{L}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ (similarly for a_i and $0 < x - a_i(x) \leq L$). □

One can also observe that the requirement of read consistency of inputs in addition to the time limit (in activity invariants) is a stronger requirement than the time limit only (in present-past invariants); this is formalized in the following theorem.

Theorem 2. *(Activity invariant implies present-past invariant) Assuming that $I = I_1, \ldots, I_n$ and $O = O_1, \ldots, O_m$, it holds:*

$$P_{act}^{L}[I][O] \Rightarrow P_{p-p}^{L}[I][O]$$

Proof. The existence of $t_1, \ldots, t_n$ for $P_{p-p}^{L}[I][O]$ is given by $a_1, \ldots, a_n$ of $P_{act}^{L}[I][O]$. In particular, $\forall t$ we set $t_i = a_i(t)$. □

A similar theorem can be formulated for the process and activity invariants. Here, the idea is that, in reality, a periodic process is actually a strict refinement of an activity with read consistency and time limit on input data. However, instead of considering the same time limit for both invariants as in previous cases, the activity invariant needs twice the time limit of the process invariant. This also complies to the well-known fact in the area of real-time scheduling: in order to achieve a particular end-to-end response time with a real-time periodic process, the period needs to be at most half of the desired response time [2]. For our invariant patterns, this fact is formalized in the following theorem.

Theorem 3. *(Process invariant implies activity invariant) Assuming that $I = I_1, \ldots, I_n$ and $O = O_1, \ldots, O_m$, it holds:*

$$P_{proc}^{L}[I][O] \Rightarrow P_{act}^{2L}[I][O]$$

Proof. Without loss of generality let us assume that $|I| = |O| = 1$. Given $t \in \mathbb{T}$ let $p = \lceil \frac{t}{L} \rceil$. The required $a : \mathbb{T} \to \mathbb{T}$ for $P^{2L}_{act}[I][O]$ is given by R and F from $P^{L}_{proc}[I][O]$ as follows:

$$a(t) = \begin{cases} R(p-1) & \text{if } t < F(p) \\ R(p) & \text{if } t \geq F(p) \end{cases}$$

First, we prove that $0 < t - a(t) \leq 2L$. Since $p = \lceil \frac{t}{L} \rceil$, then also $(p-1) \cdot L \leq t \leq p \cdot L$. According to Definition 5, $E(p-1) \leq R(p) < F(p) \leq E(p)$, where $E(p) = p \cdot L$. Therefore, given the properties of R, F, and $a(t)$, we have $E(p-2) \leq R(p-1) \leq a(t)$ and $a(t) < t$. Together, we have $(p-2) \cdot L \leq a(t) < t \leq p \cdot L$. Therefore, $0 < t - a(t) \leq 2L$.

Further, a is non-decreasing since R and F are non-decreasing. Thus, from $P^{L}_{proc}[I][O]$ we get $P^{2L}_{act}[I][O]$. □

Similarly, it holds that the exchange invariant pattern is a refinement of the activity invariant pattern with lag equal twice the period of the exchange invariant pattern plus the time for distributed transfer of the knowledge, as formulated by the following theorem.

Theorem 4. *(Process invariant implies activity invariant) Assuming that $I = I_1, \ldots, I_n$ and $O = O_1, \ldots, O_m$, it holds:*

$$P^{L,T}_{ens}[I][O] \Rightarrow P^{2L+T}_{act}[I][O]$$

Proof. The proof is similar to Theorem 3, differing only in the part relevant to knowledge transfer over network. For the purpose of the proof, we denote $R_i(p) = a_i(R(p)), \forall p \in \mathbb{N}$ for R and a_i from $P^{L,T}_{ens}[I][O]$.

Given $t \in \mathbb{T}$ let $p = \lceil \frac{t}{L} \rceil$. The required $a_i : \mathbb{T} \to \mathbb{T}$ for $P^{2L+T}_{act}[I][O]$ is given by R_i and F from $P^{L,T}_{ens}[I][O]$ as follows:

$$a_i : (t) = \begin{cases} R_i(p-1) & \text{if } t < F(p) \\ R_i(p) & \text{if } t \geq F(p) \end{cases}$$

First, we prove that $0 < t - a_i(t) \leq 2L + T$. Since $p = \lceil \frac{t}{L} \rceil$, then also $(p-1) \cdot L \leq t \leq p \cdot L$. According to Definition 6, $E(p-1) - T \leq R(p) - T \leq R_i(p) < F(p) \leq E(p)$, where $E(p) = p \cdot L$ (recall that $x - a^{ens}_i(x) \leq T$). Therefore, given the properties of R_i, F, and $a(t)$, we have $E(p-2) - T \leq R_i(p-1) \leq a(t)$ and $a(t) < t$. Together, we have $(p-2) \cdot L - T \leq a(t) < t \leq p \cdot L$. Therefore, $0 < t - a(t) \leq 2L + T$.

Further, a_i is non-decreasing since R_i and F are non-decreasing. Thus, from $P^{L,T}_{ens}[I][O]$ we get $P^{2L+T}_{act}[I][O]$. □

5.2 Pipeline decomposition

Here, we present a logical framework that would enable for formal reasoning about refinement in a particular form of decomposition – *pipeline decomposition*. Specifically, we focus on the level of activity predicates, as they represent a suitable level of abstraction, generalizing both process and ensemble predicates.

As an important observation, the fact that a decomposition is actually a refinement of the parent predicate is, with respect to time, largely affected by sharing of predicate variables among the child predicates. Thus, we introduce the concept of *dependency chain*. A vector of activity predicates forms a dependency chain if some of the output variables of a predicate in the vector are among the input variables of the next predicate in the vector. This is formalized in the following definition.

For brevity, we introduce the following notation. Given an activity (lag/process/ensemble) invariant $P^{L}_{act}[I_1, \ldots, I_n][O_1, \ldots, O_m]$, $In(P)$ denotes the set $\{I_1, \ldots, I_n\}$, while $Out(P)$ denotes the set $\{O_1, \ldots, O_m\}$.

Definition 7. *(Dependency chain) Each vector $\left(P_1{}^{L_1}_{act}, \ldots, P_{p}{}^{L_p}_{act}\right)$ of invariants forms a* dependency chain *iff:*

$$\forall i \in \{1, \ldots, p-1\}\, \exists O, I :$$
$$O \in Out(P_i) \wedge I \in In(P_{i+1}) \wedge O = I$$

In fact, the more dense sharing of variables among child predicates the harder it is to determine whether the decomposition is actually a refinement solely on the predicate-pattern level (without interpreting the semantics of the predicates).

Thus, we start with the simplest practically-relevant form of decomposition – *pipeline decomposition*. In a pipeline decomposition the children reflect simple pipeline-like flows among the corresponding activities that refine the parent activity. A formal interpretation is given in the following definition.

Definition 8. *(Pipeline decomposition) Having a parent invariant $P^L_{act}[I_1, \ldots, I_n][O_1, \ldots, O_m]$, a set of child invariants $\left\{ P_i{}^{L_i}_{act}[\ldots][\ldots], i = 1..p \right\}$ forms a* pipeline decomposition *of P^L_{act} iff:*

(i) each input variable of the parent is an input variable of exactly one child:

$$\forall I \in In(P)\ \exists! j \in \{1, \ldots, p\} : I \in In(P_j)$$

(ii) either all input variables of a child are input variables of the parent, or the child has only one input variable which is at the same time the only output variable of exactly one other child:

$$\forall i \in \{1, \ldots, p\} :$$
$$(In(P_i) \subseteq In(P)) \vee \begin{pmatrix} \exists! i' \in \{1, \ldots, p\}, i \neq i' : \\ |In(P_i)| = 1 \wedge In(P_i) = Out(P_{i'}) \end{pmatrix}$$

(iii) each output variable of the parent is an output variable of exactly one child:

$$\forall O \in Out(P)\ \exists! j \in \{1, \ldots, p\} : O \in Out(P_j)$$

(iv) either all output variables of a child are output variables of the parent, or the child has only one output variable which is at the same time the only input variable of exactly one other child:

$$\forall i \in \{1, \ldots, p\} :$$
$$(Out(P_i) \subseteq Out(P)) \vee \begin{pmatrix} \exists! i' \in \{1, \ldots, p\}, i \neq i' : \\ |Out(P_i)| = 1 \wedge Out(P_i) = In(P_{i'}) \end{pmatrix}$$

An example is the decomposition of (2) into (5) and (6) in the case study.

Intuitively, the definition of pipeline decomposition requires the children to reflect simple pipeline-like flows among the corresponding activities that refine the parent activity.

For pipeline decomposition, a straightforward rule for determining refinement can be formulated. In a correct refinement, provided that the decomposition is logically consistent with the parent predicate when not considering time, the lag of the parent predicate should be at least the sum of the lags of the predicates in the longest (in terms of time) pipeline (i.e., dependency chain) of the decomposition. Indeed, this intuitive observation confirmed in our predicate formalism as demonstrates the following theorem.

Theorem 5. *(Activity invariant pipeline refinement) Having invariant $P^L_{act}[I_1, \ldots, I_n][O_1, \ldots, O_m]$ and its pipeline decomposition $\mathcal{D} = \left\{ P_1{}^{L_1}_{act}, \ldots, P_{p}{}^{L_p}_{act} \right\}$, the decomposition is a refinement of the parent, i.e., it holds that $P_1{}^{L_1}_{act} \wedge \cdots \wedge P_{p}{}^{L_p}_{act} \Rightarrow P^L_{act}$, if:*

(i) $P_1 \wedge \cdots \wedge P_p \Rightarrow P$, i.e., the decomposition is logically consistent without considering time

(ii) for each dependency chain $\mathcal{C} = \left(P_{i_1}{}^{L_{i_1}}_{act}, \ldots, P_{i_q}{}^{L_{i_q}}_{act} \right)$ in $\mathcal{D}$ it holds $\sum_{n=i_1}^{i_q} L_n \leq L$, i.e., the lag of the parent invariant is at least the sum of the lags of the longest (in terms of time) dependency chain among the child invariants.

Proof. To prove the above theorem, we need to prove that given $\mathcal{D}$, P, and the assumptions (i) and (ii), the following lemma holds:

$$P_1{}^{L_1}_{act} \wedge \cdots \wedge P_{p}{}^{L_p}_{act} \Rightarrow (P_1 \wedge \cdots \wedge P_p)^L_{act}$$

Then, the correctness of the theorem is an immediate result of this lemma and the assumption (i). To prove the lemma, let $Q^L_{act} \stackrel{\text{def}}{=} (P_1 \wedge \cdots \wedge P_p)^L_{act}$.

From the assumption of pipeline decomposition, we get that each input variable of $P_i, i = 1..p$ is either an input variable of Q, or the only output variable of exactly one $P_j, i \neq j$; similar for output variables. This means that for each dependency chain $\mathcal{C} = \left(P_{i_1}{}^{L_{i_1}}_{act}, \ldots, P_{i_q}{}^{L_{i_q}}_{act} \right)$ in $\mathcal{D}$, all the input variables of $P_{i_1}{}^{L_{i_1}}_{act}$ are input variables of Q^L_{act}, while all the output variables of $P_{i_q}{}^{L_{i_q}}_{act}$ are output variables of Q^L_{act}. Also, for each $P_m{}^{L_m}_{act}, P_n{}^{L_n}_{act}, m \in i_1, \ldots, i_{q-1}, n = m + 1$, $P_m{}^{L_m}_{act}$ has exactly one output variable which it at the same time the one input variable of $P_n{}^{L_n}_{act}$.

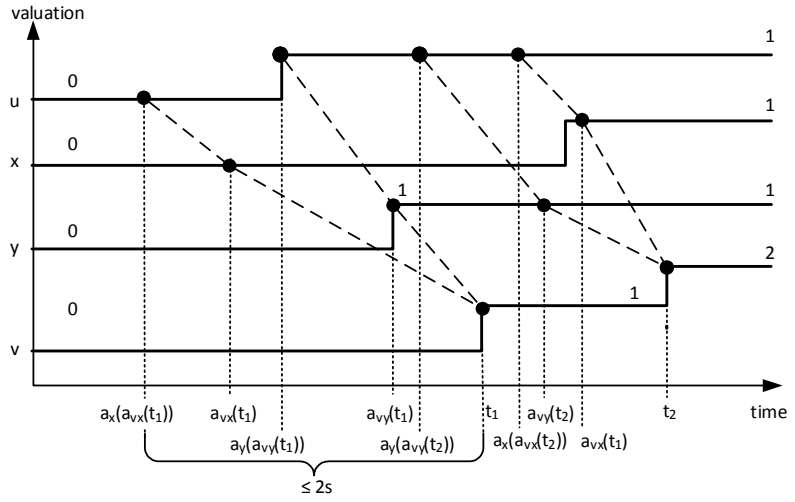

Figure 3: *A counterexample illustrating the importance of the pipeline refinement assumption in Theorem 5.*

Without loss of generality, let us assume that the dependency chain $\mathcal{C}$, its first invariant ${P_{i_1}}_{act}^{L_{i_1}}$ in particular, has only one input variable (i.e., $I_\mathcal{C}$). Also, let us assume that $\mathcal{C}$, its last invariant ${P_{i_q}}_{act}^{L_{i_q}}$ in particular, has only one output variable (i.e., $O_\mathcal{C}$). This assumption is safe since the multiple input/output variables can be merged into one as they are referred exclusively in $\mathcal{C}$.

For the variable $I_\mathcal{C}$, we define the $a_\mathcal{C} : \mathbb{T} \to \mathbb{T}$ required for Q_{act}^{L} (according to the Definition 4) as follows:

$$a_\mathcal{C}(t) \stackrel{\text{def}}{=} a_{i_1}\left(a_{i_2}\left(\dots a_{i_q}\left(t\right)\dots\right)\right)$$

where $a_{i_1}, \dots, a_{i_q}$ are taken from to ${P_{i_1}}_{act}^{L_{i_1}}, \dots, {P_{i_q}}_{act}^{L_{i_q}}$.

Because $\sum_{n=i_1}^{i_q} L_n \leq L$ and $0 < x - a_{i_1}(x) \leq L_{i_1}, \dots, 0 < x - a_{i_q}(x) \leq L_{i_q}$, it holds that $0 < x - a_\mathcal{C} \leq L$.

The assumption of the above lemma (i.e., ${P_1}_{act}^{L_1} \wedge \dots \wedge {P_p}_{act}^{L_p}$) and the properties of the dependency chain $\mathcal{C} = \left({P_{i_1}}_{act}^{L_{i_1}}, \dots, {P_{i_q}}_{act}^{L_{i_q}}\right)$ give us the following corollary:

$$\begin{gathered}
P_{i_1}(I_\mathcal{C}[a_{i_1}(a_{i_2}(\dots a_{i_q}(t)\dots))], O_{i_1}[a_{i_2}(\dots a_{i_q}(t)\dots)]) \wedge O_{i_1} = I_{i_2} \wedge \\
P_{i_2}(I_{i_2}[a_{i_2}(a_{i_3}(\dots a_{i_q}(t)\dots))], O_{i_2}[a_{i_3}(\dots a_{i_q}(t)\dots)]) \wedge O_{i_2} = I_{i_3} \wedge \\
\vdots \\
P_{i_q}(I_{i_q}[a_{i_q}(t)], O_\mathcal{C}[t])
\end{gathered}$$

By combining these corollaries for each dependency chain in the pipeline decomposition $\mathcal{D}$ of Q (i.e., each input and output variable of Q), we get:

$$Q\left(I_1\left[a_1\left(t\right)\right], \dots, I_n\left[a_n\left(t\right)\right], O_1\left[t\right], \dots O_n\left[t\right]\right)$$

where I_i, O_i, and a_i correspond to the dependency chain $\mathcal{C}_i$ in $\mathcal{D}$.

By combining all the above facts, we get:

$${P_1}_{act}^{L_1} \wedge \dots \wedge {P_p}_{act}^{L_p} \Rightarrow Q_{act}^{L}$$

□

5.3 More complex types of refinement

The assumption of pipeline decomposition in the previous theorem is essential for its correctness. To support this claim, we present the following counterexample.

Counterexample. Consider the parent invariant $P_p \stackrel{\text{def}}{=} (v = 2u)^{2s}_{act}[u][v]$, that is decomposed into three sub-invariants:

$$P_x \stackrel{\text{def}}{=} (x = u)^{1s}_{act}[u][x]$$
$$P_y \stackrel{\text{def}}{=} (y = u)^{1s}_{act}[u][y]$$
$$P_z \stackrel{\text{def}}{=} (v = x + y)^{1s}_{act}[x, y][v].$$

This decomposition is not a pipeline decomposition, because of the dependencies $P_x \stackrel{x}{\rightarrow} P_b$ (reads of P_b depend on P_x because of sharing x) and $P_y \stackrel{y}{\rightarrow} P_b$.

If we dropped the requirement of pipeline decomposition in Theorem 5, the theorem would ensure that this decomposition is a refinement.

However, if we consider the trace illustrated in Figure 3, it is obvious that although the trace is valid for all the sub-invariants P_x, P_y, and P_z, it is not valid for the parent invariant P_p, as there cannot be an $a_p(t)$, such that $v[t_1] = 1 = 2 * u[a_p(t_1)]$.

That means that although the theorem claims that the decomposition forms a correct refinement, the knowledge valuation in Figure 3 shows that it does not.

□

The reason why the theorem does not work for the counterexample is that while the parent works with the valuation of a at a single time instant, the decomposition employs the valuation of a at two different time instants (by aliasing to x and y). Indeed, this observation applies in general. Moreover, for some decompositions it is not even possible to formulate similar theorem.

This observation has two important consequences for our future work. Firstly, we would like to find patterns of decomposition, for which no such theorem can be formulated and thus which should be avoided. Next, we would like to extend the set of decomposition patterns for which the previous theorem holds.

References

[1] Dhaminda B Abeywickrama, Nicola Bicocchi, and Franco Zambonelli. SOTA: Towards a General Model for Self-Adaptive Systems. In *Proc. of 21st WETICE*, pages 48–53. IEEE, 2012.

[2] Giorgio C Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 3rd edition, 2011.

[3] Nikola Serbedzija, Stephan Reiter, Maxmilian Ahrens, Jose Velasco, Carlo Pinciroli, Nicklas Hoch, and Bernd Werther. Requirement Specification and Scenario Description of the ASCENS Case Studies. Deliverable D7.1, 2011. Available online: `<http://www.ascens-ist.eu/deliverables>`.