Representing and Communicating Context in Multiagent Systems

Sonia Rode and Roy M. Turner(✉)

School of Computing and Information Science,
University of Maine, Orono, ME 04469, USA
{sonia.rode,rturner}@maine.edu

Abstract. Context-aware agents operating in a cooperative multiagent system (MAS) can benefit from establishing a shared view of their context, since this increases coherence and consistency in the system's behavior. To this end, agents must share contextual knowledge with each other. In our prior work on context-mediated behavior, agents used frame-based contextual schemas (c-schemas) to explicitly represent and reason about context. While an expressively rich approach, the lack of formal structure poses problems for mutual understanding of c-schemas among agents in a MAS. As we are interested MASs with heterogeneous agents, not only will agents represent c-schemas in idiosyncratic ways, but the set of c-schemas known by each agent will differ. In this paper we propose a new, related representation of contextual knowledge using description logic and a shared ontology, and we present a technique for communicating contextual knowledge while respecting bandwidth limitations.

Keywords: Multiagent systems · Communicating contextual knowledge · Context representation

1 Introduction

Multiagent systems (MASs) are groups of intelligent agents that interact, usually to carry out a set of goals. They are of interest for a variety of tasks, from autonomous exploration to data collection (e.g., [6]) to e-commerce. A MAS may be cooperative, in which case the agents work together to achieve common goals, or the individual agents may be self-interested and each work to satisfy their own goals, which may or may not align with some global set of goals.

While a great deal of work has been done on the problem of ensuring that individual agents behave appropriately for their context, much of it reported in this conference series, much less has focused on agents working in a MAS. However, context-appropriate behavior is just as important for an agent in a MAS, and it is important that the MAS as a whole behaves appropriately for its context.

The problem is more difficult than for a single agent. The contexts of individual agents now *always* include other agents, which may be unpredictable to

H. Christiansen et al. (Eds.): CONTEXT 2015, LNAI 9405, pp. 257–270, 2015.
DOI: 10.1007/978-3-319-25591-0_19

some extent and which are themselves behaving in ways influenced by their own contexts. There is also the opportunity for agents to gather information from others to better understand their context, but at the cost of added complexity, effort, and time.

While we could focus only on agent-level context recognition and hope that globally-appropriate behavior will emerge from the interactions of the agents, this suffers from problems similar to agent-level control of planning in multiagent systems, primarily a lack of global coherence. Instead, the MAS, or at least a subset of its agents, should attempt to share information about the context to arrive at a shared, more complete "partial global context" (to borrow from partial global planning [7]). The individual agents, as well as any control mechanisms for the MAS as a whole, can then take the shared context into account when determining how to achieve individual goals, organize the agents, assign tasks, and coordinate agents' actions.

The ability to reason and communicate about the context implies that the context is explicitly represented. It also implies that there is a *message protocol* for communicating about the context, and that there is a way for different agents to represent contexts and agree on the meaning of contextual knowledge. This implies that there exists a representation language for contextual knowledge as well as a shared ontology that the agents can refer to for terms' meanings.

In earlier work [18], we described an approach to multiagent context-appropriate behavior that we called distributed context-mediated behavior, which we refer to here as multiagent context-mediated behavior, or MASCon.[1] This approach relies on agents communicating about their perceived context, knowledge, goals, and percepts in order to arrive at a representation of the MAS's global context (the context representation, or CoRe). The process involves context representation, local context assessment, communication, and a distributed assessment of the global context.

This paper focuses on the communication aspects of MASCon, including how contextual knowledge is represented to facilitate communication (and reasoning) about context. We focus first on representation, and describe the description logic-based representation of contexts and contextual knowledge, an ontology for contextual knowledge, and how that knowledge is represented as c-schemas. The c-schemas themselves can be viewed collectively as forming a kind of ontology for contexts. We then discuss context-related communication in MASCon. Part of this involves a message protocol for communicating about context that attempts to minimize bandwidth needed, (synergistic with any data compression that might be used) which is a key concern for some domains (e.g., a MAS consisting of underwater vehicles). The other part is deciding what to communicate about the context in order for agents to arrive at a shared understanding.

[1] The name seems appropriate, since just as a mascon is a concentration of mass that affects (e.g.) a satellite's orbit, our approach relies on a concentration of contextual knowledge to affect a MAS's behavior.

2 Representation Language

Agents need a shared language and ontology in order to communicate about anything, not just their context, and there has been much work on both in artificial intelligence, especially in the area of multiagent systems. Unfortunately, most work on knowledge representation and ontologies has not focused on contextual knowledge per se, but rather on domain and problem-solving (e.g., planning) knowledge. This is largely because context has seldom been considered as a first-order concept, but instead has been treated implicitly.

Context *has* been considered a first-order concept in some formal logic work, especially in the context community (e.g., [4,8,12]), and in some non-logical approaches (e.g., [3,9]). Our previous work has also addressed this by creating explicit representations for contexts (c-schemas) and for the contextual knowledge they must contain [17].

Unfortunately, our prior work lacked a formal representation language, and the ontology and semantics were idiosyncratic to each project and somewhat ad hoc. This is problematic if agents are to interact with others that may not have the same designers or reasoning mechanisms, such as would be the case in some open multiagent systems (e.g., autonomous oceanographic sampling networks [6]). In addition, the representation was frame-based, which has some beneficial properties, especially knowledge clustering, but for which there are no really good, widely-accepted reasoning mechanisms as there are, say, for formal logic.

What is needed, then, is an ontology for contextual knowledge and a way to represent it for communication that has a well-defined, formal basis, for which there are tractable reasoning mechanisms, and that is amenable to being related to a shared ontology.

For these reasons, we are now basing our contextual knowledge representation on *description logic* [1], a widely-used formalism in multiagent systems and the semantic web [2]. There has been other work on representing contextual knowledge as description logic, for example the work of Wang et al. [19], which was based on the Web Ontology Language (OWL). However, since their representation of context does not include guidance for behavior, it is not sufficient for our purposes.

We assume that most readers will have some familiarity with description logic (DL), and only a quick overview is presented here to allow others to understand terms in the rest of the paper. DL is a set of languages based around the idea of sets of individuals, restrictions on set membership, operators, and subsumption. A description of a set of individuals is termed a *concept*, for example, `AUV` (autonomous underwater vehicle). Concepts are viewed as having *roles* that can be used to restrict the individuals that are members of the set; for example, `(AND AUV (SOME hasColor Yellow))`[2] would denote the set of yellow AUVs. This example also shows an operator, conjunction, and the existential quantifier.

[2] Sometimes written $\texttt{AUV} \sqcap \exists\, \texttt{hasColor.Yellow}$.

Determining *subsumption* is the primary inference type in DL: if A and B are concepts (i.e., descriptions of sets), then A is said to subsume B if $B \subseteq A$. For example, the atomic concept `AUV` subsumes the more restricted description above for yellow AUVs. Although relatively straightforward, for some description logics, subsumption checking is intractable in the worst case.

The particular DL we use in our work is a version of the language L1 as specified by Teege [15], which allows concept union, concept intersection, existential role restriction, minimum cardinality role restriction, and role composition. Axioms can be defined using these operators and used as concept definitions. The operators provide sufficient expressive power for our purposes. The L1 language has the property of *structural subsumption*, which means that the subsumption test on concepts always reduces to subsumption tests of single clauses. A clause is a description that cannot be further decomposed into a conjunction. The reason we require structural subsumption is to allow for an efficient algorithm for communicating contexts, which will be discussed in detail later.[3]

The version of L1 we use adds datatypes, equivalent to the way one of the Web Ontology Language (OWL) variants, OWL-DL [13], uses them. Datatype roles are permitted, which are similar to regular roles but with data values (e.g., integers, strings, etc.) as opposed to concepts as the fillers. This does not interfere with the use of structural subsumption in our algorithm. For our purposes, datatype roles are treated like regular roles, and we consider that a data value "subsumes" another whenever the two values are of the same type.

A concept such as `Yellow` or `AUV` is an *atomic concept*. These concepts live in an *ontology*, an `isa` hierarchy that directly shows the set–subset relationships between the concepts. Figure 1 shows a portion of the ontology we use in this project, for example (with subtrees not shown for some concepts).

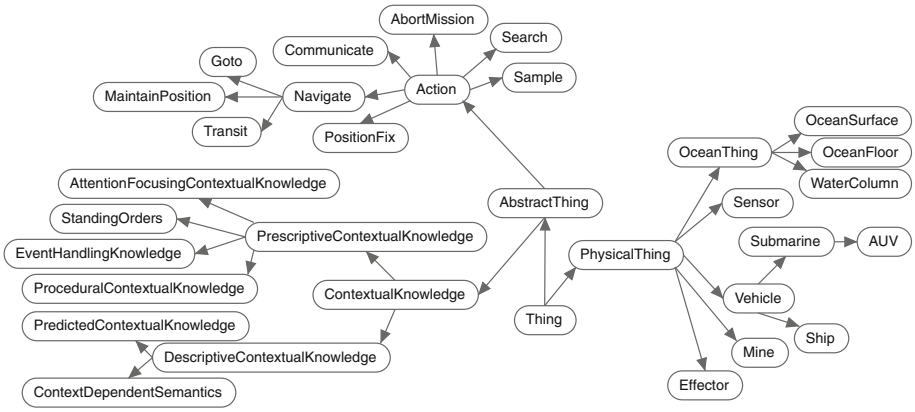

Fig. 1. A portion of the ontology. `Thing` is the top-level concept. Some subtrees are hidden to save space.

[3] Structural subsumption is weaker than logical subsumption [1].

All concept definitions as well as all axioms in DL belong to what is called the reasoner's *terminological box*, or TBox. In the case of this project, all agents will have some common knowledge in their TBoxes, that is, their shared knowledge, including the ontology, will mostly reside there.

3 Contextual Knowledge

There are two main concerns when explicitly representing knowledge about contexts: representing the contexts themselves, and representing the kinds of contextual knowledge they can contain.

MASCon, like all of our projects based on context-mediated behavior, represents contexts as knowledge structures called *contextual schemas*, or c-schemas. In the past, these have been frame-like structures with roles defining the kinds of knowledge being represented: knowledge for handling unanticipated events, for modifying goal priorities, and so forth. In this project, we have largely done away with this frame-like nature. Instead, c-schemas are primarily containers for description logic statements (concept descriptions and axioms) that apply in the represented context (cf. Guha's [10] microtheories).

A c-schema contains several types of knowledge about the represented context, each of which is best represented as a type of knowledge in its own right, i.e., a concept in the ontology. This allows the reasoner to easily determine what the knowledge is and how it is meant to be applied in the situation. Some researchers in the pervasive computing community (e.g., [19]) have also developed ontologies for context, but the kinds of contextual information used in that community tend to be only a subset of what is needed for context-sensitive behavior in agent-based and multiagent based systems.

We can broadly classify the needed contextual knowledge as being either *descriptive* or *prescriptive*. The former describes the features expected in the context, that is, the features of situations that are instances of the context. This knowledge is used in assessing the context, and it can be used to generate predictions about unseen features of the situation and to help understand newly-seen features. Part of descriptive knowledge is also any context-dependent semantics, for example, what a fuzzy logic or description logic concept might mean in the context that is different from its normal meaning. Prescriptive knowledge tells the agent how to behave in the context. There are several kinds: knowledge about goals and their context-appropriate priority, ways to achieve goals in the context, how to recognize and handle unanticipated events, and how to set non-goal-based behavioral parameters appropriately (e.g., sonar status, recommended depth envelope, etc.). In the case of multiagent systems, actions would also include such things as how/when/what to communicate, how to organize the group of agents, authority relationships, if any, and so forth.

Each kind of contextual knowledge is present in the agents' shared ontology, as shown in Fig. 1 (`StandingOrder` represents behavioral parameter settings). This allows agents to communicate about the contents of c-schemas without the problem of being misunderstood. The definitions of concepts in the ontology contain not just their name, but also their roles and their definitions in

PredictedContextFeatures:
(expectsPresenceOf *some* AbstractThing) *or*
(expectsPresenceOf *some* PhysicalThing)
ContextDependentSemantics:
(hasFuzzyFeature *some*
(AbstractThing *and*
(hasFuzzyMembershipFunction *some* FuzzyMembershipFunction)))
StandingOrders:
(hasActivePeriod *some* ContextLifeCycle)*and*
(hasOperationalSetting *some*
((SelfState *and* (hasAdvisedLowerBound *some* Number)) *or*
(SelfState *and* (hasAdvisedUpperBound *some* Number)) *or*
(SelfState *and* (hasAdvisedValue *some* ValuePartition))))
EventHandlingKnowledge:
(handlesEvent *some* Event) *and* (hasImportance *some* DefaultValuePartition)
and (respondsWithAction *some* Action)
AttentionFocusingKnowledge:
(definesGoal *some* AbstractThing) *and*
(hasCost *some* DefaultValuePartition) *and*
(hasDegreeExpected *some* DefaultValuePartition) *and*
(hasImportance *some* DefaultValuePartition) *and* (isAchievedBy *some* Action)
ProceduralKnowledge:
(definesAction *some* Action)

Fig. 2. Some definitions of contextual knowledge concepts

terms of other concepts. Figure 2 shows some of the definitions for our contextual knowledge.

The actual concept descriptions in a c-schema will make use of these concepts; for example, the description:

```
(AND EventHandlingKnowledge (SOME handlesEvent PowerFailure)
                            (SOME hasImportance High)
                            (SOME respondsWithAction AbortMission))
```

describes a piece of event-handling knowledge telling an AUV that is suffering a power failure that the event is very important and can best be handled in this context by aborting the mission.

Within a c-schema, a piece of contextual knowledge (an assertion) is associated with a name that is unique across all of the agent's knowledge. For shared ("prototype") c-schemas that are part of the MAS's common knowledge, all agents know these unique names. We require this to reduce bandwidth via our message protocol (see below).

In addition to a name, each assertion can also have associated metadata that is not part of the DL description. This is useful for knowledge that would be inconvenient or impossible to represent using the description logic in use. For example, we would like each concept within a c-schema to have an associated certainty factor (CF) representing the agent's certainty that the concept occurs

or is relevant to the context; this is used by MASCon in context assessment, making predictions, etc. However, if the CFs were represented as part of the assertion, then they would be taken into account during subsumption, causing subsumption that would otherwise succeed to instead fail due solely to differing CFs. Consequently, CFs are represented as metadata.

Figure 3 shows an example of part of a contextual schema in our approach, in this case, one that represents being in the context of performing a sampling mission. The names of the pieces of contextual knowledge are unimportant for our purposes, but note that each piece has a description and a piece of metadata, the certainty factor. The c-schema predicts (or matches) that the mission area is large (`BSM-1`) and that the agent has a sampling mission active (`BSM-2`). What "large" (`Broad`) means is also defined in terms of a fuzzy membership function (`BSM-3`). This is one way in which the semantics of terms used by the agent are context-dependent in our approach. The c-schema also suggests a behavioral parameter setting that is appropriate for the context (a "standing order"), i.e., that obstacle sensitivity should be `High` (`BSM-4`). The portion shown also contains some event-handling knowledge about sensor failures as well as some action information.

```
BSM-1(0.60): (AND PredictiveContextualFeature
                (SOME expectsPresenceOf
                     (AND MissionArea (SOME hasFuzzyValue Broad))))
BSM-2(0.34): (AND PredictiveContextualFeature
                (SOME expectsPresenceOf (AND SamplingMission
                                         (SOME hasSamplingTarget Thing))))
BSM-3(0.41): (AND ContextDependentSemantics
                 (SOME hasFuzzyFeature
                      (AND Broad
                        (SOME hasFuzzyMembershipFunction
                             (AND ShoulderFunction
                               (SOME hasLocalMinAt (AND Float 5.0 mi^2))
                                                   (SOME hasLocalMaxAt
                                                          (AND Float 20.0 mi^2)))))))
BSM-4(0.59): (AND StandingOrder
                (SOME hasActivePeriod DuringContext)
                (SOME hasOperationalSetting
                      (AND ObstacleSensitivity (SOME hasAdvisedValue High))))
SM-5(0.064): (AND EventHandlingKnowledge
                (SOME handlesEvent SensorFailure)
                (SOME hasImportance High)
                (SOME respondsWithAction
                      (AND TransferData (SOME hasObject
                                              (AND Sensor
                                                (SOME hasStatus FailureImminent)))
                                                (SOME toObject MAS))))
BSM-7(0.55): (AND EventHandlingKnowledge
                (SOME handlesEvent SamplingComplete)
                (SOME hasImportance High)
                (SOME respondsWithAction Transit))
BSM-8(0.74): (AND ProceduralKnowledge
                (SOME definesAction
                     (AND Transit (SOME hasIndex Survey))))
```

Fig. 3. Part of the contextual schema `BroadSamplingMissionCtx` (Format: `label(CF): description`)

Conceptually, contextual schemas themselves are part of the ontology. Each c-schema describes a concept corresponding to a context, or set of situations. C-schemas exist within generalization/specialization hierarchies in much the same way concepts are related an ontology. For example, the context "in a harbor" is a generalization of "in Portsmouth Harbor". In addition, agents share knowledge about c-schemas representing prototype contexts.

However, c-schemas are different than other concept descriptions. First, they contain a significant amount of knowledge that is not in the form of DL roles (e.g., metadata, information to index other related c-schemas, etc.). Second, although they do have specialization/generalization relationships, these can represent other c-schemas that they were derived from or that they can be found from (in memory) instead of true subclass relationships. Third, unlike an ontology, both the set of c-schemas and their relationships to one another are expected to change relatively frequently as an agent experiences new situations that lead to creating new c-schemas or modifying existing ones. And fourth, unlike ontology concepts, many c-schemas will be idiosyncratic to particular agents, since not all agents will experience the same contexts as they operate.

For these reasons, we treat c-schemas differently than other parts of an agent's ontology. An agent has a separate c-schema memory that changes over time as it gains experience. The schema memory is assumed to organize c-schemas in generalization/specialization hierarchies that can be traversed based on features of the situation to find c-schema(s) matching the situation [17,18]. Such a memory is essentially a set of dynamic discrimination networks that change based on the memory's contents. As an agent gains experience, it will create and store new c-schemas in this memory, and it may learn new connections between existing c-schemas. Shared prototype c-schemas are considered fixed across the agents, corresponding in some ways to a shared "upper ontology". However, how c-schemas are indexed in agents' memory and any idiosyncratic c-schemas derived from the prototypes will likely differ from agent to agent.

4 Communicating About Context

In MASCon, agents communicate about their context during distributed context assessment as well as when deciding what the context means in terms of their behavior. If only the prototype contexts were considered, communication would be trivial: just an identifier for the shared c-schema would need to be sent. However, it is more likely over time, as agents learn new contexts, that agents will each believe that the current situation is an instance of one of their own known idiosyncratic contexts.

We do not want the agents just to send the complete contents of such c-schemas to others. One reason is bandwidth. In many domains of interest, bandwidth is quite limited; for example, in the underwater vehicle domain, maximum bandwidth is on the order of 60 kbit/s or less [14]. Consequently, saving bandwidth is critical for MASs operating in those environments. The second reason is a matter of focus. If all information is sent, then the other agent has to try to

match a large amount of knowledge against all of its own c-schemas; if we can provide some commonality, then the receiver can focus immediately on its own c-schema and the differences between it and what was sent.

In our approach, an agent makes use of its shared contextual knowledge, represented as prototypical c-schemas, to communicate only what is needed to allow the recipient to regenerate the idiosyncratic contexts from its own prototypical contextual schemas. This is reminiscent of earlier work on the agent communication language COLA [16], which also was concerned with limiting bandwidth by appeal to shared knowledge. While at the current time we are focused on communicating contextual knowledge, our approach is not incompatible with a COLA-based communication system.

There are two major problems to be addressed for agents communicating about contextual knowledge. The first is what *message protocol* to use when exchanging messages. The second is determining what to send.

4.1 Message Protocol

MASCon's message protocol focuses on the different kinds of relationships between knowledge in an idiosyncratic context and in prototype ancestors. Communicating about an idiosyncratic context will require multiple messages, since the prototypical context will have to be identified, then differences from it will need to be communicated. Which message types are sent is determined by the algorithms described in the next section.

Figure 4 shows the grammar for our message protocol. Strings on the right-hand side of `<ck-code>` are abbreviations for our six types of contextual knowledge: predicted context features, context-dependent semantics, standing orders,

```
          MESSAGE ::= ALL-MSG | ALL-EX-MSG | SOME-MSG | NEW-MSG |
                      MOD-MSG | START | END
            START ::= "CTX"
              END ::= "CTX-END"
          ALL-MSG ::= ‘‘ALL’’ <ck-type>? <proto> <new-cf>*
       ALL-EX-MSG ::= ‘‘ALL’’ <ck-type>? <proto> ‘‘EXCEPT’’ <ck-code>+
                      <new-cf>*
         SOME-MSG ::= SOME <ck-code>+ <new-cf>*
          NEW-MSG ::= ‘‘NEW’’ <full-ck-descrip> <new-cf>
          MOD-MSG ::= ‘‘MODIFIES’’ <ck-code> <dl-difference>+ <new-cf>?
        <ck-type> ::= ‘‘PCF’’ | ‘‘CDS’’ | ‘‘SO’’ | ‘‘EHK’’ | ‘‘AFK’’ | ‘‘PK’’
          <proto> ::= a name of a prototypical context
         <new-cf> ::= float
        <ck-code> ::= name of contextual knowledge item in prototypical
                      context
<full-ck-descrip> ::= a KRSS (this is a DL syntax) description that
                      is-a ContextualKnowledge
  <dl-difference> ::= a KRSS description
```

Fig. 4. The message protocol

```
;; AbortMission goal has high importance, has medium cost,
;; and is achieved by action Abort:
rm-afk-1(0.93): (AND (SOME definesGoal AbortMission)
                     (SOME hasImportance High)
                     (SOME hasCost Medium)
                     (SOME isAchievedBy Abort))
;; AbortMission has medium importance, has medium cost, and is
;; achieved by aborting to sea floor}:
sm-afk-1(0.89): (AND (SOME definesGoal AbortMission)
                     (SOME hasImportance Medium)
                     (SOME hasCost Medium)
                     (SOME isAchievedBy
                     (AND Abort (SOME hasObject SeaFloor))))
```

Fig. 5. Two partially-matching pieces of contextual knowledge

event-handling knowledge, attention-focusing knowledge, and procedural knowledge.

When an idiosyncratic context K contains some of the same contextual knowledge as a prototype ancestor (shared) c-schema P, an `ALL-MSG`, `ALL-EX-MSG`, or `SOME-MSG` is used. Suppose P has three pieces of event-handling knowledge with identifiers *p1*, *p2*, and *p3*. Then the message

`ALL EHK P 0.9 0.8 0.7`

that K has all the event-handling knowledge from P. The float values are the certainty factors for the three pieces of knowledge in K. The ordering of the certainty factors in the message correspond to the order in which the event-handling knowledge is found in P. In contrast, the message

`ALL EHK P EXCEPT p2 0.9 0.7`

means that K includes all the event-handling knowledge from P except *p2*. We can refer to the pieces of knowledge by name, as we can rely on their order, since the representations of the prototypes are known to both the sender and receiver as shared knowledge. Alternatively, the message

`SOME p1 p3 0.9 0.7`

has the same meaning as the `ALL-EX` message, and is the better choice in this case because it is shorter.

Up to this point, we have been concerned with the case in which K may not match P exactly, but some or most of its corresponding contextual knowledge does match. However, corresponding pieces of contextual knowledge may only partially match in many cases. This case is handled by the `MOD-MSG` message type.

The two descriptions of `AttentionFocusingKnowledge` shown in Fig. 5 have some role restrictions in common and some that differ. If we suppose that `rm-afk-1` belongs to a prototypical c-schema, we can use a `MOD-MSG` to express `sm-afk-1`. The `<dl-difference>+` part of a `MOD-MSG` message will describe the differences between `rm-afk-1` and `sm-afk-1` and allow for the agent receiving the message to reconstruct `sm-afk-1`.

A `NEW-MSG` message, which sends a verbatim description of knowledge, is used as a last resort when none of the other message types can capture a piece of knowledge. The circumstances in which a `NEW-MSG` is used are outlined in the next section.

4.2 Deciding What to Send

We have devised an algorithm by which agents can generate a set of messages to completely describe an idiosyncratic context. The algorithm, which assumes an agent can correctly retrieve the prototypical ancestor(s) of an idiosyncratic context, is divided into two phases. The first phase generates message types `ALL-MSG`, `ALL-EX-MSG`, and `SOME-MSG`, which are the message types that deal with direct matches between idiosyncratic and prototypical knowledge. The second phase generates message types `MOD-MSG` and `NEW-MSG`, which cover all the rest. Both parts of the algorithm require a DL reasoning engine.

Direct Match Algorithm. Let C_I be an idiosyncratic c-schema and C_P be the set of its prototype ancestors, and let ck_i refer to a piece of knowledge in C_I. To find direct matches for contextual knowledge in C_I, the DL reasoner is used to look for concept synonyms in the combined knowledge of C_P for each ck_i.

Our goal is to partition the set of directly matched ck_i into subsets, where each subset is covered by a single message from the types `ALL-MSG`, `ALL-EX-MSG`, or `SOME-MSG`, such that the total number of bytes of the messages is minimized. Unfortunately, this is an exponential-time problem. Consequently, we use a greedy algorithm to approximate this partitioning. The algorithm repeatedly loops through the six contextual knowledge types until all the direct matches have been handled by a message. It finds the prototype context with the most unhandled direct matches of the current knowledge type, and uses this prototype context as a reference point to create a message covering this knowledge. Thus at each iteration we pick a subset and produce a message for it. The messages that cover each subset in the partition can then be sent, with their certainty factors based on the idiosyncratic context C_I.

Computing Differences. In addition to messages for the directly-matching contextual knowledge, the agent must send messages for the rest of the knowledge in the idiosyncratic c-schema C_I. These messages will have to be of types `MOD-MSG` or `NEW-MSG`, each covering a single piece of knowledge ck_i. Let $M_I = \{ck_i | ck_i \text{ has no direct matches}\}$. The algorithm iterates through each $ck_i \in M_I$, determining which message type to use.

The algorithm attempts first to use a `MOD-MSG` message, since that should be shorter than using a `NEW-MSG` message. To do this, it checks to see if there is a piece of knowledge ck_p from C_P that can be used as a point of reference for ck_i. The differences between ck_p and ck_i are then expressed in the `<dl-difference>+` portion of the `MOD-MSG` message.

Our algorithm for computing differences between ck_i and ck_p makes use of the DL *subtraction operation* [15]. If B and A are two DL concepts and A subsumes B, then the subtraction (difference) operation, $B - A$, gives a new concept such that each piece of information in B that is present in A is removed. For DLs with structural subsumption, a simple implementation of this operator is possible.

In our case, we can find the differences even if the one concept does not subsume the other, since we know that the two are both of the same `Contextual Knowledge` type: they contain the same restriction clauses and only differ in the

role ranges for the clauses. Consider again the two pieces of knowledge shown in Fig. 5, where `rm-afk-1` corresponds to ck_p and `sm-afk-1` is ck_i. The `MOD-MSG` message expressing `sm-afk-1` is

```
MODIFIES rm-afk-1 (SOME hasImportance Medium)
  (AND Abort
   (SOME hasObject seaFloor)) 0.89
```

Rules outline how the recipient can reconstruct `sm-afk-1` based on the message contents. Let d be a description in the `<dl-difference>+` portion of a `MOD-MSG` M, p be the description of the `<ck-code>` in M, and `type(p)` be the `ContextualKnowledge` subclass to which p belongs. Then d is interpreted as follows:

1. If d is a restriction on role r, a required role of `type(p)`, d is meant to replace the restriction on role r found in p.
2. If d is a restriction on any other role, it is meant to be ANDed to p.
3. If d is a conjunction containing a primitive base concept, let g be a superclass of d such that g is found in p. Then d is meant to replace each instance of g in p.
4. If d is of any other form, it is an error.

Sometimes it is not possible to create a `MOD-MSG` message for two pieces of differing contextual knowledge, for example in the case where the only `Contextual Knowledge` type in common is `ContextualKnowlege` itself. The rules for reconstructing a `MOD-MSG` allow successful creation of a `MOD-MSG` referencing any ck_p that is the same `ContextualKnowledge` bottom-level type as ck_i. However in some cases all possible ck_p for ck_i will have no subsuming role ranges for the required roles. In this case the `MOD-MSG` produced will contain all the conjoined clauses from ck_i, which is essentially the same as what is contained in a `NEW-MSG`. In this case, the `NEW-MSG` is shorter than using `MOD-MSG` and referencing a piece of prototypical knowledge.

5 Evaluation

The message generation algorithms have been implemented and tested using a randomly generated c-schema hierarchy. This was done to avoid accruing the extensive domain knowledge of the AUV domain needed to create a set of realistic contexts, which is not the focus of this work. Instead, well-structured contexts were generated based on the ontology.

The context-generation mechanism is implemented in Common Lisp and uses the reasoner RACER [11] for DL inference. The program first reads the description of the ontology, stored as an OWL ontology file. To create a complete c-schema, several pieces of each `ContextualKnowledge` type are generated and combined. To create a piece of contextual knowledge, for each role in its definition, a suitable concept is chosen as the value. If there are unfilled roles in that concept, then the process continues recursively.

A small number of c-schemas were created to serve as prototype contexts and placed in a context hierarchy. Idiosyncratic contexts were then generated by choosing prototypes as parents, then combining and randomly modifying their knowledge. Modifications included replacing concepts with their siblings or descendants and adding role restrictions.

For a preliminary evaluation, twenty idiosyncratic c-schemas were generated from four prototype c-schemas. The message generation procedure was run on each idiosyncratic c-schema, and the resulting messages were processed by a message receiver procedure that implemented the rules for interpreting messages. The c-schema produced by the receiver procedure matched the original c-schema each time. Further evaluation will be aimed at comparing the number of bytes of the generated messages to the minimum number of bytes required to encode the c-schema to determine the effectiveness of bandwidth-reducing heuristics.

6 Conclusion and Future Work

This paper describes an approach to context representation and communication to support multiagent context assessment by allowing agents to share their individually-known contexts with each other in an efficient manner. An ontology for contextual knowledge, represented using description logic, has been developed, and the representation of contexts themselves as c-schemas is also very much like an ontology. A message protocol and algorithms to support its use have been developed to allow an agent to decide which pieces of contextual knowledge it needs to send and how to send them.

The work reported is at an early stage, and so at this point, evaluation has been limited. The next step is to perform much more extensive evaluation to determine strengths/weaknesses of the approach and to quantify the efficiency of context communication in this approach.

Beyond representation and communication in MASCon, we are working on the problem of how agents can negotiate to come to an agreement on their shared context. To arrive at a consensus, agents must be able to evaluate others' knowledge based on their own. Our DL representation facilitates many possible techniques for an agent to compare pieces of knowledge. After receiving a contextual knowledge message, an agent can look for concept ancestors and descendants in its own evoked c-schemas, where ancestors represent more general and descendants represent more specialized knowledge. This can help an agent determine what aspects of the received knowledge it agrees with. An agent can also find the least common subsumer [5] of two knowledge descriptions, which finds the largest set of commonalities between two descriptions. In addition to techniques for evaluating knowledge, we are extending the message protocol to include message types for negotiation. Messages will be added for agreeing and disagreeing about received contextual knowledge and reasons for disagreement.

References

1. Baader, F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Sci. Am. **284**(5), 28–37 (2001)
3. Brezillon, P., Pasquier, L., Pomerol, J.C.: Reasoning with contextual graphs. Eur. J. Oper. Res. **136**(2), 290–298 (2002)
4. Buvač, S.: Quantificational logic of context. In: Working Notes of the IJCAI 1995 Workshop on Modelling Context in Knowledge Representation and Reasoning (1995)
5. Cohen, W.W., Borgida, A., Hirsh, H.: Computing least common subsumers in description logics. In: AAAI, pp. 754–760 (1992)
6. Curtin, T., Bellingham, J., Catipovic, J., Webb, D.: Autonomous oceanographic sampling networks. Oceanography **6**(3), 86–94 (1993)
7. Durfee, E.H., Lesser, V.R.: Using partial global plans to coordinate distributed problem solvers. In: IJCAI, pp. 875–883 (1987)
8. Giunchiglia, F.: Contextual reasoning. Epistemologia **16**, 345–364 (1993)
9. Gonzalez, A.J., Stensrud, B.S., Barrett, G.: Formalizing context-based reasoning: a modeling paradigm for representing tactical human behavior. Int. J. Intell. Syst. **23**(7), 822–847 (2008)
10. Guha, R.: Contexts: a formalization and some applications. Ph.D. thesis, Stanford University (1991)
11. Haarslev, V., Möller, R.: RACER system description. In: Leitsch, A., Nipkow, T., Goré, R.P. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–705. Springer, Heidelberg (2001)
12. McCarthy, J.: Notes on formalizing context. In: IJCAI, pp. 555–560 (1993)
13. McGuinness, D.L., Van Harmelen, F.: OWL web ontology language overview. Technical report, W3C, W3C Recommendation, February 2004. www.w3.org/TR/owl-features
14. Song, H., Hodgkiss, W.: Efficient use of bandwidth for underwater acoustic communication. J. Acoust. Soc. Am. **134**(2), 905–908 (2013)
15. Teege, G.: Making the difference: a subtraction operation for description logics. KR **94**, 540–550 (1994)
16. Turner, E.H., Chappell, S.G., Valcourt, S.A., Dempsey, M.J.: COLA: a language to support communication between multiple cooperating vehicles. In: Proceedings of the Symposium on AUV Technology (AUV 1994), pp. 309–316. IEEE (1994)
17. Turner, R.M.: Context-mediated behavior. In: Brézillon, P., Gonzalez, A. (eds.) Context in Computing: A Cross-Disciplinary Approach for Modeling the Real World Through Contextual Reasoning, pp. 523–540. Springer, New York (2014)
18. Turner, R.M., Rode, S., Gagne, D.: Toward distributed context-mediated behavior for multiagent systems. In: Brézillon, P., Blackburn, P., Dapoigny, R. (eds.) CONTEXT 2013. LNCS, vol. 8175, pp. 222–234. Springer, Heidelberg (2013)
19. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using OWL. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications, pp. 18–22 (2004)