Towards Reactive Multi-Context Systems

Gerhard Brewka

Leipzig University, Informatics Institute, Postfach 100920, 04009 Leipzig, Germany
brewka@informatik.uni-leipzig.de

Abstract. Among the challenges faced by the area of knowledge representation (KR) are the following ones: firstly, knowledge represented in different knowledge representation languages needs to be integrated, and secondly, certain applications have specific needs not typically fulfilled by standard KR systems. What we have in mind here are applications where reasoners, rather than being called by the user in order to answer some specific query, run online and have to deal with a continuous stream of information. In this paper we argue that multi-context systems (MCS) are adequate tools for both challenges. The original MCS approach was introduced to handle the integration problem in a principled way. A later extension to so-called managed MCS appears to provide relevant functionality for the second challenge. In this paper we review both MCS and managed MCS and discuss how the latter approach needs to be further developed for online applications.

1 Introduction

Research in knowledge representation (KR) and, more generally, information technology faces at least the following two problems:

1. A large variety of formats and languages for representing knowledge has been produced. A wealth of tools and formalisms is now available, including rather basic ones like databases or the more recent triple-stores, and more expressive ones like ontology languages (e.g., description logics), temporal and modal logics, nonmonotonic logics, or logic programs under answer set semantics, to name just a few.
   This diversity of formalisms poses some important challenges. There are many situations where the integration of the knowledge represented in such diverse formalisms is crucial. But how can this be achieved in a principled way?
2. 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, for instance expert systems, configuration or planning problems where a specific answer to a 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 observes a particular system, informing the user in case something unforeseen/uninteded happens. This different usage of KR systems again poses important challenges.

Let's illustrate these problems with two examples. For the first problem, assume your home town's hospital has

P. Cabalar and T.C. Son (Eds.): LPNMR 2013, LNAI 8148, pp. 1–10, 2013.

- a patient database (e.g. an Oracle database),
- a disease ontology (written in a particular description logic),
- an ontology of the human body (using OWL),
- an expert system describing the effects of different medications (using a nonmonotonic reasoning formalism, say disjunctive logic programming).

Needless to say that it is in the patients' best interest to integrate all available knowledge. But how? Translating everything into a single all-purpose formalism is certainly not a solution. First of all, no standardized, universal knowledge representation language exists, and there are very good reasons for this (e.g. specific modeling needs or complexity considerations). Secondly, even if there were such a language, most probably remodeling the information would be too cumbersome and costly. What seems to be needed is a principled way of integrating knowledge expressed in different formats/languages/logics.

For the second problem consider an assisted living scenario where people in need of support live in an apartment which is 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). Apparently the system is not explicitly called by a user to become active. It rather is continuously online and must be able to process a continuous stream of information rather than a fixed KB.

This shift in view clearly has an impact on KR formalisms. Most importantly, since the system is permanently online, 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 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 certainly 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.

We believe nonmonotonic multi-context systems (MCS) [1] are promising tools for addressing both problems. The original MCS framework was explicitly developed to handle problem 1, the integration of diverse KR formalisms. In Sect. 2 we recall the basic ideas underlying this approach. In a nutshell, an MCS consists of reasoning units - called contexts for historical reasons [8] - 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 contexts themselves can be viewed as parts of

an agent's knowledge connected to other parts, but they can also be viewed as single agents willing to listen to and incorporate information from other agents.

The original framework was aimed at modelling the flow of information among contexts, consequently the addition of information to a context was the only possible operation. Of course, it is easy to imagine other operations one may want to perform, for instance revisions which keep the underlying KB consistent instead of simple additions. To capture arbitrary operations MCS were later generalized to so called managed MCS (mMCS) [3], a general and flexible framework that we will briefly discuss in Sect. 3. The possibility to have arbitrary operators is what, as we believe, makes mMCS suitable tools for the kind of online applications we discussed earlier. Sect. 4 describes some of the necessary steps that need to be done. In particular, what is required is an instantiation of the general mMCS framework with operations suitable to model focussing and forgetting. The systems we have in mind are *reactive* in the sense that they modify themselves to keep system performance up and in response to potential emergencies.

In cases where knowledge integration is not an issue (that is, where a single context is sufficient) and where moreover the underlying KR formalism is rule based the separation between context knowledge base and bridge rules may become obsolete. We briefly illustrate this in Sect. 5 using ASP as the underlying formalism. Sect. 6 concludes and points to open research questions.

2 Nonmonotonic Multi-Context Systems

The basic idea underlying MCS is to leave the diverse formalisms and knowledge bases untouched, and to equip each context with a collection of so-called bridge rules in order to model the necessary information flow among contexts.

Bridge rules are similar to logic programming rules (including default negation), with an important difference: they allow to access other contexts in their bodies. Using bridge rules has several advantages: the specification of the information flow is fully declarative; moreover, information - rather than simply being passed on as is - can be modified in various ways:

- we may translate a piece of information into the language/format of another context,
- we may pass on an abstraction of the original information, leaving out unnecessary details,
- we may select or hide information,
- we may add conclusions to a context based on the absence of information in another one,
- we may use simple encodings of preferences among parent contexts,
- we can even encode voting rules, say based on majorities etc.

The semantics of MCS is defined in terms of equilibria: a belief state assigns a belief set to each context C_i. Intuitively, a belief state is an equilibrium whenever the belief set selected for each C_i is acceptable for C_i's knowledge base *augmented by the heads of C_i's applicable bridge rules.*

The history of MCS started in Trento. Advancing work in [7,9], the Trento School developed monotonic heterogeneous multi-context systems [8] with the aim to integrate

different inference systems. Here reasoning within as well as across contexts is monotonic. The first, still somewhat limited attempts to include nonmonotonic reasoning were done in [10] and [4]. To allow for reasoning based on the *absence* of information from a context, in both papers default negation is allowed in the rules. In this way contextual and default reasoning are combined.

The nonmonotonic MCS of [1] substantially generalized these approaches, by accommodating *heterogeneous* and both *monotonic* and *nonmonotonic* contexts. They are thus capable of integrating "typical" monotonic logics like description logics or temporal logics, and nonmonotonic formalisms like Reiter's default logic, answer set programs, circumscription, defeasible logic, or theories in autoepistemic logic. The currently most general MCS variant, the so-called managed MCS (mMCS) [3] allow for arbitrary user-defined operations on the context knowledge bases, not just augmentations. They will be discussed in the next section.

Here is a more formal description of multi-context systems as defined in [1]. MCS build 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 knowledge-base a set of acceptable belief sets.

A *multi-context system (MCS)* $M = (C_1, \ldots, C_n)$ is a collection of contexts $C_i = (L_i, kb_i, br_i)$ where L_i is a logic, $kb_i \in KB_{L_i}$ is a knowledge base and br_i is a set of bridge rules of the form:

$$s \leftarrow (c_1{:}\, p_1), \ldots, (c_j{:}\, p_j), not(c_{j+1}{:}\, p_{j+1}), \ldots, not(c_m{:}\, p_m). \tag{1}$$

such that $kb \cup \{s\}$ is an element of KB_{L_i}, $c_\ell \in \{1, \ldots, n\}$, and p_ℓ is element of some belief set of BS_{c_ℓ}, for all $1 \leq \ell \leq m$. For a bridge rule r, we denote by $hd(r)$ the formula s while $body(r)$ denotes the set $\{(c_{\ell_1}{:}\, p_{\ell_1}) \mid 1 \leq \ell_1 \leq j\} \cup \{not(c_{\ell_2}{:}\, p_{\ell_2}) \mid j < \ell_2 \leq m\}$.

A belief state $S = (S_1, \ldots, S_n)$ for M consists of belief sets $S_i \in BS_i$, $1 \leq i \leq n$. A bridge rule r of form (1) is applicable wrt. S, denoted by $S \models body(r)$, iff $p_\ell \in S_{c_\ell}$ for $1 \leq \ell \leq j$ and $p_\ell \notin S_{c_\ell}$ for $j < \ell \leq m$. We use $app_i(S) = \{hd(r) \mid r \in br_i \wedge S \models body(r)\}$ to denote the heads of all applicable bridge rules of context C_i wrt. S.

The semantics of an MCS M is then defined in terms of equilibria, where an *equilibrium* is a belief state $(S_1, \ldots, S_n)$ such that $S_i \in ACC_i(kb_i \cup app_i(S))$, $1 \leq i \leq n$.

3 Managed MCS: Beyond Information Flow

Although nonmonotonic MCS are, as we believe, an excellent starting point to address the problems discussed above, the way they integrate knowledge is still somewhat limited: if a bridge rule for a context is applicable, then the rule head is simply added to the context's knowledge base (KB). Although this covers the flow of information, it does not capture other operations one may want to perform on context KBs. For instance, rather than simply *adding* a formula ϕ, we may want to delete some information, or to *revise* the KB with ϕ to avoid inconsistency in the context's belief set. We are thus

interested in generalizations of the MCS approach where specific predefined operations on knowledge bases can be performed.

A first step into this direction are argumentation context systems (ACS) [2]. They specialize MCS in one respect, and are more general in another. First of all, in contrast to nonmonotonic MCS they are homogeneous in the sense that all reasoning components in an ACS are of the same type, namely Dung-style argumentation frameworks [5]. The latter are widely used as abstract models of argumentation. However, ACS go beyond MCS in two important aspects:

1. The influence of an ACS module M_1 on another module M_2 can be much stronger than in an MCS. M_1 may not only provide information for M_2 and thus augment the latter, it may directly affect M_2's KB and reasoning mode: M_1 may invalidate arguments or attack relationships in M_2's argumentation framework, and even determine the semantics to be used by M_2.
2. A major focus in ACS is on *inconsistency handling*. Modules are equipped with additional components called *mediators*. The main role of the mediator is to take care of inconsistencies in the information provided by connected modules. It collects the information coming in from connected modules and turns it into a consistent update specification for its module, using a pre-specified consistency handling method which may be based on preference information about other modules.

Managed MCS (mMCS) push the idea of mediators even further. They allow additional operations on knowledge bases to be freely defined; this is akin to management functionality of database systems. We thus call the additional component *context manager*. In a nutshell (and somewhat simplifying) the features of mMCS are as follows:

- Each logic comes with a set of operations O.
- An operational statement is an operation applied to a formula (e.g. insert(p), delete(p), revise(p), ...).
- Bridge rules are as before, except for the heads which now are operational statements.
- A management function: $mng : 2^{Opst} \times KB \rightarrow 2^{KB}$, produces a collection of KBs out of set of operational statements and a KB.
- A managed context consists of a logic, a KB, a set of bridge rules (as before), together with the new part, a management function.
- An mMCS is just a collection of managed contexts.

Regarding the semantics, a belief state $S = (S_1, \ldots S_n)$ contains - as before - a belief set for each context. To be an equilibrium S has to satisfy the following condition: the belief set chosen for each context must be acceptable for one of the KBs obtained by applying the management function to the heads of applicable bridge rules and the context's KB. More formally, for all contexts $C_i = (L_i, kb_i, br_i, mng_i)$: let S_i be the belief set chosen for C_i, and let Op_i be the heads of bridge rules in br_i applicable in S. Then S is an equilibrium iff, for $1 \leq i \leq n$,

$$S_i \in ACC_i(kb') \text{ for some } kb' \in mng_i(Op_i, kb_i).$$

Management functions allow us to model all sorts of modifications of a context's knowledge base and thus make mMCS a powerful tool for describing the influence contexts

can have on each other. Of course, the framework is very general and needs to be instantiated adequately for particular problems. As a short illustrative example let us consider an instantiation we call revision-based MCS. The main goal here is to keep each context's KB consistent when information is added, that is, we want to guarantee consistency of belief sets in equilibria.

Assume the KB's logic has a single operation inc (include). For a formula p, $inc(p)$ intuitively says: incorporate p consistently into your KB. Two things can go wrong: the formulas to be included in a particular situation

1. may be inconsistent with each other, or
2. may be inconsistent with the context KB.

For 1 we introduce preferences among bridge rules. More precisely, we represent a total preorder on bridge rules by using indexed operations $inc_1, inc_2, \ldots$ where a lower index represents higher priority. Given a collection of indexed inclusion operations we can now identify preferred sets of formulas as follows: we pick a maxi-consistent subset of inc_1-formulas (i.e. formulas appearing as arguments of inc_1), extend the set maxi-consistently with inc_2-formulas etc.

For 2 we assume a consistency preserving base revision operator

$$rev : KB \times KB \rightarrow 2^{KB}$$

that is, $rev(kb_1, kb_2)$ may potentially produce alternative outcomes of the revision, however each outcome is consistent whenever kb_2 is. We can now define the management function as follows: for

$$Op = \{inc_1(p_{1,1}), \ldots, inc_1(p_{1,m}), \ldots, inc_k(p_{k,1}), \ldots, inc_k(p_{k,n})\}$$

let:

$$kb' \in mng(Op, kb) \text{ iff } kb' \in rev(kb, F) \text{ for some preferred set } F \text{ of } Op.$$

Each belief set in each equilibrium now is apparently consistent.

Here is a specific example. Let C be a context based on propositional logic, its KB is

$$\{July \rightarrow \neg Rain, Rain \rightarrow Umbrella, July\}.$$

C has 2 parent contexts; C_1 believes $Rain$, C_2 believes $\neg Rain$. C_1 more reliable wrt. the weather. C thus has the following bridge rules:[1]

$$\{inc_1([\neg]Rain) \leftarrow 1{:}[\neg]Rain;\ inc_2([\neg]Rain) \leftarrow 2{:}[\neg]Rain\}.$$

As C_1 is preferred to C_2 the single preferred set is $\{Rain\}$.

To fully specify the management function we still need to define the revision operator. We do this as follows: $K' \in rev(K, F)$ iff $K' = M \cup F$ for some maximal $M \subseteq K$ consistent with F. Now we obtain the following two acceptable belief sets for C:

$$Th(\{Rain, July \rightarrow \neg Rain, Rain \rightarrow Umbrella\})$$
$$Th(\{Rain, July, Rain \rightarrow Umbrella\}).$$

[1] We use square brackets in the rules to represent optional parts; each rule with $[\neg]$ thus actually represents 2 rules.

4 Reactive MCS: A Sketch

In this section we discuss some of the issues that need to be addressed for applications like the assisted living scenario we described in the introduction. We believe managed MCS are an excellent starting point for the following reasons:

- they offer means to integrate sensor information from different sources, handling inconsistencies if needed,
- the management function provides capabilities to modify KBs which appear essential to keep the sizes of knowledge bases manageable.

Nevertheless, the general managed MCS framework obviously needs to be further modified, respectively instantiated, to become reactive. What we aim for is a specialization of the (potentially generalized) managed MCS framework suitable for online applications. Here we identify some of the relevant changes.

First of all, it is useful to introduce different types of contexts:

- observer contexts which are "connected to the real world via sensors; these contexts keep track of (time-stamped) sensor readings,
- analyzer contexts which reason about the current situation and in particular detect emergencies; they obtain relevant information from sensing contexts via bridge rules and generate alarms if needed,
- control contexts which make sure the system focuses on the right issues; this includes dynamically setting adequate time windows for information, increasing the frequency of sensor readings if relevant/dangerous things happen, making sure outdated/irrelevant information is deleted/disregarded to keep the system performance up.

Next, the management function needs to be instantiated adequately for purposes of focusing and forgetting:

- a language of adequate operations for focusing and forgetting needs to be defined,
- ideally the performed operations may also depend on the actual system performance,
- it would be highly useful if the management function were able to restrict recomputations to specific, relevant contexts.

Finally, we anticipate that preferences will play an essential role:

- for inconsistency handling among different sensor readings,
- to handle more important emergencies with high priority,
- to mediate between what's in the current focus and the goal not to overlook important events.

We thus will need to equip MCS with expressive and flexible preference handling capabilities.

As pointed out earlier, in online applications the major object of interest is the system behaviour over time. For this reason we next define runs of reactive MCS, an essential basic notion:

Definition 1. *Let M be a managed MCS with contexts $C_0, \ldots, C_n$ ($C_0, \ldots, C_k$ are observer contexts). Let $Obs = (Obs^0, Obs^1, \ldots)$ be a sequence of observations, that is, for $j \geq 0$, $Obs^j = (Obs_i^j)_{i \leq k}$, where Obs_i^j is the new (sensor) information for context i at step j. A run R of M induced by Obs is a sequence*

$$R = Kb^0, Eq^0, Kb^1, Eq^1, \ldots$$

where

- *$Kb^0 = (Kb_i^0)_{i \leq n}$ is the collection of initial knowledge bases, Eq^0 an equilibrium of Kb^0,*
- *for $j \geq 1$ and $i \leq n$, Kb_i^j is the knowledge base of context C_i produced be the context's management function for the computation of Eq^{j-1}, and $Kb^j = (Kb_i^j)_{i \leq n}$,*
- *for $j \geq 1$, Eq^j is an equilibrium for the knowledge bases*

$$(Kb_0^j \cup Obs_0^j, \ldots, Kb_k^j \cup Obs_k^j, Kb_{k+1}^j, \ldots, Kb_n^j).$$

5 Reactive ASP: A Bottom Up Approach

In the last section we sketched some of the issues that need to be addressed in order to turn MCS into a reactive formalism suitable for online applications. The basic idea was to handle reactivity by adequate operations in bridge rules. We now consider cases where the integration of information from different sources is not an issue and where we work with a single context. The separation between context and bridge rules is still relevant as the bridge rules (which now should better be called operational rules as they do no longer bridge different contexts) implement the focusing and forgetting strategies of the context.

However, if the single context we work with is itself rule based, then strictly speaking the separation of bridge/operational rules from the rest of the program becomes obsolete. We may as well use operational rules within the formalism itself. For instance, assume we use logic programs under answer set semantics.[2] Some of the rules in the program may have operational statements in their heads which simply are interpreted as operations to be performed on the program itself. Again, this allows us to represent the strategy for maintaining the knowledge base manageable declaratively. When the program is run, the strategy is realized by a self-modification of the program.

This is reflected in the following notion of a run of a reactive answer set program (RASP) P, that is an ASP program which has some rules with operational statements in the heads. Intuitively, the behaviour of RASP P is characterized as follows:

1. P computes an answer set S_0, during the computation the current information is frozen until the computation is finished,
2. the set of operations to be performed is read off the answer set S_0 and P is modified accordingly, at the same time observations made since the last computation started are added,

[2] For some important steps towards stream reasoning with answer set programming see also the work of Torsten Schaub's group, e.g [6].

3. the modified program computes a new answer set, and so on.

This is captured in the following definition of a run:

Definition 2. *A run of a reactive answer set program P induced by a sequence of sets of observations $(Obs_0, Obs_1, \ldots)$ is a sequence $(S_0, S_1, \ldots)$ of answer sets satisfying the following conditions:*

1. *S_0 is an answer set of $P_0 = P$.*
2. *For $i \geq 0$, S_{i+1} is an answer set of $P_{i+1} = Mod_i(P_i) \cup Obs_i$, where $Mod_i(P_i)$ is the result of modifying P_i according to the operational statements contained in S_i.*

This definition implies new information obtained while the last answer set was computed is always included in the new. modified program. In certain situations, for instance if parts of the new knowledge are outside the current focus, it may even be useful to disregard pieces of the new information entirely. Formally this can be captured by letting $P_{i+1} = Mod_i(P_i \cup Obs_i)$ in item 2 of the definition above.

6 Discussion and Future Work

In this paper we discussed some of the issues that need to be addressed if KR wants to meet the challenges of certain types of applications where continuous online reasoning is required. We sketched a top down approach, instantiating the managed MCS approach accordingly. We also briefly described a bottom up approach based on a related extension of the ASP framework.

We obviously left many questions open. The major open issue is the specification of a suitable language for the operational statements which are relevant for forgetting and focusing. The operations should allow to set the window size for specific sensor/information types dynamically, keeping relevant information available. They also should make it possible to specify the system's focus depending on events pointing to potential problems/emergencies. Focusing may lead to more regular checks for specific information, whereas other information may be looked at only from time to time.

Of course, focusing bears the danger that new problems may be overlooked. Ideally, we would like to have a guarantee that every potential emergency is checked on a regular basis, even if it is not in the current focus. In addition, it would be very useful to take information about the current system performance into account to determine what information to keep and what to give up. This would lead to a notion of resource-aware computation where part of the sensor information made available in each step of a run reveals how the system currently is performing.

The notions of a run we defined both for managed MCS and for reactive ASP is built on a credulous view: in each step a single equilibrium, respectively answer set, is computed and taken as the starting point for the next step. There may be scenarios where a skeptical approach built on what is believed in all (or in some preferred) equilibria/answer sets is more adequate. It may even be useful to switch between the credulous and skeptical approach dynamically.

Finally, if memory is not an issue (but computation time is) then rather than deleting irrelevant information one could as well keep it but put it aside for a certain time.

The available information would thus be divided in a part to be forgotten, a part to be kept but disregarded for the time being, and a part currently in the focus of attention.

In conclusion, a lot remains to be done in KR to fully solve the challenges of integration and online reasoning. Nevertheless, we believe promising ideas are already around and addressing the open problems will definitely be worth it.

Acknowledgements. Some of the ideas presented here are based on discussions with Torsten Schaub and Stefan Ellmauthaler. The presented work was partly funded by Deutsche Forschungsgemeinschaft, grant number FOR 1513.

References

1. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: Proc. AAAI 2007, pp. 385–390. AAAI Press (2007)
2. Brewka, G., Eiter, T.: Argumentation context systems: A framework for abstract group argumentation. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 44–57. Springer, Heidelberg (2009)
3. Brewka, G., Eiter, T., Fink, M., Weinzierl, A.: Managed multi-context systems. In: Proc. IJCAI 2011, pp. 786–791 (2011)
4. Brewka, G., Roelofsen, F., Serafini, L.: Contextual default reasoning. In: Proc. IJCAI 2007, pp. 268–273 (2007)
5. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–358 (1995)
6. Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T.: Stream reasoning with answer set programming: Preliminary report. In: Proc. KR 2012, pp. 613–617 (2012)
7. Giunchiglia, F.: Contextual reasoning. Epistemologia XVI, 345–364 (1993)
8. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics or: How we can do without modal logics. Artif. Intell. 65(1), 29–70 (1994)
9. McCarthy, J.: Generality in artificial intelligence. Commun. ACM 30(12), 1029–1035 (1987)
10. Roelofsen, F., Serafini, L.: Minimal and absent information in contexts. In: Proc. IJCAI 2005 (2005)