Multi-Context Systems for Reactive Reasoning in Dynamic Environments

Gerhard Brewka and Stefan Ellmauthaler and Jörg Pührer

Abstract. We show in this paper how managed multi-context systems (mMCSs) can be turned into a reactive formalism suitable for continuous reasoning in dynamic environments. We extend mMCSs with (abstract) sensors and define the notion of a run of the extended systems. We then show how typical problems arising in online reasoning can be addressed: handling potentially inconsistent sensor input, modeling intelligent forms of forgetting, selective integration of knowledge, and controlling the reasoning effort spent by contexts, like setting contexts to an idle mode. We also investigate the complexity of some important related decision problems and discuss different design choices which are given to the knowledge engineer.

1 Introduction

Research in knowledge representation (KR) faces two major problems. First of all, a large variety of different languages for representing knowledge - each of them useful for particular types of problems - has been produced. There are many situations where the integration of the knowledge represented in diverse formalisms is crucial, and principled ways of achieving this integration are needed. Secondly, most of the tools providing reasoning services for KR languages were developed for offline usage: given a knowledge base (KB) computation is one-shot, triggered by a user, through a specific query or a request to compute, say, an answer set. This is the right thing for specific types of applications where a specific answer to a particular problem instance is needed at a particular point in time. However, there are different kinds of applications where a reasoning system is continuously online and receives information about a particular system it observes. Consider an assisted living scenario where people in need of support live in an apartment equipped with various sensors, e.g., smoke detectors, cameras, and body sensors measuring relevant body functions (e.g., pulse, blood pressure). A reasoning system continuously receives sensor information. The task is to detect emergencies (health problems, forgotten medication, overheating stove,...) and cause adequate reactions (e.g., turning off the electricity, calling the ambulance, ringing an alarm). The system is continuously online and has to process a continuous stream of information rather than a fixed KB.

This poses new challenges on KR formalisms. Most importantly, the available information continuously grows. This obviously cannot go on forever as the KB needs to be kept in a manageable size. We thus need principled ways of forgetting/disregarding information. In the literature one often finds sliding window techniques [13] where information is kept for a specific, predefined period of time and forgotten if it falls out of this time window. We believe this approach is far too inflexible. What is needed is a dynamic, situation dependent way of determining whether information needs to be kept or can be given up. Ideally we would like our online KR system to guarantee specific response times; although it may be very difficult to come up with such guarantees, it is certainly necessary to find means to identify and focus on relevant parts of the available information. Moreover, although the definition of the semantics of the underlying KR formalism remains essential, we also need to impose procedural aspects reflecting the necessary modifications of the KB. This leads to a new, additional focus on runs of the system, rather than single evaluations.

Nonmonotonic multi-context systems (MCSs) [5] were explicitly developed to handle the integration problem. In a nutshell, an MCS consists of reasoning units - called contexts for historical reasons [15] - where each unit can be connected with other units via so-called bridge rules. The collection of bridge rules associated with a context specifies additional beliefs the context is willing to accept depending on what is believed by connected contexts. The semantics of the MCS is then defined in terms of equilibria. Intuitively, an equilibrium is a collection of belief sets, one for each context, which fit together in the sense that the beliefs of each context adequately take into account what the other contexts believe.

The original framework was aimed at modeling the flow of information among contexts, consequently the addition of information to a context was the only possible operation on KBs. To capture more general forms of operations MCSs were later generalized to so called managed MCSs (mMCSs) [7]. The main goal of this paper is to demonstrate that this additional functionality makes managed MCSs particularly well-suited as a basis for handling the mentioned problems of online reasoning systems as well. The main reason is that the operations on the knowledge bases allow us to control things like KB size, handling of inconsistent observations, focus of attention, and even whether a particular context should be idle for some time.

However, to turn mMCSs into a reactive online formalism we first need to extend the framework to accommodate observations. We will do so by generalizing bridge rules so that they have access not only to belief sets of other contexts, but also to sensor data. This allows systems to become reactive, that is, to take information about a dynamically changing world into account and to modify themselves to keep system performance up.

The rest of the paper is organized as follows. We first give the necessary background on mMCSs. We then extend the framework to make it suitable for dynamic environments, in particular we show

---

1 A former version [3] of this paper has been accepted for publication in the Proc. of ECAI-14. This work has been partially supported by the German Research Foundation (DFG) under grants BR-1817/7-1 and FOR 1513.

2 Institute of Computer Science, Leipzig University, Germany, email: {brewka,ellmauthaler,puehrer}@informatik.uni-leipzig.de
how observations can be accommodated, and we define the notion of a run of an MCS based on a sequence of observations. The subsequent sections address the following issues: handling time and the frame problem; dynamic control of KB size; focus of attention; control of computation (idle contexts). We finally discuss the complexity of some important decision problems.

2 Background: Multi-Context Systems

We now give the necessary background on managed MCSs which provides the basis for our paper. We present a slightly simplified variant of mMCs here as this allows us to better highlight the issues relevant for this paper. However, if needed it is rather straightforward (albeit technically somewhat involved) to extend all our results to the full version of mMCs. More specifically we make two restrictions: 1) we assume all contexts have a single logic rather than a logic suite associated contexts. The latter will change dynamically during updates, as we will see later, and it is thus convenient to keep them separate. Bridge rules will be separated due to technical reasons (i.e., better presentation of the later introduced notion of a run).

An mMCS builds on an abstract notion of a logic $L$ as a triple $(KB_L, BS_L, ACC_L)$, where $KB_L$ is the set of admissible knowledge bases (KBs) of $L$, which are sets of KB-elements ("formulas"); $BS_L$ is the set of possible belief sets, whose elements are beliefs; and $ACC_L : KB_L \rightarrow 2^{BS_L}$ is a function describing the semantics of $L$ by assigning to each KB a set of acceptable belief sets.

**Definition 1** A context is of the form $C = \langle L, ops, mng \rangle$ where

- $L$ is a logic,
- $ops$ is a set of operations,
- $mng : 2^{ops} \times KB_L \rightarrow KB_L$ is a management function.

For an indexed context $C_i$ we will write $L_i$, $ops_i$, and $mng_i$ to denote its components.

**Definition 2** Let $C = \langle C_1, \ldots, C_n \rangle$ be a tuple of contexts. A bridge rule for $C_i$ over $C$ ($1 \leq i \leq n$) is of the form

$$op \leftarrow a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_m,$$

such that $op \in ops_i$ and every $a_k$ ($1 \leq k \leq m$) is an atom of form $c \in c$, where $c \in \{1, \ldots, n\}$, and $b$ is a belief for $C_c$, i.e., $b \in S$ for some $S \in BS_{cte}$. For a bridge rule $r$, the operation $bd(r) = op$ is the head of $r$, while $bd(r) = \{a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_m\}$ is the body of $r$.

**Definition 3** A managed multi-context system (mMCS) $M = \langle C, BR, KB \rangle$ is a triple consisting of

1. a tuple of contexts $C = \langle C_1, \ldots, C_n \rangle$,
2. a tuple $BR = \langle br_1, \ldots, br_n \rangle$, where each $br_i$ is a set of bridge rules for $C_i$ over $C$,
3. a tuple of KBs $KB = \langle kb_1, \ldots, kb_n \rangle$ such that $kb_i \in KB_{L_i}$.

A belief state $S = \langle S_1, \ldots, S_n \rangle$ for $M$ consists of belief sets $S_i \in BS_{L_i}$, $1 \leq i \leq n$. Given a bridge rule $r$, an atom $c \in bd(r)$ is satisfied by $S$ if $p \in S_r$ and a negated atom not $c \in bd(r)$ is satisfied by $S$ if $p \notin S_r$. A literal is an atom or a negated atom. We say that $r$ is applicable wrt. $S$, denoted by $S \models bd(r)$, if every literal $l \in bd(r)$ is satisfied by $S$. We use $app(S) = \{bd(r) \mid r \in br_i, S \models bd(r)\}$ to denote the heads of all applicable bridge rules of context $C_i$ wrt. $S$.

The semantics of an mMCS $M$ is then defined in terms of (local) equilibrium, where an equilibrium is a belief state $S = \langle S_1, \ldots, S_n \rangle$ satisfying the following condition: the belief set chosen for each context must be acceptable for the KBs obtained by applying the management function to the heads of applicable bridge rules and the KB associated with the context. More formally, for all contexts $C_i = \langle L_i, ops_i, mng_i \rangle$ let $S_i$ be the belief set chosen for $C_i$. Then $S$ is an equilibrium if, for $1 \leq i \leq n$,

$$S_i \in ACC_i(kb_i') = mng_i(app_i(S), kb_i).$$

Management functions allow us to model all sorts of modifications of a context’s KB and thus make mMCSs a powerful tool for describing the influence contexts can have on each other.

3 Reactive Multi-Context Systems

To make an mMCS $M$ suitable for reactive reasoning in dynamic environments, we have to accomplish two tasks:

1. we must provide means for the MCS to obtain information provided by sensors, and
2. we have to formally describe the behavior of the MCS over time.

Let us first show how sensors can be modeled abstractly. We assume that a sensor $Pi$ is a device which is able to provide new information in a given language $L_{Pi}$ specific to the sensor. From an abstract point of view, we can identify a sensor with its observation language and a current sensor reading, that is, $Pi = \langle L_{Pi}, \pi_i \rangle$ where $\pi_i \subseteq L_{Pi}$. Given a tuple of sensors $Pi = \langle Pi_1, \ldots, Pi_k \rangle$, an observation $Obs$ for $Pi$ (Pi-observation for short) consists of a sensor reading for each sensor, that is, $Obs = \langle \pi_1, \ldots, \pi_k \rangle$ where for $1 \leq i \leq k$, $\pi_i \subseteq L_{Pi}$. Each context must have access to its relevant sensors. Contexts already have means to obtain information from outside, namely the bridge rules. This suggests that the simplest way to integrate sensors is via an extension of the bridge rules: we will assume that bridge rules in their bodies can not refer to contexts, but also to sensors.

**Definition 4** A reactive multi-context system (rMCS) over sensors $Pi = \langle Pi_1, \ldots, Pi_k \rangle$ is a tuple $M = \langle C, BR, KB \rangle$, as in Def. 3 except that the atoms $a_i (1 \leq i \leq m)$ of bridge rules in BR for context $C_i$ of form $\Pi$ can either be a context atom of form $c \in c$ or a sensor atom of form $o \in obs$, where $s$ is an index determining a sensor (1 $\leq s \leq k$) and $o \in L_{Pi}$ is a piece of sensor data.

The applicability of bridge rules now also depends on an observation:

**Definition 5** Let $Pi$ be a tuple of sensors and $Obs = \langle \pi_1, \ldots, \pi_k \rangle$ an $Pi$-observation. A sensor atom $o \in obs$ is satisfied by $Obs$ if $o \in \pi_s$; a literal not $o \in obs$ is satisfied by $Obs$ if $o \notin \pi_s$.

Let $M = \langle C, BR, KB \rangle$ be an rMCS with sensors $Pi$ and $S$ a belief state for $M$. A bridge rule $r$ in $BR$ is applicable wrt. $S$ and $Obs$, symbolically $S \models_{Obs} bd(r)$, if every context literal in $bd(r)$ is satisfied by $S$ and every sensor literal in $bd(r)$ is satisfied by $Obs$.
Instead of \( \text{app}_i(S) \) we use \( \text{app}_i(S, \text{Obs}) = \{ \text{bd}(r) \mid r \in \text{br}_i \land S \models \text{obs}, \text{bd}(r) \} \) to define an equilibrium of an rMCS in a similar way as for an mMCS:

**Definition 6** Let \( M = (C, \text{BR}, KB) \) be an rMCS with sensors \( \Pi \) and \( \text{Obs} \) a \( \Pi \)-observation. A belief state \( S = \{S_1, \ldots, S_n\} \) for \( M \) is an equilibrium of \( M \) under \( \text{Obs} \) if, for \( 1 \leq i \leq n \),

\[
S_i \in \text{ACC}(\text{mng}_i(\text{app}_i(S, \text{Obs}), kb_i)).
\]

**Definition 7** Let \( M = (C, \text{BR}, KB) \) be an rMCS with sensors \( \Pi \), \( \text{Obs} \) a \( \Pi \)-observation, and \( S = \{S_1, \ldots, S_n\} \) an equilibrium of \( M \) under \( \text{Obs} \). The tuple of KBs generated by \( S \) is defined as \( KB^S = \langle \text{mng}_1(\text{app}_1(S, \text{Obs}), kb_1), \ldots, \text{mng}_n(\text{app}_n(S, \text{Obs}), kb_n) \rangle \). The pair \( (S, KB^S) \) is called full equilibrium of \( M \) under \( \text{Obs} \).

We now introduce the notion of a run of an rMCS induced by a sequence of observations:

**Definition 8** Let \( M = (C, \text{BR}, KB) \) be an rMCS with sensors \( \Pi \) and \( O = (\text{Obs}^0, \text{Obs}^1, \ldots) \) a sequence of \( \Pi \)-observations. A run of \( M \) induced by \( O \) is a sequence of pairs \( R = ((S^0, KB^0), (S^1, KB^1), \ldots) \) such that

1. \( (S^0, KB^0) \) is a full equilibrium of \( M \) under \( \text{Obs}^0 \),
2. for \( i > 0 \), \( (S^i, KB^i) \) is a full equilibrium of \( (C, \text{BR}, KB^{i-1}) \) under \( \text{Obs}^i \).

To illustrate the notion of a run, let’s discuss a simple example. We want to model a clock which allows other contexts to add timestamps to sensor information they receive. We consider two options. We will first show how a clock can be realized which generates time internally by increasing a counter whenever a new equilibrium is computed. We later discuss a clock based on a sensor having access to “objective” time. In both cases we use integers as time stamps.

**Example 1** Consider a context \( C \) whose KBs (and belief sets) are of the form \( \{\text{now}(t)\} \) for some integer \( t \). Let \( kb^{C^0} = \{\text{now}(0)\} \). Assume the single bridge rule of the context is \( \text{incr} \leftarrow \), which intuitively says time should be incremented whenever an equilibrium is computed. The management function is thus defined as

\[
\text{mng}_c(\{\text{incr}\}, \{\text{now}(t)\}) = \{\text{now}(t+1)\}
\]

for all \( t \). Since the computation of the (full) equilibrium is independent of any other contexts and observations, the context just increments its current time whenever a new equilibrium is computed. Each run of an rMCS with context \( C \) will thus contain for \( C \) the sequence of belief sets \( \{\text{now}(1)\}, \{\text{now}(2)\}, \{\text{now}(3)\}, \ldots \). The example illustrates that the system may evolve over time even if there is no observation made at all.

It is illustrative to compare this with a context \( C_{e^0} \) which is like the one we discussed except for the bridge rules which now are the instances of the schema

\[
\text{set}(\text{now}(T+1)) \leftarrow e^c: \text{now}(T).
\]

The management function correspondingly becomes

\[
\text{mng}_c(\{\text{set}(\text{now}(t+1))\}, \{\text{now}(t)\}) = \{\text{now}(t+1)\}
\]

for all \( t \). Note that in this case no equilibrium exists! The reason for this is that by replacing \( \text{now}(0) \) with \( \text{now}(1) \) the precondition for the rule sanctioning this operation goes away. Special care thus needs to be taken when defining the operations.

In the rest of the paper we will often use an alternative approach where “objective” time is entered into the system by a particular sensor \( \Pi \). In this case each update of the system makes time available to each context via the current sensor reading of \( \Pi \).

In Example 1 we already used a bridge rule schema, that is, a bridge rule where some of the parts are described by parameters (denoted by uppercase letters). We admit such schemata to allow for more compact representations. A bridge rule schema is just a convenient abbreviation for the set of its ground instances. The ground instances are obtained by replacing parameters by adequate ground terms. We will admit parameters for integers representing time, but also for formulas and even contexts. In most cases it will be clear from the discussion what the ground instances are, in other cases we will define them explicitly. We will also admit some kind of basic arithmetic in the bridge rules and assume the operations to be handled by grounding, as is usual, say, in answer set programming. For instance, the bridge rule schema

\[
\text{add}(p(T+1)) \leftarrow c:p(T), \text{not } c:-p(T+1)
\]

which we will use to handle the frame problem in the next section has ground instances \( \text{add}(p(4)) \leftarrow c:p(0), \text{not } c:-p(1), \text{add}(p(2)) \leftarrow c:p(1), \text{not } c:-p(2), \text{etc.} \)

Although in principle parameters for integers lead to an infinite set of ground instances, in our applications only ground instances up to the current time (or current time plus a small constant, see Sect. 6) are needed, so the instantiations of time points remain finite.

In the upcoming sections we describe different generic modeling techniques for rMCSs. For concrete applications, these techniques can be refined and tailored towards the specific needs of the problem domain at hand. To demonstrate this aspect, we provide a more specific example from an assisted living application.

**Example 2** Although Bob suffers from dementia, he is able to live in his own apartment as it is equipped with an assisted living system that we model by means of an rMCS. Assume Bob starts to prepare his meal. He leaves the kitchen to go to the bathroom. After that, he forgets he has been cooking, goes to bed and falls asleep.

The rMCS should be able to recognize a potential emergency based on the data of different sensors in the flat that monitor, e.g., the state of the kitchen equipment and track Bob’s position.

Our rMCS M has three contexts C = \{C_{stt}, C_{hst}, C_{sh}\} and sensors \( \Pi = (\Pi_{pow}, \Pi_{tmp}, \Pi_{pos}) \). \( C_{stt} \) is the kitchen equipment context that monitors Bob’s stove. Its formulas and beliefs are atoms from at_{stt} = \{pw(on), pw(off), tm(cold), tm(hot)\} representing the stove’s power status (on/off) and a qualitative value for its temperature (cold/hot). The logic L_{stt} = \{u_{stt}, v_{stt}, ACC_{stt}\} of \( C_{stt} \) has a very simple semantics ACC_{stt} in which every knowledge base \( kb \) has only one accepted belief set coinciding with the formulas of \( kb \), i.e., ACC_{stt}(kb) = \{kb\}. The bridge rules for \( C_{stt} \) over C are

\[
\text{setPower}(P) \leftarrow \text{switch}(P)@\text{pow}.
\]

\[
\text{setTemp}(\text{cold}) \leftarrow T@\text{tmp}, T \leq 45.
\]

\[
\text{setTemp}(\text{hot}) \leftarrow T@\text{tmp}, 45 < T.
\]
that react to switching the stove on or off, registered by sensor \( \Pi_{pow} \), respectively read numerical temperature values from sensor \( \Pi_{tmp} \) and classify the temperature value as cold or hot. The management
function mng_{\text{app}}(app, kb) =
\{\text{pw}(\text{on}) | \text{setPower}(\text{on}) \in \text{app} \lor \text{setPower}(\text{off}) \notin \text{app}\} \cup
\{\text{pw}(\text{off}) | \text{setPower}(\text{on}) \notin \text{app} \lor \text{setPower}(\text{off}) \in \text{app}\} \cup
\{\text{tm}(t) | \text{setTemp}(t) \in \text{app}\}

ensures that the stove is considered on when it is switched on or
when it is not being switched off and already considered on in the old
knowledge base kb. Otherwise, the KB constructed by the manage-
ment function contains the atom pw(\text{off}). Context C_{\text{kb}} keeps track
of Bob’s position. The location of sensor $\Pi_{\text{pos}}$ is given by $L_{\text{pos}} = \{\text{enters(kitchen)}, \text{enters(bathroom)}, \text{enters(\text{bedroom})}\}$ and non-
empty sensor readings of $\Pi_{\text{pos}}$ signal when Bob has changed rooms.
The semantics of $C_{\text{kb}}$ is also ACC_{\text{kb}} and its bridge rules are given
by the schema

\text{setPos}(P) \leftarrow \text{enters}(P)@\text{pos}.

The management function writes Bob’s new position into the KB
whenever he changes rooms and keeps the previous position, oth-
erwise. $C_{\text{kb}} = \{\text{hr}_{\text{kb}}, \text{ops}_{\text{kb}}, \text{mng}_{\text{kb}}\}$ is the context for detecting emer-
gencies. It is implemented as an answer-set program, hence the ac-
ceptable belief sets of $V_{\text{kb}}$ are the answer sets of its KBs. The bridge
rules of $C_{\text{kb}}$ do not refer to sensor data but query other contexts:

\text{extVal(\text{oven}(P, T))} \leftarrow \text{kt:pw}(P), \text{kt:tm}(T).
\text{extVal(\text{humanPos}(P))} \leftarrow \text{hu:pos}(P).

The answer-set program $k_{\text{b}}$ is given by the rule

\text{emergency} \leftarrow \text{oven(on, hot)}, \text{not humanPos(kitchen)}.

The management function of $C_{\text{kb}}$ that adds information from
the bridge rules temporarily as input facts to the context’s KB is given
by mng_{\text{kb}}(app, kb) =

\{\text{kk}(\{\text{oven}(P, T) \leftarrow P \in \{\text{on, off}\}, T \in \{\text{cold, hot}\}\} \cup
\{\text{humanPos}(R) \leftarrow \text{enters}(R) \in L_{\text{pos}}\})\cup
\{\text{on}(p, t) \leftarrow \text{extVal(\text{oven}(p, t)) \in \text{app}\}\cup
\{\text{humanPos}(r) \leftarrow \text{extVal(\text{humanPos}(r)) \in \text{app}\}}\}

Consider the sequence $O = (\text{obs}^0, \text{obs}^1)$ of $O$-observations with
$\text{obs}^0 = \{\pi_{\text{pw}}, \pi_{\text{temp}}, \pi_{\text{pos}}\}$ for $0 \leq i \leq 1,$ $\pi_{\text{pw}} = \{\text{switch(on)}\},$
$\pi_{\text{temp}} = \{16\}, \pi_{\text{pos}} = \{\text{enter}(\text{kitchen})\}, \pi_{\text{pos}} = \{\text{enter}(\text{bathroom})\}$, and $\pi_{\text{pos}} = \emptyset$ for all other $\pi_{\text{pos}}$. Then, $(S^0, KB^0)$
is a full equilibrium of $M$ under $\text{obs}^3$, where

$S^0 = \{\{\text{pw}(\text{on}), \text{tm}(	ext{cold})\}, \{\text{pos}(\text{kitchen})\},$
\{\text{oven(on, cold), humanPos(kitchen)}\}\}.

and $\text{KB}^0$ equals $S^0$ except for the last component which is
$k_{\text{kb}} \cup \{\text{oven(on, cold) \leftarrow humanPos(kitchen) \leftarrow}\}$. Moreover,
$((S^0, KB^0), (S^1, KB^1))$ is a run of $M$ induced by $O$, where

$S^1 = \{\{\text{pw}(\text{on}), \text{tm}(	ext{hot})\}, \{\text{pos}(\text{bathroom})\},$
\{\text{oven(on, hot), humanPos(bathroom), emergency})\}.$

4 Handling sensor data

In this section we discuss how to model an rMCS where possibly
inconsistent sensor data can be integrated into a context $C_f$. To this
end, we add a time tag to the sensor information and base our treat-
ment of time on the second option discussed in the last section, that
is, we assume a specific time sensor $\Pi_t$ that yields a reading $\pi_t$ of the
actual time of the form $\text{now}(t)$ where $t$ is an integer.

Let $\Pi_1, \ldots, \Pi_m$ be the sensors which provide relevant informa-
tion for $C_f$ in addition to $\Pi_t$. Then $C_f$ will have bridge rules of the form

\text{add}(P, T, j_t) \leftarrow P@j_t, \text{now}(T)@t

where the operation add is meant to add new, time tagged informa-
tion to the context.

We assume the readings of a single sensor at a particular time point
to be consistent. However, it is a common problem that the readings
of different sensors may be inconsistent with each other wrt. some
context-dependent notion of inconsistency. To handle this we foresee a
management function $mng_j$, that operates based on a total prefer-
ence ranking of the available sensors. The third argument of the add
operation provides information about the source of sensor informa-
tion and thus a way of imposing preferences on the information to be
added. Without loss of generality assume $j_1 \ldots > j_m$, that is,
sensor $\Pi_{j_1}$ has highest priority.

Now let $\text{add}(S)$ be the set of add-operations in the heads of bridge
rules active in belief state $S$. We define

$\text{Add}_{j_1}(S) = \{(p, t) | \text{add}(p, t, j_1) \in \text{add}(S)\}$

and for $1 < i \leq m$ we let $\text{Add}_{j_i}(S) = \text{Add}_{j_{i-1}}(S) \cup$

$\{(p, t) | \text{add}(p, t, j_i) \in \text{add}(S), (p, t) \text{ consistent with } \text{Add}_{j_{i-1}}(S)\}$.

Finally, we define $mng_j(\text{add}(S), kb) = kb \cup \text{Add}_{j_m}(S)$.

This shows how the management function can solve conflicts
among inconsistent sensor readings based on preferences among the
sensors. Of course, many more strategies of integrating inconsistent
sensor data can be thought of which we are not going to discuss in
the paper. Please also note that the bridge rules do not necessarily
have to pass on sensor information as is to the context. They may as
well provide the context with some abstraction of the actual readings.
For instance, the sensor temperature information $\text{temp} = 55$ may be
transformed into qualitative information by a rule schema like

$\text{add}(\text{temp} = \text{high}, T, j_t) \leftarrow \text{temp} = x@j_t, 45 \leq x \leq 65,$
\text{now}(T)@t.$

We next present a way to address the frame problem using bridge
rules when sensors are not guaranteed to provide complete informa-
tion about the state of the environment in each step. In this case we
want to assume, at least for some of the atoms or literals observed at
time $T - 1$ which we call persistent, that they also hold at time $T$.
Assume $p$ is some persistent observable property. Persistence of $p$
is achieved by the following bridge rule schema:

$\text{add}(p(T)) \leftarrow \text{now}(T)@t, j_t:p(T - 1),$ not $j_t: \neg p(T)$.

Please note that in order to avoid non-existence of equilibria as
discussed at the end of Sect. 3 the use of this rule schema for the
frame problem presupposes that information about $p$ valid at time
$T - 1$ remains available and is not deleted by any other bridge rule.

5 Selective forgetting and data retention

To illustrate our approach we discuss in this section a context $C_d$
which can be used for emergency detection in dynamic environ-
ments. Assume there are $m$ potential emergencies $E_1, \ldots, E_m$ we
want the context to handle. The role of $C_d$ is to check, based on the observations made, whether one or more of the emergencies $E_i$ are suspected or confirmed. Based on information about potential emergencies $C_d$ adjusts the time span observations are kept. This is the basis for intelligent forgetting based on dynamic windows.

We do not make any assumption about how $C_d$ works internally apart from the following:

- $C_d$ may signal that emergency $E_i$ is suspected ($\text{suspect}(E_i)$) or confirmed ($\text{confirm}(E_i)$).
- $C_d$ has information about default, respectively actual window sizes for different observations ($\text{def}(w, p, x), \text{win}(p, x)$), and
- about the number of time steps observations are relevant for particular emergencies ($\text{rel}(p, e, x)$).

Given facts of the form mentioned above, here is a possible collection of bridge rules for the task. The operation $\text{set}$ sets the window size to a new value, deleting the old one. To signal an alarm, information is added to the context KB via the operation $\text{alarm}$.

\[
\text{set}(\text{win}(P, X)) \leftarrow d: \text{def}. \text{win}(P, X), \text{not } d: \text{suspect}(E) \\
\text{set}(\text{win}(P, Y)) \leftarrow d: \text{rel}(P, E, Y), d: \text{suspect}(E) \\
\text{alarm}(E) \leftarrow d: \text{confirm}(E)
\]

Finally, we have to make sure deletions of observations are performed in accordance with the determined window sizes:

\[
\text{del}(p(T')) \leftarrow \text{now}(T) \otimes t, d: \text{win}(P, Z), T' < T - Z.
\]

The management function just performs additions and deletions on the context KB. Since additions always are tagged with the current time, whereas deletions always refer to an earlier time, there can never be a conflict.

We have so far described a form of focusing where a time window is extended based on a specific suspected event. The next example shows a different form of focusing where specific information is generated and kept only during there is a potential danger in a particular room.

Example 3 Continuing Example 2, we show how context $C_{ig}$ can focus on specific rooms if there is a potential emergency. For the kitchen there is a threat if the stove is on, and it then becomes important to track whether someone is in the kitchen. Assume $C_{ig}$ has a potential belief $\text{pw}(\text{on}, T)$ expressing the stove is on since $T$. Focusing on the kitchen can be modeled by following the ASP-rule in $C_{ig}$’s KB:

\[
\text{focus}(\text{kitchen}) \leftarrow \text{pw}(\text{on}, T).
\]

In addition we will need a bridge rule, which keeps track whether Bob is absent from a room in case that room is in the current focus:

\[
\text{add}(\text{absence}(R, T)) \leftarrow \text{now}(T) \otimes t, \text{ig}: \text{focus}(R), \\
\text{not ig}: \text{humanpos}(R), \\
\text{not ig}: \text{absence}(R, T'), T' < T.
\]

as well as a bridge rule to forget the absence in a room if it is no longer necessary. There the $\text{delAll}$ operator removes all occurrences of absence with respect to a given room $R$ from the KB of the context.

\[
\text{delAll}(\text{absence}, R) \leftarrow \text{ig}: \text{humanpos}(R), \\
\text{delAll}(\text{absence}, R) \leftarrow \text{not ig}: \text{focus}(R).
\]

With those modifications it is possible to generate an alert if Bob was too long away from the kitchen although the stove is active.

6 Control of computation

In this section we show how it is possible - at least to some extent - to control the effort spent on the computation of particular contexts. We introduce a specific control context $C_0$ which decides whether a context it controls should be idle for some time. An idle context just buffers sensor data it receives, but does not use the data for any other computations.

Let’s illustrate this continuing the discussion of Sect. 5. Assume there are $k$ different contexts for detecting potential emergencies as described earlier. The RMCS we are going to discuss is built on an architecture where each detector context $C_i, 1 \leq i < k$ is connected via bridge rules with the control context. $C_0$ receives information about suspected emergencies and decides, based on this information, whether it is safe to let a context be idle for some time.

We now address the question what it means for a detector context to be idle. A detector context $C_i$ receives relevant observations to reason whether an emergency is suspected or confirmed. In case $C_i$ is idle, we cannot simply forget about new sensor information as it may become relevant later on, but we can buffer it so that it does not have an effect on the computation of a belief set, besides the fact that a buffered information shows up as an additional atom in the belief set which does not appear anywhere in the context’s background knowledge.

To achieve this we have to modify $C_i$’s original bridge rules by adding, to the body of each rule, the context literal not 0:idle(i). This means that the bridge rules behave exactly as before whenever the control context does not decide to let $C_i$ be idle.

For the case where $C_i$ is idle, i.e. where the belief set of $C_0$ contains 0:idle(i), we just make sure that observations are buffered. This means that for each rule of the form

\[
\text{add}(P, T, j) \leftarrow P \otimes j, \text{now}(T) \otimes t
\]

in the original set of bridge rules we add

\[
\text{bf}(P, T, j) \leftarrow P \otimes j, \text{now}(T) \otimes t, 0: \text{idle}(T).
\]

The operation $\text{bf}$ just adds the atom $\text{bf}(p, t, j)$ to the context (we assume here that the language of the context contains constructs of this form). As mentioned above, this information is not used anywhere in the rest of the context’s KB, it just sits there for later use.

The only missing piece is a bridge rule bringing back information from the buffer when the context is no longer idle. This can be done using the bridge rule empty buffer $\leftarrow$ not 0:idle(i). Whenever the management function has to execute this operation, it takes all information out of the buffer, checks whether it is still within the relevant time window, and if this is the case adds it to the KB, handling potential inconsistencies the way discussed in Sect. 4.

The control context uses formulas of the form idle(i, t) to express context i is idle until time t. We intend here to give a proof of concept, not a sophisticated control method. For this reason we simply assume the control context lets a detector context be idle for a specific constant time span $c$ whenever the detector does not suspect an emergency. This is achieved by the following bridge rule schemata:

\[
\text{add}(\text{suspect}(K)) \leftarrow K: \text{suspect}(E) \\
\text{add}(\text{idle}(K, T + c)) \leftarrow \text{now}(T) \otimes t, \text{not 0:suspect}(K), \\
\text{not 0:idle}(K, T'), T' < T + c
\]

Provided information of the form idle(i, t) is kept until the actual time is $t + 2$, the last 2 conditions in the second rule schema guarantee that after being idle for period $c$ the context must check at
least once whether some emergency is suspected. To avoid a context staying idle forever, we assume the management function deletes information of this form whenever \( t \) is smaller than the current time minus 1. One more rule schema to make sure information about idle contexts is available in the form used by detector contexts:

\[
\text{add(idle}(K) \leftarrow \text{now}(T) @ t, 0; \text{idle}(K, T'), T \leq T'.
\]

7 Complexity

We want to analyze the complexity of queries on runs of rMCSs. For simplicity we do not consider parametrized bridge rules here, and assume that all knowledge bases in rMCSs are finite and all management functions can be evaluated in polynomial time.

Definition 9 The problem \( Q^3 \), respectively \( Q^4 \), is deciding whether for a given rMCS \( M \) with sensors \( \Pi \), a context \( C_i \), of \( M \), a belief \( b \) for \( C_i \), and a finite sequence of II-observations \( O \) it holds that \( b \in S_i \) for some \( S^j = (S_1, \ldots, S_n) \) \((0 \leq j \leq n)\) for some run, respectively all runs, \( R = ((S^1, KB^1), \ldots, (S^n, KB^n)) \) of \( M \) induced by \( O \).

As the complexity of an rMCS depends on that of its individual contexts we introduce the notion of complexity family along the lines of Eiter et al. [10]. To do so, we need to focus on relevant parts of belief sets by means of projection. Intuitively, among all beliefs, we only need to consider beliefs that we want to query and beliefs that contribute to the application of bridge rules for deciding \( Q^3 \) and \( Q^4 \). Given \( M, \Pi, C_i \), and \( b \) as in Definition 9 the set of relevant beliefs for a context \( C_j \) of \( M \) is given by \( RB_j(M, i; b) = \{ b' | r \in br_j, h.b' \in bd(r) \lor \text{not } h.b' \in bd(r) \} \cup \{ b | i = j \} \). A projected belief state for \( M \) and \( i; b \) is a tuple \( S^j_{M,i} = (S_1 \cap RB_1(M, i; b), \ldots, S_n \cap RB_n(M, i; b)) \) where \( S = (S_1, \ldots, S_n) \) is a belief state for \( M \). The context complexity of \( C_j \) in \( M \) w.r.t. \( i; b \) for a fixed II-observation \( O \) is the complexity of deciding whether for a given projected belief state \( S \) for \( M \) and \( i; b \), there is some belief state \( S' = (S'_1, \ldots, S'_n) \) for \( M \) with \( S'^j_{M,i} = S \) and \( S'_j \in ACC_j(mng_j(app(S, O), kb_j)) \) for all \( 1 \leq j \leq n \). The system’s context complexity \( \mathcal{CC}(M, i; b) \) is a (smallest) upper bound for the context complexity classes of its contexts. Our complexity results are summarized in Table 1.

| \( \mathcal{CC}(M, i; b) \) | \( Q^3 \) | \( Q^4 \) |
|-------------------------------|--------|--------|
| \( \Sigma^P \) \((i \geq 2)\) | \( \Pi^P \) | \( \Pi^P \) |
| \( \Pi^P \) | \( \Pi^P \) | \( \Pi^P \) |
| \( \Pi^P \) | \( \Pi^P \) | \( \Pi^P \) |

Table 1. Complexity of checking \( Q^3 \) and \( Q^4 \) (membership, completeness holds given hardness for \( \mathcal{CC}(M, i; b) \)).

Membership for \( Q^3 \): a non-deterministic Turing machine can guess a projected belief state \( S^j = (S_1, \ldots, S_n) \) for all \( m \) observations in \( O \) in polynomial time. Then, iteratively for each of the consecutive observations \( obs_j \), first the context problem can be solved polynomially or using an oracle (the guess of \( S^j \) and the oracle guess can be combined which explains that we stay on the same complexity level for higher context complexity). If the answer is ‘yes’, \( S^j \) is a projected equilibrium. We can check whether \( b \in S_i \), compute the updated knowledge bases and continue the iteration until reaching the last observation. The argument is similar for the co-problem of \( Q^4 \). Hardness: holds by a reduction from deciding equilibrium existence for an MCS when \( \mathcal{CC}(M, i; b) \) is polynomial and by a reduction from the context complexity problem for the other results.

Note that \( Q^3 \) and \( Q^4 \) are undecidable if we allow for infinite observations. The reason is that rMCSs are expressive enough (even with very simple context logics) to simulate a Turing machine such that deciding \( Q^3 \) or \( Q^4 \) for infinite runs solves the halting problem.

8 Discussion

In this paper we introduced reactive MCSs, an extension of managed MCSs for online reasoning, and showed how they allow us to handle typical problems arising in this area. Although we restricted our discussion to deterministic management functions, two sources of non-determinism can be spotted by the attentive reader. On the one hand, we allow for semantics that return multiple belief sets for the same knowledge base, and, on the other hand, non-determinism can be introduced through bridge rules.

The simplest example is guessing via positive support cycles, e.g., using bridge rules like \( \text{add}(a) \leftarrow c:a \cdot \text{that allow (under the standard interpretation of add) for belief sets with and without formula } a \cdot \text{Multiple equilibria may lead to an exponential number of runs. In practice, non-determinism will have to be restricted. A simple yet practical solution is to focus on a single run, disregarding alternative equilibria. Here, one might ask which is the best full equilibrium to proceed with. In this respect, it makes sense to differentiate between non-deterministic contexts and non-determinism due to bridge rules.}

In the first case, it is reasonable to adopt the point of view of the answer-set programming (ASP) paradigm, i.e., the knowledge bases of a context can be seen as an encoding of a problem such that the resulting belief sets correspond to the problem solutions. Hence, as every belief set is a solution it does not matter which one to choose. Thus, if the problem to be solved is an optimisation problem that has better and worse solutions, this could be handled by choosing a context formalism able to express preferences so that the semantics only returns sufficiently good solutions. For preferences between equilibria that depend on the belief sets of multiple contexts, one cannot rely on intra-context preference resolution. Here, we refer the reader to preference functions as proposed by Ellmauthaler [12]. One might also adopt language constructs for expressing preferences in ASP such as optimization statements [14] or weak constraints [9]. Essentially, these assign a quality measure to an equilibrium. With such additional quality measures at hand, the best equilibrium can be chosen for the run.

As to related work, there is quite some literature on MCSs by now, for an overview see [6]. Recently an interesting approach to belief change in MCSs has been proposed [13]. Other related work concerns stream reasoning in ASP [13] and in databases: a continuous version of SPARQL [3] exists, and logical considerations about continuous query languages [19] were investigated. Koválski’s logic-based framework for computing [17] is an approach which utilizes first order logic and concepts of the situation- and event-calculus in response to observations. Updates on knowledge-bases, based upon the outcome of a given semantics where also facilitated for other formalisms, like logic programming in general. There the iterative approaches of EPI [11] and EVOLP [1] are the most prominent. Note that none of these related approaches combines a solution to both knowledge integration and online reasoning, as we do.

The idea of updates to the knowledge-base was also formalised for database systems [2].

For a related alternative approach using an operator for directly manipulating KBs without contributing to the current equilibrium, we refer to the work by Gonçalves, Knorr, and Leite [15].
REFERENCES

[1] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira, ‘Evolving logic programs’, in 8th European Conference on Logics in Artificial Intelligence (JELIA 2002), eds., Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, volume 2424 of Lecture Notes in Computer Science, pp. 50–61. Springer, (September 2002).

[2] Chitta Baral, Jorge Lobo, and Goce Trajcevski, ‘Formal characterizations of active databases: Part ii’, in 5th International Conference on Deductive and Object-Oriented Databases (DOOD 1997), eds., François Bry, Raghu Ramakrishnan, and Kotagiri Ramamohanarao, volume 1341 of Lecture Notes in Computer Science, pp. 247–264. Springer, (1997).

[3] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, ‘C-SPARQL: a continuous query language for RDF data streams’, International Journal of Semantic Computing, 4(1), 3–25, (2010).

[4] G. Brewka, ‘Towards reactive multi-context systems’, in 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013), pp. 1–10, (2013).

[5] G. Brewka and T. Eiter, ‘Equilibria in heterogeneous nonmonotonic multi-context systems’, in AAAI’07, pp. 385–390, (2007).

[6] G. Brewka, T. Eiter, and M. Fink, ‘Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources’, in Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, 233–258, Springer, (2011).

[7] G. Brewka, T. Eiter, M. Fink, and A. Weinzierl, ‘Managed multi-context systems’, in JICAI’11, pp. 786–791, (2011).

[8] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer, ‘Multi-context systems for reactive reasoning in dynamic environments’, in 21st European Conference on Artificial Intelligence (ECAI 2014), (2014). To appear.

[9] F. Buccafurri, N. Leone, and P. Rullo, ‘Strong and weak constraints in disjunctive datalog’, in 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1997), pp. 2–17, (1997).

[10] T. Eiter, M. Fink, P. Schüller, and A. Weinzierl, ‘Finding explanations of inconsistency in multi-context systems’, in Proc. KR’10, (2010).

[11] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits, ‘A framework for declarative update specifications in logic programs’, in Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), ed., Bernhard Nebel, pp. 649–654. Morgan Kaufmann, (2001).

[12] S. Ellmauthaler, ‘Generalizing multi-context systems for reactive stream reasoning applications’, in Proceedings of the 2013 Imperial College Computing Student Workshop (ICCSW 2013), pp. 17–24, (2013).

[13] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub, ‘Stream reasoning with answer set programming: Preliminary report’, in Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2012), (2012).

[14] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, A users guide to gringo, clasp, clingo, and iclingo, Potassco Team, 2010.

[15] F. Giunchiglia and L. Serafini, ‘Multilanguage hierarchical logics or: How we can do without modal logics’, Artif. Intell., 65(1), 29–70, (1994).

[16] R. Gonçalves, M. Knorr, and J. Leite, ‘Evolving multi-context systems’, in 21st European Conference on Artificial Intelligence (ECAI 2014), (2014). To appear.

[17] R. A. Kowalski and F. Sadri, ‘Towards a logic-based unifying framework for computing’, CoRR, abs/1301.6905, (2013).

[18] Y. Wang, Z. Zhuang, and K. Wang, ‘Belief change in nonmonotonic multi-context systems’, in 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013), pp. 543–555, (2013).

[19] C. Zaniolo, ‘Logical foundations of continuous query languages for data streams’, in 2nd International Workshop on Datalog in Academia and Industry (Datalog 2.0), pp. 177–189, (2012).