1 Characterizations, Architecture, and Interface

This documents describes the general, conceptual architecture of the DynaCon framework. The lead responsible for this deliverable is Siemens.

2 Introduction

Cyber-physical systems (CPS) must adapt to various changes of exogenous or endogenous conditions, e.g., failures of system components, or varying environmental situations. This adaptation to change can be handled through re-configuration by knowledge-based reasoning systems. However, these systems are usually part of a static environment, and need to be embedded into a dynamic setting of CPS in a cloud/fog environment with sensors creating streamed data.

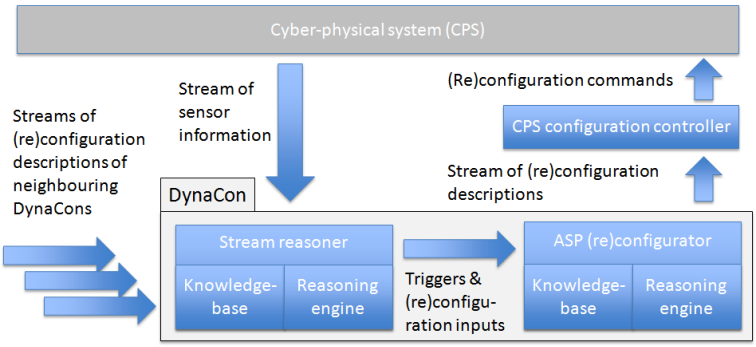


Figure 1: DynaCon Architecture of the Proposal

As shown in Figure 1, the stream of sensor information generated in the CPS environment is observed by the stream reasoner (SR). The SR performs temporal reasoning and eventually triggers the re-configurator by sending the required inputs. Based on these inputs, the re-configurator generates a stream of configurations, which are passed to the a configuration controller. The controller analyses the differences between the current resp., new configurations and generates the required commands to reconfigure the CPS. DynaCon extends also naturally to a multi-agent scenario, where different autonomous CPS interact with each other.

We start with the initial architecture of the proposal, as shown in Figure 1, and re-design the architecture by developing more characterizations of a CPS based on the requirements of the use cases.

3 Characterizations

3.1 Cyber-Physical System

The field of CPS started with an early definition worded at the CPS-Summit in 2008 as:[1] "Such systems use computations and communication deeply embedded in and interacting with physical processes to add new capabilities to physical systems. These CPS range from minuscule (pace makers) to large-scale (the national power-grid)."

The above description is still generic, as a consequence several refinements were published the following years. For instance, in the EU 2020 project CyPhERS,[2] a more detailed distinction was elaborated, where the authors of CyPhERS started with the analysis of the five domains manufacturing, health, smart grid, transportation, and smart cities, which lead to the derivation of following generic CPS characteristics (Törngren et al. 2017):

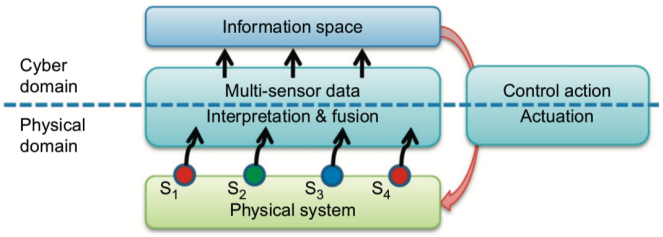
- *Technical emphasis*: The aim of CPS is the integration of physical and embedded systems with communication and software (IT) systems, which leads to (a) the approach how physical and embedded system parts are interacting, and (b) how communication is used to distribute the processing.
- *Cross-cutting aspects*: This characteristics includes systems properties (i.e., safety), jurisdiction (i.e., standards), and governance (i.e., operations).
- *Level of automation (LoA)*: The LoA describes which activities are automated and to what degree the automation reaches, since CPS should act to some extend independently of humans.
- *Life-cycle integration*: The characteristics describes the integration into existing products, services, and (IT) systems.

DynaCon focuses on the technical aspects of CPS; the other three characteristics are out of scope of the project. The above description of technical aspects is still too generic and an important step in CPS-related research is the development of a formal representation and analysis based on mathematical constructs.

The authors of (Sarkar et al. 2017) boil down a CPS to a monitor, control, and actuation cycle, where a CPS is embedded into the information flow of they cycle, and a separation between the cyber and physical domain exists. This suggests that monitoring and control can be formally represented. The authors also hilight the need for the interpretation and fusion of the (analog) sensor information generated in the physical domain.

Bagade et al. provide a good classification of formal representations that can be used to describe a CPS (Bagade, Banerjee, and Gupta 2017):

[1] https://cps-vo.org/cps-summit/2008
[2] http://www.cyphers.eu/



CPS: combing physical space with information space via computing, communication and control.

Sarkar, S., 2011. Autonomous Perception and Decision Making in Cyber-Physical Systems. PhD thesis, The Pennsylvania State University Graduate School, State College, PA 16802.

Figure 2: Monitoring and Control in a CPS

- *Logical modeling*: Different classes of temporal logics such as linear temporal logic (LTL) (Pnueli 1977), metric temporal logic (MTL) (Koymans 1990), or signal temporal logic (STL) (Donzé et al. 2012) are used to model abstract constraints of dynamical systems using a qualitative notion of time.
- *State-based modeling*: State-based models (also called automata) are one of the standard formal constructs used in control theory and were lifted to CPS, where the CPS is described by control states and dynamic state transitions. One example are the frequently used hybrid automata (Henzinger 2000), which are capable of combining discrete computing models with continuous variables.
- *Information theory*: CPS can be understood as a communication system, where the information content used by a process is estimated using the well-know Shannon's equation (Shannon 1948).
- *Multi-agent representation*: A CPS can be represented by a set of entities that interact using a form of control message exchange. A controller in each entity receives the control messages and computes a decision in order to minimize a global objective.

Besides the logical modeling, which gives the formal constructs for stream reasoning and re-configuration modules, the multi-agent representation, which we will call multi-context system, can be envisioned to describe our DynaCon architecture.

3.2 Multi-context System

First, we model a CPS from a KR&R perspective, where multi-context systems (MCS) (Brewka and Eiter 2007) and managed MCS (Brewka et al. 2011) are

a suitable framework to capture the cyber domain of a CPS. MCS are a well-established approach for describing the interaction between different, independent systems using bridge rules. MCS keep the different formalism and knowledge bases (called contexts) as they are, and use bridge rules to model the information flow among them. A bridge rule is similar to logic programming rule (as used in Datalog) with the extension of allowing access to the other contexts in the rule body. The bridge rules can add new knowledge or revise existing knowledge in the context.

We introduce a MCS framework for a CPS, where we define a separation of concerns with the following four contexts:

- *Producers* are contexts that generate information. Sensors that produce data streams collected in the CPS environment can be viewed as such;
- *Monitors* are contexts that observe and aggregate data streams from producers and feed information to the configurators, e.g., a C-ITS stream reasoner (SR);
- *Configurators* are contexts that calculate the setup through re-configuring the CPS, e.g., a signal phase configurator.
- *Actuators* are contexts that change the setup in the CPS environment according to the out of the configurators.

Depending on the situation, the separation of concerns can be weakened, and actuators become an integral part of the producers.

3.3 Classification of the Use Cases

First, we differentiate the use cases according to generic CPS requirements. Then, we refine some criteria in more detail.

3.3.1 CPS Requirements

We evaluate our use cases regarding "generic" requirements for CPS. For this, we describe in "Fog/cloud" the existence of a fog- and/or cloud-environment. In "Sense", we evaluate the (existing) deployment of sensors in the field and the need for sensor fusion and analog-to-digital conversion. In "Com", we describes the means of communication between fog and cloud. In "Ctrl", we describes the type of controller that is needed, and in "Agg" the possibility of *aggregate effects* that is also called anharmonic oscillation in control theory. Some use cases have currently open-loop control operations, which means that user intervention is needed to re-configure the systems. Finally, in "Agent" we describe whether several DynaCon systems could co-exist in an multi-agent setting. Following, the evaluation for each us case of the above criteria:

Use Case	Fog/cloud	Sense	Com	Ctrl	Agg	Agent

Use Case	Fog/cloud	Sense	Com	Ctrl	Agg	Agent
Kelag	Both	Y	Radio[3]	Discrete/continuous, open-loop	N	N
LTE	Cloud	N	Files[4]	Discrete, open-loop	N	N
Net4You	Both	N	Files[5]	Discrete, open-loop	N	N
Siemens C-ITS	Both	Y	V2X[6]	Discrete/continuous, closed-loop	Y	Y
Siemens CPPS	Both	Y	ProfiNet[7]	Discrete/continuous, closed-loop	Y	Y

The criteria of the fog/cloud-environment differs in each use case and will be worked out in the next section. We assume that communication and sensors are the means that generate (data) streams, so we will discuss both criteria in the section on streaming.

The controller in a normal CPS setting needs to (re)act on continuous sensor data. As shown in Figure 1, we have a stream reasoner placed in-between, which transforms the continuous into discrete values, which we call also aggregated values. Differing from a normal CPS setting, we still shave a controller in-place, but the DynaCon controller (which we call the controller module) is coupled with a re-configurator and optionally with a decision module. Therefore, the control actions are calculated by the re-configurator and encoded in use-case specific commands by the controller module.

The first three use cases have currently open-loop control operations, which means that user intervention is needed to re-configure the systems. One aim of the DynaCon project is to support closed-loop control operations with less user intervention. Note that aggregation effects only occur in the closed-loop systems.

Multi-agent systems are out of the scope of this project due to their increased complexity. The last two use cases could be extended to a multi-agent environment.s

[3] Proprietary communication protocol
[4] Database exports
[5] Access to the log files of controlled servers
[6] https://en.wikipedia.org/wiki/Vehicle-to-everything
[7] https://en.wikipedia.org/wiki/PROFINET

3.3.2 System Topology

In a specific fog/cloud-environment, we distinguish between different types of topologies, where $n \leftrightarrow m$ states that n many fog (also called edge) devices communicate with m many command and control (C&C) servers in the cloud:

Use Case	Topology	Comments
Kelag	$n \leftrightarrow 1$	n edge devices directly report to one C&C server
LTE	$1 \leftrightarrow 1$	Only direct user input on the server
Net4You	$n \leftrightarrow 1$	n web servers with one C&C server
Siemens C-ITS	$n \leftrightarrow 1$, $n \leftrightarrow m$	n RSUs reporting to one or more traffic C&C servers
Siemens CPPS	$n \leftrightarrow 1$, $n \leftrightarrow m$	n edge devices report to one or more C&C servers

From the above evaluation, we can conclude that the LTE use-case is not a "classical" CPS; the fog-environment is only indirectly connected to the cloud-side, since the input is loaded via an (log) database. The other use cases, to a different degree, have (a) a separation between a physical and cyber domain, and (b) different types of topologies, whereas the Siemens use-cases will include the most complex settings, where we might have $n \leftrightarrow m$ units interacting with each other.

3.3.3 Streaming Characteristics

The authors of (Stonebraker, Çetintemel, and Zdonik 2005) introduced eight requirements for real-time stream processing systems, which were re-evaluated in 2017 by (Dell'Aglio et al. 2017). The characteristics *volume*, i.e., handling large quantities of data, *velocity*, i.e., streaming data that is fast changing and updated, and *latency*, i.e., reacting by giving answers in a timely fashion, do not need more clarification. *Incompleteness* and *noise* describe the occurrence of missing or erroneous data due to the malfunctioning of sensors or communications links. "DM" captures the complexity of the domain model, which could include the description of topologies, processes, and topic models. We ignore the characteristic *user intention*, we will do not cover direct user involvement with the system, since this is only in the LTE use case of relevance.

Use Case	Volume	Velocity	Variety	Latency	Incomplete	Noise	DM
Kelag	Low	Med	Low	Med	Yes	Yes	Med
LTE	Med	Low	Med	High	Yes	Maybe	High
Net4You	High	High	Low	Low	Yes	Yes	Low
Siemens C-ITS	High	High	Med	Low	Yes	Yes	Med
Siemens CPPS	Med/High	High	High	Med	Yes	Yes	High

The streaming characteristics are just an initial evaluation. Details need to be worked out in use case. This preliminary evaluation shows that we have slow changing, low volume streams as in the LTE use case, but also high volume and velocity streams that require low latency networks as in C-ITS use case.

4 Architectural and Modules

In Figure 3, we give an overview of our extended DynaCon architecture involving several producers (P) and actuators (A), two monitors (M), and a single re-configurator (C). One concern/task can combine various components such as stream reasoners, controller modules, re-configurator, and domain model. The component boxes and the communication flow arrows marked blue are generic features of our design. These components are use case independent, i.e., the stream reasoners could be build on top of an "off-the-shelve" engine such as Hexlite (Fink et al. 2013). The communication protocol is standardized over all covered use-cases and described in Deliverable 5.1. The component boxes marked green are use-case specific, hence a custom implementation will be provided. The dashed boxes are optional components and might be ignored in certain use cases. Furthermore, in some use cases the entire cloud-side is missing, hence the controller and re-configurator is directly activated by a user.

Figure 1 also illustrates the communication channels that are covered in detail in Deliverable 5.1. The communication between fog and cloud can be adapted according to the each use case with the following three options:

(a) Direct calls between the fog/cloud components are possible, since they are located on the same system, e.g., the Kelag use case. This allows the merging of the SR and re-configurator into a single component.

(b) In a distributed environment, we intend to use a simple protocol such as UDP, for the communication between cloud and fog, e.g., the Siemens C-ITS use case. SR and re-configurator are then separate components.

(c) No fog components exists, hence the user interacts directly with controller and re-configurator, e.g., the LTE use case.

4.1 Stream Reasoner Module

The SR has the overall concern of making the input with a variable data rate from the CPS accessible to the re-configurator by (i) detecting events, (ii) discretizing and accumulating data streams, and (iii) sending the results via the event and process information channels with limited data rate. In detail, the SR has the following concerns:

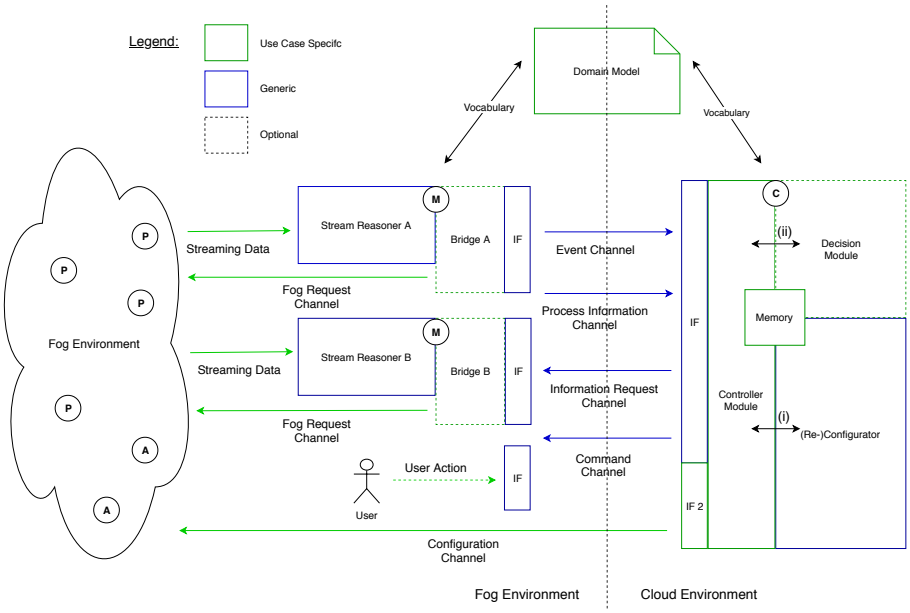


Figure 3: DynaCon Architecture

- Obtaining streaming data input in various formats, discrete or continuous, from the cyber-physical system by observing and processing this input nearly in real time.
- Detecting events in input streams and notifying the re-configurator about these events.
- Collecting accumulated and projected information about the cyber-physical system.
- Transmitting accumulated information to the re-configurator.

A bridge module might be used to manage the SR. This module handles the starting, ending, and locally reconfiguration of the reasoner with parameters sent by the controller module (e.g., windows size) via the command channel.

In particular, the purpose of a SR is *not* necessarily deciding when the re-configurator should perform a new configuration. However, for more simple cases the SR could trigger the re-configurator directly.

Example. For instance in the Siemens C-ITS use case, the SR is either a spatial-stream ontology-based query answering (Eiter, Parreira, and Schneider 2017) or an Hexlite (Fink et al. 2013) engine.

4.2 Controller Module

The controller module is designed to manage use case specific tasks on cloud side. The module is located between the interface and re-configurator and is needed to manage, set-up, and activate the re-configurator.

The controller manages the communication with the fog side, i.e. receiving messages from the event and process information channel. Acting on the event and the collected process information the controller activates the re-configurator, which includes also the the preparation of ASP (disjunctive Datalog) programs like:

- the static domain model,
- the re-configuration problem encoding, and
- selected process information.

Some use cases additionally require a decision module that is polled to distinguish different re-configuration scenarios. An important task of the controller module is the maintenance of actual states of fog-side objects obtained via the process information channel or by actively making information requests to obtain the missing information. For state maintenance, the controller has access to a (possibly persistent) memory module, i.e., a database, which might keep the log of the process information channel over a longer time.

4.3 Re-Configuration Module

The Re-Configuration Module (RCM) is providing re-configurations upon request by the controller module. In the DynaCon Architecture, it is the component that directly communicates with the (lazy-grounding) ASP solver and as such, it prepares the direct input for the ASP solver.

The RCM makes use of existing information and consults the memory for such information, which depending on the use case may be legacy configuration(s), difference information (deltas) to legacy configuration(s), reusable (first-order) conflicts, reusable search decisions, or any combination of these.

For obtaining a re-configuration, the RCM can utilize the configurator, i.e., the lazy-grounding ASP solver, or construct a reconfiguration by other means, possibly even without querying the configurator. The RCM may also attempt reconfiguration by breaking the task down into several smaller configuration problems to be solved by the configurator, and then constructing a full reconfiguration by combining the solutions of the smaller problems in a meaningful way. The RCM can run in close connection with the configurator supplying it legacy information on demand, e.g., when reusing search decisions.

Due to the heterogeneity of the DynaCon use cases, the RCM may also be realized in a virtual way in cases where no reconfiguration attempts are desired and each reconfiguration shall be treated as a new configuration.

4.4 Decision Module

The decision module is responsible for deciding about any actions of the module to take, which includes a) starting of the new computations or reconfigurations, b) communication of the new configurations to units in the network, and c) reconfiguration of a specific SR. It is regarded as the logic respectively dedicated decision component of the re-configuration unit for this task, while the re-configuration module serves to provide configurations that match a given specification, yet does not take any decision about whether re-configuration should be attempted and for what specification precisely.

The separation of concerns between the decision module and the configuration module detangles deciding about whether a reconfigurations should be attempted and actually computing a new configuration. The modularization of these tasks makes it possible to replace components independently and it fosters maintainability.

The decision module interacts with the controller module, which provides information obtained via the interface from the SR to the decision module. The controller module then gets in return the outcome of the decision process for the input provided.

No particular logic or decision mechanism is compulsory for the decision module, nor its physical implementation. The decision module can be virtual, in particular if decisions and corresponding actions are straight from the data stream that are obtained. On the other hand the decision making may access background knowledge and information e.g., about the current session and/or previous sessions for its computation.

As regards the DynaCon use cases, the decision making is rather heterogeneous and has different dimensions of conceptual complexity. Therefore, no formalization of the notion of decision module and its interfaces will be developed for DynaCon.

Example. For example, deciding whether a traffic light should be reconfigured might be simply determined by using a threshold, for instance, while a more elaborated approach might refer to historical data and data from the neighborhood; deciding about reconfiguration of a larger cluster of traffic lights in a large city might can be more knowledge intense.

4.5 Domain Model

We assume that the SR and configurator share one domain with the same constants referring to the same objects and concepts. To perform a mapping on the interface is out of the scope of this work with several possible solutions, for instance ontology alignment methods (Ehrig 2006) could be applied.

A simple domain model just includes a collection of terms, i.e., objects ids, describing the events, process information, and configuration items. This is often not sufficient, and we need a more elaborate model, which is captured by a domain ontology that includes classes, class hierarchies, properties, and relations. We express the ontology in Datalog as our modeling language, thus allowing us to reuse the domain model in the SR and re-configurator. As we use Datalog as our language, we restrict ourselves to ontology languages that are Datalog-rewritable, which is feasible with RDF(S) or OWL 2 RL (Krötzsch 2012) dialects.

Example. For instance in the C-ITS use case, the classes would relate to the traffic participants and the road network, which would need relations to describe the intersection topologies. Also a class for events is defined, which includes the different events and their severity that might appear in the use case.

4.6 Communication and Message Structure

The communication between fog and cloud-side can be adapted according to the use case. We outline the following possible options:

(a) Direct calls between the fog/cloud components are possible, since they are located on the same system, e.g., the Kelag use case;

(b) In a distributed environment, we intend to use a simple protocol such as UDP, for the communication between cloud and fog, e.g., the C-ITS use case;

(c) No fog components exists, hence the user interacts directly with controller and configurator, e.g., the LTE use case.

Each channel transports customized messages encoded as *Datalog* facts, which includes the intended data that relate to events, process information, requests, and commands. A predefined (base) message structure is always given, which is specified according to each use case and the vocabulary given by the domain model. The base message structure includes events that are defined by a set of actors, targets, locations, and temporal items. Events then are (possibly) linked to process information through actors, location, and temporal items, where one process information message carries observations that are either simple values or aggregations with an unit (e.g., km/h for the speed of vehicles).

5 Stream Reasoner Interface

To justify design decisions regarding the stream reasoner (SR) Interface, we first define a separation of concerns between the SR and the configurator. The SR has the following overall concern:

Making the (variable data rate) input from the cyber-physical system accessible to the configurator via two channels with limited data rate: (i) events and (ii) accumulated data streams.

The Stream Reasoner has at least two independent channels where it can actively send information to the configurator:

```
* Event Channel
* Accumulated Process Information Channel
```

The Stream Reasoner has two channels for receiving commands from the configurator:

```
* Command Channel
* Information Request Channel
```

Important: There is also an independent Configuration Channel and an interface IF 2, which are responsible for sending the result of the configurator to the devices (as producers) in the fog. We do not cover this, since handling the conversion and communication of specific configurations commands of the device should be considered in each use case individually.

5.1 Event Channel

The Event Channel has the purpose of sending events to the configurator, which were detected in the input stream by the Stream Reasoner. This channel has a variable rate of transmitted information because there can be extended periods of time where no events are detected.

Events are predefined occurrences of (unexpected) changes in our streaming data that are detected by the SRs or are started by user interactions. We follow the Complex Event Processing (CEP) and Event Extraction (EE) communities and define events as a tuple: $m_e = \langle e, a, t, l, d, p \rangle$, where each position represents the following:

- *EventType*: e is either a simple label or class. A class is linked to the domain model, putting the event into a larger context. For instance, in the C-ITS use case, "trafficJamStarted" or "accident" are even types;
- *Source*: a is the source, the entity that is involved in generating and sending the event. E.g., in the C-ITS use case this could be the RSU, or vehicles. In the CEP context this would be the actor;
- *Targets*: t is the set of systems or objects, where an event is targeted to. This implies that the target could react in a (predefined) manner on that event. Target can be empty, if only one receiver is per se listening. Again, in the C-ITS use case the target would be the traffic light configurator;
- *Locations*: l is a set of locations that represent simple labels or objects that have some spatial or topological extent. E.g., in the C-ITS use case,

the location might be a specific intersection where an accident happend; *PS: Question is if we only define the extent of the event or the extent of all the objects involved.*

- *Time*: d is a set of temporal items, that can be either time points or intervals. We assume a general timeline (a synchronised clock), where all components agree on.
- *Parameter*: p is a set of tuples of the form $\langle v, u \rangle$, where v is either a numeric value, an aggregation, or an qualitative value, which is specified by an optional unit u. Information concerning the event can be attached here.

The rate of events transmitted over this channel should be low enough so that the configurator can process all events, evaluate them, and potentially react by performing a reconfiguration.

Datalog Encoding.

```
event(eventType, sourceID, targetID, locationID, time, parameter).
```

Example. An example for events in this channel in the C-ITS use case for configuring traffic lights are "TrafficJamStarted" and "TrafficJamStopped" events: $m_p = \langle trafficJamStarted, RSU\_i100, , i100, 12{:}50{:}10, \langle \rangle \rangle$.

5.2 Process Information Channel

The Accumulated process information channel has the purpose of either periodically (push-based) or on demand (pull-based) providing certain accumulated process information to the configurator. The pull-based request is trigger through the Information Request Channel.

The purpose of this channel is to provide the configurator with information that are needed to parametrize the configuration problem. The channel is managed by the controllers on both sides, and might include the storage of the information. Hence, process information messages carry sets of observations $\{m_{p_1}, ..., m_{p_n}\}$, where each tuple is defined as: $m_p = \langle e, a, t, l, u, p \rangle$, where e is a label defining the information type, and a is the source t are the target, and u is an optional unit; p is a set of tuples of the form $\langle t, v \rangle$, where t is a time point/interval, v is either a numeric value, an aggregation, or an qualitative value.

Datalog Encoding.

```
information(infoType1, sourceID, targetID, locationID, time, value1, unit1).
information(infoType2, sourceID, targetID, locationID, time, value2, unit2).
...
```

Example. An example for the C-ITS use case could be the "number of cars in lane X" for all lanes X on specific intersection. A message for a specific lane

"i100_l1" on intersection "i100" showing the average nr of cars during one 1m would look like: $m_p = \{\langle avgNrCars, RSU\_i100, , i100\_l1, Nr, 12{:}50{:}10, 10\rangle$.

5.3 Information Request Channel

The Information Request Channel has the purpose of permitting the configurator to query certain process information instead of periodically receiving such information.

A request can either be a s set of process information items, which are combined and filtered according to some specifications. Or, it can contain some additional information supplementing the process information. For instance, the current signal plan for each intersection could be requested via this channel.

They first five parameter of a process information message, i.e., $m_p = \langle e, a, t, l, d\rangle$, can act as a filter for a particular request. The filter then can be either a specific value, an empty value (ignoring that position), or an asterisk (representing all values).

Example. We could requesting the "average nr of cars" during the last 10m for all lanes on intersection i100 as follow: $m_p = \langle avgNrCars, RSU\_i100, , *, 10m\rangle$. But also a request for the current signal plan for RSU i100 on intersection i100 could be performed by: $m_p = \langle signalPlan, RSU\_i100, , , \rangle$

Datalog Encoding.

```
request(infoType1, sourceID, targetID, locationID, time).
```

5.4 Command Channel

The command channel has the purpose of sending Stream Reasoner commands to the interface/controller of the fog-side. This channel permits the configurator to select the type of information to be received from the Stream Reasoner and to configure parameters for detecting events and for accumulating process information. Moreover it permits to send commands such as reset to the Stream Reasoner, where all volatile information is discarded and accumulation starts anew.

The following types of messages can be transmitted over this channel:

- A target as defined in the events
- Stream Reasoner commands

Commands, parameters and possible values, as well as the format of a knowledge base update operation, need to be defined individually for each use case. Following is a list of possible commands to a Stream Reasoner:

- Reset

- Set parameter (Parameter, Value)
- Get parameter (Parameter) -> Value is returned
- Activate/deactivate rules or queries
- Update knowledge base (Update Operation)

Datalog Encoding.

```
command(reset, sourceID, targetID).
command(setParameter, sourceID, targetID, parameterID, <filter>, value).
command(getParameter, sourceID, targetID, parameterID, <filter>) .
```

5.5 Additional Considerations

5.5.1 Configuration Channel

The configuration channel has the purpose of sending proprietary sets of configurations back to the CPPU on the fog-side, e.g., in the C-ITS use case this will be the specific RSU and signal controller. Since the communication and structure of this task depends on each use case, we do not cover the generation and conversion of configurations to CPPU specific commands in this document.

5.5.2 Synchronisation

Rate Limiting

The above mentioned concern of the limitation of the rate of events and the rate of accumulated process information has to be ensured by the reasoning within the SR. This can be the actual reasoner implementation, the knowledge representation, or filters at the event output interface.

> An example for realizing event rate limiting in the knowledge representation would be to replace simple thresholds for detecting events with hysteresis-based event detection methods.
>
> An example for realizing accumulated process information limiting in the knowledge representation would be to define a fixed set of objects for which a fixed set of process information can be transmitted per time unit, where fixed means that these sets do not depend on input to the SR but only on the (fixed) configuration of the SR in this concrete application. An advantage of such a definition of a fixed set of objects is, that a limited rate of information in the stream can be enforced without modifying the Stream Reasoner or its knowledge base.
>
> An example for realizing such limiting as an output filter would be to drop certain events according to priorities before sending them to the configurator.

6 Instantiation for an Application

We show the instantiation in the Siemens C-ITS use case, and refer to the use case specific document for the other use cases.

6.1 Siemens C-ITS

Cooperative Intelligent Transportation Systems (C-ITS) are the setting for this use case. The edge devices are roadside units (RSU) that observe V2X communication messages of traffic participants (i.e., cars and traffic lights). The cloud is represented by a traffic control centre that is responsible for managing the overall traffic and the different RSUs. The initial goal is the detecting of undesired traffic events, i.e., accidents or traffic jams, and the second goal is finding a reconfiguration of traffic light signal plans that ameliorates the effects of the undesired event. Concretely, we aim to improve vehicle throughput compared to an unmodified system for undesired traffic events.

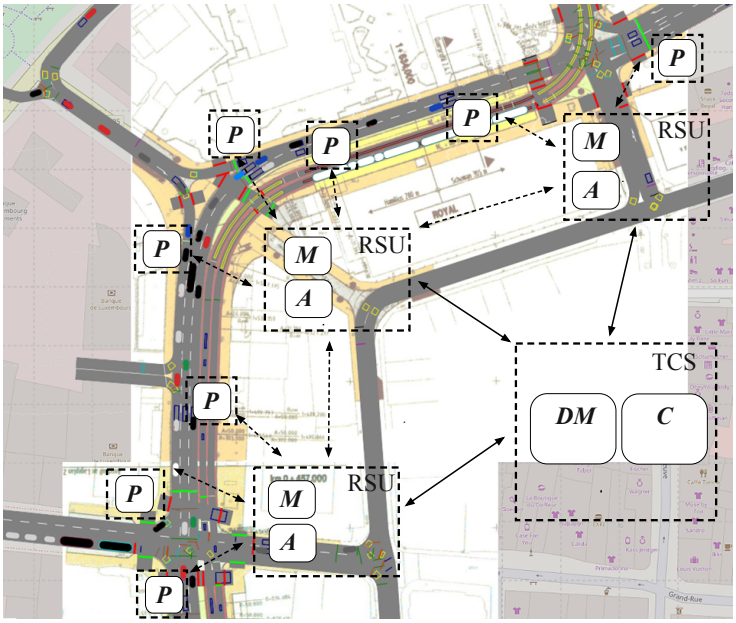


Figure 4: C-ITS Architecture Example

In Figure 4, we show an example for a C-ITS instantiation of our general architecture. The producers (P) are in this setting either vehicles that communicate via V2X messages to the surrounding or traffic light (TL) installations that manage their assigned intersection. Each intersection has one edge device, the RSU, which is connected to the TL installations and receives the V2X messages from

the vehicles.[8] The RSUs acts as monitor (M) and actuator (A) by analyzing the V2X messages streams, detecting events such as accidents, and updating the (new) signal plans to the connected TL installation. Resulting, the stream reasoner would be located on the RSU, and sending (detected) events and (local) process informations to the traffic control centre (TCS), which is our cloud server and has the tasks of re-configuring the TL installations using the decision module (DM) to trigger the re-configuration. Note that the configurator (C) should be on the TCS, since only there all the events and local information is available. Since, we have several RSUs deployed, and vehicles have their own capabilities of monitoring and changing the surrounding, we naturally have also multi-agent environment.

References

Bagade, P., A. Banerjee, and S.K.S. Gupta. 2017. "Validation, Verification, and Formal Methods for Cyber-Physical Systems." In *Cyber-Physical Systems: Foundations, Principles and Applications*, edited by Houbing Song, Danda Rawat, Sabina Jeschke, and Christian Brecher, 175–91. Elsevier Inc.

Brewka, Gerhard, and Thomas Eiter. 2007. "Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems." In *Proceedings of the National Conference on Artificial Intelligence*, 1:385–90.

Brewka, Gerhard, Thomas Eiter, Michael Fink, and Antonius Weinzierl. 2011. "Managed Multi-Context Systems." In *IJCAI 2011*, 786–91.

Dell'Aglio, Daniele, Emanuele Della Valle, Frank van Harmelen, and Abraham Bernstein. 2017. "Stream Reasoning : A Survey and Outlook a Summary of Ten Years of Research and a Vision for the Next Decade." *Data Science*, no. 1: 59–83.

Donzé, Alexandre, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott Smolka. 2012. "On Temporal Logic and Signal Processing." In.

Ehrig, Marc. 2006. "Ontology Alignment: Bridging the Semantic Gap." In *Semantic Web and Beyond: Computing for Human Experience*.

Eiter, Thomas, Josiane Xavier Parreira, and Patrik Schneider. 2017. "Spatial Ontology-Mediated Query Answering over Mobility Streams." In *ESWC 2017*, 219–37. Springer International Publishing.

Fink, Michael, Stefano Germano, Giovambattista Ianni, Christoph Redl, and Peter Schüller. 2013. "ActHEX: Implementing Hex Programs with Action Atoms." In *LPNMR 2013*, 317–22. Berlin, Heidelberg: Springer Berlin Heidelberg.

[8] https://new.siemens.com/global/en/products/mobility/road-solutions/connected-mobility-solutions/sitraffic-vehicle2x.html

Henzinger, Thomas A. 2000. "The Theory of Hybrid Automata." In *Verification of Digital and Hybrid Systems*, 265–92. Berlin, Heidelberg: Springer Berlin Heidelberg.

Koymans, Ron. 1990. "Specifying Real-Time Properties with Metric Temporal Logic." *Real-Time Systems* 2 (4): 255–99.

Krötzsch, Markus. 2012. "OWL 2 Profiles: An Introduction to Lightweight Ontology Languages." In *Reasoning Web 2012*, edited by Thomas Eiter and Thomas Krennwallner, 112–83. Springer Berlin Heidelberg.

Pnueli, Amir. 1977. "The Temporal Logic of Programs." In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, Usa, 31 October - 1 November 1977*, 46–57.

Sarkar, S., Z. Jiang, A. Akintayo, S. Krishnamurthy, and A. Tewari. 2017. "Probabilistic Graphical Modeling of Distributed Cyber-Physical Systems." In *Cyber-Physical Systems: Foundations, Principles and Applications*, edited by Houbing Song, Danda Rawat, Sabina Jeschke, and Christian Brecher, 265–85. Elsevier Inc.

Shannon, C. E. 1948. "A Mathematical Theory of Communication." *Bell System Technical Journal* 27 (3): 379–423.

Stonebraker, Michael, Uğur Çetintemel, and Stan Zdonik. 2005. "The 8 Requirements of Real-Time Stream Processing." *SIGMOD Rec.* 34 (4): 42–47.

Törngren, SM., F. Asplund, S. Bensalem, J. McDermid, R. Passerone, H. Pfeifer, A. Sangiovanni-Vincentelli, and B. Schätz. 2017. "Characterization, Analysis, and Recommendations for Exploiting the Opportunities of Cyber-Physical Systems." In *Cyber-Physical Systems: Foundations, Principles and Applications*, edited by Houbing Song, Danda Rawat, Sabina Jeschke, and Christian Brecher, 3–14. Elsevier Inc.