PUC

ISSN 0103-9741

Monografias em Ciência da Computação

n$^{o}$ 02/12

# An Algorithm for Distributed Cooperative Reasoning over Global Context States

**Alexandre Skyrme**

**Markus Endler**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900**

**RIO DE JANEIRO - BRASIL**

Monografias em Ciência da Computação, No. 02/12
Editor: Prof. Carlos José Pereira de Lucena

ISSN: 0103-9741
January, 2012

# An Algorithm for Distributed Cooperative Reasoning over Global Context States

Alexandre Skyrme    Markus Endler

askyrme@inf.puc-rio.br , endler@inf.puc-rio.br

**Resumo.** Em muitas ocasiões do nosso cotidiano queremos interagir espontaneamente com pessoas desconhecidas próximas para compartilhar ideias, conversar, economizar tempo/dinheiro ou ajudar um ao outro. Para este fim, é necessário identificar situações de contexto compartilhado que dependem de fontes distribuídas de contextos locais de usuários. Até o momento, a maioria dos trabalhos que investigam mecanismos para suportar a descoberta espontânea e a interação entre usuários móveis, ainda não exploraram meios para detecção automática de Estados Globais de Contexto (EGC) comuns. Neste artigo propomos uma abordagem para raciocínio distribuído e um algoritmo que determina um estado global distribuído de contexto entre nós/pares que potencialmente interagem entre si. Também avaliamos a complexidade do algoritmo - através de simulações - e identificamos que a convergência do algoritmo depende muito dos padrões de mobilidade dos usuários e da quantidade mínima de nós que contribuem para o raciocínio, em vez da volatilidade dos contextos locais.

**Palavras-chave:** raciocínio, contexto, algoritmo, distribuído, cooperativo

**Abstract.** In many occasions of our daily lives, we want to spontaneously interact with nearby strangers for sharing ideas, chatting, saving time/money, or helping each other. For that purpose, it is necessary to identify shared context situations that depend on distributed sources of users' local context. So far, most of the work that investigates mechanisms to support spontaneous discovery and interaction among mobile users has not yet explored means of automatic detection of common Global Context States (GCS). In this paper, we propose a distributed reasoning approach and algorithm that determines a distributed Global Context State among potentially interacting nodes/peers. We also evaluate the complexity of the algorithm - through simulation - and identify that the convergence of the algorithm depends very much on the users' mobility pattern and the requested minimum number of contributing peers, rather than on the volatility of the local contexts.

**Keywords:** reasoning, context, algorithm, distributed, cooperative

**In charge of publications :**

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

# Contents

## 1 Introduction

Computer aided reasoning, or automated reasoning, concerns how to infer or confirm, completely automatically or semi-automatically, an information which is not explicitly available based on a set of known information. It is one of the central problems of Artificial Intelligence (AI) [1] and a number of systems, both centralized and distributed, have been developed to support it. While centralized systems may avoid communication issues, distributed reasoning systems can offer greater scalability, increased flexibility and better reliability. This paper focuses in the latter type.

Distributed reasoning systems have been studied for some time, especially as means to process the vast amounts of data that compose the semantic web [2–4]. However, as mobile devices with built-in sensors enjoy increasing popularity, many distributed pervasive applications have been proposed and implemented. As these applications are intrinsically context-aware [5], context reasoning became an important issue, and *distributed context reasoning* [6, 7], i.e. the ability to infer or detect a global state of context, became a necessary feature. By Global Context State we mean a specific combination of the local context states experienced (or sensed) by a group of nodes/agents, and that is relevant to the pervasive application. However, distributed context reasoning raises many challenges such as: ability to handle the dynamic set of nodes/agents and the associated topology of the system, modeling entities, their local and the Global Context States, handling the potentially heterogeneous nature of context information, tracking the concurrent and independent changes of constituent parts of the global context and management/coordination of the reasoning process. In this work, we will concentrate on the latter two challenges.

Reasoning can be of many sorts, such as *fuzzy logic-based*, *Case-Based Reasoning (CBR)*, *ontology-based*, *rule-based*, *distributed cooperative reasoning* [8], or a combination of them. Whereas fuzzy logic-based reasoning aims at modeling the imprecise modes of reasoning for decision making in an environment of uncertainty and imprecision [9], in CBR the idea is to rely on previous experience, i.e. a solution of new problems happens by retrieving relevant prior cases and adapting them to fit the new situation. Ontology-based reasoning, on the other hand, is mainly focused on formally describing how the entities that compose a domain can be classified and how they relate to each other; many ontology applications (such as OWL [10] and Description Logic (DL) variations [11]) can be used to support it. Rule based reasoning employs rules/statements in some language (usually Description Logic) expressing situations of interest. It is implemented through "reasoning engines" that evaluate the rules and trigger those that have their antecedents satisfied. Distributed cooperative reasoning simply adds to the previous techniques the lack of central control and reasoning structures, relying on a distributed algorithm that supports a coordinated reasoning process performed by several nodes/agents. Of all possible forms of reasoning, in this paper we will address distributed cooperative and rule-based reasoning, using Description Logic.

A distributed cooperative reasoning system is typically comprised of independent nodes/agents with some means to communicate among themselves [12]. Each node/agent runs a reasoning engine, and is capable of inferring or checking a new piece of information that is not explicitly available at its local knowledge base. Communication architectures frequently used in such reasoning systems include blackboard or message passing systems. Blackboard systems rely on a shared data structured (called a blackboard) where an node/agent can post information, as well as read and act on information posted by other nodes/agents. Message passing systems, on the other hand, rely on nodes/agents sending messages to other nodes/agents and receiving messages from other nodes/agents as part of a reasoning process.

Another common distinction in distributed cooperative and rule-based reasoning is the choice between *data partitioning* and *rule partitioning*. In data partitioning, data is distributed among nodes/agents and no single node/agent knows all information available globally; whereas all rules are applied to each data subset. In rule partitioning, rules are split, meaning that each nodes/agent only checks parts of the rules against its local data. However, in order to check the original rule (the conjunction of the parts) on the global state of all data available, nodes/agents have to exchange data among each other. In this work, we focus on distributed rule-based reasoning with rule partitioning, and how to accomplish it in message passing systems.

In distributed pervasive applications, an important issue for reasoning becomes the determination of the system's *Global Context State* (GCS) [13]. Concrete examples of such a Global Context State shared among a group of mobile users are: **quality of connectivity -** all user devices are connected through a high-speed wireless connection, enabling them to use a collaboration app with high communication demands; or **location -** all users are located within less then 200 meters from a common meeting place, so that they may gather in a few minutes; or else, **phone settings -** all users have their device set to normal ring-tone, implying that all members of the group are mutually available for some consultation or chatting.

As mentioned before, Global Context State is just the combination of all the nodes/agents *local contexts* at a same instant of time, and determining such a state is essentially an instance of the Global Predicate Evaluation (GPE) problem [14] for unstable predicates. As well known from Distributed Algorithms theory, effectively detecting unstable predicates in a asynchronous distributed system is complex and exponential in the number of nodes/agents and local states, even if performed by a central server. However, by assuming that the node/agent's clocks are approximately synchronized (that is made possible through current communication and GPS technology present on most mobile devices), that remote communication is reliable and has upper-bound delays, and that a Global Context State is only application-relevant if it remains stable during a minimum amount of algorithm rounds (see section 3.1), it is possible to devise algorithms that converge towards the distributed detection of such global states.

The determination of the Global Context State has several applications in distributed pervasive systems. For example, consider the following scenario: a conference attendee is interested in meeting with other nearby attendees who are idle (e.g. waiting in the conference hall) and who share similar interests or expertise. In this scenario, distributed reasoning could be used to compare the local context state of all attendees (i.e. current location, interest/s and expertise) in order to identify which attendees, if any, match the *meeting criteria* set by the seeking attendee. This would require the reasoners at the device of each attendee (further called, *Peer Reasoners*) to exchange their local context state information (e.g. current locations) and cooperatively select which attendees match the criteria. We should further consider the existence of an additional reasoner, the *Ambient Reasoner*, which would act as a communication/coordination hub for the reasoning process and also provide public information about the ambient-specific context, such as the conference program, the layout of the rooms, the room-specific planned activities, etc. This "ambient context state", e.g. the current activity in each room, could also be relevant for the matching: for example, only attendees located in the conference hall, the registration desk, or in the restaurant should be considered for the matching.

The above examples of matching criteria would be defined by a Description Logic rule, and would better be split in parts, to be evaluated independently by each of the

Peer Reasoners (at the attendee's devices) on the one side, and the Ambient Reasoner, on the other side. The former would handle rule parts/predicates more closely related to the attendee's context data and preferences (e.g. current location, if available or busy, affiliation, interests, etc.), and the latter would process the parts of the rule that refer to public information (e.g. place-specific activities) and which contain predicates matching issues, such as a precise definition of user co-location or proximity.

In this paper we describe an algorithm used for cooperative detection of distributed Global Context States, that is the basis for distributed cooperative and rule-based reasoning. The main contributions of this paper are to define the concept of rule-based Global Context States (GCS), as well as some required properties and premises, to propose an algorithm to determine a distributed GCS among potentially interacting peers and to simulate the execution of the proposed algorithm. The paper is structured as follows: in section 2 we discuss some related work on distributed (context) reasoning for pervasive applications. In section 3 we define the concept of rule-based Global Context State, list some premises about the system, expected properties of a solution and introduce the algorithm, including its pseudo-code. In section 4 we explain how we simulated the algorithm's execution and present extensive results of the simulation. Finally, in section 5 we present some concluding remarks concerning our approach and future work.

## 2 Related Work

The present work is largely based on the work by Viterbo and Endler [15, 16], which explores distributed rule-based reasoning (with rule partitioning) and is aimed at Ambient Intelligence applications. It proposes a simple two-tier model comprised of a user/client side and an ambient side and a protocol that must be executed by both sides to converge towards the evaluation of a partitioned rule. It also formalizes the notion of (decentralized) cooperative reasoning, discusses the necessary stability conditions required for convergence, and presents a middleware system called DRS that implements the distributed reasoning protocol. In this work, we extend the aforementioned two-tier approach to work in with an open set of nodes contributing to the reasoning process, i.e. detection of the subset of nodes whose local context satisfies the rule's antecedent. Also, while Viterbo and Endler were primarily concerned with laying out the foundations of distributed cooperative reasoning, this work focuses more on the distributed algorithm defining the interactions and synchronization among the nodes to detect a Global Context State (which actually occurred) defined through a global rule's antecedents.

In [17], Padovitz, Loke and Zaslavsky propose and formalize an approach that enables individual nodes to reason about common context situations by considering distributed information. However, instead of presenting a specific approach for collaborative reasoning, their work's main focus is on context model transformations that allow nodes to obtain a merged perspective of the common context. Since our work does not consider heterogeneous context models, one can see their work as complimentary to ours, since it proposes a solution for merging different context model visions.

In [18], Gu, Pung and Zhang present a peer-to-peer system, where peers are organized according to an ontology based semantic network, to support distributed reasoning for collaborative context-aware applications. Each peer in the system can act as a context producer (which provides low-level context data, usually obtained from physical sensors), a context interpreter (which is similar to a producer but is able to infer high-level contexts from low-level context data) or a context consumer (which obtains context data by querying producers or interpreters). The system supports both pull and push modes for context requests; the former mode allows a consumer to execute a single query for present context data, while the latter allows a consumer to subscribe to a context event and be informed of related context changes for a period of time. The algorithm we present in this paper works in a similar fashion to the aforementioned push mode; it allows for continuous monitoring of context changes in a distributed environment. In our algorithm, however, we're not concerned with intermediate context states or with informing the user of intermediate context updates, but rather with verifying if a certain Global Context State holds while a user is interested in it. Also, in our work we expect all participating nodes to be homogeneous, or to be able to provide the same type of information, while the system presented by Gu, Pung and Zhang could support heterogeneous peers, which have different sensing capabilities, and then have a context interpreter combine the data in order to infer a high-level context. Lastly, their system is fully distributed and thus does not rely on a central entity such as the Ambient Reasoner which is necessary in our algorithm; it is worth noticing, though, that the prototype used to evaluate their system used standard desktop computers to run context producers and interpreters, which suggests these roles may not be suitable for execution in mobile devices.

Distributed Reasoning Architecture for a Galaxy of Ontologies (DRAGO) is a distributed reasoning system implemented as a peer-to-peer architecture, in which every peer registers a set of ontologies and mappings [19]. In DRAGO, the reasoning operations are implemented using local reasoning over each registered ontology and by coordinating with other peers when local ontologies are semantically connected with the ontologies registered in other peers. The reasoning with multiple ontologies is performed by a combination of local reasoning operations, internally executed in each peer for each distinct ontology.

P2P-DR [8] is a system for distributed reasoning focused on Ambient Intelligence (AmI), which uses a peer-to-peer model and accounts for potential conflicts which might happen during the reasoning process. Each node holds independent (local) information, expressed as rules, and is also able to exchange information with neighbor nodes, by means of *bridging rules*. Potential conflicts which may arise from global consolidation of local information are dealt with by considering bridging rules as defeasible (can be overridden) and defining trust levels between nodes in order to settle disputes between conflicting rules. The authors point out context data can be inconsistent, for instance due to imprecise or faulty sensors, or become ambiguous, when conflicting data is reported by different sources. They highlight that ambient environments host nodes that are heterogeneous and dynamic in nature and thus might not be able to communicate directly or even be aware of all nearby nodes.

A peer-to-peer inference system (P2PIS [20]) is a network of peer theories. Each peer has a finite set of propositional formulas and can be semantically related by sharing variables with other peers. A shared variable between two peers is in the intersection of the vocabularies of the two peers. Not all the variables in common in the vocabularies of two peers have to be shared by them. Besides, two peers may not be aware of all the variables that they have in common but only of some of them. In a P2PIS, no peer has the knowledge of the global P2PIS theory. P2PIS distributed algorithm splits clauses if they share variables of several peers. Each piece of a split clause is then transmitted to the corresponding theory to find its consequences. The consequences that are found for each piece of split clause must then be re-composed to get the consequences of the clause that had been split.

DRAGO, P2P-DR and P2PIS propose distributed reasoning solutions considering data distributed over different elements in an AmI system. The main concern of DRAGO is to reason in distributed environments overcoming the barrier of the heterogeneous knowledge representation that independent entities in a AmI system are very likely to employ. DRAGO relies on predefined mappings to align different ontologies. In a similar way, P2P-DR and P2PIS are peer-to-peer frameworks in which peers can communicate with a subset of the other available peers to import the knowledge necessary to answer queries based on mappings that define how their local knowledge relates to their peers' knowledge. In such way, P2P-DR and P2PIS are capable of performing inference to answer queries that check if a rule is true or false, in which the knowledge, i.e., set of literals that represent context information, is fully distributed in a peer-to-peer system. Nevertheless, P2P-DR and P2PIS are not capable of answering queries with variables. Moreover, DRAGO, P2P-DR and P2PIS are also limited by the fact that in practical implementations of AmI it is not feasible to build in advance mappings of all possible pairs of different ontologies that may be needed. On the other hand, our approach is not a fully decentralized peer-to-peer system, as it relies on the Ambient Reasoner for mediation and coordination. Finally, unlike DRAGO and P2PIS, in our work we do not handle heterogeneous ontologies.

## 3 Proposed Algorithm

In this section we describe an algorithm for determination of a distributed global state as the foundation for distributed coordinated reasoning on Global Context States. Before this, though, we introduce some key concepts.

### 3.1 Global Context States

A *Global Context State (GCS)* is defined by a global condition, or constraint, which may refer to the local context states of all (or some) the nodes in the system, including the ambient node[1]. Such global condition is commonly expressed as the conjunction of antecedent predicates, *R_i*, in a Description Logic (DL) rule of the form $R_1 \wedge R_2 \wedge ... \wedge R_k \Rightarrow C$. In such rule, the consequent $C$ is usually an action, and the predicates $R_i$ describe relations between concrete context data facts at one or more nodes, and may also contain free variables (denoted by, $?v, ?w$, etc.) which of which ranges over context facts/items (of given type or attribute), in a single node. When a rule like the above is evaluated, these free variables are bound to sets of concrete context facts of any of the nodes. For example, consider that a node N local's context state has only facts *(sentMsg, addr1,Hi)*, and *(sentMsg, addr2,Hello)*. Then, evaluating predicate *sentMsg(?a, ?msg)*, on N's context state, would cause the binding of the pair of variables ($?a, ?msg$) to the set T_N = {*<addr1, Hi>*, *<addr2, Hello>*}. Consequently, the result of evaluating a DL rule with free variables $?v_i$, (i= 1 . . . N) on a Global Context State, is thus a set of tuples $< c_1, ..., c_N >$, where $c_i$ is a data item found in any of the nodes' local context state, that has been bound to the free variable $?v_i$, such that the tuple elements $c_i$ cause the rule's antecedent $R_1 \wedge R_2 \wedge ... \wedge R_k$ to be satisfied. Note that if the Global Context State does not satisfy the rule's antecedent, the set of tuples is empty.

However, since the local context state of each node changes spontaneously, and independently of the other nodes' context state, the global condition may be satisfied only for a limited amount of time. Thus, the main objective of *distributed reasoning over Global Context States* is to detect if the Global Context State matches the rule's antecedent part $R_1 \wedge R_2 \wedge ... \wedge R_k$, and if this is the case, to execute the action of the rule's consequent $C$.

[1]A local context state is defined by a finite set of n-ary tuples (e.g. attribute-value pairs), called context data facts. E.g. (Battery, 50%), (Connection,3G), (Lat-Long-Location, -22.9784231, -43.2338216), etc.

For this, the reasoners on the different nodes must run a distributed algorithm enabling them to exchange partial reasoning results (the bindings of free variables with their local context facts) until some tuple (i.e. the combination of all partial results) satisfies the rule's antecedent.

In order to illustrate the above concepts, let's look at a very simple example: consider three nodes, n1, n2 and n3, which repeatedly throw individual dice (*d*1, *d*2 and *d*3) asynchronously, i.e. at random instants. Making an association with the aforementioned, the current die number plays the role of the node i context state (which here would be a single, unary tuple (*di*)). So, a rule to identify the global state in which *d*3's number is odd, and is the sum of the other two die numbers, would be:

$$\text{even(?d1)} \wedge \text{odd(?d2)} \wedge \text{odd(?d3)} \wedge \text{equals(?d3,?d1+?d2)} \Rightarrow \text{OddSumObtained}$$

In this case, the only tuple set that satisfies this rule is T= {<2, 1, 3>, <4, 1, 5>, <2, 3, 5>}, where the elements of each tuple are T_n1, T_n2, and T_n3, respectively.

In this particular case, we can see that the above rule can be split in a way that each of the reasoners becomes responsible for one of the predicates even/odd() - to be checked whenever the node throws a die - and one reasoner will evaluate predicate equals(). Moreover, the reasoners have to exchange their data whenever some of them detects that a new die throw - a change of its local context state - satisfies its local predicate. In this specific example, only the reasoner with predicate equals() also has to take into account the latest data received from the other two reasoners. However, there are other examples of rules where any partitioning and assignment of predicates to the nodes, causes nodes to share more than just one free variable, so that data does not flow only towards one node, as in the example, but that all nodes have to exchange partial results of their predicate evaluations with some, or all, the other nodes.

As a second example, consider two mobile devices, n1 and n2, which are connected through a wireless transmission technology (e.g. Bluetooth), and that n1 should send a large file, say F, to device n2, but only if its own and n2's remaining battery lifetime (captured by variables ?BT1, and ?BT2, respectively) are sufficiently large, and otherwise, the file transfer should be postponed. Let's also assume that F's transmission time FTT, can be calculated - using predicate transTime() - from the file's size and the current quality of the wireless connection as perceived by each node, captured by variables ?QoC1 and ?QoC2, respectively. Then, the following rule could express whether the system of two nodes share an appropriate Global Context State - i.e. sufficient battery lifetime on both devices - for doing the file transfer.

$$\begin{gathered}\text{transTime(?F, ?QoC1, ?FTT1)} \wedge \text{greater(?BT1, ?FTT1)} \wedge \text{greater(?BT1, ?FTT2)} \\ \wedge\, \text{transTime(?F, ?QoC2, ?FTT2)} \wedge \text{greater(?BT2, ?FTT2)} \wedge \text{greater(?BT2, ?FTT1)} \\ \Rightarrow \text{FileTransferPossible(?F)}\end{gathered}$$

In this rule, clearly the first three predicates of the antecedent can be evaluated/checked by n1's reasoner, while latter three predicates should be better checked at n2.[2]. Therefore, also in this example, n1 and n2 must continuously update their estimated file transmission times FTTx, so that both reasoners detect the adequate Global Context State to start the file transmission. This continuous update of shared free variables among the reasoning nodes will be further explained in section 3.3.

First, however, we must precisely define in which case a Global Context State (GCS) is *considered to occur*. Since the local contexts that constitute a GCS can change in arbitrary and system unknowably ways, we must consider the occurrence of a GCS in relation

[2]We also assume that node n1 informs n2 about F's size.

to real time. This is quite different than the concept of a distributed system's global state [21,22], which is determined only by program-produced local events at the nodes and by events associated to their interactions, and hence there is no concern about time. On the other hand, in the case of GCS, we have to consider that the nodes have a notion of time. In particular, all nodes must record the run of real time by periodic events, a clock tick, $\Delta t$. This assumption does not require the nodes to have the same clock tick counter, nor to have their clock tick at the same real time. It just enforces that their $\Delta t$ do not drift from each other. With this, it is possible to define:

**Definition 1** *A* GCS *has occurred* **iff** *its constituent local states have overlapped during at least* $2 * \Delta t$*, from the perspective of each node contributing with a constituent local state.*

This definition essentially says that only global states which remain stable for a minimum period of time are *de facto* considered, while a quick overlap of constituent local states should be ignored. The $2 * \Delta t$ limit is required due to the fact that nodes may not have their clock ticks synchronized, being incremented exactly at the same moment. In the following section we will show that this $2 * \Delta t$ is a function of the system model parameters.

Based on the definition of a Global Context State (*GCS*), we can now discuss the general required properties of any distributed solution for reasoning over Global Context State, and our system model.

## 3.2 Required Properties and Premises

Any solution/algorithm for distributed rule-based reasoning over a GCS must have the following properties:

**Convergence:** if the Global Context State (*GCS*) satisfying the antecedents of a rule R remains stable for a sufficiently long period of time, then eventually the algorithm will evaluate that the rule R has been satisfied.

**Safety:** if the algorithm detects that the antecedents of rule R have been satisfied, then the corresponding *GCS* has actually occurred at some moment during the processing of the algorithm.

The convergence property leaves open the possibility that the solution never converges, if the *GCS* defined by the rule stays valid only for short times (yet longer than $2 * \Delta t$). In such case, due to the required exchange of messages between the reasoners, it may be impossible for them to collectively grasp this short-lived Global Context State. However, it is worth noting that the convergence property does not state that the *GCS* will be valid at the moment when the algorithm detects it, but that it detects that the GCS has in fact occurred in the past. This detection delay may be inevitable due to the processing and message transmission latencies. The safety property, on the other hand, requires the solution to be correct in that it never detects a *GCS* that has never occurred in the sense of Definition 1. Since this definition is based on the notion of time, it requires a synchronous model of a distributed system, i.e. where the maximum processing time and message transmission latency are constant and well-known. Moreover, we make other assumptions about the distributed system, which are summarized in the following list:

- Each node is capable of doing all local processing related to one incoming request (i.e. handle any incoming or outgoing messages plus evaluate any DL predicates in regard to its local context state in less than $\lambda$ time units;
- Nodes do not fail, and stationary nodes can be found and are always reachable by any other node. However, the total number of mobile nodes is variable;
- Message delivery is reliable and follows FIFO ordering;
- All nodes have a unique ID and communication address/endpoint;
- The clocks of all nodes do not drift from each other;
- The timer period at each node ($\Delta t$) is much larger than the communication and processing latencies (i.e., $\delta + \lambda \ll \Delta t$);
- Each reasoner on a node checks its local context state (local_Cxt) periodically and does this in an atomic way and a negligible amount of time.
- The periodic check of local context state is stateful, meaning that it not just grasps the momentaneous state of local context resources/sensors, but also registers any change of context state that might have occurred since the previous check, even if this change was very quick.

Without this last assumption about statefulness of context probing, it would be impossible to ensure the safety property, i.e. that a global state that did not actually occur - e.g. the overlap was less than $2 * \Delta t$ - would not be detected by the nodes. In order words, we need the testimonies of very quick local context changes that invalidate the occurrence of associated Global Context States.

## 3.3 Overview of the Algorithm

As previously mentioned, we assume that the system has a single stationary node which is associated with a place/location (e.g. the conference site), and which will be the communication mediator and coordinator of the reasoning process. This node will execute the Ambient Reasoner (*Amb*), and for the cooperative reasoning, all mobile nodes will interact only with this *Amb*. From this point on, from our algorithm's perspective, we will refer to mobile nodes simply as *peers* or by the name of the corresponding role they play in the algorithm.

The algorithm essentially works as follows: whenever an application on a a Requesting Peer (ReqP) needs to evaluate a Global Context State, it submits the corresponding DL rule to the middleware at this peer[3] . The rule is then partially evaluated at ReqP, taking into account its current local context state, and then forwarded to the Ambient reasoner (*Amb*) with the partial results (i.e. a set of tuples denoted by T_I)[4], produced by ReqP. When this message is received by *Amb* it will also partially evaluate some parts of the rule based on its local context state and the partial results received from ReqP. This results in yet another new set of possible partial results (denoted by, T_A), but where some of the rule's free variables (related to the other peers' context facts) are still unbound, i.e. undefined. Then, *Amb* broadcasts the rule and the partial results (T_I and T_A) to all other

[3]We assume that split of the rule is pre-defined by the application and sent as part of the submission.

[4]In the remainder, we use following T_suffix convention: I = Requesting peer, A= *Amb* and O = other contributing peer.

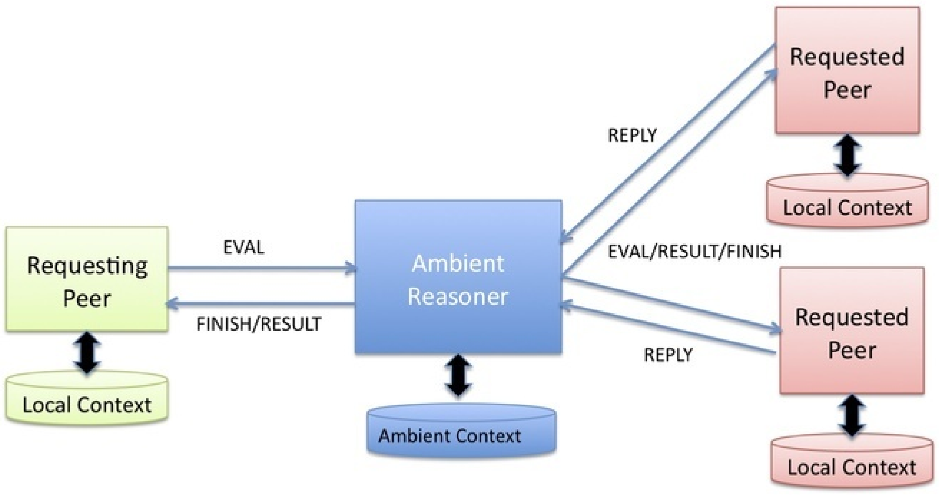


**Figure 1: Architectural diagram of the components used in our algorithm.**

participating peers, requesting them to reply whenever their local context states, together with the partial results T_I and T_A, satisfy the rule antecedents.

Figure 1 shows a diagram which represents the main architectural components used in the algorithm.

## 3.4 Pseudocode

In this section we present the *pseudo code* of the algorithm, which consists of ECA (Event Condition Action) clauses, where the *Event* part is either the start event (**init**), a new request - from the application - to check a rule R (only at ReqP), the arrival of a message (**receive(sender, msg-type, arguments)**), or a **timeout** from a timer set previously. We assume that each peer has a single event queue and that events are fetched from this queue one at a time, and that the *Action*-parts of ECA clauses are executed atomically. Each peer has its own variables and timer, and data exchange only occurs through the communication primitives send(), receive() and broadcast().

### Notation, Variables and Primitive Functions

Here we shortly present the notation, the main variables, function names and message types that appear in the pseudocode.

| **Variable/Symbol** | **Usage** |
|---|---|
| RID | system-wide unique ID for a reasoning request |
| R_I, R_A, R_O | rule part assigned to a RequestingPeer (I), AmbReasoner (A), and ParticipatingPeer (O), respectively |
| R [T_X ] | Antecedent of DL rule R (or part of it), where all variables referring to items of the local context at peer X have been bound to corresponding elements in the tuple set T_X |
| T_I, T_A, T_O | tuple sets produced by a RequestingPeer (I), AmbReasoner (A), and ParticipatingPeer (O), respectively, with some bindings of free variables to their local context state data facts |
| waiting | boolean flag set when a peer is waiting for message(s) from another peer |
| response_bag | set of tuples of the form (PeerID, RID, count, T_PeerID, ref_PeerID), where T_P is the partial evaluation from peer P, i.e. it is the set of its local context states that satisfies the peer's local part of the rule |
| ref_P | name/address of peer P, which is used for making the peers that contributed to a global state mutually aware of each other at the end of the reasoning process |
| count | number of continuous timer periods that the local context state (from a Peer) satisfied the local part of the rule |
| min_size | minimum number of peers that are required to contribute to the global state expressed by the rule |

| **Primitives** | **Functionality** |
|---|---|
| eval(RID, R, Cxt): T | returns the set of tuples T with current value(s) for context variables that satisfy rule R for request RID |
| set_timer() | sets a new timeout |
| send(), receive() | primitive communication functions (first arg. is destination, 2nd arg. is message type) |
| extract(): RefList | extract the contributing peer's references from the response bag relative to a RID |
| deliver_result() | middleware layer returns the reasoning result (addresses of the contributing peers) to the application which requested the reasoning |

| **Message Type** | **Meaning and Source** |
|---|---|
| EVAL | request to evaluate if the local context state (local_Cxt) satisfies the rule with some of its variables instantiated by other peers (may be sent by ReqP or *Amb* ) |
| REPLY | peer reply with the context state that partially satisfies the local part of the rule (sent by a participating peer to *Amb* ) |
| RESULT | delivery of a response_bag with a global state that was detected on at least min_size peers (sent by *Amb* ) |
| FINISHED | termination of the distributed reasoning process (sent by *Amb* ) |

## Requesting Peer (ReqP)

```
**init** ⇒ {waiting = FALSE}
**when** new request for rule R = R_I ∧ R_A ∧ R_O ⇒ {
    RID = new_request();
    T_I = eval(RID, R_I, local_Cxt);
    count = 0;
    send (Amb, EVAL, count, {R_A [T_I ] ∧ R_O [T_I ] }, my_ref);
    set_timer();
    waiting = TRUE;
}
**when** timeout ∧ waiting ⇒ {
    T_I = eval(RID, R_I, local_Cxt);
    **if** (T_I changed) **then** count = 0 **else** count = count +1;
    send (Amb, EVAL, RID, count, T_I );
    set_timer();
}
**when** receive (Amb, RESULT, RID, response_bag) $\wedge$ waiting $\Rightarrow$ {
    (ref_$O_1$, ref_$O_2$, ...) = extract(responde_bag, RID);
    deliver_result( RID, ref_$O_1$, ref_$O_2$, ...);
}
**when** receive (Amb, FINISHED, RID) $\wedge$ waiting $\Rightarrow$ waiting = FALSE;
```

## Ambient Reasoner (Amb)

```
**init** $\Rightarrow$ {waiting = FALSE; response_bag = $\emptyset$; count = 0; finished = FALSE;}
**when** receive (ReqP, EVAL, RID, c, (R_A [T_I ] $\wedge$ R_O [T_I ]), ref_I) $\Rightarrow$ {
    T_A = eval( RID, R_A[T_I], Amb_Cxt);
    set_timer();
    waiting = TRUE;
    broadcast (EVAL, RID, count, R_O[T_I,T_A]);
}
**when** (receive (ReqP, EVAL, RID, c, T_I) $\vee$ timeout) $\wedge$ waiting $\Rightarrow$ {
    **if** (T_I changed) **then** response_bag = $\emptyset$;
    T_A = eval(RID, R_A[T_I, response_bag), Amb_Cxt);
    **if** (T_A changed) **then** count = 0 **else** count = count +1;
    broadcast (EVAL, RID, count, (T_A, T_I));
    set_timer();
}
**when** receive(Peer_*i*, REPLY, RID, count, T_O_*i*) $\wedge$ waiting $\Rightarrow$ {
    **update** response_bag with (Peer_*i*, RID, count, T_O_*i*, ref_O_*i*);
    **remove from** response_bag any record (P, R, c, T_O, ref_O) where c $\in$ 0,1;
    **if** (card(response_bag) $\geq$ min_size) **then** finished = TRUE;
}
**when** finished $\Rightarrow$ {
    waiting = FALSE;
    $\forall$ P $\in$ ({ ReqP } *c* response_bag ) send (P, RID, RESULT, response_bag);
    broadcast (RID, FINISHED);
    finished = FALSE;
}
```

Note: min_size may be a constant configured at system start-up, or may be provided by the requesting peer (ReqP).

## Participating Peer (Peer_*i*)

```
**init** $\Rightarrow$ waiting = FALSE;
**when** receive (Amb, EVAL, RID, (R_O[T_A,T_I]) $\Rightarrow$ {
    T_O = eval( RID, R_O[T_A,T_I], local_Cxt);
    set_timer();
    waiting = TRUE;
    counter = 0;
    **if** (R_O[T_I,T_A,T_O] is satisfied)
        **then** send (Amb, REPLY, RID, count, T_O, my_ref);
}
**when** (receive (Amb, EVAL, RID, new(T_I,T_A) ) $\vee$ timeout ) $\wedge$ waiting $\Rightarrow$ {
    T_O = eval( RID, R_O[T_A,T_I], local_Cxt);
    if (T_O changed) then count = 0 else count = count + 1;
    if (R_O[T_I,T_A,T_O] is satisfied) $\vee$ count == 0)
        then send (Amb, REPLY, RID, count, T_O, my_ref);
    set_timer();
}
when receive (Amb, RESULT, RID, response_bag) $\Rightarrow$ waiting = FALSE;
when receive (Amb, FINISHED, RID, $\emptyset$) $\Rightarrow$ waiting = FALSE;
```

## 3.5 Discussion

Although the formal proof of the algorithm's correctness, i.e. its convergence and safety properties, is beyond the scope of this paper, we feel the need to informally discuss it here.

As mentioned in section 3.2, convergence cannot be guaranteed. However, it is intuitive that if the Global Context State (*GCS*) is satisfied by the combined local contexts of at least min_size peers, and stays satisfied for a sufficiently long period of time ($\gg 2 * \Delta t$), then this will not cause the Requesting Peer to send any modified EVAL messages to the Ambient Reasoner, and all Participating Peers will also start sending REPLY messages with count $\geq 1$, so that eventually the Ambient Reasoner will have received min_size REPLY messages from the Peers, and will move to state finished.

The safety property, i.e. that the algorithm will never notify a *GCS* which never occurred, is heavily based on the assumption that local context state checks by the peers are stateful (cf. last item in section 3.2). Thus, if a *GCS* has not occurred, it means that at least at one of the peers (a Requesting or Participating Peer) the operation changed will return true, causing the peer to either send a new EVAL request or send a REPLY message with count $\geq 1$. This fact, combined with the assumptions that (i) the peers' clocks do not drift, and (ii) the timer period of all peers ($\Delta t$) is much larger than their communication and processing latencies, will guarantee that this global context instability will always reach the Ambient Reasoner before it might conclude that min_size peers have witnessed a stable *GCS*. Thus, the Ambient Reasoner will not finish the reasoning process, but will keep waiting for the Global Context State to stay satisfied during a longer time interval.

# 4 Simulation

## 4.1 Sinalgo

In order to simulate the proposed algorithm's execution we used Sinalgo [23], a free open-source Java framework for validating and testing network algorithms in mobile networks. Sinalgo allows for quick prototyping of network algorithms in Java, easy extensibility and customization, two and three dimensional network graphs, plus synchronous and asynchronous simulations. It includes a network graph visualization utility which can be used to visually inspect an algorithm's execution steps.

Sinalgo's extension points are called *models*. Each model includes a small set of predefined options already implemented in the standard Sinalgo distribution. The following models are included with Sinalgo:

**Mobility** Defines if and how each node's position varies over time;

**Connectivity** Defines when two nodes are within communication range;

**Distribution** Defines how to initially place the nodes in the network graph (i.e., defines the nodes' initial positions);

**Interference** Defines if overlapping communications can interfere with each other;

**Reliability** Defines how reliable message transmission is (i.e., defines if and under what circunstances messages can be dropped while in transit);

**Transmission** Defines how long messages take to be transmitted.

Creating a project in Sinalgo typically involves carrying out the following tasks:

- Implementing the nodes behavior, which will usually include defining message structures that will be used for communication between nodes;
- Implementing, if necessary, customized globally visible methods (such as methods to collect statistics data, write log files, take some action when a node is added or removed from the simulation, customize the drawing of the network graph or check if the simulation has finished);
- Extending any of the existing standard models or even implementing additional models as needed;
- Configuring the project and some of the simulation parameters using a standard XML configuration file.

As previously mentioned, Sinalgo supports both synchronous and asynchronous simulations. While asynchronous simulations are purely based on events which lack temporal concurrence, synchronous simulations are based on rounds, which are simply fixed time slots where events can occur in parallel. In an asynchronous simulation, nodes are only able to act when one of the following events ocurr: a message arrives or a timer is fired. Usually, in asynchronous mode, the reaction to an event will involve creating other events and thus the execution flow of the simulation will be comprised of a sequence of chained events.

In a synchronous simulation, on the other hand, the framework keeps track of a global clock which has its ticks incremented by one at the beginning of each round. After incrementing the global clock, Sinalgo executes global methods defined to be executed prior to each round and global timers that have fired in the current round. It then moves nodes according to their mobility model and updates the connections (communication links) between nodes based on the connectivity model. After that, the framework executes each nodes' *step*, which is the main action sequence performed by the node and essentially includes: executing node specific methods defined to be executed prior to each round, executing node specific timers that have fired in the current round, handling received messages and executing node specific methods defined to be executed after each round. Finally, after executing each nodes step, Sinalgo executes global methods defined to be executed after each round and then checks if the simulation has terminated. So, to summarize, the basic calling sequence of a synchronous simulation in Sinalgo is as follows:

1. Increment global clock
2. Execute global pre-round methods
3. Execute global timers that have fired

4. Update nodes' position
5. Update nodes' connectivity
6. Execute nodes' step
    6.1. Execute node specific pre-round methods
    6.2. Execute node specific timers that have fired
    6.3. Handle messages received by node
    6.4. Execute node specific post-round methods
7. Execute global post-round methods
8. Check if simulation has terminated

Overall, Sinalgo turned out to be a very adequate option to implement our algorithm and explore some simulation scenarios. Its ease of use combined with its flexibility to extend and customize its behavior make it an invaluable tool.

## 4.2 Scenario and Variables

We propose a general simulation scenario similar to the one mentioned in section 1: during a small conference, an attendee is interested in engaging in a group discussion with nearby attendees who share a common interest. For that purpose, the attendee must rely on a reasoning process to assess the interests of other attendees, to determine if there are enough attendees nearby that share a common interest in order to have a group discussion and, in case so, to find out who are these attendees. The attendee could also benefit from knowing if there are any available rooms in order to host the group discussion, although room unavailability will not stop the discussion from happening if there are enough attendees with a common interest nearby.

We assume all attendees are potentially interested in having group discussions and, thus, are willing to share their own interests and their current location in order to support the reasoning process. We also assume attendees can move within the conference grounds, in order to attend sessions in different rooms, and that their interests change as a result of attending sessions about different topics. This effectively means that both attendees positions and interests change throughout time. Room availability to host group discussions also changes throughout time, as conference sessions have different time schedules. In addition, since it is a small conference, we take for granted that the organizers are able to provide reliable connectivity access with a single central communication hub.

In this scenario, the attendee who is interested in finding nearby attendees with a common interest plays the role of the Requesting Peer (ReqP), as it wants to evaluate a Global Context State; the other attendees play the role of Participating Peers (ParP), as they contribute to the reasoning process with partial results; and finally, the central communication hub provided by the conference organizers plays the role of the Ambient Reasoner (Amb), since it is used to mediate and coordinate the reasoning process. According to our algorithm, we assume that for all reasoning purposes, attendees only interact with the central communication hub, and not directly between themselves. Table 1 summarizes the mapping between the entities involved in the scenario, in our algorithm and in Sinalgo.

| **Scenario** | **Algorithm** | **Sinalgo** |
|---|---|---|
| Attendee interested in promoting group discussions | Requesting Peer | Node (PeerNode class) |
| Remaining attendees | Participating Peer | Node (PeerNode class) |
| Central comm. hub | Ambient Reasoner | Node (AmbientReasoner class) |

**Table 1: Mapping between the proposed scenario, our algorithm and Sinalgo entities.**

The Global Context State looked for in this scenario could be expressed by the following Description Logic rule, which would be evaluated cooperatively among the Ambient Reasoner, the Requesting Peer and the Participating Peers:

$$\mathsf{InCenter(?ReqP,\ ?Region)} \wedge \mathsf{CurrentInterest\ (?ReqP,\ ?I)} \wedge \mathsf{InsideRegion(?P,\ ?Region)}$$
$$\wedge\ \mathsf{CurrentInterest\ (?P,\ ?J)} \wedge \mathsf{Equals(?I,\ ?J)} \wedge \mathsf{Available(?Room)} \Rightarrow \mathsf{NotifyAll(?ReqP,\ ?P)}$$

In this rule, variables ?ReqP and ?P bind, respectively, to the Requesting Peer and any Participating Peers that are nearby, while ?Region describes the dynamic perimeter region around ?ReqP (as it moves), ?I and ?J bind to the current interest of the Requesting and Participating Peers, respectively, and ?Room binds to the name/number of the rooms that are available at the moment. As rule's predicates are self-explanatory, one could imagine that InCenter() and CurrentInterest() - its 1st occurrence in the rule - are evaluated at the Requesting Peer, while each of the Participating Peers evaluates InsideRegion() and CurrentInterest() - 2nd occurrence - , and Equals() and Available() would be evaluated at the Ambient Reasoner. It is worth mentioning that the above rule is just one of several possible rules that describe the Global Context State. Other, perhaps more detailed, rules could be used instead of this one.

Both types of attendees (Requesting Peer and Participating Peers) and the central communication hub (Ambient Reasoner) are implemented in Sinalgo as standard simulation nodes, each with its own class. Initial node deployment relies on a circle pattern (Circle distribution model) with the central communication hub in the center and attendees around it. Network edges, Sinalgo's abstraction of communication links between nodes that are within communication range, are created only between attendees and the central communication hub. Since we consider the conference's wireless network to be reliable, we do not use interference, we use reliable message delivery (ReliableDelivery reliability model) and constant transmission time for messages (ConstantTime message transmission model in Sinalgo) equal to 1 round.

Attendees' mobility is implemented according to two different models. The first model is Sinalgo's standard RandomWayPoint, which moves each node to a randomly selected waypoint and then waits at that point for a certain amount of time, before choosing another waypoint and repeating the same procedure. Both the movement's speed and the waiting time are determined according to standard Sinalgo distributions (Gaussian and Poisson, respectively). The second model, which we implemented based on the previous model, is called *Random Waypoint with Fixed Meeting Points*. It works in a similar fashion to the standard RandomWayPoint model, however, it allows for up to two fixed meeting points to be configured. When it selects a new waypoint, it has a configurable probability of choosing one of the meeting points instead of a random waypoint. Consequently, it is possible to use this model's configuration in order to increase the likehood of attendees with similar interests getting near each other. The central communication hub is placed at the center of the conference grounds and does not move (NoMobility). All the models used in the simulation are summarized in table 2.

| **Model Type** | **Model(s) Used** |
|---|---|
| Mobility (Attendees) | RandomWayPoint |
| | Random Waypoint w/ Meeting Points |
| Mobility (Comm. Hub) | NoMobility |
| Connectivity | StaticConnectivity |
| Distribution | Circle |
| Interference | (not used) |
| Reliability | ReliableDelivery |
| Transmission | ConstantTime (= 1) |

**Table 2: Summary of Sinalgo models used in the simulation.**

We assume that each attendee switches between three different interests: A, B or C. Interests are chosen randomly and last for a random period of time, measured in simulation rounds, after which they may change. The minimum and maximum durations for interests are configured by custom parameters in Sinalgo's standard configuration file; by default they are set to 25 and 40, respectively. It is also possible to configure the timeout ($\Delta t$), measured in simulation rounds, before each entity participating in the reasoning process will need to reevaluate its own context and act accordingly as predicted by the algorithm.

For our simulation we chose to use Sinalgo's standard deployment field, which is 1000x1000, in order to simulate the conference grounds. We consider that an attendee is nearby another attendee when the distance between them is less than 250. In this manner, a group discussion can only happen when enough attendees that share a common interest have gathered within a 250 radius of the attendee who is interested in promoting the discussion. The amount of attendees needed to start a group discussion can be configured by means of establishing the *minimum response bag size*. Moreover, since we are not concerned with short lived Global Context States which might not be detected by the algorithm, we require that attendees must maintain a common interest and stay within range for at least twice the timeout value ($2 * \Delta t$).

We used Sinalgo's standard CustomGlobal class in order to implement a global monitor which detects when the simulation is over, maintains some statistics and tracks the detection delay between reasoning occurs and the simulation finishes. In order to check if the algorithm has finished its execution, it checks at the end of each round if all peers have been notified of the end of the reasoning process (in effect it checks if all peers have received a FINISH message). It also keeps track of all Participating Peers by inspecting them at every round; it checks, for each of them, if its interest matched the interest of the Requesting Peer when it was evaluated and if its distance to the Requesting Peer was within the acceptable range when it was evaluated. Doing so allows it to keep track of how long Participating Peers who share an interest with the Requesting Peer and are near it have stayed that way; thus it can determine, externally to the algorithm, when reasoning should become stable.

In figure 2a we present a screenshot of the standard Sinalgo deployment field already loaded with a central communication hub, identified by the blue square in the middle with the letters AR (for Ambient Reasoner) and 10 attendees, identified by the person icon which display the node's identification number in the middle and the current interest on top. The circle around attendee 2 identifies it as the Requesting Peer and determines the range of attendees considered to be nearby. In figure 2b we present another screenshot of the simulation, this time after reasoning has finished. The Requesting Peer is identified by the green color, while the nearby peers who share a common interest (C, in this case) are identified by the blue color. In figures 3a and 3b we present, respectively, screenshots of running simulations both with and without (two) meeting points; red icons represent peers who are too distant or who do not share an interest with the Requesting Peer (pictured in black), while green icons represent peers who are close enough to the Requesting Peer and share an interest with it. Meeting points, placed at (250,250) and (750,750), are represented by a common meeting point icon with four arrows pointing inwards.

We propose three settings for the simulation. In each setting we work with a fixed number of attendees (20) excluding the one who originates the reasoning process (Requesting Peer) and we measure the amount of time needed for reasoning to stabilize, the total number of point-to-point messages exchanged and the detection delay (the time between reasoning has stabilized and the algorithm finishes its execution). In the simulation, timeouts ($\Delta t$) occur synchronously every 10 rounds; this effectively means that every 10 rounds each participating entity will reevaluate its context and act accordingly. A brief description of each setting follows.

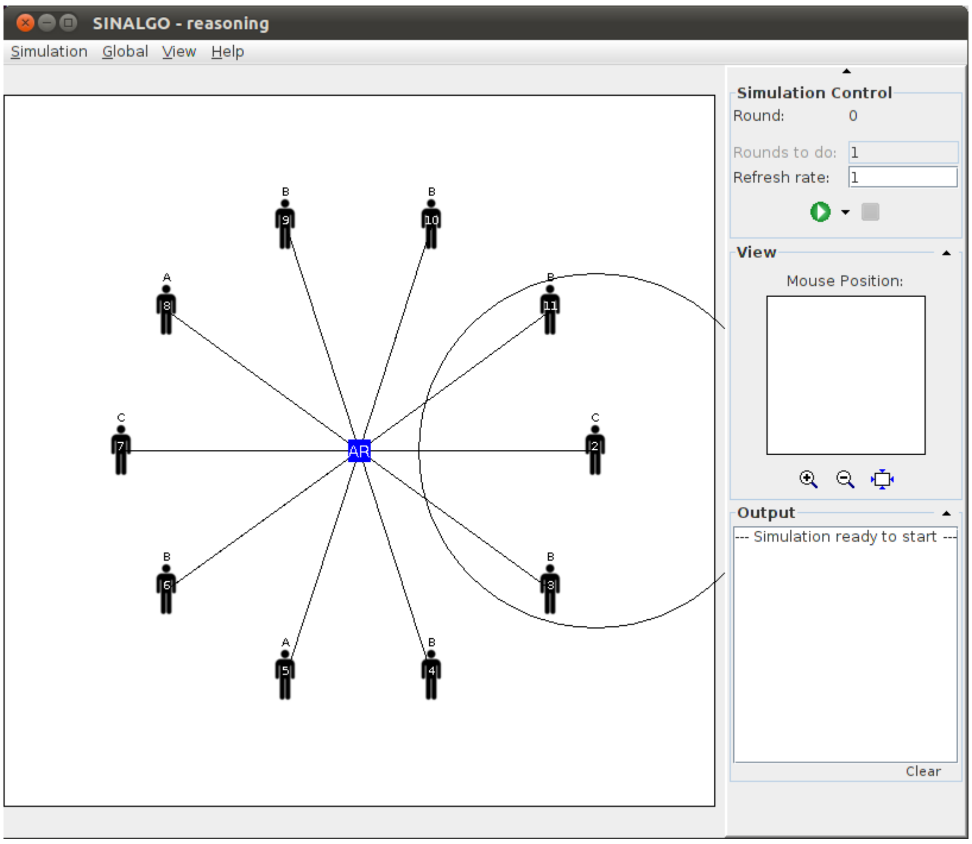
In the first setting, we vary the minimum response bag size based on percentages, in steps of 5% and up to 35%, of the number of attendees (corresponding to response bag sizes from 1 to 7) and the mobility model. We experiment both with the standard RandomWayPoint mobility model, varying the movement's speed, and with the custom Random Waypoint with Fixed Meeting Points model, varying the amount of fixed meeting points available at the conference grounds, but always with a 50% chance of choosing a fixed meeting point instead of a random waypoint. In this setting we are interested in observing how the mobility models, as well as the relation between the number of attendees and the minimum response bag size, influence the simulation. Table 3 summarizes the key variables of the first simulation setting.

In the second setting, we vary the range (minimum and maximum values) that defines how long a local context (interest) lasts based on the amount of time required for a Global Context State to be considered stable by the algorithm ($2 * \Delta t$, where $\Delta t = 10$). In this setting we are interested in observing how much local context volatility influences the reasoning process, in particular in terms of the time taken to finish reasoning and the amount of messages exchanged. We use both the Random Waypoint with Fixed Meeting Points mobility model, with one and two meeting points, and the standard RandomWayPoint model. Table 4 summarizes the key variables of the second simulation setting.

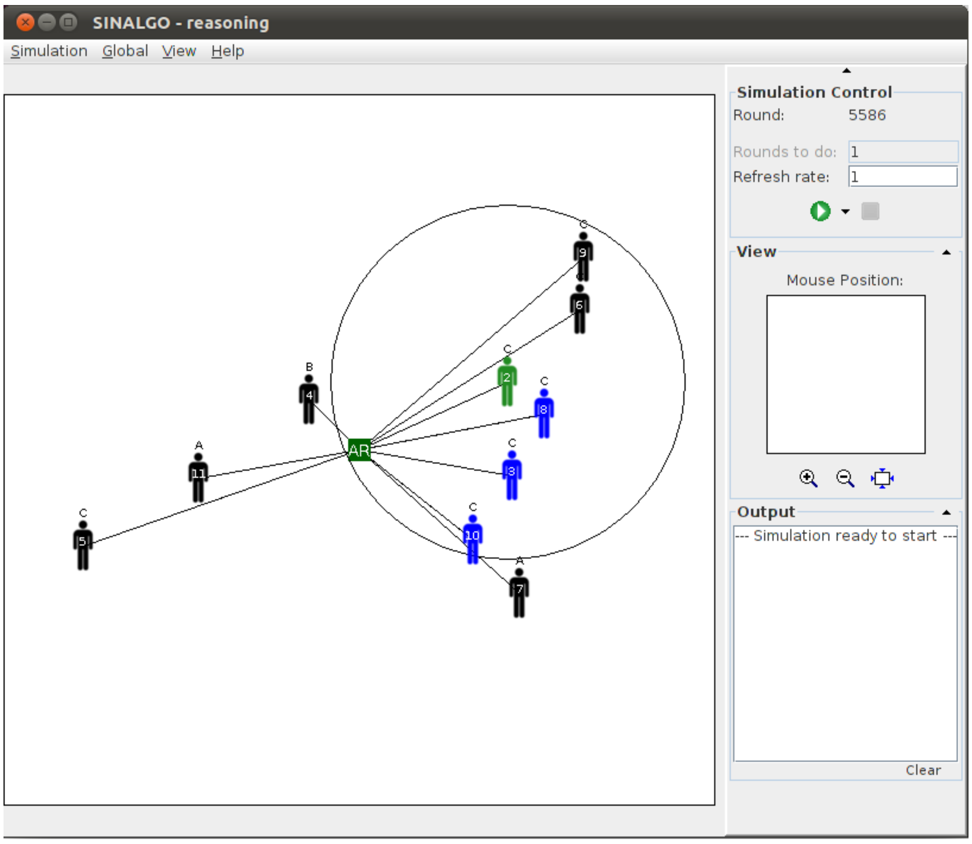
In the third setting, we vary the probability of attendees heading to a meeting point, from 25% to 75%. In this setting we are interested in observing how much the probability of attendees heading to meeting points and the existence of meeting points influence the reasoning process. We stick to the Random Waypoint with Fixed Meeting Points mobility model, with both one and two meeting points. Table 5 summarizes the key variables of the second simulation setting.

## 4.3 Results

In this section we present the results for the simulation settings described in section 4.2. Each variation of a setting was executed 10 times and the numbers shown in this section correspond to the rounded average of these executions. We present, for each setting, the simulation duration (measured in Sinalgo rounds) and the number of messages ex-

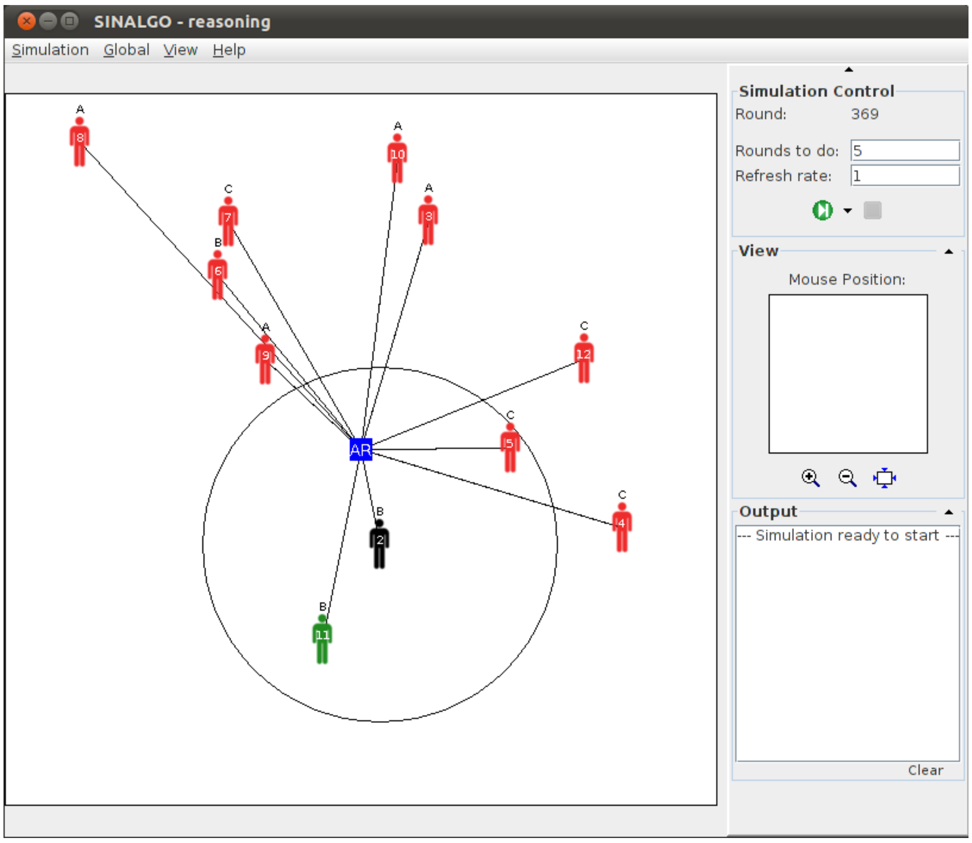


(a) Reasoning about to start.



(b) Reasoning finished.

**Figure 2: Screenshots of a starting simulation and of a finished simulation in Sinalgo.**



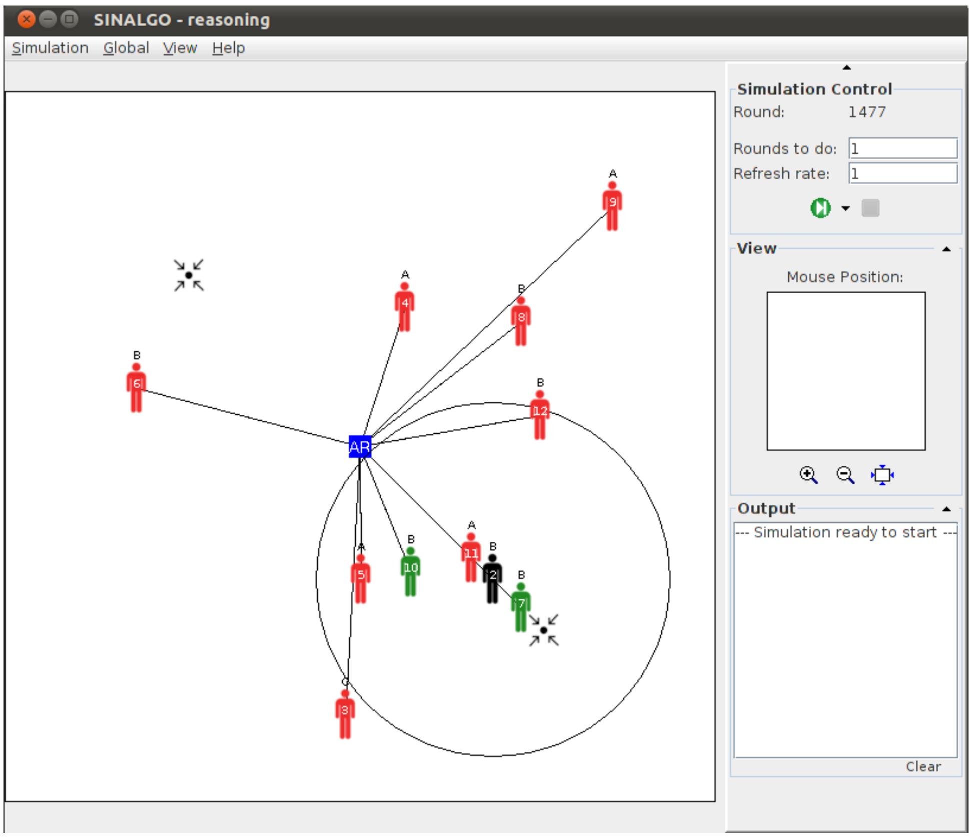
Round: 369

Rounds to do: 5

Refresh rate: 1

```
--- Simulation ready to start ---
```

(a) Reasoning without meeting points.



Round: 1477

Rounds to do: 1

Refresh rate: 1

```
--- Simulation ready to start ---
```

(b) Reasoning with two meeting points.

**Figure 3: Screenshots of running simulations in Sinalgo.**

| Variable | Possible Values | Parameters |
|---|---|---|
| Attendees | 20 | n/a |
| Min. Response Bag Size | 1, 2, 3, 4, 5, 6, 7 | n/a |
| LocalContextDuration | (random) | min=25, max=40 |
| Mobility Model | RandomWayPoint ″slow″ | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; |
| | RandomWayPoint ″fast″ | Speed distribution=Gaussian, mean=10, variance=2; WaitingTime distribution=Poisson, lambda=20; |
| | Random Waypoint w/ One Meeting Point | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; GoToMeetingPointProb value=0.5; NumMeetingPoints value=1; MeetingPoint1 x=250.0 y=250.0; |
| | Random Waypoint w/ Two Meeting Points | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; GoToMeetingPointProb value=0.5; NumMeetingPoints value=2; MeetingPoint1 x=250.0, y=250.0; MeetingPoint2 x=750.0, y=750.0; |

**Table 3: Summary of key variables for the first simulation setting.**

changed.

When accounting for messages exchanged during the algorithm's execution, we separate point-to-point messages (i.e., messages sent directly from specific peers to the Ambient Reasoner or in the opposite direction) and broadcast messages (i.e., messages sent from the Ambient Reasoner to all peers). We choose not to present the total number of broadcast messages, as that number is rather deterministic: the Ambient Reasoner only does a broadcast once it receives an EVAL message from the Requesting Peer; the Requesting Peer only sends an EVAL message once it timeouts; thus, we expect broadcasts to occur in regular timeout intervals, which means the total number of broadcasts can be estimated by dividing the simulation duration by the timeout. Indeed, our results showed that assumption to be true.

We also choose not to present the detection delays observed during the simulations. The detection delay is the number of rounds between the moment that the Global Context State becomes stable (i.e., there are enough replies with count $\geq 2 * \Delta t$ ) and the algorithm finishes its execution. Our results showed that it is a constant value (3 rounds), which corresponds to the time needed for the Participating Peers to send their last REPLY message to the Ambient Reasoner and for the Ambient Reasoner to send the FINISH and RESULT messages.

For each result we also calculated the standard deviations, which are not shown in this paper. We observed high standard deviation for the majority of the results, mostly within 50% to 100% of the corresponding mean. We believe this is a consequence of the

| Variable | Possible Values | Parameters |
|---|---|---|
| Attendees | 20 | n/a |
| Min. Response Bag Size | 4 | n/a |
| LocalContextDuration | (random) | min=10, max=25<br>min=25, max=40<br>min=40, max=55 |
| Mobility Model | RandomWayPoint | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; |
| | Random Waypoint w/ One Meeting Point | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; GoToMeetingPointProb value=0.5; NumMeetingPoints value=1; MeetingPoint1 x=250.0 y=250.0; |
| | Random Waypoint w/ Two Meeting Points | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; GoToMeetingPointProb value=0.5; NumMeetingPoints value=2; MeetingPoint1 x=250.0, y=250.0; MeetingPoint2 x=750.0, y=750.0; |

**Table 4: Summary of key variables for the second simulation setting.**

random variables used in the simulations. Furthermore, we believe the high standard deviations do not weaken the conclusions we draw throughout this section based on the simulation results.

### 4.3.1 Simulation Setting 1

Table 6 presents the simulation results for the first setting, where we vary the minimum response bag size and the mobility model. The results demonstrate some of the algorithm's characteristics.

First, and perhaps more obvious, it is clear that as we increase the minimum response bag size, the longer the reasoning process lasts. This is expected, as a greater minimum response bag size demands that more attendees have a similar interest and are near each other. Figure 4 depicts how the minimum response bag size influences the duration of the simulation. It shows that as we linearly increase the minimum response bag size, the duration of the simulation increases exponentially. This implies not only that the minimum response bag size is a key factor for the performance of the algorithm, but also that the algorithm is better suited for reasoning among small groups of peers.

Concerning the mobility models, the results show that the faster the attendees move, the longer the reasoning process lasts, as it is more difficult to get the required number of attendees with a similar interest in the same area. Moreover, the results show that the introduction of meeting points help to promote colocation of attendees, increasing the likehood of attendees with similar interests getting near each other and thus reducing the duration of the reasoning process. This can also be observed in figure 4, which shows

| Variable | Possible Values | Parameters |
|---|---|---|
| Attendees | 20 | n/a |
| Min. Response Bag Size | 4 | n/a |
| LocalContextDuration | (random) | min=25, max=40 |
| | Random Waypoint w/ One Meeting Point | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; GoToMeetingPointProb value=0.25, 0.50, 0.75; NumMeetingPoints value=1; MeetingPoint1 x=250.0 y=250.0; |
| | Random Waypoint w/ Two Meeting Points | Speed distribution=Gaussian, mean=5, variance=1; WaitingTime distribution=Poisson, lambda=20; GoToMeetingPointProb value=0.25, 0.50, 0.75; NumMeetingPoints value=2; MeetingPoint1 x=250.0, y=250.0; MeetingPoint2 x=750.0, y=750.0; |

**Table 5: Summary of key variables for the third simulation setting.**

how having a single meeting point mostly results in a shorter reasoning process than having two meeting points, as attendees will likely gather at a single location and thus the probability that attendees with similar interests will get near each other increases.

Another expected result is that the longer the simulation lasts, the more messages need to be exchanged between the Participating Peers and the Ambient Reasoner. Figure 5 depict how the simulation duration influences the number of point-to-point messages sent. As it can be observed, the amount of point-to-point messages exchanged tends to hold a linear relation to the duration of the simulation.

### 4.3.2 Simulation Setting 2

Table 7 presents the simulation results for the second setting, where we vary the local context volatility; contrary to the previous setting, here we use a constant number of attendees (20) and a minimum response bag size (4). The results for this setting allow us to observe different characteristics of the proposed algorithm.

Probably the most interesting pattern which can be observed is that less volatile (longer lasting) local contexts result in shorter reasoning processes than more volatile local contexts. This is most likely due to the fact that the more stable the local contexts are, the more time (and thus the greater the chance) there is that attendees with similar interests will get near each other at some point in time. Figure 6 depicts how local context volatility influences the duration of the simulation. Naturally, consistent with the first setting's results, the number of exchanged point-to-point messages also increases as the duration of the simulation increases.

### 4.3.3 Simulation Setting 3

Table 8 presents the simulation results for the second setting, where we vary the probability of attendees heading to meeting points; we use a constant number of attendees (20)

| **Mobility Model** | **Min. Resp. Bag Size** | **Simulation Duration** | **Messages Point-to-Point** |
|---|---|---|---|
| RandomWayPoint ”slow” | 1 | 32 | 18 |
| | 2 | 224 | 142 |
| | 3 | 460 | 306 |
| | 4 | 1,745 | 1,194 |
| | 5 | 22,215 | 15,081 |
| | 6 | 159,169 | 108,496 |
| | 7 | 1,465,423 | 1,000,618 |
| RandomWayPoint ”fast” | 1 | 67 | 45 |
| | 2 | 187 | 142 |
| | 3 | 1,808 | 1,319 |
| | 4 | 8,225 | 6,002 |
| | 5 | 130,747 | 95,512 |
| | 6 | 984,876 | 719,306 |
| | 7 | 21,931,631 | 16,022,897 |
| Random Waypoint w/ One Meeting Point | 1 | 35 | 21 |
| | 2 | 86 | 60 |
| | 3 | 187 | 143 |
| | 4 | 293 | 229 |
| | 5 | 2,239 | 1,635 |
| | 6 | 15,055 | 11,216 |
| | 7 | 61,588 | 45,817 |
| Random Waypoint w/ Two Meeting Points | 1 | 32 | 19 |
| | 2 | 107 | 78 |
| | 3 | 130 | 92 |
| | 4 | 555 | 423 |
| | 5 | 4,170 | 3,040 |
| | 6 | 27,096 | 19,968 |
| | 7 | 145,695 | 107,353 |

**Table 6: Results for simulation setting 1.**

| **Mobility Model** | **Context Duration Range** | **Simulation Duration** | **Messages Point-to-Point** |
|---|---|---|---|
| RandomWayPoint | 10-25 | 15,294 | 13,608 |
| | 25-40 | 3,325 | 2,287 |
| | 40-55 | 1,948 | 1,132 |
| Random Waypoint w/ One Meeting Point | 10-25 | 2,149 | 2,033 |
| | 25-40 | 497 | 373 |
| | 40-55 | 355 | 219 |
| Random Waypoint w/ Two Meeting Points | 10-25 | 3,525 | 3,271 |
| | 25-40 | 1,137 | 841 |
| | 40-55 | 637 | 405 |

**Table 7: Results for simulation setting 2.**

and a minimum response bag size (4). The results for this setting allow us to observe yet other characteristics of the proposed algorithm.

The results for this setting clearly show that adding meeting points to the simulation result in shorter reasoning processes, as depicted in figure 7. Moreover, it shows

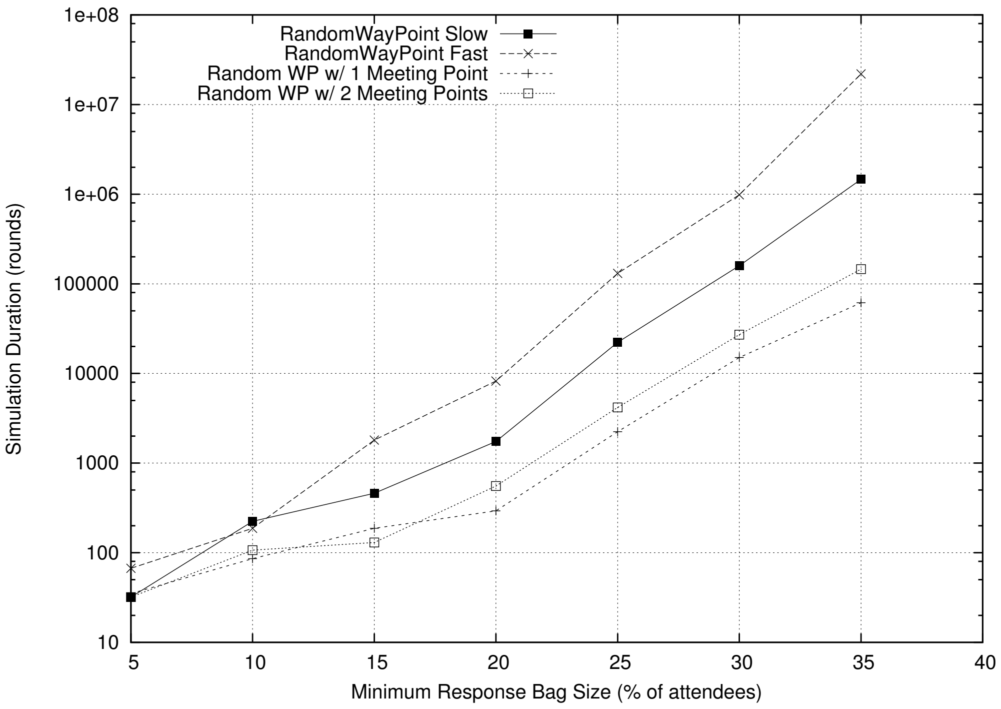


**Figure 4: Minimum response bag size influence on the duration of the simulation for setting 1.**

| Mobility Model | Probability of Heading to Meeting Point | Simulation Duration | Messages Point-to-Point |
|---|---|---|---|
| Random Waypoint w/ One Meeting Point | 25% | 1964 | 1383 |
| | 50% | 470 | 358 |
| | 75% | 219 | 179 |
| Random Waypoint w/ Two Meeting Points | 25% | 2057 | 1438 |
| | 50% | 1934 | 1402 |
| | 75% | 560 | 429 |

**Table 8: Results for simulation setting 3.**

that adding a single meeting point shortens the reasoning process more than adding two meeting points, which is also consistent with the previous settings. This is especially true as we increase the probability of attendees heading to meeting points.

## 5 Conclusion

In this paper, we have proposed a distributed algorithm for detecting Global Context States among an arbitrary set of mobile peers, presented its applicability for cooperative reasoning for pervasive applications, and have conducted several simulation experiments to evaluate the performance of the algorithm in different mobility and context volatility scenarios.

The obtained results show that the convergence time of the algorithm grows exponentially with the percentage of peers that are required to contribute to the Global Context State (i.e. the relative size of *response_bag*), but is not so much sensitive to context volatil-

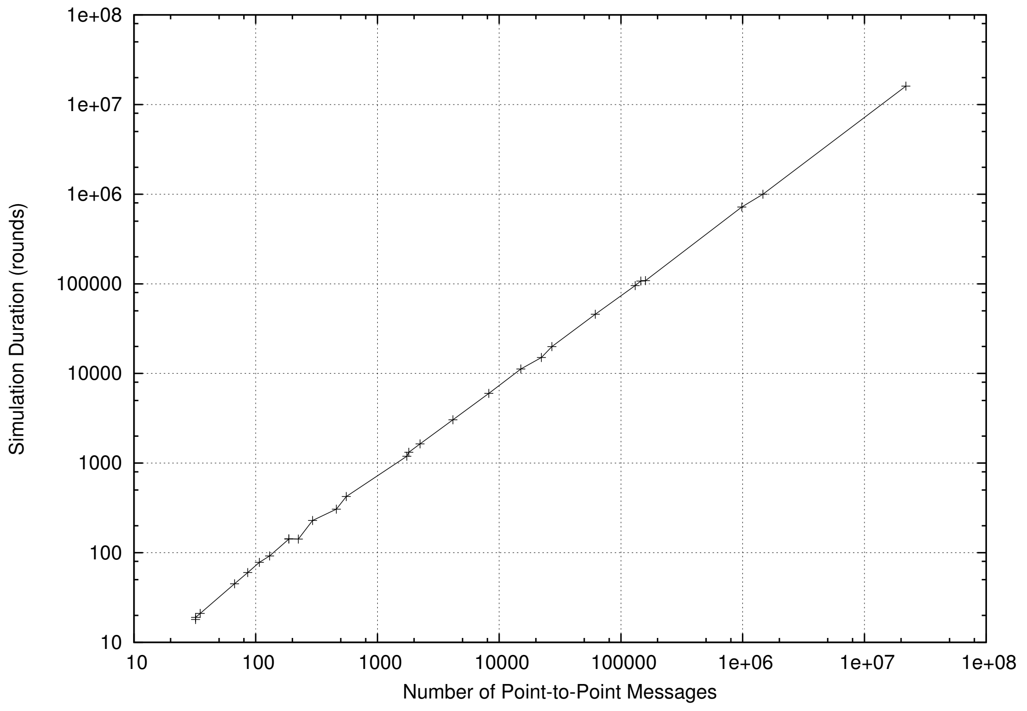


**Figure 5: Simulation duration influence on the number of exchanged point-to-point messages in setting 1.**

ity. Moreover, the results show that with more regular mobility patterns, e.g. with some fixed "meeting points", the convergence time drops significantly, and that the communication complexity is proportional to the convergence time, making it feasible for such scenarios with less mobility entropy. We have also identified that the detection delay is very small and almost constant, suggesting that Global Context States will be informed timely to users. Collectively, theses results suggest that the approach can be applied in practice in situations where the percentage of contributing peers is less than 35% of the total number of peers, when there exists some user clustering points, such as meeting points or coffee tables, and the Global Context State is defined by a few context variables, which do not change very frequently.

A limitation of our algorithm is the fact that it requires a careful calibration of the timer period (i.e. $\Delta t$), which is inherently dependent on the application domain. However, once the minimum period of stability of a Global Context State to be inferred is determined, then our algorithm does a fairly good job in detecting it. Another point of criticism may be the premise that all peers must adopt exactly the same timer periodicity. However, if we assume that each peer is expected to run the same client program so as to have the ability to perform the decentralized reasoning, then this client would of course use a common timer periodicity. Also, in regard to the assumption that local clocks don't drift from each other, we believe that current processor clock technology is already capable of guaranteeing this property for periods of time which exceed, by large, the time scale of the expected time of use of our system.

It is worth noting also that in our current implementation of the algorithm (used in the simulations) the peers and the Ambient Reasoner perform only the detection of the specific Global Context State of the small conference scenario (Section 4.2). Hence, our implementation does not yet support general purpose reasoning of DL rules - expressing Global Context States - nor the rule splitting and distribution process among the peers. Hence, as part of future research we plan to introduce DL rule processing engines at the

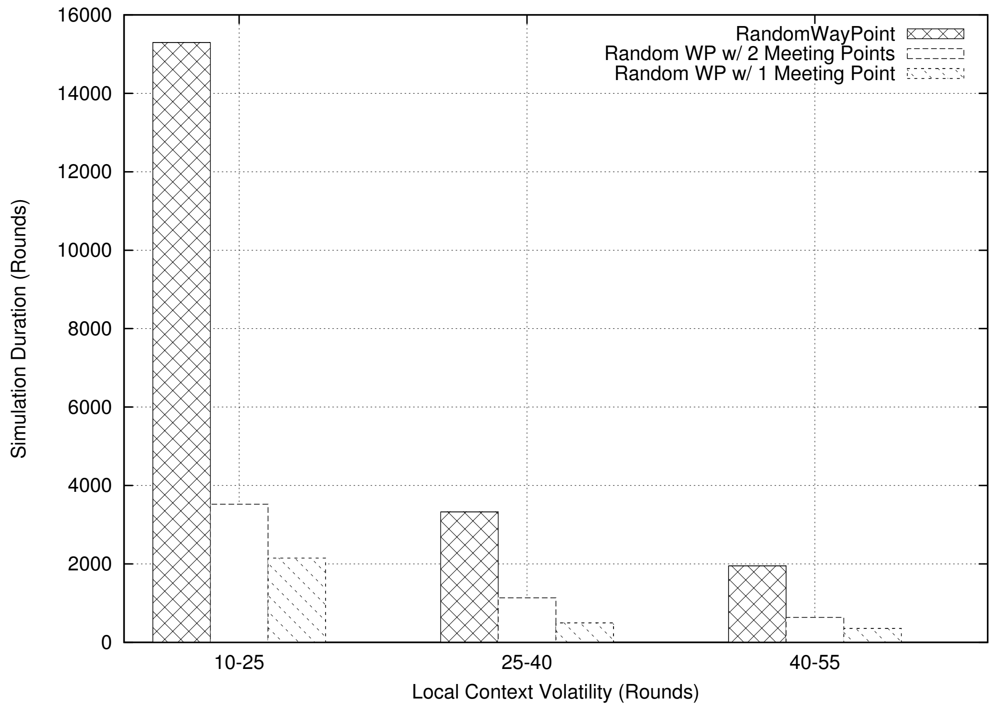


**Figure 6: Local context volatility influence on the duration of the simulation.**

peers, and evaluate the performance of the complete reasoning algorithm.

## Future work

This work is just a first step towards a decentralized reasoning approach, and we envisage several possible lines of future work, both as improvements of the algorithm, as well as in regard to the reasoning approach, as a whole.

Regarding the algorithm, in our simulations we only evaluated the convergence of the algorithm using two simple local context variables: location and a small set of (three) interests. It would be interesting, though, to evaluate the convergence on scenarios where the global context depends on more local context variables, possibly with different volatilities. Still concerning the algorithm's convergence, we showed that the minimum response bag size appears to be central to that matter, however we did not delve into trying to refine and optimize the convergence time, which could result in some improvement to the algorithm. Also, we believe some of the algorithm's premisses could be weakened in order to allow for further testing and adjustments to the algorithm. Examples of weakened premisses include different evaluation timeouts at each peer and local clocks with a small drift. Ultimately, these adjustments could result in an asynchronous algorithm.

Regarding the reasoning approach, we believe it could be interesting to define a fully decentralized approach. In this approach, the Ambient Reasoner could become a role. Thus, instead of relying on a single pre-determined peer, any peer with sufficient resources and low mobility could assume the Ambient Reasoner role throughout the reasoning process.

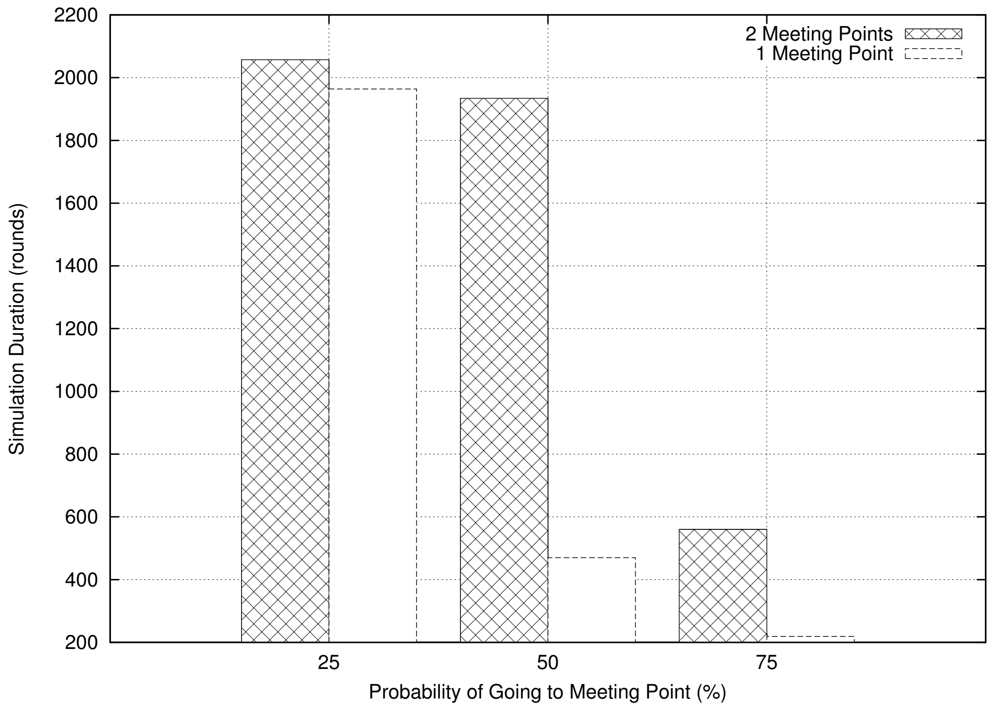


**Figure 7: Probability of going to a meeting point influence on the duration of the simulation when there are one and two meeting points.**

## References

[1] LUGER, G. F.. **Artificial Intelligence: Structures and Strategies for Complex Problem Solving**. Addison Wesley, 2005.

[2] OREN, E.; KOTOULAS, S.; ANADIOTIS, G.; SIEBES, R.; TEN TEIJE, A. ; VAN HARMELEN, F.. **Marvin: Distributed reasoning over large-scale semantic web data**. Journal of Web Semantics, 7(4):305–316, 2009.

[3] SCHLICHT, A.; STUCKENSCHMIDT, H.. **Towards distributed ontology reasoning for the web**. 2008 IEEEWICACM International Conference on Web Intelligence and Intelligent Agent Technology, 1:536–539, 2008.

[4] SERAFINI, L.; TAMILIN, A.. **DRAGO: Distributed reasoning architecture for the semantic web**. In: Gómez-Pérez, A.; Euzenat, J., editors, THE SEMANTIC WEB: RESEARCH AND APPLICATIONS, volumen 3532 de **Lecture Notes in Computer Science**, p. 155–162. Springer Berlin / Heidelberg, 2005.

[5] SCHILIT, B.; ADAMS, N. ; WANT, R.. **Context-aware computing applications**. In: IN PROCEEDINGS OF THE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, p. 85–90. IEEE Computer Society, 1994.

[6] NURMI, P.; FLORÉEN, P.. **Reasoning in context-aware systems**. Technical report, University of Helsinki, Department of Computer Science, December 2004. MobiLife.

[7] BETTINI, C.; BRDICZKA, O.; HENRICKSEN, K.; INDULSKA, J.; NICKLAS, D.; RANGANATHAN, A. ; RIBONI, D.. **A survey of context modelling and reasoning techniques**. Pervasive Mob. Comput., 6:161–180, April 2010.

[8] BIKAKIS, A.; ANTONIOU, G.. **Distributed Reasoning with Conflicts in an Ambient Peer-to-Peer Setting**. In: Mühlhäuser, M.; Ferscha, A. ; Aitenbichler, E., editors,

CONSTRUCTING AMBIENT INTELLIGENCE, volumen 11 de **Communications in Computer and Information Science**, p. 24–33. Springer Berlin Heidelberg, 2008.

[9] ZADEH, L.. **Fuzzy logic**. IEEE Computer, 21(4), 1988.

[10] World Wide Web Consortium (W3C). **OWL 2 – Web Ontology Language**. http://www.w3.org/TR/owl2-overview.

[11] BAADER, F.. **Description logics**. In: REASONING WEB: SEMANTIC TECHNOLOGIES FOR INFORMATION SYSTEMS, 5TH INTERNATIONAL SUMMER SCHOOL 2009, volumen 5689 de **Lecture Notes in Computer Science**, p. 1–39. Springer–Verlag, 2009.

[12] RICH, E.; KNIGHT, K.. **Artificial Intelligence**. Tata McGraw Hill Education Private Limited, 2010.

[13] AMIGONI, F.; GATTI, N.; PINCIROLI, C. ; ROVERI, M.. **What planner for ambient intelligence applications?** IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 35(1):7–21, Jan 2005.

[14] BABAOGLU, Ö.; MARZULLO, K.. **Consistent global states of distributed systems: Fundamental concepts and mechanisms**. Technical Report UBLCS-93-1, U. Bologna, Jan 1993.

[15] VITERBO, J.; ENDLER, M.. **Decentralized reasoning in ambient intelligence**. Software Engineering Workshop (SEW), 2009 33rd Annual IEEE, 0:115–124, 2009.

[16] FILHO, J. V.. **Decentralized Reasoning in Ambient Intelligence**. PhD thesis, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), 2009.

[17] PADOVITZ, A.; LOKE, S. W. ; ZASLAVSKY, A. B.. **Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems**. IEEE Transactions on Systems, Man, and Cybernetics, Part A, 38(4):729–742, 2008.

[18] GU, T.; PUNG, H. K. ; ZHANG, D.. **Peer-to-Peer Context Reasoning in Pervasive Computing Environments**. In: PROCEEDINGS OF THE 2008 SIXTH ANNUAL IEEE INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS, p. 406–411, Washington, DC, USA, 2008. IEEE Computer Society.

[19] SERAFINI, L., T. A.. **Drago: Distributed reasoning architecture for the semantic web**. In: PROCEEDINGS OF ESWC 2005, número 3532 em LNCS. Springer, 2005.

[20] CHATALIC, P.; NGUYEN, G. H. ; ROUSSET, M. C.. **Reasoning with inconsistencies in propositional peer-to-peer inference systems**. In: PROCEEDING OF THE 17TH EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE (ECAI), p. 352–356. IOS Press, 2006.

[21] MARZULLO, K.; NEIGER, G.. **Detection of global state predicates**. In: PROC. 5TH INTL. WORKSHOP ON DIST. ALGORITHMS (WDAG), p. 254–272. Springer-Verlag, 1991.

[22] SCHWARZ, R.; MATTERN, F.. **Detecting causal relationships in distributed computations: In search of the holy grail**. Distributed Computing, p. 149–174, 1994.

[23] GROUP, E. Z. . I. . T. . D. C.. **Sinalgo – simulator for network algorithms**. Website. http://disco.ethz.ch/projects/sinalgo.